Malware Evolution

Unraveling Malware Genomics: Synergistic Approach using Deep Learning and Phylogenetic Analysis for Evolutionary Insights

M.Sc. Thesis

Akash Amalan

Delft University of Technology

Malware Evolution

Unraveling Malware Genomics: Synergistic Approach using Deep Learning and Phylogenetic Analysis for Evolutionary Insights

by

Akash Amalan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on May 24, 2024

Student Number	4682505
Project Duration	November, 2023 - May 2024
Thesis Committee:	Prof. G. Smaragdakis, TU Delft
	Dr. Tom Viering, TU Delft
	Dr. Harm Griffoen, TU Delft

TU Delft

Preface

During my studies, I often found myself torn between pursuing courses in cybersecurity and artificial intelligence. Fortunately, I had the opportunity to attend Prof. Georgios Smaragdakis's lectures on network security, which I thoroughly enjoyed. When I approached him about combining my passion for AI with cybersecurity, he enthusiastically supported the idea and even accepted my proposal to use AI to trace the evolution of malware samples. Despite his initial doubts, he embraced the challenge.

I was fortunate to have Dr. Tom Viering, an expert in deep learning, on my committee as a daily supervisor. Regardless of the time, he was always available to address my questions, whether they pertained to machine learning or phylogenetics. His unwavering support and willingness to engage in discussions were invaluable, and I am deeply grateful for his guidance, which was instrumental to the success of my work. I would also like to thank Stefan Op de Beek and Dr. Harm Griffioen for reading my thesis and being part of the committee.

I would also like to extend my gratitude to VirusTotal for allowing me to use some of their samples for research and providing metadata for all the samples. Additionally, I appreciate AnyRun for granting me a license to perform dynamic analysis on their platform using their API. Without these tools, validating my approach would have been impossible. Finally, I would like to thank my parents for their constant support, my close friends Daniel, Luke, and Tony for their unwavering support, and Karthik and Yuqian for their constructive criticism of the ”blackbox” nature of my approach.

Akash Amalan
Delft, May 2024

Abstract

The rapid advancement of artificial intelligence technologies has significantly increased the complexity of polymorphic and metamorphic malware, presenting new challenges to cybersecurity defenses. Our study introduces a novel bioinformatics-inspired approach, leveraging deep learning and phylogenetic analysis to understand the evolutionary dynamics of such malware. By analyzing a dataset of 103,883 malware samples, we transformed extracted features using pseudo-static, dynamic, and image analyses into embeddings with deep learning techniques, combining them into what we refer to as the ”genome” of malware. These combined embeddings were used to construct phylogenetic trees employing the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) and the Neighbor-Joining (NJ) method.We were the first to utilize OpenAI’s state-of-the-art embeddings for converting pseudo-static and dynamic features into embeddings. In addition, we discovered that transfer learning with ResNet-50 is highly effective compared to traditional CNNs, producing better image embeddings that outperform others in terms of classification accuracy.

We also introduced new validation techniques for phylogenetic trees, making use of VirusTotal timestamps and embedding drift analysis. These methods confirmed that the NJ method was more accurate. Furthermore, we developed techniques to simplify the analysis of these extensive phylogenetic trees, enabling efficient derivation of relationships within and between malware families. The insights from our NJ-built phylogenetic trees closely align with public data and lay a foundation for generating evolutionary-informed signatures that enhance tailored detection strategies. Our method has significantly expedited the process of identifying connections among 538 malware families by dramatically reducing the timeframe from months or years to just weeks – much faster than traditional reverse engineering approaches for tracing malware evolution.

Contents

List of Figures

List of Tables

1 Introduction

1.1. Motivation

The ongoing struggle against malware is commonly compared to a perpetual ”cat and mouse game”, in which security measures often find themselves in a position of catching up to newly emerged malicious variants. These variants are usually inherently polymorphic, metamorphic, or both, consistently undergoing evolution to present new challenges to security analysts [1]. Traditionally, efforts to counteract such malware have relied heavily on static analysis [2], an approach that focuses on evaluating the components of a file. Static analysis led to signatures using attributes, such as the information contained within Portable Executable (PE) headers, the organizational layout of code, and entropy.

While these signatures are helpful, their effectiveness depends on accessing the original, unaltered code. However, as malware developers increasingly use packing techniques to obscure the original code and functionalities, signatures generated from static analysis may prove insufficient [3]. To address packing, malware researchers have turned to dynamic analysis. Dynamic analysis has proven highly effective by adding precise behavioural indicators as signatures. These signatures are widely used in most antivirus software; for example, they are implemented in Avira and Windows Defender disguised as Yara rules [4].

Over the past decade, researchers have shown a growing interest in utilizing machine learning methodologies to classify malware based on static and dynamic features. Several studies [5] [6] [7] [8] have used decision trees, random forests, and gradient boosting to achieve state-of-the-art accuracy in malware classification. Likewise, others have delved into deep learning architectures [9] [10] [11] such as deep belief networks and LSTMs. Interestingly, some endeavours [12] [13] have focused on integrating computer vision into malware detection by transforming malware into images. These advancements have not only improved classification accuracy but also offered a more comprehensive understanding of the unique traits exhibited by malware families.

Yet, these approaches frequently overlook the evolutionary characteristics of malware, typically lagging in the ongoing ”cat and mouse game”. Instead, understanding and mapping the evolutionary trends of malware could benefit both proactive and reactive security tactics.

Firstly, modelling malware evolution enables specialists to develop predictive frameworks that predict potential mutations. This forward-thinking strategy permits defenders to improve or adjust their protective measures before the arrival of new variants. As a result, defences that are successful against parent strains could also be effective against their offspring, attributed to the shared genetics. Secondly, identifying the genealogical ties among malware families can enhance the formulation of detection signatures. By embedding features unique to a malware lineage within these signatures, it’s possible to protect against later variants, especially those that maintain key functionalities or code fragments from their forebears. This approach may subsequently reduce false positives, addressing a common challenge in malware detection.

Grasping the nuanced differences between a parent malware and its subsequent variants also enables the development of tailored detection strategies. Such strategies can be adjusted to recognize the slight modifications characteristic of new malware variants, significantly increasing the precision and reliability of detection mechanisms. Lastly, studying the evolution of malware over time provides valuable insights into the tactics, techniques, and procedures used by cyber adversaries. Understanding these strategies is vital not only for creating effective technical defences against malware but also for developing broader security strategies that can anticipate and counteract evolving threats. This holistic approach may allow organizations to stay one step ahead in the ongoing battle against cyber threats.

Beyond the potential benefits, a critical aspect that is often overlooked is the impact of Artificial Intelligence on malware development, particularly the challenge it presents in terms of scale. Generative models[14] [15] [16], including ChatGPT, have facilitated the creation of scalable and polymorphic malware. For example, BlackMamba[17] employs a large language model(LLM) to produce a keylogger that evolves with each iteration, eluding predictive antivirus software. Additionally, the emergence of ”zero-click” worms[18] is especially troubling; these worms propagate without any user interaction. Researchers such as Ben Nassi from Cornell Tech have illustrated how these worms exploit generative AI systems to autonomously conduct malicious activities[19]. Ben Nassi notes that such worms can harvest and utilize sensitive data to compromise new hosts through ordinary activities like email replies, subsequently storing and spreading the data without activating standard defenses.

Similarly, Dark AI, which repurposes LLMs for malicious purposes, significantly accelerates the creation of complex malware [20] [21]. This technology automates the coding process, quickly producing malware variants that exploit new vulnerabilities. An example of Dark AI at work is DarkBert, a variant of Google Bert tailored for malicious use [22] [23]. DarkBert integrates with other malware components to dynamically generate attack code, orchestrating sophisticated multi-stage breaches and adapting attack strategies to be highly effective against current cybersecurity measures. Moreover, Dark AI enables malware to adapt during deployment based on its environment. For example, if malware senses it is being analyzed in a virtual machine or sandbox, it can change its behavior to avoid detection or even disable itself to conceal its mechanisms. These advancements indicate that malware creators are capable of producing malware at scale. Therefore, it is essential to track the evolution of malware on a similar scale. Traditional methods like reverse engineering often take months or even years, making them insufficient for keeping pace with the rapid evolution of malware.However, as shown later in this study, utilizing techniques from AI and Bioinformatics can improve our ability to trace these developments effectively and at scale.

Despite these concerns, there has been little to no research on modelling the evolution of malware. A frequently underestimated analogy lies in the parallels between biological viruses and computer malware, as both exhibit mutation and inheritance of traits from their progenitors. Beyond these basic similarities, biological viruses and computer malware engage in adaptive responses to environmental pressures, such as host immunity or cybersecurity measures, respectively. Additionally, they both can exhibit rapid rates of evolution, facilitated by their ability to generate numerous variants in a short period. This rapid evolution is often driven by the competitive necessity to circumvent defences—biological viruses evolving to breach cellular defences and malware to evade detection software. Understanding these analogies enriches the conceptual framework for modelling malware evolution, suggesting that strategies employed in mapping the evolution of virology could serve a foundation for malware evolution.

1.2. Research question

This paper introduces the use of phylogenetic trees in studying malware evolution. Constructing phylogenetic trees involves input sequences or a similarity matrix between species. In our method, we first extract representations called embeddings from features collected through pseudo-static, dynamic, and image analyses. These individual embeddings are combined to create a comprehensive yet compact representation, conceptually similar to DNA. This concatenated embedding is used to generate a similarity matrix, which serves as the basis for constructing a phylogenetic tree using Neigbout Joining(NJ) and Unweighted Pair Group Method with Arithmetic Mean(UPGMA).

The primary research question guiding this study is: *How can deep learning be used to trace the evolution of malware families by identifying shared software and behavioural components through static, dynamic, and image analysis methods using phylogenetic trees?*

To address this overarching question, several subquestions will be explored:

1. Which analysis type yields the best embeddings and does combining them improve performance?
2. Are embeddings, merged or individual, useful for downstream tasks?
3. How does our approach on classifying malware based on images compare to previous work?
4. Which phylogenetic tree construction method using distances produces the most accurate representation of malware evolution using VirusTotal timestamps?
5. How can embedding drift analysis be employed as an alternative method for validating phylogenetic trees?
6. How do clusters formed by visualizing malware embeddings with t-SNE, UMAP, and PCA align with lateral(leaf to leaf) distances in a phylogenetic tree built with NJ method?
7. Do outliers alter the topology of a phylogenetic tree constructed using the NJ method by changing the Most Recent Common Ancestor (MRCA)?
8. Do the relationships identified through inter-family analysis using the NJ method correlate with public cybersecurity insights and with the pseudo-static and dynamic features of the malware?
9. Are there any similarities between the inter-family relationships of UPGMA and NJ?

1.3. Contributions

In this work, we make the following contributions:

- We demonstrate how static analysis can be used with dynamic analysis(psuedo- static analysis) to overcome some of the effects of packing.
- We provide a dataset of features derived from pseudo-static analysis of 103,883 malware samples.
- We showcase the application of deep learning to generate embeddings from malware samples. To our knowledge, this is the first study to employ OpenAI's advanced embeddings for transforming malware features into embeddings.
- We explore how the resultant embeddings can be used for downstream tasks such as malware clustering and detection by finetuning only the last layer.
- We implement two main ways, NJ and UPGMA, for constructing phylogenetic trees from distance matrices.
- We introduce new techniques to validate phylogenetic trees using VirusTotal timestamps, as there are currently no known methods to validate phylogenetic trees when applied to malware.
- We introduce methods to simplify the tree, which consists of 103,883 leaves, making it challenging to visualize or derive any relationships. This simplification is crucial for targeted analysis within and between malware families.
- Our method scales effectively, allowing us to derive relationships from 538 families within a matter of weeks.

1.4. Report Structure

The rest of this paper is structured as follows:

Chapter 2, titled Background, explores the use of machine learning in malware analysis and discusses conventional malware analysis techniques. It also examines different methods for constructing phylogenetic trees to set the stage for understanding the subsequent chapters in this paper.In chapter 3, we introduce our dataset MalwareGenome and other public datasets we use.

In Chapter 4, Methodology, we introduce a pipeline that starts with extracting embeddings and ends with constructing phylogenetic trees. This chapter provides in-depth insight into the pipeline, including details about the datasets used and additional information necessary to replicate our work. Chapter 5, Results, presents the outcomes of various experiments aimed at addressing the subquestions outlined in the introduction. Looking ahead to Chapter 6, the Discussion reflects on the implications of our findings in the context of related works. This chapter explores the practicality and limitations of using embeddings and phylogenetic trees to understand malware evolution and proposes future works. Finally, Chapter 7 summarizes the key findings and contributions of our approach.

2 Background

This chapter introduces the foundational concepts essential for understanding malware analysis. We start by examining the different types of malware and the common file types they typically target. Then, we explore how machine learning and deep learning techniques are used to classify malware. Lastly, we highlight various methods of constructing phylogenetic trees that map the evolutionary relationships among species to trace their development and spread.

2.1. Malware Fundamentals

Before we delve into the intricacies of malware analysis, it's essential to have a basic understanding of malware. This section will cover various types of malware and their file formats.

2.1.1. Types Of Malware

The classification of malware into specific categories can be somewhat vague, but literature commonly divides malware into two broad types based on their evasion tactics: polymorphic and metamorphic[1].

Polymorphic Malware

Polymorphic malware poses a major challenge for cybersecurity defenses as it constantly alters a part of its code or signature with each new infection [3]. This type of malware employs different evasion tactics, such as modifying encryption keys and the order of subroutines, to effectively conceal its presence. A critical component of polymorphic malware is its reliance on distinct encryption keys for each instance. These keys are used to encrypt the malware payload, ensuring that each version appears different to antivirus systems. By utilizing these encryption keys and regularly reordering subroutines, the malware creates difficulty for antivirus software in detecting consistent patterns [3]. Importantly, these alterations are automatically carried out by the malware rather than being manually orchestrated.

Metamorphic Malware

Metamorphic Malware differs significantly from its polymorphic counterpart due to its unique evasion methods. Unlike polymorphic malware, which relies on encryption keys to alter parts of its code or signature, metamorphic malware completely rewrites its entire codebase with each iteration [24]. This extensive self-modification leads to fundamentally unique instances upon each infection, rendering traditional detection methods based on established patterns or signatures ineffective.

Metamorphic malware achieves this through the use of sophisticated *metamorphic engines*. These engines enable the malware to translate its code into a temporary representation, modify this representation, and then re-translate it back into machine code. Notably, these transformations do not require the use of encryption keys, which are a hallmark of polymorphic malware techniques.Metamorphic engine employs advanced techniques such as instruction substitution, register renaming, and dead code insertion to significantly alter its code structure, making detection difficult . It also uses control flow alteration and code integration with obfuscation to disrupt standard pattern recognition [3]. These strategies ensure each iteration of the malware is distinct enough to evade traditional signature-based antivirus defenses, posing substantial challenges to cybersecurity measures.

Advanced Evasive Malware

In recent years, malware has been using both polymorphic and metamorphic evasion techniques [25]. These advanced evasive malware variants utilize the adaptability of polymorphic malware, which changes parts of its code or signatures with each infection, and merge it with the metamorphic strategy of completely rewriting their entire codebase without the use of encryption keys.

This strategic and sophisticated integration not only enables the malware to alter superficial aspects of its code to avoid detection but also to fundamentally reconstruct its entire structure. As a result, these malware variants significantly enhance their capability to evade traditional and modern detection systems. Table 2.1 shows recent examples of different types of malware classified by their evasion techniques. For a more extensive list of examples, refer to Appendix A

Table 2.1: Classification of Malware Based on Evasion Techniques

Type	Malware	Description
Polymorphic	Mirai [26]	Known for its devastating DDoS attacks, Mirai's polymorphic variants complicate the detection process, leading to increased infection rates across IoT devices.
Polymorphic	Gafgyt [27]	Specializes in launching DDoS attacks, continuously changing attack vectors and evolving encryption, making it hard to trace and neutralize.
Polymorphic	MooBot [28]	As a direct offshoot of Mirai, exploits vulnerabilities quickly, adapting its scanning and exploitation techniques, infecting a wide array of devices.
Metamorphic	ZMist (ZMistfall) [29]	Integrates into and modifies executable files using advanced metamorphic techniques to rearrange and rewrite its code.
Metamorphic	Win95/Regswap [30]	Targets Windows 95/98 files, swaps code segments to obscure its presence, making detection challenging.
Advanced Evasive	IcedID [31]	A banking Trojan that employs injection techniques and evasion tactics to avoid detection, spreading via malspam campaigns.
Advanced Evasive	Emotet [32]	Initially a banking Trojan, now a sophisticated malware delivery service, known for its rapid spread and delivery of various payloads.
Advanced Evasive	GandCrab [33]	A ransomware-as-a-service that evolved through continuous updates, utilizing both polymorphic and metamorphic techniques.

Note that in our research, we analyze a wide range of malware from both public and private datasets without categorizing them by evasion technique. Our methodology is based on the observation that malware often inherits traits from its predecessors - traits that are crucial to our study and will be further discussed in subsequent chapters. While we acknowledge the categorization of malware, it is primarily to illustrate the diverse evolutionary paths of malware, not as a focal point of our analysis.

2.1.2. File Formats

Our primary focus is on the three most prevalent malware formats: the Portable Executable (PE) format for Windows, the Executable and Linkable Format (ELF) for Unix/Linux systems, and the DOS format for older DOS environments. Including the DOS format is particularly important for tracing the evolution of malware, as some variants display characteristics reminiscent of DOS-based architectures. Each format has its sections and headers, which are essential for the executable's operation, facilitating security analyses and aiding in the identification and breakdown of malware. An overview of important aspects of the different formats are presented in Table 2.2.

Table 2.2: Detailed Components of PE, ELF, and DOS Formats

Aspect	**PE Format**	**ELF Format**	**DOS Format**
Signature	MZ followed by `PE\0\0`	`\x7FELF`	”MZ” Signature
Header	PE Header	ELF Header	DOS Header
MS-DOS Compatibility	MS-DOS 2.0 Compatible EXE Header, Stub Program	Not applicable	Entire format
OEM Information	OEM Identifier and OEM Information	Not applicable	Not applicable
Section Headers	Section Headers for code, data, resources	Section Headers for .text, .data, etc.	Not explicitly defined
Program Headers	Not directly applicable	Program Headers for segment loading	Not applicable
Import/Export Info	Import Table, Export Table	Symbol Table (imports/-exports through dynsym)	Not applicable
Dynamic Linking	Import Table, Export Table, Base Relocations	Dynamic Section, Relocation Sections	Not applicable
Debug Information	Debug Directory	Debug Section	Not standard
TLS Support	Thread Local Storage	Sectional support (e.g., .tdata, .tbss)	Not applicable
Resources	Resources Section	Not standard, managed externally	Not applicable
Symbol Table	COFF Symbol Table	Symbol Table for linking	Not applicable
Relocations	Base Relocations Section	Relocation Sections (.rela, .rel)	Relocation Table
Entry Point	Defined in Optional Header	Entry point in ELF Header	Defined in Header
Raw Data Sections	Code, Data, Resources	.text, .data, .bss, etc.	Program Code and Data
Microsoft COFF	COFF Header, Section Headers, Raw Data	Not applicable	Not applicable
Dynamic Symbol Table	Not directly applicable	Dynamic Symbol Table (dynsym)	Not applicable
32-bit vs. 64-bit	Optional Header (PE32 vs. PE32+)	`e_ident[EI_CLASS]` (ELFCLASS32 vs. ELFCLASS64)	Not applicable
DLL vs. Executable	`Characteristics` field in COFF Header	`e_type` field (ET_DYN vs. ET_EXEC)	Not applicable
Subsystem Information	Specifies subsystem (e.g., Windows GUI or console)	Not applicable	Not applicable
Alignment of Sections	Section alignment details in Optional Header	Section alignment constraints in ELF Header	Not applicable
Permissions	Permissions for code and data sections	Permissions detailed in Program Headers	Not applicable

Continued on next page

Table 2.2 – *Continued from previous page*

Aspect	PE Format	ELF Format	DOS Format
Checksums	Checksum in Optional Header	Optional checksum for integrity	Not applicable
Virtual Address vs. Physical Address	Manages both virtual and physical addressing	Virtual addressing in Program Headers	Not applicable
Compression and Encryption	Supports encrypted or compressed sections	May contain compressed sections	Not applicable
File Extension Association	.exe, .dll, etc.	.o, .so, .elf, etc.	.exe, .com, etc.
Loadable vs. Linkable	Distinguishes between loadable sections	Differentiates loadable from linkable sections	Not applicable
Versioning Information	Version numbers in Optional Header	Version fields in ELF Header	Not applicable
Endianness	Endianness specified in Optional Header	Endianness field in ELF Header (`e_ident[EI_DATA]`)	Not applicable
Built-in Support for Threads	Thread Local Storage details	Support for thread-specific storage in sections	Not applicable
Usage in Industry	Widely used in Windows environments	Common in Unix/Linux systems	Historically used in DOS systems

Portable Executable format(PE)

PE[34] format is the foundation of file execution in Windows, encompassing applications packaged as executables and dynamic link libraries. Designed for versatility, it supports a wide range of applications built for the operating system. The 64-bit PE format structure, outlined in Figure B.1, includes vital components such as the program's code, data sections, and metadata that are crucial for the Windows loader to effectively manage program execution.

At the start of each PE file, there is a distinctive signature sequence acting as an identifier: the `'MZ'` signature signifies its executable nature, while the `'PE\0\0'` signature marks the beginning of the actual PE header. Following this, the Machine field specifies which system architecture or processor type is intended for running that particular executable. The key difference between 64-bit and 32-bit PE formats lies in the optional header. In 32-bit PE files, the optional header's Magic number is 0x10b to ensure compatibility with 32-bit memory address spaces. Conversely, for 64-bit PE files, the Magic number is 0x20b signaling the use of expanded field sizes for addressing, including support for larger memory space. For additional information, please see the first section of Appendix B.

Executable and Linkable Format(ELF)

ELF[35] is the standard for executables in Unix and Linux systems, featuring an ELF Header that is essential for providing metadata to prepare the operating system for execution. This metadata includes the file type—whether it's an executable, a relocatable file, or a shared object file (similar to DLLs in Windows), the machine architecture (specifying whether the file is compiled for 32-bit or 64-bit systems), and the program's entry point (the initial memory address for execution). The header also helps differentiate between executable files (`ET_EXEC`) and shared object files (`ET_DYN`), which are crucial for dynamic linking, akin to DLLs. The ELF signature, represented as `\x7fELF`, clearly marks the file as an ELF format. These details are vital for ensuring the system correctly manages and executes files, with their distinctions visually depicted in Figure B.2, underscoring the ELF format's critical role in system operations. For more details refer to the second section of Appendix B.

Disk Operating System (DOS)

The DOS format[34] stands out for its straightforward design and pivotal role in the historical development of computer software, acting as a fundamental precursor to the more sophisticated Portable Executable (PE) format prevalent in Windows systems. Its unique architecture, designed for DOS environments, is easily identified by the DOS Header, which bears the distinctive "MZ" signature in homage to its developer, Mark Zbikowski. For more details, refer to the last section of Appendix B.

2.2. High-Level Overview of Machine Learning Techniques

This section covers basic machine learning methods, starting with an explanation of model parameters and the processes of training and testing. It also discusses the importance of cross-validation and double cross-validation for fine-tuning hyperparameters and providing a fair evaluation of models. Next, it introduces supervised learning techniques, including logistic regression and k-nearest neighbors, and unsupervised methods such as hierarchical clustering, all of which are used in our study. Finally, it looks at dimensionality reduction strategies such as t-SNE, PCA, and UMAP, which help visualize high-dimensional data in simpler, lower-dimensional spaces.

2.2.1. Fundamentals

This section briefly covers the concept of parameters, outlines the training and testing phases, and discusses the most commonly used train-test split.

Parameters

Parameters in a machine learning model are the adjustable elements that the model fine-tunes during training [36]. These internal variables, whose values are derived from the training data, enable the model to adapt and make predictions.For instance, in logistic regression, the parameters are the weights (coefficients) and biases. Weights adjust the influence of each input feature (variables that are fed into the model) on the prediction, while biases offer an additional offset, improving the model's ability to fit the data.

Training

Training[37] is the process of teaching a machine learning model to make predictions or decisions from the training data.This data consists of features (input variables) and labels (desired outcomes). During training, the model uses the features to make predictions and iteratively adjusts its parameters to minimize the difference between these predictions and the actual labels (outcomes) of the data. This adjustment is typically done through a loss function which quantifies the error between the true labels(ground truth) and predictions, and an optimization algorithm, such as gradient descent, that iteratively reduces this error.

Testing

During the testing phase, we assess the performance of the trained model on unseen data to evaluate its generalization capabilities [37]. The main goal of testing is to measure how well the model can apply learned patterns to new instances. Unlike during training, the model parameters stay constant during testing and no further learning takes place. We use various metrics such as accuracy, F1 score, ROC AUC score to measure the model's generalization performance. In cybersecurity research, there is often a focus on accuracy and the F1 score, as evident in the following sections. Therefore, we will use these same metrics to assess the generalization performance of our models.

Train/Test Split

To accommodate the phases mentioned above, we typically split our dataset into training and testing sets using a stratified 70/30 [38] train-test split for preliminary evaluation in our study. This stratified approach is important because the malware datasets we use in this study are highly imbalanced, with some classes of malware being much less frequent than others. A stratified split ensures that each class is represented proportionally in both training and test sets, according to its original distribution in the complete dataset [39]. This method is beneficial for ensuring that minority classes are represented and evaluated in both train and test set.

Overfitting Versus Underfitting

Overfitting and underfitting can significantly affect the performance of machine learning models. Overfitting occurs when a model is overly complex, fitting the training data closely but struggling to generalize well to new, unseen data [37] . To address overfitting, strategies include simplifying the model by choosing simpler models or fewer features, using regularization techniques like L1 or L2 to penalize large weights, and increasing the amount of training data to help the model learn more general patterns [37].

On the other hand, underfitting occurs when a model is too simple to capture the data's underlying patterns [37]. To address underfitting, one can select a more powerful model with more parameters, add more or better-quality features to expose the model and reduce constraints that might be limiting the model's learning capacity. These adjustments help ensure that the model can both fit the training data adequately and perform well on new data [37].

2.2.2. Cross Validation

In this study, we extensively use cross-validation to conduct a comprehensive evaluation of machine learning models. This technique [40] systematically assesses the model's performance by segmenting the dataset into various subsets, known as folds. Each fold is utilized as a test set once, with the other folds used for training. The folds are systematically rotated to ensure each one serves as the test set.

As will be discussed in our methodology, we typically opted for $k = 10$ for our cross-validation, balancing the trade-off between computational time and the thoroughness of the evaluation. This approach ensures that each of the ten parts is rotated to serve as the test set in turn, allowing us to use the entire dataset for both training and testing. A key aspect of this process is that the model is reinitialized for every fold. This ensures that each training and testing cycle is conducted with a fresh model, preventing any learning from previous folds from influencing the outcomes. This method maintains the integrity of the evaluation by ensuring that each data point contributes independently to both training and testing phases.

2.2.3. Hyperparameters

Hyperparameters are parameters whose values are set before the training process begins and are not derived from the data. Unlike model parameters, which are learned automatically during training, hyperparameters are used to control the learning process itself, directly influencing the behavior of the training algorithm and the performance of the model [37]. Common examples include the learning rate, which dictates how quickly a model adjusts its parameters during training; or the number of hidden layers and neurons in a neural network.These settings are important as they affect how well the model learns. To train hyperparameters we mainly use Grid Search [41]. Grid Search is a technique used to identify the best hyperparameters for a machine learning model. It involves defining a grid of possible values for each hyperparameter and systematically testing different combinations of these values.

2.2.4. Nested Cross Validation

Nested cross-validation is a robust approach employed to tune hyperparameters and thoroughly assess machine learning models. This method functions through two layers: an outer loop, which evaluates model performance across outer segments (folds), and an inner loop, responsible for fine-tuning hyperparameters within each outer fold. This is illustrated in figure 2.1.

In our study, we employ a stratified cross-validation method consisting of 10 outer folds and 5 inner folds, which helps maintain a nearly equal class distribution within each fold. The outer loop divides the dataset into ten folds, where each fold serves as a unique test set on a rotational basis, and the remaining folds form the training set. Concurrently, the inner loop further splits the training set into five folds using the same stratified approach, creating inner training and validation folds. Within these inner training folds, the model is initialized with various hyperparameters determined by a grid search and subsequently evaluated on the inner validation folds.

After determining the best hyperparameter configuration from the inner loop, the model is reinitialized with these parameters and trained on the full training subset of the outer loop. Subsequently, it is tested on the corresponding outer test subset to evaluate key performance metrics. Once all outer folds are processed, the metrics from each test are aggregated to provide a comprehensive assessment of the model's performance across the entire dataset.

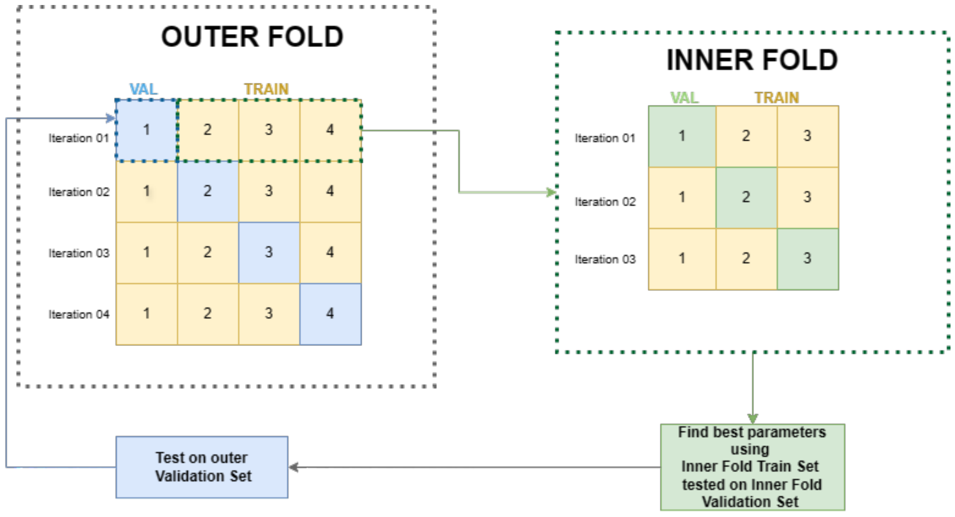

OUTER FOLD
VAL
TRAIN
Iteration 01
Iteration 02
Iteration 03
Iteration 04
INNER FOLD
Find best parameters using Inner Fold Train Set tested on Inner Fold Validation Set
Test on outer Validation Set

Figure 2.1: Example of Nested Crossvalidation with 4 outer folds and 3 inner folds

2.2.5. Supervised Learning

In supervised learning, the training data provided to the algorithm contains desired solutions, known as labels [37]. This method is mainly used when the labels are well defined and directly linked to the input data.

Classification is a typical example of supervised learning in which the objective is to predict categorical labels (classes) of new instances (samples) based on a dataset of labeled examples. The process involves training a model to identify the relationships between the features of the samples and their respective classes, which are predefined. Once trained, the model can then apply these learned relationships to classify new unseen instances into predefined classes. Classification can be divided into two primary types. Binary classification involves categorizing a sample into one of two classes, while multi-class classification entails categorizing samples into more than two classes. In our work, we will primarily use multi-class classification to distinguish between malware families by treating the families as distinct classes. However, we will also employ binary classification to distinguish between benign and malicious malware later in this study.

Logistic Regression

Logistic Regression is a machine learning technique traditionally used for binary classification. It models the probability that a given input—which includes the features of a data point—belongs to a specific class (the output) [42]. The algorithm outputs a probability score indicating how likely it is that the input falls into that class.To determine the class, a threshold, often set at 0.5 for binary classification, is used. If the probability score is greater than or equal to the threshold, the input is classified into the respective class; if below, it is assigned to the alternative class.

For situations involving more than two classes, Logistic Regression can be adapted using the One-vs-Rest (OvR) strategy or multinomial logistic regression [43]. In our work, we use multinomial logistic regression. Multinomial logistic regression extends the binary technique directly to handle multiple classes, providing a probability score for each class simultaneously without splitting the problem into several binary classifications [37]. In this case, the class with the highest probability score is selected as the final ouptut.

k-Nearest Neighbours

In the k-NN algorithm, the input consists of the features of a data point whose label needs to be determined [37]. The ”k” represents the count of nearest neighbors that the algorithm consults to arrive at its prediction. It calculates the distance between the input data point and other points in the dataset using metrics like Euclidean or Manhattan distance. The prediction outputs the most frequent class among the nearest neighbors. For a detailed list of hyperparameters considered for these calculations, please see Appendix C.

2.2.6. Unsupervised Techniques

On the other hand, unsupervised learning algorithms learn patterns exclusively from unlabeled data [37].These methods automatically discover the underlying structure of the data without any supervision. Clustering is a typical example of unsupervised learning, where data points are grouped into clusters such that items within the same cluster are more similar to each other than to those in other clusters.

Heirachial Clustering

In our work we use a clustering method called UPGMA which is a Heirachial Clustering algorithm with average linkage[44]. Hierarchical clustering creates a multilevel hierarchy of clusters through either agglomerative ”bottom-up” approach—where each observation starts as its own cluster and pairs are merged as one moves up the hierarchy—or a divisive ”top-down” approach, which starts with all observations in one cluster and performs recursive splits as one moves down [45]. The process hinges on the linkage method used to determine the distance between sets of observations, based on pairwise distances. The most common types of linkage methods include Single Linkage, where the minimum distance between pairs of points from two clusters dictates merging; Complete Linkage, which considers the maximum distance between pairs; Average Linkage, using the average distance [46].

2.2.7. Dimensionality Reduction Techniques

Dimensionality reduction is the transformation of data from a high-dimensional space into a suitable low-dimensional space. In our study, we will use these techniques to visualize high-dimensional data (1000-dimensional data) in 2 dimensions. More specifically, we will use t-Distributed Stochastic Neighbor Embedding (t-SNE), Principal Component Analysis (PCA), and Uniform Manifold Approximation and Projection (UMAP).

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a non-linear technique used to reduce the dimensionality of high-dimensional data for effective exploration and visualization, typically mapping the data into two or three dimensions. Its primary objective is to preserve the local structure of the data, meaning that points close to each other in the high-dimensional space are projected to be close in the lower-dimensional representation [47].

Principal Component Analysis (PCA)

PCA is a linear dimensionality reduction technique that transforms high-dimensional data into low dimensions while attempting to capture as much variability in the data as possible [37]. It achieves this by identifying the directions, called principal components, along which the variation of the data is maximized.

Uniform Manifold Approximation and Projection (UMAP)

UMAP is a non-linear reduction technique known for preserving local and the broader global structures within high-dimensional data [48]. This method balances the retention of local structures, which encapsulate the proximities and interactions among neighboring data points, with the preservation of global structures, which represent the overall distribution and spatial organization of data clusters across the dataset. The ability to fine-tune this balance is critically dependent on the selection and adjustment of the UMAP algorithm’s hyperparameters. By modulating these parameters, UMAP can be tailored to prioritize more granular local details or more expansive global relationships.

2.2.8. Deep Learning

Deep learning has gained significant popularity in malware analysis due to its sophisticated approach to handling data. In contrast to traditional machine learning, which often relies on manual feature selection and relatively simple predictive models, deep learning automates the process of feature extraction and typically utilizes multi-layered neural networks. These complex networks are capable of learning rich and hierarchical replresentations of data, making them highly effective at detecting subtle and intricate patterns in malware that may evade simpler models.

This subsection gives a high-level overview of the deep learning architectures that have become influential in the development of malware detection and classification systems. First, we will introduce Neural Networks(NNs), which form the foundation of many deep learning models, then move on to Convolutional Neural Networks(CNNs), and finally discuss transformers.

Neural Networks (NNs)

Neural networks are sophisticated computational models inspired by the intricate networks of neurons in biological brains [37]. These artificial networks are structured with an input layer, which receives the input features, hidden layers that process these features, and an output layer that delivers the final predictions as shown in figure 2.2 .

The hidden layers are where the majority of computation within a neural network occurs. Each neuron in these layers takes the output from the previous layer's neurons, applies a set of weights that signify the importance of this input, and adds a bias, which allows the model to better fit the data. The result is then passed through an activation function -often a non-linear function like ReLU, sigmoid, or tanh. This step is crucial as it introduces non-linearity to the model, enabling it to capture complex and abstract patterns. Without non-linear activation functions, the neural network would be incapable of solving problems beyond the scope of linear classification and regression.

The output layer receives the transformed data from the last hidden layer and translates it into a format suitable for the problem at hand. In the case of classification tasks, which are the focus of our study, this layer outputs a series of output scores, called logits, corresponding to each class. These scores are subjected to a softmax function—an operation that converts raw scores into a probability distribution, ensuring that the sum of probabilities for all classes equals one. The predicted class is then determined by selecting the one with the maximum probability from this distribution.

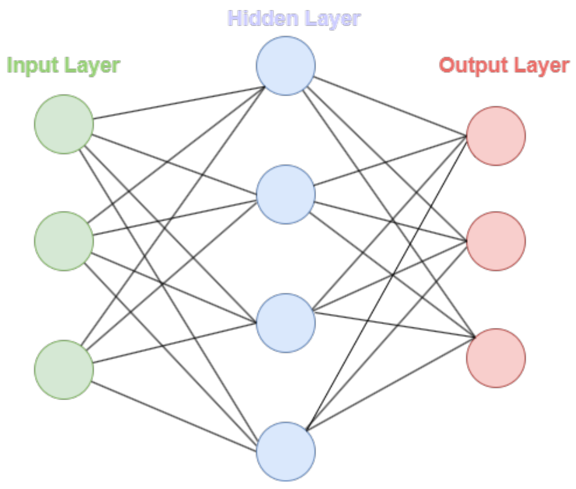


Figure 2.2: Neural Network with 3 input neurons, 1 hidden layer with 4 neurons and 3 output neurons

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized type of neural network designed for processing data with a grid-like topology, such as images [37]. Unlike standard fully connected networks where each input is connected to every neuron, CNNs utilize a unique architecture that includes convolutional layers to efficiently handle spatial data. The convolutional layers use small, square matrices of weights, known as filters or kernels, which move across the input data performing elementwise multiplications and summing them up, thus capturing spatial hierarchies and features such as edges and textures.

A key feature of convolutional layers is parameter sharing, where the same filter (and hence the same set of weights) is used across the entire input. This approach not only drastically reduces the number of parameters the network needs to learn but also decreases computational requirements, making CNNs significantly more efficient than fully connected networks. This efficiency comes from the ability of a single filter to detect the same feature throughout the entire input, promoting translational invariance and helping the network generalize better to new, unseen data.

Beyond convolutional layers, CNNs typically employ pooling layers, which reduce the spatial dimensions (height and width) of the input volumes, further simplifying the network computations. Commonly used pooling methods include max pooling and average pooling, which summarize the features detected in patches of the feature map. Following the convolutional and pooling layers, CNNs usually conclude with one or more fully connected layers, which integrate the learned features into predictions or classifications [49]. These layers are similar to those in standard neural networks, processing the flattened output of the previous layers to produce the final output.In computer vision, CNNs have become the architecture of choice due to their effectiveness in handling and interpreting visual data. A well known architecture that use many CNNs is ResNet (Residual Network), which we use in this study for image analysis [50].

Transformers

Transformers are an advanced architecture in deep learning, particularly prominent for processing sequential data such as text and time series [51]. Unlike models that rely on sequential processing (like RNNs and LSTMs [52]), transformers use a mechanism called attention, which allows the model to weigh the significance of different parts of the input data simultaneously. This parallel processing capability enables them to learn the context of any part of the sequence in relation to all other parts, improving efficiency and performance in tasks that involve understanding the entire sequence as a whole. For example, in language tasks, this means a transformer can evaluate the relevance of earlier words in a sentence when interpreting the meaning of a word later in the sequence.

Transformers typically consist of an encoder and a decoder. The encoder processes the input data into a continuous representation that holds all the learned insights about the input. The decoder then uses this representation, along with previous output elements, to generate the final output sequence. This architecture has proven highly effective, particularly in natural language processing tasks such as translation, text summarization, and content generation, setting new benchmarks for accuracy and flexibility in handling diverse and complex datasets.There are many examples of application of transfomers. Some of the recent examples include :

- BERT (Bidirectional Encoder Representations from Transformers) is a model in natural language processing (NLP) developed by Google [53]. It is able to train language representations bidirectionally. This means that for each word in a sentence, BERT looks at the words before and after it—rather than in just one direction—to get the full context, allowing it to understand language in a more nuanced manner . BERT has been applied successfully in numerous NLP tasks, including question answering sentiment analysis, and language translation.
- Longformer extends the capabilities of BERT by tackling one of its main limitations: handling long documents [53]. Standard BERT implementations are typically constrained to sequences of 512 tokens due to memory limitations of the attention mechanisms in transformers. Longformer introduces an attention mechanism that scales linearly with sequence length making it efficient for processing documents like legal texts or scientific papers that can be thousands of words long. It achieves this by combining a sliding window approach with global attention mechanisms on selected tokens, which allows the model to maintain a broader context over longer texts.

In our study, we employ OpenAI's state-of-the-art embeddings for text and code, which are developed using a method called contrastive pre-training on unsupervised data [54]. These embeddings are created through a process where a Transformer encoder processes the input data to generate dense vector representations. These representations are specifically designed to capture the semantic core of the text or code, which is crucial for tasks that depend on a deep understanding of content and context. The efficacy of these embeddings is demonstrated by their improved performance across various applications, such as linear-probe classification, semantic search, and large-scale information retrieval.

2.3. Related Works: Malware Analysis

There are three principal approaches to malware analysis: static, dynamic, and image-based. In our methodology, we will individually utilize each approach to generate embeddings, which will then be concatenated to form a comprehensive final embedding. In this section, we will review recent advancements in all three key analysis techniques. Although methods in literature are typically employed for malware classification rather than generating embeddings, they are pertinent to our study as we utilize classification techniques to validate the embeddings.

2.3.1. Static Analysis

Static analysis is a fundamental technique in malware analysis, involving inspecting a program's code without running it [55]. This method helps to identify specific functionalities such as function calls and hardcoded elements like domains or IP addresses, which are then used to develop detection signatures for identifying malware. In this section, we will first explore traditional rule-based approaches, highlighting tools such as Yara. We will then discuss the key features employed in malware detection and classification, and examine how these features are integrated with machine learning(ML) and deep learning(DL) models.

Rule Based Approaches

Rule-based approaches, exemplified by tools such as YARA, play a crucial role in static analysis by allowing analysts to create customized rules that identify and classify malware based on specific textual or binary patterns [56].These methods are integral to detecting known malware types and are commonly used by antivirus programs as part of their signature-based detection systems.

However, rule-based approaches have limitations, primarily due to their reliance on static indicators, which renders them less effective against evolving malware variants [57]. To address these shortcomings, our approach does not depend on rule-based methods. Instead, we ultimately build a phylogenetic tree that facilitates the creation of signatures that consider potential evolutionary pathways, including the spread of variants.

Machine Learning and Deep Learning Approaches

In this subsection, we will initially discuss the key features commonly used in machine learning and deep learning for malware classification. Subsequently, we will explore models used in both machine learning and deep learning approaches.

Features Used for Classification

Previous studies have established a strong foundation for extracting features in malware detection from binary executables. Ahmadie et al.[58] and Alazab et al.[59] focused on bytegrams, opcode grams, API calls, section-based features, and the frequency of static API calls in Windows binaries, specifically to detect malware, including zero-day threats. Similarly, other studies [60] [61][62][63] [64] [65] proposed using PE file headers, opcode frequency, byte entropy, function lengths, and strings from disassembled executables, among others, as features, to classify different types of malware.Each of these studies claim that that such features are highly effective in detecting or classifying malware.

Therefore, in our study we incorporate these features—byte histograms,byte entropy, opcode frequencies, and function lengths, pe header features, to name a few. Rather than exhaustively cataloging all API calls, we refine our focus to target entry, exit points, imports and exports. Similarly, for string analysis, inspired by shafiq et al [66] and Rama et al. [67] , we employ a composite feature vector that includes the total number of strings, their average length, distributions of printable characters, and counts of specific patterns such as URLs and file paths. This strategy efficiently condenses the information, maintaining key data in a form that is readily applicable to our model as will be discussed later in methodology.

Machine Learning Approaches

Numerous studies have applied supervised learning techniques to malware classification. For example, Ahmadi et al.[58] used an XGBoost classifier and achieved high accuracy on the Microsoft Malware Classification Challenge dataset. Alazab et al.[59] employed supervised learning algorithms such as KNNs and NNs to analyze malware using the frequency of static API calls. Similarly, Babbar et al.[68]

focused on using the K-Nearest Neighbors algorithm for Android malware detection by studying interactions with system APIs. Others, like Sahs et al. [69], utilized a one-class Support Vector Machine to profile benign behavior and identify malware based on a binary vector representing Android application permissions.

Other works preferred using ensemble models. For example, Santos et al.[61] utilized Polynomial Kernel classifiers and Random Forests to differentiate between benign and malicious samples using opcodes. Zhu et al. [70] implemented an ensemble Rotation Forest model for Android malware classification, demonstrating the effectiveness of combining multiple classifiers for improved classification accuracy.

While these models report high accuracy they primarily handle numerical or categorical data, our research integrates a broader array of features including both string and numerical data. The challenges posed by diverse data types are not as effectively addressed by traditional models like SVMs or Random Forests. To address these challenges, we use deep learning, specifically leveraging state-of-the-art embeddings from OpenAI that are adept at processing diverse data formats. These embeddings convert diverse features into a vector of floats, which can subsequently be employed as input for malware classification using any supervised learning technique. For classification, we employ logistic regression—a linear model—to validate our embeddings. This approach is chosen because good-quality embeddings should facilitate effective classification even with simple linear models. Logistic regression is advantageous as it generally involves fewer hyperparameters than more complex models like neural networks or ensemble methods as used in prior work.

There also have been works that use unsupervised methods. For example, Tian et al.[62] use clustering in combination with statistical tests to analyze function length patterns, enabling them to classify seven Trojan families. They choose clustering to delve into deeper insights regarding function lengths, despite having access to ground truth data that could support supervised learning. Conversely, Suarez et al. [71] created "Dendroid," a system that combines text mining and information retrieval with hierarchical clustering to produce dendrograms which they claim to mimic phylogenetic trees. Their approach employs single linkage clustering, which they liken to phylogenetic analysis, to provide evolutionary insights into malware families.

Our research recognizes the advantages of clustering but primarily utilizes supervised learning techniques, making use of available ground truth data, to validate the embeddings sourced from OpenAI. Moreover, for a more precise analysis of evolutionary relationships among malware, we adopt established bioinformatics techniques like UPGMA and Neighbour Joining, instead of single linkage clustering, which is uncommon in phylogenetic studies [44] [72] [73]. These methods better align with the standard practices in phylogenetic analysis, providing a more accurate framework for building phylogenetic trees.

Deep Learning Approaches

In our previous discussion, we highlighted our intention to use deep learning techniques. It is beneficial to explore the work of others in this field to understand the various approaches that have been employed. In deep learning, it's often feasible to convert a feature classifier into a feature extractor, making the approaches in literature applicable for both generating embeddings and classifying malware.

Dahl et al. [74] use random projections to compress sparse binary features through neural networks, improving classification performance compared to traditional models. Others such as Krčál et al. [75] and Raff et al. [76] employ convolutional neural networks (CNNs) for analyzing raw byte sequences of executable files. Krčál's architecture [75] is designed to reduce false positives, whereas CNN of Raff et al. adeptly handles long sequences and diverse data types.

Our approach differs from those that use convolutional neural networks to process raw byte sequences. Instead of employing CNN based architectures for this, we reserve them for image analysis, where executable files are converted into byte sequences and then transformed into image formats—a method that shows more promise, as will be discussed in the next subsection. Instead, our strategy mirrors that of Rizvi et al. [63], who extract features like strings and byte entropy and utilize attention-based networks to classify malware executables. However, rather than using attention-based networks directly, we employ sota embeddings from OpenAI, which internally uses transformer architectures[54].

2.3.2. Image Analysis

Computer vision, a technique that enables computers to interpret and analyze visual data, has become increasingly popular in malware analysis. In this section, we will initially review the algorithms that convert binary files into image formats, mainly PNGs. Subsequently, we will examine how these images are transformed to be compatible with different models. Finally, we will explore the diverse architectural approaches employed by studies to classify these images.

Malware To Image conversion

Several studies have focused on transforming executable files into grayscale images to classify malware.For example, Ni et al. [77] extract opcodes from malware, apply SimHash to generate binary hashes, and convert these into grayscale values to visualize malware fingerprints. Similarly, Nataraj et al. [78], Kancherla and Mukkamala [79] treat binary data as arrays of 8-bit unsigned integers that are reorganized into two-dimensional arrays to produce grayscale images. Similarly, approaches by Gaikwad et al. [80], and Kumar et al.[81] involve direct mapping of binary file bytes onto visual formats to highlight malware structures. Han et al. [82] map bytes directly onto bitmap images, while Gaikwad et al. [80] and Kumar et al. [81] arrange binary code of executable files into two-dimensional arrays, converting these arrays into grayscale images.

On the other hand, Demmese et al.[83] argue that rgb-based encoding, as opposed to grayscale, produces more compact images by reducing pixel space by a third, thereby minimizing distortion during image resizing. Similarly, other studies [84] [85] [86] highlight that rgb images may be better than gray scale for malware classification. Building on this, Han et al.[82] adopt a more colorful approach by extracting opcode sequences from malware samples and assigning specific RGB colors to different opcode sequences. Demensce et al.[83] use a technique where each pixel represents three sequential bytes, effectively capturing more data per row and ensuring consistent byte-level alignment across the image. Similarly, Vu et al.[85] assign rgb colors based on byte value categorization and Shannon entropy calculations. This technique transforms binary data into color images where different colors denote various data types and entropy levels.

Surprisingly, many studies [84] [87] [88] [89] [90] adopted 'bin2png' method developed by Sultanik to create RGB images from binaries[91]. This method processes the binary data in three-byte chunks, where each chunk represents one RGB pixel: the first byte for red, the second for green, and the third for blue. To ensure the byte count aligns with this segmentation, all files are zero-padded at the end to make them divisible by three. The dimensions of the resulting image are automatically determined to be as close as possible to a multiple of three. Given the use of bin2png in the public dataset Malevis and its adapted versions in MalNet, which we also utilize, we have opted to employ this method for converting our malware executables to maintain consistency.

Image Transformations

In the literature, a few studies choose not to use any transformations and adapt their methodologies to work with original image sizes [78] [79] [81]. Conversely, many works implement resizing techniques. Some [92] [86] [93] [94] [95] explicitly utilize bilinear interpolation while others employ PIL [96] [97] [98] [99] , which typically involves either bilinear or nearest neighbor methods according to the code. Additionally, a few studies[100] [101] [102] [102] specify the use of LANCZOS interpolation.

Given the lack of detailed explanations about the benefits or effects of various resizing methods in studies, we have chosen bilinear interpolation. This decision is primarily because it is the default resizing method in the transformers library that we utilize for processing our images.

Architectures

In the field of image analysis, CNNs (Convolutional Neural Networks) are widely employed. Studies such as those by Sang et al.[77] focus on analyzing malware through grayscale visual fingerprints using CNNs. Similarly, Gilbert et al. [103] and Kumar et al. [81] apply CNNs to examine grayscale images created from binary executables, achieving high accuracy in malware detection and classifcation. Others such as Han et al.[82] and He et al.[86] use CNNs on RGB-colored pixel images for malware detection. Differing from these approaches, Yakura et al. [104] enhance their CNN with an attention mechanism to pinpoint critical regions within binary sample images.

Some works [105] [106] have taken it a step further by utilizing transfer learning with pre-trained models like VGGNet, initialized with ImageNet weights, to classify malware. A study conducted by

Bhodia et al. [107] shows that transfer learning with architectures such as VGGNet and InceptionV3 generally yields better results than traditional CNNs especially when handling large public datasets.

Motivated by this, we use a ResNet-50 model, initialized with ImageNet weights, as the foundational architecture for image analysis. Our approach involves initially training this model on publicly available datasets, then further refining it by fine-tuning on our specific dataset.

2.3.3. Main Limitation: Static and Image Analysis

Works in both static and image analysis work with the assumption that the malware executables are unpacked or do not take that into consideration. There are studies such that of kumar et al. [105] who claim that their architecture is resilient to packing by showing that it can still classify UPX packed samples but do not extend this analysis to other packers.

In our work, we address this limitation by extracting memory dumps from running the malware and subsequently retrieving the executable from memory—a process we will elaborate on later. This approach is effective as it typically captures the executable when most forms of runtime obfuscation or encryption have been resolved to facilitate execution.

Furthermore, the literature we reviewed mainly focuses on malware in Windows executables or Android applications, but it largely overlooks ELF samples. An analysis by Michael Potuck [108] shows that, in 2023, around 52% of malware attacks have targeted Windows platforms. Meanwhile, Linux systems have experienced 47% of attacks, mainly involving ELF formats like Mirai and Bashlite, indicating a rising interest in Linux because of its growing exposure and advanced attack methods. In contrast, only a marginal 1% of attacks were directed at macOS. Therefore, our study includes samples from PE, DOS, and ELF formats, reflecting our aim to facilitate a comprehensive study of malware evolution.

2.3.4. Dynamic Analysis

In this subsection, we will initially discuss features commonly used in machine learning and deep learning for malware classification in dynamic analysis. Subsequently, we will explore models used in both machine learning and deep learning approaches.

Features used for Classification

Many behavioral features are utilized for malware classification. Mohaisen et al. [109] analyze system, network, and registry behaviors to detect anomalies. Simiarly, Galal et al. [110] focuses on process creation, process termination and registry.

Other studies [111] [112] [113], [114] focus primarily on leveraging API calls for behavioral classification. In contrast, research by Phode et al. [115] adopts a more extensive approach, encompassing a variety of metrics including the number of processes, CPU usage, packet sizes, and memory usage, alongside API calls, to provide a thorough analysis of system impact.

Drawing inspiration from these works, our research involves monitoring API calls and extends this by indirectly assessing command executions through the examination of processes that are initiated or terminated. We also give special consideration to registry behaviors, including tracking changes, additions, or deletions of registry keys. While we do not explicitly monitor network behaviors or the quantity of processes, we focus on file interactions, tracking files that are opened, written to, and deleted.

Machine Learning Architectures

Most studies employing machine learning in malware detection have predominantly utilized Support Vector Machines(SVMs) and Random Forests (RFs). For example, both Liang et al. [112] and Galal et al. [110] have implemented SVMs and RFs to assess their feature sets. While studies like Mohaisen et al.[109] have investigated Decision Trees and KNNs, these techniques are relatively less prevalent in the field.

Deep Learning Architectures

In deep learning applications for malware detection, sequential networks like Recurrent Neural Networks(RNNs) and Long short-term memory(LSTMs) are frequently used. For example, Maniath et al. [116] leveraged LSTMs to classify ransomware, while Rhode et al. [115] utilized RNNs for broader malware classification tasks. Less frequently used methods involve denoising autoencoders, as implemented by David et al. [117] and Stacked AutoEncoders, utilized by Hardy et al. [118], for malware detection.

Our approach

Similar to static analysis, many architectures in previous work transform textual features into categorical or numerical features before feeding it traditional machine learning algorithms like Random Forest or decision trees. Instead of following these conventional methods, we use state-of-the-art embeddings from OpenAI, which leverage Transformers—considered an advancement over LSTMs and RNNs, to process the textual features into embeddings.We believe our approach is not only notably simpler than feature transformation but also offers greater expressiveness. This is because it allows us to directly use complex dependencies within the text that may be lost when reduced to categorical as done in prior works.

2.3.5. Practical Applications

Despite the perception that machine learning and deep learning might not be widely used in real-world applications, many antivirus (AV) vendors effectively employ these technologies for malware detection and classification. For instance, Avira [119] leverages deep learning for feature extraction and malware classification, while Kaspersky [120] utilizes supervised learning throughout its detection pipeline, primarily employing ensemble methods like Random Forests and Gradient Boosted Trees. Kaspersky also implements clustering to identify hidden data structures and group similar objects or features.

Similarly, Windows Defender [121], utilizes deep learning models such as Deep CNN-BiLSTM to effectively handle sequential tasks encountered in natural language processing. This model is used to detect various attacker tactics, including the identification of malicious PowerShell scripts associated with malware. Other AV vendors like Sophos [122] and MalwareBytes [123] also employ machine learning to boost their malware detection capabilities. Therefore, both machine and deep learning are widely utilized by these private antivirus vendors.

2.4. Open Source Datasets of Malware Executables

Open source datasets are often limited and can quickly become outdated due to the rapidly evolving nature of malware. In this section, we will review a selection of available open source datasets and highlight the public datasets that we will use for our research.

Binaries for static and behavior analysis

Nadeem et al.[124] recently highlighted the scarcity of open-source malware datasets and the rapid obsolescence of existing collections such as VX Heavens, Drebin, and MalGenome. While these datasets were initially comprehensive, they have struggled to keep pace with the evolving nature of malware threats. Projects like Stratosphere IPS [125] have produced datasets like CTU-13 and IoT-23 focusing on network traffic from botnets and IoT malware; however, these do not meet our needs as our research requires binary data rather than network traffic analysis.

Prominent datasets like the Kaggle Microsoft Malware dataset[126] present significant challenges. Alterations such as modifying the header to hinder malware execution and removing SHA hashes compromise the reliability of these datasets. These changes prevent dynamic analysis due to malformed headers and complicate validation with tools like VirusTotal, making it unusable for our research.

Other specialized databases, like the Motif collection compiled by Joyce et al. [127], offer extensive information through modified PE malware samples and detailed metadata. However, they also modify their samples to prevent execution.

Similarly, the EMBER dataset [128] provides numerous PE samples primarily as features, including byte entropy and strings. While these features contain rich content, we cannot use them directly; we need the binary to create images and execute the malware. Other private datasets [129] [130] [131] have only 20-30 malware samples with claimed family labels and do not have a sha for verification purposes. These factors render such datasets unreliable for this study's objectives.

The majority of these datasets primarily focus on PE files, often neglecting crucial formats like DOS and ELF that are essential for tracking malware evolution in our study. Malware Bazaar[132], on the other hand, offers a broad range of malware samples and metadata across various formats. Despite challenges with accurate ground truth data and a high incidence of false positives, we carefully select samples from Malware Bazaar for our research, as detailed further in our methodology.

Another noteworthy dataset is VX Underground[133], sourced from the darknet and claimed to be the internet's largest repository of malware source code, samples, and scholarly articles. This dataset includes SHA signatures for all samples covering multiple formats, which generally correspond well with the ground truth labels provided by VirusTotal AV vendors. After confirming these SHA signatures through VirusTotal, we have incorporated a substantial number of their samples in our study.

Datasets for Image Analysis

We use 3 primary public datasets used for image analysis. The Malimg Dataset, introduced by Nataraj et al., consists of 9,339 gray scale images derived from malware samples across 25 distinct families and is commonly used to explore and develop machine learning models [134] [135] [136] [137].

Similarly, the MaleVis Dataset,an open-set collection of RGB images generated from 25 different classes of malware, is also commonly used in the literature [138] [136] [139] [140].

Lastly, MalNet [141], developed with the support of Androzoo, is a hierarchical RGB image and graph database aimed at aiding machine learning and security researchers in identifying malicious software. It stands out by providing a large collection of 1,262,024 images and graphs across 696 families organized according to the Euphony Hierarchy. We use all these datasets to initially train our resnet-50 model, and then we finetune it on our specific dataset, as detailed later in the methodology.

Our selection of the Malimg dataset, which uses a unique encoding method to create grayscale images, along with the Malevis dataset that employs bin2png encoding for RGB images, and the MalNet dataset that uses an adapted version of bin2png encoding, is requently observed in the literature. For example, Panda et al. [142] and Guven et al. [143] have both employed the Malevis and Malimg datasets for transfer learning to enhance malware detection capabilities in IoT environments despite the different encodings.

2.5. Phylogenetic Trees

Phylogenetic trees [44] [72] [73] are a recognized method in bioinformatics used to trace the evolutionary history of species. These trees graphically depict the evolutionary relationships among biological entities based on their physical or genetic similarities and differences. In our research, we adapt this method to create phylogenetic trees from our malware embeddings. The detailed process will be described in the methodology section of our paper. This section will begin with a high-level overview of phylogenetic trees, followed by a detailed explanation of the construction methods we employ, focusing particularly on Unweighted Pair Group Method with Arithmetic Mean(UPGMA) and NeighbourJoining(NJ) techniques. We will also review related work on other methods for constructing phylogenetic trees and discuss approaches used in rooting and validating these trees.

2.5.1. High level overview of Phylogenetic Tree

Phylogenetic trees are graphical representations comprising nodes and branches that delineate the evolutionary relationships among taxa. In biological terms, ”taxa” (plural of ”taxon”) are groups of organisms recognized as distinct entities by taxonomists, such as species or genera. In our study, each taxon represents an individual malware sample, and collectively, these are referred to as ”taxa.” Terminal nodes in these trees represent the specific entities under study—malware samples in our case—while internal nodes denote their theoretical common ancestors. These trees effectively map out the evolutionary trajectories of the taxa, illustrating their divergence from common origins over time [44].

2.5.2. Phylogenetic Tree Building methods

Developing phylogenetic trees involves various computational methods, each making its own assumptions about evolution’s nature and strategies for modeling evolutionary processes. This section will initially explore tree construction methods before showing some of its application in bioinformatics.

There are many ways to build phylogenetic trees. The common ones are Maximum Parsimony(MP), Neighbour Joining(NJ), Unweighted Pair Group Method with Arithmetic Mean(UPGMA), Maximum Likelihood(ML) and Bayesian method.

These methods can be grouped into Sequence-Based Methods and Distance-Based Methods. Among the Sequence-Based Methods, Maximum Parsimony (MP) seeks the simplest explanation for evolutionary changes by minimizing the total number of evolutionary events [44]. Maximum Likelihood (ML) evaluates the probability of different phylogenetic trees by applying specific statistical models of sequence evolution [44]. Bayesian Inference calculates probabilities of phylogenetic trees by integrating prior knowledge with the observed data, allowing for complex model specifications and providing measures of uncertainty [44]. These techniques are essential in studies where direct sequence comparisons illuminate evolutionary histories.

On the other hand, distance-based methods construct phylogenetic trees by analyzing the genetic distance between sequences rather than the sequences themselves. These methods are generally faster and simpler compared to sequence-based methods. Neighbor-Joining (NJ) is a popular approach that builds trees by iteratively clustering the closest pairs of operational taxonomic units based on genetic distance, optimizing tree topology for minimal total branch length [44]. Another method, Unweighted Pair Group Method with Arithmetic Mean (UPGMA), assumes a constant rate of evolution and creates a tree by grouping taxa based on the average distance to other taxa, resulting in an ultrametric tree where the distances from the root to every tip are equal [44]. These distance-based methods are particularly useful when sequence data is scarce or when rapid tree estimation is required.

In our research, we utilize distance-based methods to construct phylogenetic trees, which effectively visualize evolutionary relationships using embeddings derived from malware samples. Although sequence-based methods are used in some studies[144]—such as those employing the Needleman-Wunsch global alignment technique for opcode sequences—they prove less efficient for our objectives. These methods involve time-consuming alignment processes and introduce complexities and potential inaccuracies when converting malware into a suitable sequence format for alignment. Conversely, distance-based methods are straightforward when applied to embeddings, which are central to our methodology.

Neighbor Joining Method

The Neighbor Joining (NJ) method is a distance-based algorithm used in the construction of phylogenetic trees. Table 2.3 gives a detailed overview of the neighbour joining method.

Table 2.3: Overview of the Neighbor Joining Method

Aspect	**Details**
Method Overview	The Neighbor Joining (NJ) method is a distance-based algorithm efficient for constructing phylogenetic trees by minimizing total branch length, ideal for large datasets.
Data Preparation	**Input:** Matrix of pairwise distances reflecting the evolutionary divergence between taxa. **Leaves:** Represent the taxa being analyzed, forming the terminal nodes of the tree.
Tree Construction	**Merging**: Begins with a star-like tree, where all taxa are directly connected to a central node without any internal structure. At each iteration, a pair of taxa that minimally increases the total branch length is identified and merged into a new node. **Distance Matrix Update:** After merging, the distance matrix is updated to reflect the new distances between the new node and other taxa, setting up for the next iteration.
Tree Finalization	**Termination:** The iterative process continues until all taxa have been merged into a single phylogenetic tree that reflects the evolutionary relationships among the input taxa with the minimum total branch length. **Unrooted Tree:** The final phylogenetic tree is unrooted, where the length of each branch is proportional to the estimated evolutionary distance between nodes.
Assumptions	**Distance Matrix Accuracy:** Assumes input distances accurately reflect the evolutionary divergences. **Minimization of Tree Length:** Seeks to construct a tree that minimizes the total branch length. **Starlike Phylogeny:** Initially assumes a starlike phylogeny, simplifying early algorithm stages.

UPGMA Method

The Unweighted Pair Group Method with Arithmetic Mean (UPGMA) is a classic hierarchical clustering method with average linkage used to construct phylogenetic trees. Unlike distance-based methods that seek to minimize tree length, UPGMA assumes a constant rate of evolution across all lineages, producing a rooted tree that reflects the temporal sequence of divergences. This process is detailed in table 2.4

Table 2.4: Overview of the UPGMA Method

Aspect	Details
Method Overview	The UPGMA (Unweighted Pair Group Method with Arithmetic Mean) method is a distance-based algorithm used for constructing phylogenetic trees, characterized by its use of an arithmetic mean to calculate the distance between clusters, making it suitable for ultrametric trees where the same amount of evolutionary change is assumed along each branch.
Data Preparation	**Input:** A matrix of pairwise distances, representing the evolutionary divergence between each pair of taxa. **Leaves:** Each leaf represents a taxon (species or sequence) from the input distance matrix, forming the observed endpoints of the tree.
Tree Construction	**Cluster Formation:** UPGMA iteratively joins the two closest taxa or clusters of taxa, based on the arithmetic mean of their distances. This process continues until all taxa are included in a single hierarchical tree. **Height Calculation:** Calculates the 'height' of each node (the distance to the leaves), reflecting the time of divergence under the molecular clock assumption.
Tree Finalization	**Rooted Tree:** The resulting UPGMA tree is rooted, indicating the most recent common ancestor of all taxa. Branch lengths are proportional to the estimated time of divergence. **Molecular Clock:** Assumes a constant rate of evolution, where evolutionary change occurs uniformly across all branches.
Assumptions	**Molecular Clock:** Assumes mutations accumulate at a constant rate over time across all lineages. **Accuracy of Distance Measures:** Assumes that the provided pairwise distances accurately reflect the evolutionary time between taxa. **Equal Rates Across Lineages:** Assumes uniform evolutionary rates across all branches, which can lead to inaccuracies when different lineages have experienced varying rates of evolution.

Applications of UPGMA and NJ

While UPGMA and Neighbor Joining are both distance-based methods, their popularity in bioinformatics remains significant. For instance, Hillis et al.[145] explore the application of UPGMA in studying molecular phylogenies, while Andriani et al.[146] utilize it to track the identification and spread of the Ebola Virus. Additionally, Andrei et al.[147] employ UPGMA for analyzing type II CRISPR RNA-guided endonuclease Cas9 homologues.

Neighbor Joining is notably more prevalent due to its flexibility in not assuming a constant rate of evolution, making it suitable for a wider range of studies [148] [149] [150] [151]. Some notable examples inlcude Snrzlic et al.[152] who apply it to molecular characterization of Anisakidae larvae from the Adriatic Sea, Thangaraj et al.[153] use it to analyze four Aspergillus species, and Comas et al.[154] leverage it for DNA sequencing of monomorphic bacteria.

2.5.3. Rooting Phylogenetic Trees

Rooting a phylogenetic tree is fundamental for accurately depicting the evolutionary history of the taxa under study, as it provides directionality to the tree. This process distinguishes between ancestral and descendant relationships, essential for interpreting the sequence of evolutionary events. By establishing a temporal framework, rooted trees facilitate the reconstruction of evolutionary history and enable the analysis of evolutionary dynamics over time[155].

Among the methods discussed, UPGMA uniquely produces a rooted tree inherently due to its assumption of a molecular clock. This assumption implies that evolution occurs at a constant rate across all lineages, allowing the method to infer the timing of divergence events and thus establish a root automatically [73].

Methods for Rooting Trees

For methods that do not inherently produce a rooted tree, such as Neighbor Joining, rooting techniques must be employed to root the tree. We use two main methods to root our phylogenetic tree:

- **Outgroup Rooting:** This method involves including an outgroup in the analysis, which is known to be distantly related to the rest of the taxa (the ingroup). The root is placed on the branch that leads to the outgroup, assuming that the outgroup diverged earlier from the ancestral lineage than any of the ingroup taxa [156].
- **Midpoint Rooting:** In the absence of a clear outgroup, trees can be rooted at the midpoint of the longest path between any two taxa on the unrooted tree. This method assumes that the root lies equidistant from the tree's furthest points, aiming to minimize the variance in the distance from the root to all leaves[155].

2.5.4. Application of Phylogenetic Trees in Malware Research

As highlighted in the introduction, there is limited research on the use of phylogenetic trees in malware analysis. Nonetheless, three pioneering studies have utilized phylogenetic trees on malware samples.

Vinod et al.[144] address metamorphic malware by using the Needleman-Wunsch method to align opcode sequences, facilitating signature development. Although they incorporate phylogenetic concepts, they do not detail their tree construction techniques or analyze evolutionary relationships between malware families. Instead, their focus is on enhancing malware detection through the development of opcode sequence-based signatures, informed by phylogenetic tree evolution.

Cozzi et al.[157] explore the lineage among IoT malware families by employing binary diffing and Minimum Spanning Tree (MST) analysis as proxies for phylogenetic trees. Their approach clusters malware from similar families but does not provide a timeline of evolutionary relationships, nor does it root the trees to specify directional ancestry.

He et al.[158] introduce an efficient method for performing neighbor joining, though their work is limited to methodological development without applying it to understand relationships between malware.

Collectively, these studies demonstrate limited applications of phylogenetic trees in malware analysis. Our research is unique in its approach to use phylogenetic trees to illuminate the evolutionary dynamics of malware, integrating both inferential and validation techniques based on established bioinformatics literature.

2.5.5. Related Works: Validating Phylogenetic Trees

Validating phylogenetic trees ensures reliability of evolutionary relationships inferred from genetic or morphological data. In bioinformatics, three main approaches are commonly used to validate phylogenetic trees: cross-validation, congruence testing, and the use of temporal data.In this section we will discuss these methods and their relevance to our approach.

Several studies[159] [160] [161][162] [163] employ cross-validation techniques like jackknife and leave-one-out methods to validate phylogenetic trees. These methods assess tree robustness by systematically omitting parts of the data, examining how tree topologies change in response. These methods work under the assumption that consistent tree topologies despite data exclusion signify reliable evolutionary relationships. While primarily applied to sequence-based data, these methods present challenges: they require significant computational resources for large datasets and the removal of taxa can unexpectedly alter the tree topology.

Phylogenetic congruence tests are also utilized to validate phylogenetic trees [164] [165] [166]. These work by comparing trees derived from various data partitions or genes to ensure consistency across these different sources. High congruence supports the tree's reliability. Predominantly requiring sequence data, these tests are challenging to adapt for other data types due to their reliance on specific gene or partition comparisons.

Both cross-validation techniques and phylogenetic congruence tests present practical challenges for our purposes. While cross-validation could theoretically be applied, it is computationally intensive, especially given the large collection of malware samples in our study. Phylogenetic congruence, on the other hand, relies on sequence data, which is not applicable to our dataset.

Other studies [167] [168] [169] use temporal data to validate trees by considering the chronological order of species divergences. This approach often relies on molecular clock assumptions to estimate divergence times and typically requires sequence data. It calculates the timing of evolutionary events by analyzing the genetic differences between sequences, using known rates of mutation to infer how long ago these divergences occurred based on temporal data.

In our study, we are not interested in estimating the divergence times. Instead, we introduce a new method that relies on analyzing relative recent divergence times—a fundamental aspect of the temporal method which will be elaborated in the methodology section of our study.

3
MalwareGenome Dataset

This chapter presents our dataset, MalwareGenome, which forms the basis of our study. It includes a detailed discussion of the data sources, encompassing both public and private collections. Additionally, this chapter describes the methods we employed to verify and preprocess the samples.

3.1. Sources

The background section underscored several key challenges in using public datasets for malware analysis, such as the unreliability of ground truth, rapid obsolescence of datasets, and the modification of headers to inhibit execution.

Our public samples are sourced from MalwareBazaar and Vx-underground. Additionally, following recommendations from Nadeem et al.[124], we have also gathered private samples from entities such as VirusTotal—whom we thank—as well as other other private collections.

To address the challenges mentioned, all samples from these sources are verified to be valid, with no header modifications that could inhibit execution, and are regularly updated to stay relevant to the current malware landscape. Despite these precautions, the inherent unreliability of ground truth remains a concern, as some samples may still be mislabeled or not malicious at all. To mitigate this issue, we have established a rigorous validation process for the samples, details of which are outlined in the following subsection. This process is designed to overcome the obstacles identified in the initial discussion and ensure the integrity of our dataset.

3.1.1. Validation of Samples

To validate the samples, we rely on VirusTotal as a trusted source. The criteria for including or excluding a sample are outlined in a flowchart presented in Figure 3.1.

Referring to figure 3.1, there are two primary reasons for excluding a sample if it is not present in VirusTotal. Firstly, if a SHA hash is not identified as malicious by multiple sources, including various Antivirus(AV) vendors on VirusTotal, we consider it unreliable. Secondly, while we could upload the sample to VirusTotal to obtain a label, the temporal metadata—especially the first submission date—is crucial for validating our phylogenetic tree. Uploading the sample ourselves would incorrectly set our upload date as the initial submission date undermining the accuracy of our validation approach.

To verify the family labels, we primarily use labels from VirusTotal AV vendors. Our method involves a straightforward heuristic: if the family label does not align with the most common label from VirusTotal, we then check if at least 30 % of the AV vendor labels on VirusTotal concur. If they do, we accept the sample; otherwise, we reject it as shown in figure 3.1.

3.1.2. Preprocessing of Samples

The samples in our collection are not just executables; they are stored in folders that contain various file types including JPEGs, PEGs, Word documents, shortcut files, and DLLs. We removed these extraneous files as part of our data cleaning process. Following this cleanup, our refined dataset contained 103,883 samples, representing 538 distinct malware families across PE, ELF, and DOS formats.

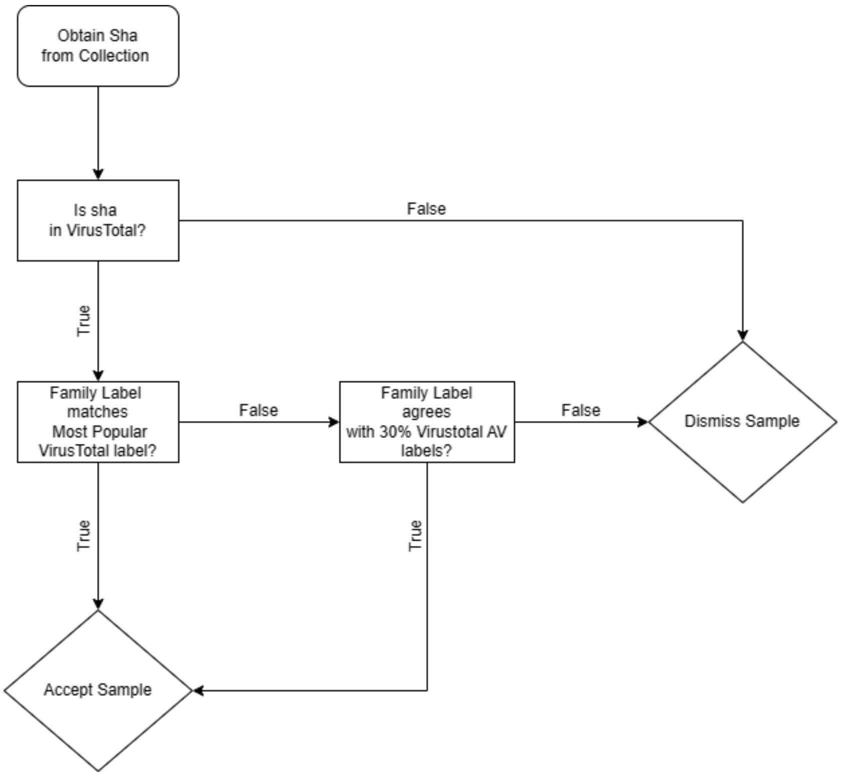

Obtain Sha from Collection
Is sha in VirusTotal?
False
True
Family Label matches Most Popular VirusTotal label?
False
Family Label agrees with 30% Virustotal AV labels?
False
Dismiss Sample
True
True
Accept Sample

Figure 3.1: Flowchart of how we validate our samples

3.1.3. Class Distribution

In figure 3.2 we highlight that the top 10 families in terms of the number of samples we have.

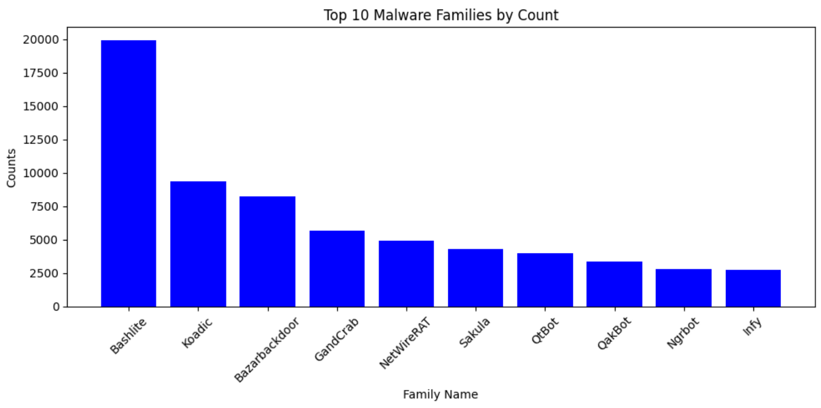


Figure 3.2: Imbalanced Family Distribution Of Our Dataset

It is immediately apparent from 3.2 that we have quite an imbalanced dataset. When dealing with such a dataset it is important to ensure both the training and testing tests are representation in terms of the distribution of the data. Therefore, we adopt our training and testing methodologies accordingly as will be pointed in the methodology section.

3.2. Public Datasets for Image Analysis

We briefly mentioned the public datasets used for image analysis in the background. In this section we will highlight some high level statistics of these datasets.

3.2.1. High level overview of Image Dataset

In our study, we employ three public image datasets: Malimg, Malevis, and MalwareNet. We combine these datasets into a single, concatenated dataset referred to as ImageDataset. This dataset will serve as the initial training set for our image analysis model, as detailed in the methodology section of our work. The Malimg Dataset is comprised of grayscale byteplot images from 25 unique malware families, processed using the grayscale conversion technique developed by Nataraj et al. [78]. Meanwhile, the Malevis Dataset includes RGB images from 25 different malware classes, and the MalNet Dataset contains images from 696 families, both of which use adapted versions of bin2png conversion technique. Collectively, these datasets represent 746 unique classes, all distinct and non-overlapping with our MalwareGenome dataset.

We also employ a benign image dataset processed using the same encoding techniques as those used for the MalNet dataset from Androzoo. To construct a new dataset for our analysis, we first extract embeddings from the benign dataset, as detailed in the methodology section. We then combine these embeddings with those from our malicious dataset, the MalwareGenome dataset. The final dataset contains 34.77% benign samples and 65.23% malicious samples.

3.2.2. Class Distribution

The distibution of classes of these datasets are presented in the figures 3.3, 3.4 and only 25 classes of 3.5. The distributions within these datasets show a pronounced imbalance, resulting in a combined dataset that is also imbalanced. This issue of dataset imbalance is a common occurrence even in publicly available datasets and will be discussed in detail in the methodology section.

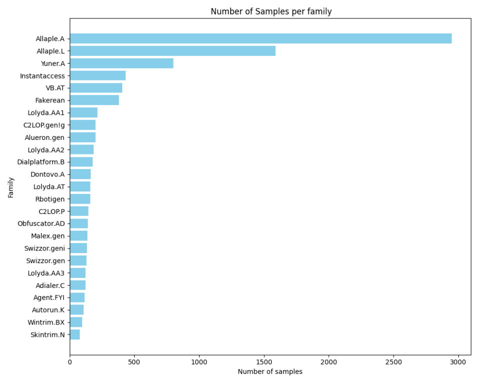


Figure 3.3: Imbalanced Family distribution of Malimg Dataset

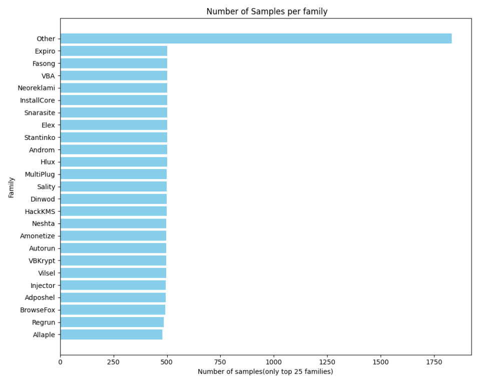


Figure 3.4: Imbalanced Family distribution of Malevis Dataset

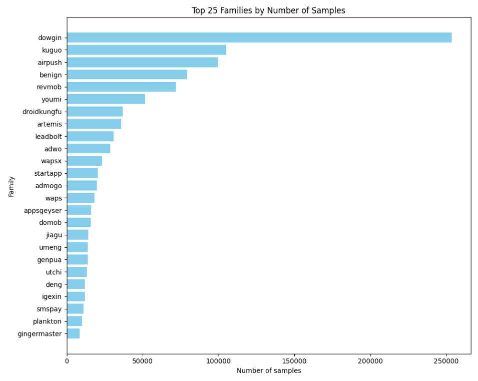


Figure 3.5: (Top 25)Imbalanced Family distribution of Malnet Dataset

4 Methodology

This chapter outlines our methodology for analyzing the evolution of malware. We start by introducing our main pipeline, which involves transforming malware into embeddings and then constructing a phylogenetic tree.The explanation of this pipeline is divided into two sections.

The first part details our approach to generating embeddings from malware samples through static, dynamic, and image analyses.It also discusses how these different types of analyses are combined into a single embedding. Additionally, we describe the experiments conducted to validate each individual embedding as well as the overall combined embedding. The second part delves into the construction of our phylogenetic tree using the combined embedding. In this section, we also explain our techniques for interpreting the tree in order to discern relationships within and among different malware families. Lastly, we highlight the experiments used to validate the accuracy of this tree.

4.1. Pipeline

Transforming malware instances into embeddings and then converting them into trees is a fairly intricate process. Our goal is to simplify this method by introducing a pipeline for creating phylogenetic trees from malware samples. Throughout each phase of the pipeline we will highlight tools, and approaches required to reproduce our approach.Our methodology is structured around four key phases, as depicted in Figure 4.1:

1. **Embedding Extraction (Embedding):** The first phase focuses on extracting embedding what we call ”DNA”, of malware aiming to capture genetic profiles from pseudo-static(static analysis on memory dumps), Dynamic and image analyses.

2. **Embedding Concatenation and Dimensionality Reduction:** Following extraction, the next step involves merging these sub-embeddings into a unified embedding for each malware sample. We do this along with reducing the total embedding dimension in a supervised fashion. This make it computationally feasible to build trees.

3. **Distance Matrix Development:** Next, we create a similarity matrix using the combined embeddings from different malware samples. This matrix quantifies the genetic divergences among various malware families, laying the groundwork for evolutionary analysis.

4. **Phylogenetic Tree Creation:** Using the distance matrix, the last step involves creating the phylogenetic tree. Two primary approaches, Neighbour Joining and UPGMA, are examined for constructing the phylogenetic tree.

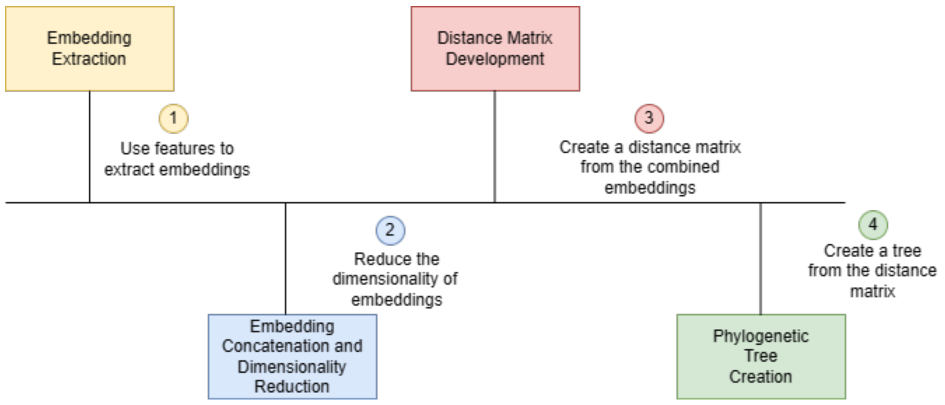

Embedding Extraction
1
Use features to extract embeddings
Distance Matrix Development
3
Create a distance matrix from the combined embeddings
2
Reduce the dimensionality of embeddings
Embedding Concatenation and Dimensionality Reduction
4
Create a tree from the distance matrix
Phylogenetic Tree Creation

Figure 4.1: High Level Pipeline

Part 1: Creation of Embeddings

In the first part, our approach involves extracting embeddings from malware samples through pseudo-static, dynamic, and image analyses. We then combine these diverse embeddings while applying dimensionality reduction to obtain the final embeddings, which we refer to as the genome of malware.

4.2. Embedding Extraction

This section outlines our methodology for extracting embeddings from malware through static analysis on dumps (pseudo-static), Dynamic analysis, and image analysis.

4.2.1. Motivation

We liken the embeddings we collect from (pseudo-static), Dynamic analysis, and image analysis to individual ”DNA” strands that collectively form the genome of a malware sample, similar to how the collective strands of DNA define biological species.

While we are pioneers in using embeddings for this comparison, organizations such as Intezer[170] have been utilizing genetic analysis for some time. Intezer identifies small code fragments as genes, which are then transformed into ”searchable tokens” for malware classification. Their analysis reveals that most examined malware shares over 50% of its code with previously identified strains[170], highlighting that the essence of malware identification lies in the preserved traits within a family. Therefore, even though we categorize malware into polymorphic, metamorphic, and evasive types based on behavior patterns—our approach is rooted in the observation that many malwares often retain significant traits from their ancestors across all categories.

4.2.2. Overview of Different Analysis Methods

We use three different analysis methods in our study. Pseudo-Static Analysis involves examining the malware's code from memory dumps to identify its structure, operations, and potential threats. This method is akin to sequencing a strand of DNA to understand its genetic blueprint, providing a detailed look into the underlying mechanics of the malware.

In Image Analysis, we create visual representations of the malware's code based on memory dumps. Although more abstract, this approach can uncover patterns, structures, or anomalies that aren't immediately apparent, similar to using electron microscopy to visualize a virus's physical structure and protein arrangements. This method offers a unique perspective on the malware's composition.

On the other hand, dynamic Analysis observes the malware in action within a controlled environment, such as a sandbox, to study its behavior, mechanisms of spread, and the effects it has on the host system. It mirrors the study of a virus's interaction with host cells in a lab setting, providing real-time insights into the malware's operational tactics and its immediate impact on systems.

4.2.3. Extraction of Memory Dumps

Before we delve into the extraction of embeddings from various analysis methods, it's important to address how we first obtain the executable file from memory dumps. This step is crucial due to the challenges associated with packing, as discussed in the background section.

Our approach tries to mitigates these issues by directly extracting the executable file from memory dumps.We apply this method to both pseudo-static and image analysis. In pseudo-static analysis, we perform static analysis on executable files extracted from memory dumps, which better reveals obfuscated and decrypted code. For image analysis, we use the executable file from memory rather than the raw executable, providing a clearer view of the malware's codebase rather than the packer. In this subsection we will highlight how we extract memory dumps using different tools from malware samples.

Tooling

We use several tools for malware analysis to enhance our investigation across various systems. DigitalOcean is one of the foundational tools we utilize, as it's a cloud infrastructure provider that allows us to scale and manage our computing resources efficiently. We emulate VirtualBox inside a digital ocean environment in order to create and manage virtual machines that emulate different operating systems. This enables safe execution of malware in a controlled environment, allowing us to observe its behavior without risking the integrity of the host system.

When analyzing memory, we use specialized tools tailored to the specific operating systems. On Windows, our primary tool is Procdump, a command-line utility developed by Sysinternals (now part of Microsoft). It generates memory dumps of processes based on specific triggers such as CPU spikes or unhandled exceptions. This tool is essential for capturing the memory state of a process during or after malware execution and provides deep insights into the operations of the malware. In contrast, for Linux or Darwin systems, we utilize Gcore—an efficient utility that creates core dumps of running processes without interrupting them. Both these tools can capture memory dumps without stopping process execution and are valuable for our monitoring and logging activities.

To futher analyze the structure of malware, we use Lief, a tool for parsing and manipulating binary files in ELF, PE, or Mach-O formats. We us lief to identifying and extracting executable sections within these files marked by `MZ` for PE files or `\x7fELF` for ELF files. By isolating these executable components, lief enables a more targeted and efficient analysis of the malware's binary structure.

Methodology

we begin by configuring a VirtualBox virtual machine on a Digital Ocean platform to emulate the target operating system for malware analysis. A snapshot of the VM's clean state is captured to serve as a restoration point for post-analysis cleanup. After executing the malware within this controlled environment, we monitor all system processes for five minutes to capture behavior and network interactions. Appropriate tools are employed to generate memory dumps: *Windows* systems use `procdump -ma` for complete memory dumps, while *Linux* systems use `gcore` to produce core dumps without stopping processes. These dumps are securely transferred to the host machine for analysis.

Sequential analysis of the memory dumps is conducted using the `lief` library to parse and identify binary formats and to determine the size of the binary from its header, which aids in establishing the endpoint of the executable within the dump. The executable segment is then extracted from the dump using the start and end points identified with `lief`, and the isolated executable is saved to a file for further detailed analysis.

Finally, the VM is restored to its pre-execution snapshot to eliminate all traces of malware and prepare the system for subsequent analyses. We operate under the strong assumption that the initial memory dump resembling a `PE/DOS` or `ELF` format is considered the main executable, allowing us to automate the extraction process across an extensive dataset of 103,883 samples.

4.2.4. Features Extraction From Psuedo-Static Analysis

This section begins by detailing the features used in our analysis. It then proceeds to explain the methods employed to extract these features from PE, ELF, and DOS executables.

High Level overview of all features

Before delving into the specifics of our feature extraction process, we present a comprehensive overview in Table 4.1, summarizing all the features utilized in our study. Additionally, we indicate whether these features have been employed in related research. It is important to note that previous studies have only focused on PE formats. In our analysis, we have extended these techniques to include ELF and DOS formats as well.

Table 4.1: We use all of the features across PE, ELF, DOS formats

Feature	PE	ELF	DOS	Prior Work
ByteHistogram	✓	✓	✓	[58] [60] [64] [66][65] [67]
ByteEntropyHistogram	✓	✓	✓	[64] [66] [65] [67]
Strings	✓	✓	✓	[69, 71] [64] [66] [67] [65]
GeneralFileInfo	✓	✓	✓	[60] [64] [66] [65]
HeaderFileInfo	✓	✓	✓	[62]
SectionInfo	✓	✓		[60] [67] [66]
ImportsInfo	✓	✓		[59]
ExportsInfo	✓	✓		[60]
EntryPoints	✓	✓	✓	[58]
ExitPoints	✓	✓	✓	[62]
Opcodes	✓	✓	✓	[61]
OpcodeOccurrences	✓	✓	✓	[61]
ImageSize	✓	✓	✓	[63]
HeaderSize	✓	✓	✓	[63]
StackReserveSize	✓			[63]
StackCommitSize	✓			[63]
HeapSize	✓	✓		[63]
LoaderFlags	✓	✓		[63]
KolmogorovCompression	✓	✓	✓	[171]
DataDirectories	✓			[66] [64] [65] [63]
MemorySize			✓	[63]
BlockEntropy			✓	
InterruptInfo			✓	
StackSize			✓	
SegmentInfo		✓		
GNUStackPhysicalSize		✓		
SectionEntropy	✓	✓		[63]

Feature Extraction Across Different Executable Formats

All of our Portable Executable (PE) features, as detailed in Table 4.1, are derived from related work. For DOS and Executable and Linkable Format (ELF) executables, we aimed to maintain a consistent feature set with PE files. However, DOS, being an older format, inherently lacks the capacity for as detailed feature extraction as PE. In the case of ELF, certain adaptations were necessary; for example, instead of the *StackReserveSize* and *StackCommitSize* features common in PE, we use *GNUStackPhysicalSize* as detailed in Table 4.1. Generic extraction methods for all features are outlined in Table 4.2. For detailed extraction processes specific to each feature type, refer to Appendix D for PE features, Appendix F for ELF features, and Appendix E for DOS features.

Table 4.2: Extraction Processes for Generic Features

Feature	**Extraction Process**
ByteHistogram	Uses `LIEF` to target sections with `MEM_EXECUTE` and `NumPy` to compute byte frequency histograms.
ByteEntropyHistogram	Analyzes entropy within executable sections using `LIEF` for section targeting and `NumPy` for entropy calculation.
Strings	Employs regex and disassembly using `Capstone` to extract ASCII strings and identify significant patterns like URLs and file paths.
GeneralFileInfo	Leverages `LIEF` to parse and extract general file metadata, such as size, and security attributes from headers.
HeaderFileInfo	Utilizes `LIEF` to gather detailed metadata from COFF and optional headers, focusing on architecture and system version.
SectionInfo	Uses `LIEF` to analyze properties like name, size, and entropy of each section, identifying roles and security features.
ImportsInfo	Uses `LIEF` to parse the import address table, detailing external library dependencies and associated API functions.
ExportsInfo	Employs `LIEF` to access and list functions in the export table, revealing capabilities offered to other processes.
EntryPoints	Utilizes `LIEF` to identify main and secondary entry points from headers and export tables.
ExitPoints	Combines `LIEF` for structure analysis and `Capstone` for disassembly to locate and analyze exit function calls.
Opcodes	Uses `Capstone` to disassemble executable sections marked with `MEM_EXECUTE` and list operational codes.
OpcodeOccurrences	Follows opcode extraction with a frequency analysis using Python dictionaries to tally opcode occurrences.
ImageSize	Computes total virtual size by aggregating section sizes using `LIEF` or `PEfile` to parse the PE format.
HeaderSize	Summarizes the combined size of all headers using `LIEF` or `PEfile` for a detailed structural assessment.
StackReserveSize	Extracts reserved stack size directly from the optional header using `LIEF` or `PEfile`.
StackCommitSize	Retrieves memory committed to the stack at start-up from the optional header using `PEfile`.
HeapSize	Parses the optional header with `PEfile` to calculate the sizes reserved and committed for the heap.
LoaderFlags	Checks loader flags set during executable loading using `PEfile` to parse the optional header.
KolmogorovComplexity	Applies `zlib` compression to executable sections identified by `LIEF` to measure data redundancy.
DataDirectories	Iterates with `LIEF` over data directories detailing import and export tables and resource sections.
MemorySize (DOS)	Parses the DOS header with `struct` to extract memory size specifications, including maximum and minimum allocation.
BlockEntropy (DOS)	Divides the file into blocks and calculates entropy for each using Python's standard libraries to assess data randomness and complexity.
InterruptInfo (DOS)	Employs `Capstone` for disassembly of the DOS executable to identify and document software interrupt instructions, providing insight into system interactions.
StackSize (DOS)	Extracts stack size directly from the DOS header using `struct`, offering insights into the initial stack setup and memory management.
SegmentInfo (ELF)	Uses `LIEF` to parse ELF segments, collecting details on segment type, physical size, and virtual address to analyze the binary's memory layout and security characteristics.
GNUStackPhysicalSize (ELF)	Utilizes `LIEF` to identify the GNU_STACK segment in ELF binaries and extracts its physical size, indicating memory allocation for the stack.
SectionEntropy	Calculates entropy within file sections across formats to detect encryption or obfuscation using tools like `NumPy`.

4.2.5. Embedding Extraction From Pseudo-Static Analysis

This section outlines our methodology for extracting embeddings from Pseudo-Static Analysis . We begin by justifying our choice of the text-embedding-3-large model over other available text-embedding model variants. Next, we adjust the format of features extracted through pseudo-static analysis to align with the input specifications of the OpenAI text-embedding-3-large model, herein referred to as the GPT-Text model. We then detail the process for generating embeddings from these adjusted features and discuss the techniques we employ to validate these embeddings.

Model

In our study, we opted for OpenAI's GPT-Text text-embedding-3-large(GPT-Text) model over alternatives such as text-embedding-ada-002 and text-embedding-3-small, due to its advanced capabilities and enhanced performance, which are crucial for efficiently handling complex JSON-formatted data. OpenAI's text embedding models are engineered to convert text into numerical vectors, facilitating a broad spectrum of applications from semantic search to multilingual information retrieval, underscoring OpenAI's dedication to improving text processing technologies.

The text-embedding-3-large model excels significantly beyond its predecessors, achieving impressive results in benchmark tests with scores of 54.9% on the MIRACL benchmark and 64.6% on the MTEB benchmark[172]. Capable of generating embeddings with up to 3072 dimensions, this model offers a richer representation of text, which is essential for analyzing intricate data structures.

Additionally, its use of Matryoshka Representation Learning[172] enables an efficient trade-off between performance and computational expense, allowing for the reduction of embedding dimensions without substantial loss of accuracy. This feature makes the text-embedding-3-large model particularly suitable for our research objectives.

Formatting

We opted to store features as JSON strings, despite the fact that the GPT-Text model accepts text. Our decision was based on our belief that using JSON helps the model correlate features with their corresponding values more effectively. In addition, JSON not only simplifies storage but also enables efficient conversion back to original data formats such as strings, numbers, arrays, or dictionaries which is useful in truncation, as we will see later.

Embedding Extraction

To generate embeddings from Pseudo-Static features, we categorize the features into two types: simple features, consisting of numerical arrays, and complex features, which encompass structures like dictionaries and lists containing both strings and numbers.

Numerical Features as Direct Embeddings

ByteHistogram and ByteEntropyHistogram are directly used for embeddings because they are in numerical array form, unlike other features such as EntryInfo, which contains a mix of strings and numbers. Simple numerical features like StackSize or KolmogorovComplexity, despite being scalar values, are not directly suitable for embeddings since they are not vectors. Instead, we utilize GPT-Text model to convert these single values into embeddings, as detailed later.

Additionally, using ByteHistogram and ByteEntropyHistogram as direct embeddings offers further benefits. The distinct signature provided by ByteHistogram outlines the frequency of byte values within a file, helping to identify malware through recognition of unique or anomalous patterns. Similarly, ByteEntropyHistogram evaluates the randomness or entropy in byte distribution, crucial for detecting advanced malware employing compression,or obfuscation techniques to evade detection. These features can thus be used directly as embeddings to differentiate malware. These two features were combined to create a composite 512-dimensional 1D embedding, with each feature contributing 256 dimensions. This forms the initial segment of the pseudo-static embedding derived from pseudo-static analysis.

Complex Features

For complex features, we input them directly into the GPT-Text model. However, the model has a truncation limit of 8191 tokens, necessitating the truncation of JSON strings. To manage this, we convert the JSON strings back into their original data structures, allowing us to apply truncation where needed, as explained next.

Truncation

Several techniques can effectively truncate JSON structures, drawing from established strategies in algorithms and data structures. These strategies include hierarchical truncation, sliding window, top-K elements, recursive truncation, and tail trimming. Hierarchical truncation prioritizes elements based on their importance to ensure that key data is preserved for longer durations. Sliding window divides text into overlapping segments to accommodate the token limit while maintaining contextual coherence.

Top-K elements selectively retain the most critical components within lists or dictionaries by considering factors like frequency or relevance. Recursive truncation applies truncation strategies to sub-elements within JSON, particularly beneficial for deeply nested structures. Tail trimming involves removing elements from the end of lists or sequences under the assumption that initial data holds greater relevance.

In our truncation strategy, we implement a straightforward greedy approach using tail trimming, which, despite its simplicity, proves to be effective. Initially, we determine the necessary token counts using the TikToken tokenizer utilized by OpenAI. We then focus on key features; entryInfo, exitInfo, imports, exports, and information about sections or segments of the executable. These features are targeted due to their extensive lists that significantly contribute to the overall token count. To optimize token reduction, we evaluate each feature by temporarily removing its last element, recording the decrease in tokens, and then restoring it. If the removal of elements from a particular feature results in more token reduction compared to other features, we prioritize its truncation by permanently removing the last elements.

As we truncate these features, we impose a minimum length requirement of 5 entries for each to retain fundamental detail. If the token count still remains above the limit, we iteratively decrease the opcodes feature, removing 10 at each step while preserving critical information with a minimum length of 20. Overall, this strategy enables us to efficiently manage the token limit constraint by systematically reducing the size of token-heavy elements and opcodes. By doing so in a controlled manner, we ensure that the reduction in tokens is achieved with minimal impact on the features' content.

Embedding Generation

The embeddings, which are 1D vectors, were created by inputting complex features into the GPT-Text model with the dimensionality set to 3000. This specific dimension was chosen because it adequately captures the context of the features. To obtain the final embedding, we combined both direct and complex embeddings, resulting in a composite one-dimensional embedding of length 3512.

Validation

To validate the Numerical and Complex embeddings derived from pseudo-static analysis, we employ logistic regression. This approach is predicated on the assumption that robust embeddings should enable a simple linear model to outperform a baseline classifier, which predicts based solely on the most frequent class without using any features. A significant improvement in logistic regression's performance over the baseline would imply good embeddings.

To test this, we modeled Logistic Regression as a linear network using cross entropy loss in PyTorch. To address class imbalances and achieve a robust evaluation, we employed stratified nested cross-validation across 10 outer folds and 5 inner folds. The grid search, which consists of values for L1 and L2 regularisation and initial learning rates, is presented in Appendix I in table I.1. Note that the learning rate is adjusted in each fold using the learning Rate Scheduler that decreases the learning rate based on the patience of validation fold. The results of this experiment are outlined in the results chapter under Experiment 1.

Table 4.3: Hyperparameters for Validation of Embeddings

Hyperparameter	Setting
Regularization	L1
Initial Learning Rate	0.01
Learning Rate Scheduler	ReduceLROnPlateau (mode: max)
Patience	10 epochs using validation accuracy
Batch Size	64
Random State	42
Optimizer	Adam

4.2.6. Features Extraction From Dynamic Analysis

This section outlines the features derived from dynamic analysis, describes the extraction process using our sandbox environments, and details how these features are transformed into embeddings.

High Level Overview of all features

Features that we used for dynamic analysis was inspired from several prior works as seen in table 4.4. This table also specifies the maximum number of entries retained for each list-based feature.

Table 4.4: We use all of the features from Dynamic Analysis as listed in the table

Feature	Entry Limit	Prior Work
Files Opened	30	[109] [110]
Files Written	30	[110] [111]
Files Deleted	30	[110]
Command Executions	20	[113] [112]
Files Attribute Changed	20	[114] [115]
Processes Terminated	20	[109]
Processes Killed	20	[113]
Processes Injected	20	[111] [112]
Services Opened	20	[115]
Services Created	20	[115]
Services Started	20	[110]
Services Stopped	20	[110]
Services Deleted	20	[110]
Windows Searched	20	[111]
Registry Keys Deleted	10	[109] [114]

Feature Extraction Process

To extract features, we ran the malware samples using Any.Run and systematically collected behavioral data through its API. In cases where the Any.Run API failed, we used VirusTotal Api to obtain the behavioral data. All extracted features are list-based and include both strings and numbers. We established entry limits for each feature, as outlined in table 4.4, in order to meet the token requirements of the GPT-Text model, which will be discussed in the next subsection.

4.2.7. Embedding Extraction From Dynamic Analysis

This section describes our approach to extracting embeddings from dynamic analysis features, using the same GPT-Text model as in the pseudo-static analysis. We explain the rationale behind truncating our features according to the limits specified in Table 4.4. Following this, we detail the process of converting these truncated features into embeddings using the GPT-Text model.

Truncation

The truncation strategy in Table 4.4 was implemented to keep feature lengths within the token limit of the GPT-Text model. Unlike the greedy strategy used in pseudo-static analysis, we set fixed entry limits for each feature to expedite processing, as dynamic analysis often generated extensive list-based features. For features that still exceeded the token limit after applying these entry limits, manual truncation was employed to meet the constraints of the GPT-Text model.

Final Embedding

The final embeddings were generated as 1000-dimensional vectors by feeding the dynamic analysis features into the GPT-text model with the dimensionality parameter set to 1000. Although this dimensionality was chosen arbitrarily, initial logistic regression tests comparing performance between 1000 and 2000 dimensions showed minimal difference, prompting us to opt for the lower dimensionality.

Exceptions
When analyzing malware from specific families, such as MagicRat , AsyncRat, we sometimes found that AnyRun's API or Virustotal API did not provide any behavioral data. In these cases, we used the average embedding for the malware family in question. However, our approach during testing differed: because we could not identify the family of a test sample in advance, we defaulted to returning a zero vector.

Validation
We employ the same validation methodology as outlined in section 4.2.5, using the identical hyperparameters and model described in table 4.3. For gridsearch values, please see table I.1. The results of this experiment are outlined in the results chapter under Experiment 1.

4.2.8. Embedding Extraction From Image Analysis

Our approach to extracting embeddings from image analysis differs from the methods used in pseudo-static and dynamic analysis. We employ a model that automatically extracts features from images, eliminating the need for manual feature selection. In this section, we motivate our decision to use the ResNet50 model for image training. We then describe our training pipeline, which begins with initial training on public datasets—MalwareNet, Malevis, and Malimg i.e ImageDataset followed by fine-tuning on our MalwareGenome dataset before extracting embeddings.

Model

We evaluated several models, including various ResNet versions and VGGNet, for our image-based malware classification task. We ultimately chose ResNet50 due to its effective balance between complexity and performance. Testing on the MaleVis dataset showed that ResNet101 quickly reached 98% validation accuracy within just 5 epochs but demanded considerable computational resources. In contrast, ResNet50 achieved the same accuracy over 20 epochs, providing a more manageable trade-off between time and performance, while simpler version like ResNet-34 achieved 95% accuracy only in 30 epochs.

We ruled out other architectures such as ResNet-110, ResNet-152, ResNet-164, ResNet-1202, and VGG Net due to hardware constraints. Despite our server's robust specifications—an A40 GPU and 1TB of RAM—these larger models exceeded the GPU's memory capacity, making them impractical for our needs. Thus, ResNet50 was selected as it offered the best compromise between performance and resource utilization for our application.

Methodology

Our methodology is explained through a 3 step pipeline is shown in Figure 4.2.Initially, we preprocess the ImageDataset to ensure compatibility with the V1 model, which is initialized with ImageNet weights. Subsequently, we fine-tune this model on our dataset, referred to as the V2 model. Finally, we extract embeddings from this fine-tuned model. In the following sections, we will elaborate on each stage of this pipeline in detail.

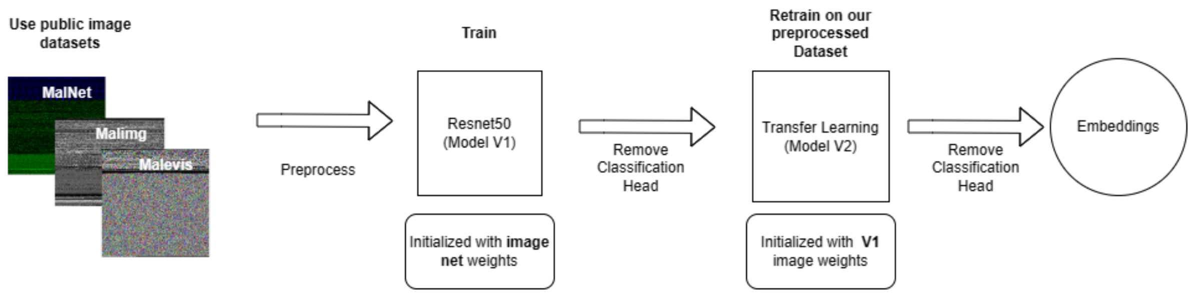

Use public image datasets
MalNet
Malimg
Malevis
Preprocess
Train
Resnet50 (Model V1)
Initialized with image net weights
Remove Classification Head
Retrain on our preprocessed Dataset
Transfer Learning (Model V2)
Initialized with V1 image weights
Remove Classification Head
Embeddings

Figure 4.2: Image Analysis Pipeline

Training Model V1

The ResNet model in our analysis is initialized with ImageNet weights, designed primarily for RGB images. To accommodate this, several preprocessing steps, before training model V1, are necessary for the ImageDataset, particularly for the Malimg Dataset, which consists only of grayscale images. First, these images are converted to RGB by duplicating the single grayscale channel into three channels, ensuring that the input format matches what the ResNet model expects.

Second, normalization of the images is performed using predefined mean values `([0.485, 0.456, 0.406])` and standard deviations `([0.229, 0.224, 0.225])` across RGB channels. This process standardizes pixel intensities, centers the data around zero, and is essential for promoting faster convergence during the training phase. Third, the images are resized to 224x224 pixels to comply with the input dimensions that ResNet models are optimized for. This uniformity is critical for consistent image processing through the model. Upon , preprocessing Model V1 was trained with a stratified train test split of 70/30. Table 4.5 highlights the hyperparameters that was used. With this setup, our model V1, achieved a 85% accuracy on the test set.

Table 4.5: Hyperparameters for Models V1 and V2

Parameter	**Value**
Optimizer	Adam
Patience	10
Batch Size	64
Random State	42
Initial Learning Rate	0.01
Learning Rate Scheduler	ReduceLROnPlateau (mode: max, factor: 0.1, patience: 10 epochs)
Weight decay	0.0001

Training Model V2

Before training on model V2, the executables from memory of MalwareGenomeDataset are first converted to images using the `bin2png` tool as discussed in the background. Then we apply the same preprocessing steps as for ImageDataset where we normalize using mean values `([0.485, 0.456, 0.406])` and standard deviations `([0.229, 0.224, 0.225])` and resize the images to 224x224 pixels. Model V2 is then initialized with the pre-trained weights from modelV1, with the exception of classification layer, which has been now replaced to accommodate an output of 538 classes. This modification is important for the process of transfer learning, as it leverages pre-existing learned patterns while tailoring the model's output to the new classification task. Just as model V1, we use the same stratified 70-30 split for training and validating model V2 using the same hyperparameters as table 4.5. After training, we found that the accuracy was 93% on the validation dataset.

Comparative Analysis of Our Approach and Traditional CNNs

We assess Model V2 by comparing its performance against the grayscale CNN architecture developed by Gilbert et al. [103] and a modified version of this architecture, which has been adapted to accept RGB (3-channel) inputs instead of grayscale (1-channel). These comparisons are conducted on our dataset using the hyperparameters outlined in Table 4.5. To ensure compatibility with Gilbert et al.'s[103] architeture, we convert our images to grayscale using the PIL library, and then compare our architecture against their architecture using stratified cross-validation with 10 outer folds. The outcomes of this comparison are discussed in Experiment 3 of the results chapter.

Embedding Generation

The final embedding is obtained by freezing all model weights and performing a forward pass for each tensor corresponding to its sample SHA. This process produces embeddings with a dimensionality of 2048. We validate the final embedding using the same approach as for pseudo-static and dynamic analyses, employing stratified nested cross-validation with logistic regression. The hyperparameters used are consistent with those detailed in table 4.3. The outcomes of this experiment are outlined in the results chapter under Experiment 1.

Image Embeddings for Benign and Malicious Malware Classification

Interestingly, image embeddings alone can be effectively used for malware detection, categorizing samples as either benign or malicious. For benign samples, we employ the dataset from Androzoo, which uses an adapted 'bin2png' encoding technique similar to our dataset. We use Model V2 to process images from Androzoo, freezing the weights for a forward pass to generate benign embeddings. These benign embeddings are then merged with our malicious embeddings to create a new dataset. In this dataset, 65.23% of the samples are labeled as malicious and 34.77% as benign.

We apply logistic regression to this binary classification task, utilizing the hyperparameters configuration in Table 4.3. To ensure robust evaluation, we conduct cross validation with 10 outerfolds and 5 inner-folds. The results of this analysis are thoroughly presented in Section 4.2 of the results chapter.

4.3. Embedding Concatenation and Reduction

In this section, we detail how we merge these embeddings into one embedding in a supervised fashion.

4.3.1. Current Approaches

Concatenating embeddings is a widely used technique in machine learning to enhance data representation by combining features from different models or sources.

Simple Concatenation is the most straightforward approach where feature vectors from different embeddings are combined into a single comprehensive vector. This technique assumes that each element of the combined vector independently contributes to the overall data representation, as noted by Zheng et al.[173]. Weighted Concatenation adjusts the influence of different embeddings before their integration by assigning specific weights. This approach is particularly useful when the significance of each embedding varies, such as in Yang et al.[174] where they use hierachial attention networks for document classification.

Feature Fusion extends beyond simple concatenation by employing operations like addition, multiplication, or more complex transformations to blend embeddings together. For instance, Zadeh et al. [175] uses feature fusion to merge embeddings from diverse modalities for multimodal sentiment analysis. Neural Networks also facilitate the integration of embeddings by using concatenation followed by dense layers or by applying attention mechanisms that dynamically assign weights to different embeddings. The Transformer model introduced by Vaswani et al.[51] is a prime example, utilizing attention mechanisms to blend and balance embeddings for various tasks such as translation and text generation.

4.3.2. Our approach

Our method involves supervised dimensionality reduction, where we stack embeddings from three different analyses: pseudo-static analysis (3512 dimensions), dynamic analysis (1000 dimensions), and image analysis (2048 dimensions). This results in a hybrid 1D embedding of 6560 dimensions, which we then dimensionally reduce through supervised classification based on malware family class labels. This approach is illustrated in Figure 4.3.

Motivation

The primary reason for reducing the dimensions of merged embeddings is to lessen computational demands. By compressing the dimensionality from 6560 to 1000, we make subsequent tasks, such as 2D visualization using methods like UMAP or t-SNE, more feasible. This reduction also simplifies validation with logistic regression and decreases the computational requirements for tree building processes. Overall, this dimensionality reduction facilitates easier handling and further analysis.

Preprocessing

Before, we reduce and combine our hybrid embeddings. we need to normalize them. Normalization is a import preprocessing step for hybrid embeddings, especially given the substantially larger magnitude values of image embeddings compared to behavioral and numerical embeddings. By ensuring all data types contribute equally, normalization prevents any single feature set from disproportionately influencing the learning process. This uniform scaling not only enhances model stability and accelerates convergence but also facilitates meaningful feature comparisons across different scales, crucial for accurate distance measurements in analyses. Moreover, consistent normalization helps improve model

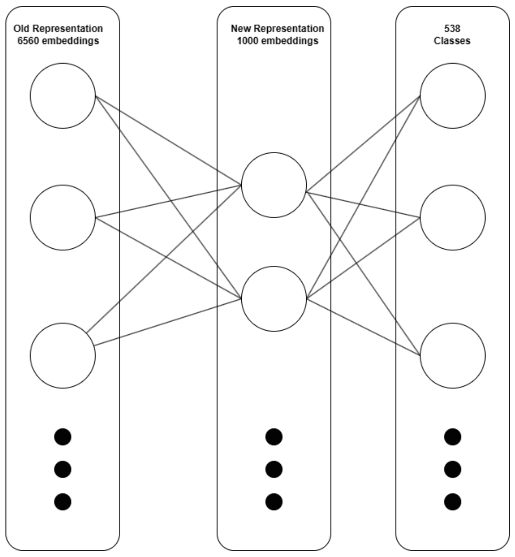


Figure 4.3: Representation Pipeline showing how we reduce embedding dimensions

generalization, preventing overreliance on specific feature magnitudes which might help it generalize on new data.

Model

Upon preprocessing, we reduce the dimensions using a 2 layer fully connected neural network with cross entropy loss, we start with 6560 input neurons, a hidden layer of 1000 and final output layer of 538 corresponding to number of malware families. We train the embeddings with the hyperparameters as summarized in table 4.6. After training we remove the final classification layer, the last layer in Figure 4.3. We then proceed to perform a forward pass with the network's weights frozen for each malware sample associated with its tensor. This step enables us to extract the final 1000-dimensional embeddings for each sample.

Table 4.6: Hyperparameters for Supervised Dimensionality Reduction

Parameter	**Setting**
Initial Learning Rate	0.01
Learning Rate Scheduler	ReduceLROnPlateau (mode: max, factor: 0.1, patience: 10 epochs)
Patience for Validation	10
Optimizer	Adam
L1 Regularization	Lambda = 0.0001

Final Embeddings

To generate the final embeddings, the classification layer was removed, and for each malware sha, the corresponding tensor was retrieved. A forward pass was then used, with the weights freezed, to produce 1000 final 1D embedding. To evaluate the final embeddings we use a logistic regression model with the same hyperparameters as in table I.1. The outcome of this experiment is highlighted in results chapter under Experiment 1. The grid search parameters are described in Appendix I under table I.1.

Evaluating Embedding Effectiveness for Temporal Classification

We assess the utility of final embeddings for tasks transferred to different contexts. Specifically, we explore their effectiveness in determining the age of malware. To do this, we organize the malware embeddings by year and train a logistic regression to predict the year based on these embeddings. The outcomes of this study are detailed in the results chapter under Experiment 2. The hyperparameters employed are outlined in in Appendix I.

Part 2: Building Phylogenetic trees

Now that we have looked at how we developed our embeddings, this part delves into using the embeddings to generate a phylogenetic tree. We first highlight how we can build a distance matrix from the embeddings and convert it into a format that's usable for building trees. Then, we discuss the methods and assumptions used to construct our phylogenetic tree using UPGMA and NJ. Lastly, we focus on validating our results and deriving relationships from our tree.

4.4. Distance Matrix Generation

This section outlines the process of constructing a distance matrix from the embeddings described in the previous part and then converting it into a PHYLIP-formatted distance matrix required for generating phylogenetic trees.

4.4.1. Known Approaches

Various approaches exist for computing distances between embeddings, each with unique characteristics suited for different applications. The Euclidean distance is the most straightforward way to measure the straight-line distance between two points in Euclidean space and is known for its intuitive nature and simplicity. Conversely, Cosine similarity evaluates the cosine of the angle between two vectors, making it especially useful for determining directional similarity independent of vector magnitude.

Manhattan distance is another method that computes the sum of the absolute differences of their coordinates, providing a metric for distance calculation that measures the path along axes at right angles. For categorical data,typically Jaccard similarity index is employed; it measures similarity between finite sample sets by dividing the size of their intersection by the size of their union. Lastly, Hamming distance is used primarily to compare strings of equal length by counting how many positions have corresponding symbols that differ. It offers a critical tool for error correction and detection applications.

When constructing phylogenetic trees, the selection of a distance metric is influenced by the characteristics of the embeddings and the assumed evolutionary model. In numerous situations, Euclidean distance is favored due to its simplicity, particularly when dealing with continuous vector space embeddings that represent measurable traits or genetic distances which can be conceptualized in geometric terms[176]. Therefore, in our approach we use euclidean distances.

4.4.2. Distance Matrix

When calculating the Euclidean distance between pairs of samples, each represented by a 1000-dimensional embedding, the process involves making pairwise comparisons across all samples. This results in $O(n^2)$ computational complexity. The procedure entails iterating over each of the n samples and performing nested iterations for every other sample to compute distances. Specifically, this means that for each sample, distances to all other samples are calculated based on their embeddings, resulting in $\frac{n(n-1)}{2}$ total distance calculations. This sequential and nested iteration approach inherently leads to a computational complexity of $O(n^2)$.

4.4.3. Converting Distance Matrix

Phylogenetic analysis tools often require specific data formats for processing, and the PHYLIP format is widely used for distance matrices [177]. This format is crucial because it allows these tools to accurately interpret the necessary data to construct phylogenetic trees. As a result, we convert our distance matrix into the PHYLIP format to ensure compatibility with phylogenetic analysis software, especially for methods like Neighbor Joining that we plan to use later.

The PHYLIP format consists of a header that indicates the number of taxa, which in our case refers to malware samples. Following the header, the format details each taxon with its name followed by the distances to all other taxa. In the next subsection, we will explain the time complexity of converting our distance matrix to PHYLIP format [178].

Time Complexity

The conversion is done by iterating systematically over an $n \times n$ matrix, where n is the number of malware samples. The conversion includes writing a header line to state the total number of samples and then formatting each sample along with its computed distances to all other samples in compliance with PHYLIP specifications. The computational complexity of this conversion is $O(n^2)$ due to the need to process each matrix element once. The end result of this conversion process is a PHYLIP-formatted distance matrix of size $103,883 \times 103,883$, encompassing all pairwise Euclidean distances between the samples in our dataset, ready for subsequent phylogenetic analysis.

4.5. Building Phylogenetic Trees

In the background we discussed five approaches were examined: Maximum Parsimony, NJ, UPGMA, Maximum Likelihood, and the Bayesian method. Only Neighbor Joining and UPGMA can be used with distance matrices, while the other methods require sequences. Therefore, in our approach we will use NJ and UPGMA.

4.5.1. UPGMA

The Unweighted Pair Group Method with Arithmetic Mean is a hierarchical clustering technique with average linkage that is used to construct phylogenetic trees [73] . It groups taxa based on pairwise distances, progressively merging the closest clusters and recalculating distances until a single cluster forms. UPGMA makes several assumptions: it requires a consistent mutation rate across all lineages (molecular clock hypothesis), accurate distance measurements between taxa, and uniform evolutionary rates across all branches.

Implementation

In our methodology, the UPGMA algorithm is implemented using the SciPy library, which offers comprehensive tools for hierarchical clustering.

We start by loading the evolutionary distances and taxa labels from a PHYLIP-formatted distance matrix to prepare the data for subsequent clustering operations. We employ the `cluster.hierarchy` module from SciPy, utilizing the `linkage` function set to the 'average' method for UPGMA clustering. This function computes the arithmetic mean distance between clusters to create a linkage matrix, which outlines the hierarchical structure of the taxa. After completing the clustering, the linkage matrix is converted into a tree structure through the `to_tree` function. This conversion enables further analysis and manipulation of the phylogenetic relationships.

To make the tree compatible with various phylogenetic visualization software, we convert the tree into a Newick format string using SciPy's `dendrogram` function. These visualizations are crucial for interpreting the evolutionary relationships within the tree, and they form an integral part of the analysis presented in subsequent sections of our study.

Time Complexity

When assessing the feasibility of applying the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) to our extensive dataset of 103,883 samples, the inherent time complexity of the algorithm is a pivotal consideration. UPGMA operates with a quadratic time complexity of $O(n^2)$, where n represents the number of samples involved [179].

The quadratic complexity arises from the algorithm's procedures. Initially, UPGMA computes pairwise distances between all samples, necessitating $O(n^2)$ operations. This step establishes the foundation for all subsequent clustering activities. In each iteration, UPGMA identifies and merges the pair of clusters with the smallest distance, a process that is repeated $n-1$ times. Each merge reduces the number of clusters by one, necessitating the recalculation of distances between the newly formed cluster and all other remaining clusters.

These recalculations, involving averaging distances from the merging clusters to all other clusters, if implemented straightforwardly, accumulate computational efforts approaching $O(n^2)$ across all iterations, significantly contributing to the total computational demand. To address this, we have integrated several optimization strategies to streamline the process. Min-heaps are employed to efficiently manage the retrieval of the minimum distance during each iteration of the clustering process, ensuring that the smallest distance is accessible in $O(1)$ time and that updates can be performed swiftly in $O(\log n)$ time. Additionally, we utilize a caching mechanism to store previously calculated distances between clusters, significantly reducing the need for redundant computations.

4.5.2. Neighbor Joining

The Neighbor-Joining (NJ) method is a clustering algorithm used for phylogenetic tree construction that does not assume a constant rate of evolution across all lineages, accommodating variable evolutionary rates [158]. It begins with a starlike phylogeny, positioning all taxa equidistant from a central point. NJ iteratively identifies pairs of taxa that minimally increase the total tree length when joined, recalculating distances between newly formed clusters and remaining taxa until a single tree is formed. Key assumptions of NJ include the accuracy of the input distance matrix, minimization of the overall tree length based on the parsimony principle, and an initial starlike phylogeny to simplify early clustering steps

Time Complexity

The Neighbor-Joining method, with its $O(n^3)$ time complexity, presents significant computational challenges, particularly for large datasets. Each iterative step in the NJ process involves merging a pair of taxa and recalculating distances to the remaining taxa, leading to extensive computational requirements. For our dataset of 103,883 samples, the cubic complexity of the NJ method results in prohibitive computational demands. Moreover, the method's predominantly sequential algorithmic nature restricts the feasibility of parallelizing processes, further exacerbating the challenges of managing such a large matrix efficiently. Given these limitations, utilizing the NJ method for our dataset was deemed unfeasible due to the intensive resource requirements and extended processing times needed.

4.5.3. RapidNJ

Given the high computational complexity of the standard Neighbor-Joining (NJ) method, especially with large datasets, we employed RapidNJ, an optimized algorithm developed by Simonsen et al.[180] that enhances computational efficiency for constructing phylogenetic trees using a modified version of NJ. RapidNJ introduces several key innovations that help reduce the overall time complexity, including optimizing the storage and access of the distance matrix using advanced data structures. This minimizes the computational overhead involved in scanning and updating the matrix. Additionally, RapidNJ does not exhaustively compare all possible pairs of taxa. Instead, it uses heuristics to quickly identify the most promising pairs for merging, thereby accelerating the iterative tree construction process.

RapidNJ employs a sparse matrix format to store the distance matrix, which significantly reduces memory usage and processing time, particularly beneficial when many distances are infinite or undefined. Instead of recalculating the entire distance matrix after each merge, RapidNJ only updates distances that are affected by the merge. This selective updating reduces the number of operations per iteration. It also uses a min-heap (or priority queue) to efficiently find the pair of clusters with the smallest distance, allowing the algorithm to quickly access the minimum distance without scanning the entire matrix. This speeds up each iteration and, combined with heuristic reductions, minimizes the computational burden of exhaustive search.

To further enhance efficiency, RapidNJ is designed to take advantage of multi-core processors by parallelizing the distance matrix update process. This parallelization allows simultaneous updates to different parts of the matrix, thereby accelerating the overall tree construction process. Additionally, RapidNJ performs computations directly on the matrix without creating additional copies, minimizing memory overhead and avoiding unnecessary data movement. It also includes options to adjust branch lengths post-calculation to ensure they are non-negative, crucial for maintaining the biological relevance and interpretability of the phylogenetic tree.

Implementation

RapidNJ is readily accessible for download as open-source software. However, for our implementation, we opted to use the BioConda platform for installation. BioConda serves as a dedicated repository within the Conda package management system, tailored specifically for the bioinformatics community.

To run RapidNJ on our dataset of 103,883 samples, we utilized 20 cores for parallel processing. The command used to run RapidNJ is:

```
rapidnj distancematrix.phy -p 20 -o newickoutput.txt -l
```

This command processes the input distance matrix using 20 cores and outputs the resulting phylogenetic tree in Newick format. The inclusion of the '-l' option adjusts any negative branch lengths to a non-negative value, typically zero, ensuring the biological plausibility of the evolutionary distances and relationships depicted in the phylogenetic tree.

Time Complexity and Trade Off

RapidNJ enhances the efficiency of phylogenetic tree construction by reducing the standard Neighbor-Joining (NJ) method's cubic time complexity to near quadratic. The worst-case time complexity for RapidNJ is $O(n^3)$, which occurs when the heuristic minimally reduces the number of recalculations needed after each merge. In the best-case scenario, particularly when the heuristics effectively minimize recalculations and quickly identify optimal taxa pairs for merging, RapidNJ can achieve a time complexity close to $O(n^2)$. This significant improvement is due to RapidNJ's advanced data structures for efficient distance matrix management and selective updating mechanisms that optimize computational steps and reduce overall processing time.

While RapidNJ offers substantial improvements in computational efficiency over the traditional Neighbor-Joining (NJ) method, it introduces certain trade-offs that could affect the accuracy and granularity of the phylogenetic trees it constructs. One significant compromise is the reduced granularity in pairwise comparisons. Unlike NJ, which exhaustively evaluates all possible pairs of taxa, RapidNJ employs heuristics to quickly identify the most promising pairs for merging. This method significantly accelerates the tree construction process but may overlook some taxa pairings that would be considered under a full NJ analysis. This reduction in granularity can potentially miss subtle evolutionary relationships, thereby affecting the tree's overall detail and accuracy. Additionally, RapidNJ's strategy to minimize recalculations of the entire distance matrix after each merge leads to the use of approximations rather than exact recalculations. This approach can result in slight inaccuracies in distance measurements compared to the more precise calculations performed by the NJ method, potentially altering the fine details of the phylogenetic tree's structure.

4.6. Validating Large-Scale Phylogenetic Trees

This section outlines the validation approach for phylogenetic trees derived via UPGMA and Neighbor-Joining methods. We first identify the timestamps used as reference points, followed by detailing the rooting process of the trees. The validation involves employing timestamps for time divergence analysis and embedding drift analysis to assess each tree. Our primary aim is to identify the tree that best corresponds with the timestamps data.

4.6.1. Timestamps

Our timestamps are sourced from VirusTotal, which provides key metadata fields including the creation date and the first submission date of malware samples. To assess their reliability, we analyzed these fields to determine the most dependable indicator of when malware was first detected. Our findings highlighted notable discrepancies between the creation and first submission dates across different malware families, underscoring the need for careful selection of the most reliable timestamp. These variations are documented in Figure 4.4, showcasing data up to 2023.

Figure 4.4 illustrates a noticeable correlation between the creation dates and first submission dates for most malware samples from 2015 to 2023. Intriguingly, some samples show creation dates as early as 1991 and even 1970, with significant instances from these early dates linked to the Fareit malware family, known for stealing system details and application credentials. Although these samples bear early timestamps, Fareit was not officially documented until 2012 [181] . Additionally, as illustrated in Figure H.1, certain samples were marked with future timestamps. Later investigation revealed that creation dates can be altered by VirusTotal users. Given these inconsistencies, we have opted to use the first submission dates as more reliable metric for our analysis. Although not perfect, these dates provide a crucial reference point for validating the phylogenetic trees.

4.6.2. Rooting Trees

Before validating our phylogenetic trees, it's imperative to ensure proper rooting. Rooting a phylogenetic tree involves identifying a root or reference point on the tree that represents the most ancestral lineage, which is essential for determining the direction of evolutionary change. This process is crucial for accurately interpreting the tree's structure as it defines the evolutionary relationships from the most ancient to the most recent species.

The UPGMA method automatically produces a rooted tree, but we also apply additional rooting methods to verify if they align better with the VirusTotal timestamps. Conversely, the NJ method yields an unrooted tree, requiring explicit rooting steps. We utilize both outgroup rooting and midpoint rooting

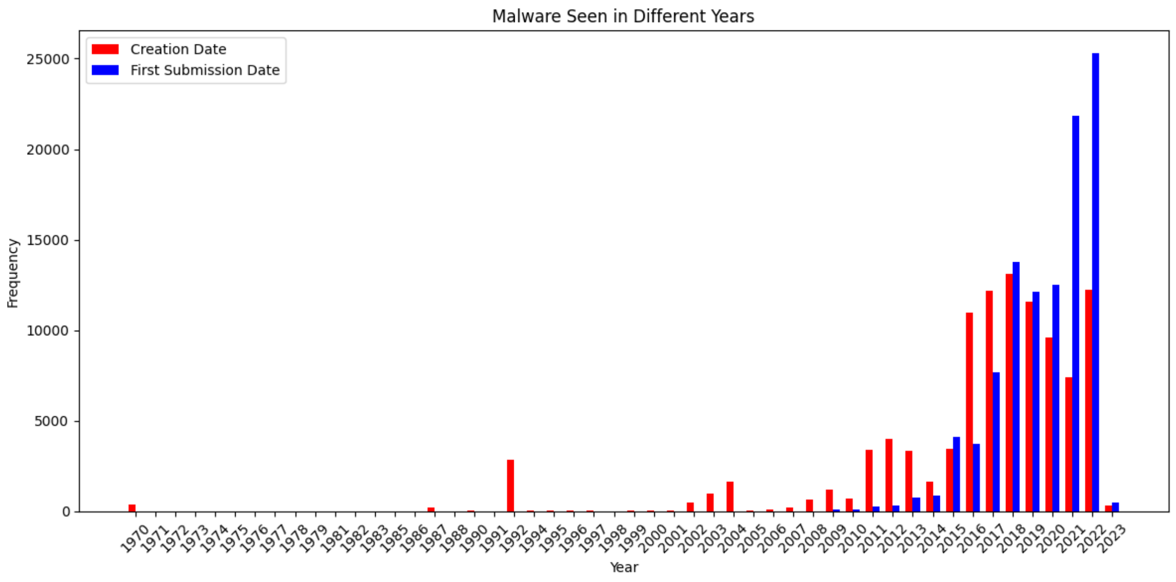


Figure 4.4: Malware Timestamps showing mismatch between Creation Date and First Submission Date

techniques for both types of trees to accurately interpret the evolutionary relationships they represent.

Outgroup Rooting

Outgroup rooting is a method in phylogenetic analysis that identifies a root on a tree by selecting one or more ’outgroup’ species. These outgroups are closely related to but outside the main group of species under study (the ’ingroup’). By comparing the ingroup to these outgroups, this method infers the most basal position on the tree, effectively placing the root and establishing the evolutionary direction from ancestral to derived lineages.

Outgroup rooting involves several key steps. First, we map our samples to timestamps obtained from VirusTotal, which reflect the discovery or submission dates of the samples. These dates serve as proxies for the evolutionary emergence of the samples, with the assumption that older samples may represent early diverging lineages.

Following this mapping process, we sort the samples in ascending order based on their timestamps to identify potential outgroups. We then tally the occurrences of each taxon’s family throughout this sorted list. Families with a minimum representation of 10 samples are eligible for outgroup selection; satisfyingly, this criterion is met by 98.98% of families in our dataset, ensuring that chosen outgroups are well-represented. Then to root the tree, we take the common ancestor of all the outgroup samples and then set that as the root. This is done by `ete3` toolkit, a Python framework designed for the analysis and visualization of trees.

Midpoint Rooting

Midpoint rooting is a method in phylogenetic analysis used to establish the root of a tree without requiring external outgroups. This technique identifies the root by placing it at the midpoint of the longest path between any two taxa on the tree, providing a balanced root location that does not lean on the phylogenetic placement of outgroups. Midpoint rooting is especially beneficial when suitable outgroups are ambiguous or when aiming for a phylogenetically neutral root position.

The process of midpoint rooting involves calculating the midpoint, which is the exact halfway point along the longest span between two tree leaves. This calculation assumes the tree is ultrametric—where all leaves are equidistant from the root—but can also be applied as a heuristic in non-ultrametric trees to find a plausible root position. After determining the midpoint, the tree is reoriented by rooting at this midpoint. This reorientation aims to minimize assumptions about the direction of evolutionary change and provides a more balanced view of the relationships within the tree. We use the same `ete3` tool as before to implement this.

4.6.3. Understanding Trees

To validate the trees using using time divergence analysis and embedding drift analysis we first define the semantics of these trees in order to facilitate effective analysis and reasoning. This subsection introduces some notation and semantics to reason about trees.

Notation

A_x Represents variant x within Family A, where A is the family name, and x indicates a specific variant within that family.

$MRCA$ Refers to the most recent ancestor from which two or more lineages or taxa have diverged.

Li Denotes the evolutionary distance from leaf i to an ancestor, quantifying the divergence time between a specific leaf (representing a sample in our dataset) and its ancestor.

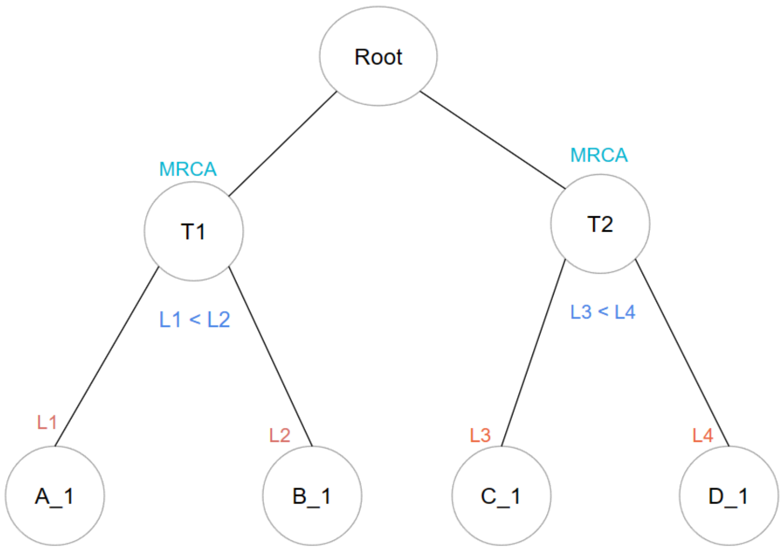


Figure 4.5: Simple Phylogenetic Tree

In phylogenetic trees, ancestors are conceptualized as theoretical nodes, which serve as inferred common points of divergence among lineages. These ancestral nodes are not directly observable but are inferred from genetic, morphological, or other data types that indicate shared lineage traits among descendants. Due to the impossibility of directly sampling these ancestral states in the present, they are modeled based on probable scenarios of evolutionary history that account for the observed diversity among current taxa.

Conversely, leaves on the tree are modeled as concrete samples from the dataset. These leaves represent actual data points collected from existing organisms, fossils, or genetic sequences and serve as the endpoints of the tree's branches; but the conext of our malware samples, the leaves are the samples . Each leaf directly connects the theoretical model of the tree to empirical evidence, tracing back to a specific sample in the dataset and thereby establishing a tangible link between the tree's structure and the real-world entities it represents. For instance, in our sample phylogenetic tree shown in Figure 4.5, leaves such as A_1 from Family A, B_1 from Family B, C_1 from Family C, and D_1 from Family D represent actual data points collected from specific lineages. The MRCA of A_1 and B_1 is denoted as $T1$, marking the theoretical node from which both lineages diverge. Similarly, the MRCA of C_1 and D_1 is denoted as $T2$.

4.6.4. Time Divergence Analysis

As mentioned in the background section, while there are numerous methods for validating phylogenetic trees, most of them are designed for sequence data. Methods that can use distance data, such as jackknife cross-validation, require significant computational resources. When it comes to temporal data, the literature predominantly focuses on estimating divergence times but lacks comprehensive methods for validating trees using this type of data.

In our method, we adapt the concept used for estimating divergence times to focus on relative early divergences instead. This adjustment enables us to evaluate the tree's structural validity by examining the sequence and relative timing of divergences rather than their exact chronological order. We will now discuss our approach in detail.

Our Approach

Our phylogenetic analysis primarily targets early divergences within the tree, particularly those near the leaves rather than the deeper, more complex branches near the root. This strategy aligns with the established understanding in phylogenetic research that closer to the root, relationships become less clear due to extensive genetic changes over time, a concept extensively discussed by Felsenstein et al. [182].

Unlike typical studies that use a molecular clock to estimate temporal divergences, our approach employs a first-order assumption by focusing on the most recent common ancestors (MRCAs) of leaf nodes. We interpret the branch lengths leading to these MRCAs as indicators of early divergence, assuming that shorter branch lengths imply more recent divergence events. This method is grounded in the fundamental principles of phylogenetic analysis, where tree topology and branch lengths provide insights into lineage splits without quantifying exact temporal distances, as noted by Page & Holmes [183].

For practical illustration, consider the simple phylogenetic tree shown in Figure 4.5. Observations such as $L1$ being shorter than $L2$ lead us to infer that A_1 diverged before B_1, and similarly, C_1 before D_1. However, relationships between A_1 and C_1 or D_1 remain undetermined due to different MRCAs. We align these inferences with timestamps from VirusTotal for the corresponding leaves and assess the accuracy of these inferred sequences. To validate the alignment, we utilize a straightforward metric: the proportion of correct comparisons over the total comparisons within the tree. For example, if timestamps indicate A_1 appeared before B_1 and C_1 after D_1, but our analysis shows C_1 diverged first, the result is one correct and one incorrect comparison, yielding 50%.

Implementation

In our approach to validate the chronological order of leaves in a phylogenetic tree, we first initialize counters for correct pairings and total comparisons. For each leaf, we identify its direct ancestor and gather all descendant leaves from this ancestor(mrca). Next, we compute the evolutionary distances between each pair of these descendant leaves.

As we process each pair, we compare the evolutionary distances to the chronological order indicated by timestamps associated with each leaf. If the relative ordering of distances corresponds to the chronological order of timestamps, the pairing is considered correct. We continue this comparison across all possible leaf pairs, updating our counts of total comparisons and correct pairings. Finally, we report the correct pairings by the total number of comparisons. This is detailed in Algorithm 1 of Appendix J.

4.6.5. Embedding Drift Analysis

In our study, we propose another approach to validating phylogenetic trees by examining the concept of embedding drift, specifically focusing on the comparison of mutation rates between the UPGMA and Neighbor-Joining (NJ) methods. UPGMA assumes a constant rate of molecular evolution, whereas NJ does not, making this comparation useful.

Starting with each family, we systematically calculate the Euclidean distances between embeddings of successive years. Specifically, for each variant in a family, we compare it with other variants in subsequent years, recording the minimum and maximum distances observed. For example, if we consider a hypothetical family 'A' with variants 0 through 6, as depicted in Figure 4.6, we would measure distances starting with variant 0 compared to variants 1 to 6. An identical embedding between variant 0 and any subsequent variant, such as variant 5, would result in a minimum distance of zero, indicating no drift. Conversely, the maximum distance would suggest significant divergence; for instance, the maximum distance could occur between variant 0 and variant 4 as their embeddings are most dissimilar.

This process is repeated for each variant, where variant 1 would then be compared only to variants 2 through 6, and so forth, ensuring each comparison captures the drift over time from the reference variant. This step-by-step analysis helps map out the trajectory of genetic drift for each variant, illustrating how each diverges from its predecessors and contemporaries over time.

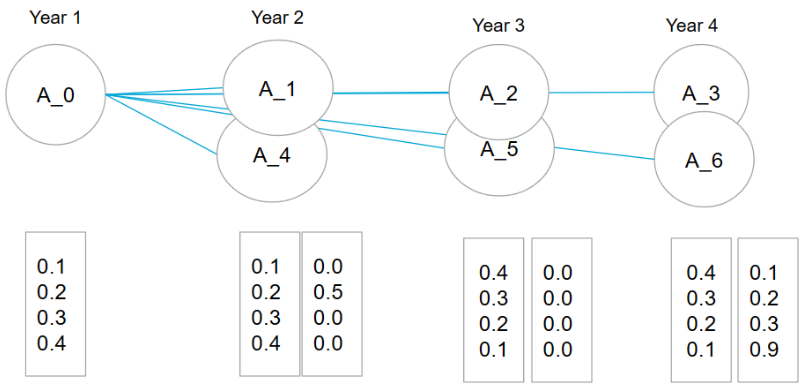


Figure 4.6: Embedding Drift Analysis: Comparing the euclidean distance between malware from different years

By aggregating these metrics across all families, we aim to determine whether mutation rates are uniform across the dataset. Uniform mutation rates would imply that both minimum and maximum distances are consistent across families, indicating a steady rate of change. Significant variations in these distances would highlight non-uniform mutation rates, challenging the assumptions of constant molecular evolution and providing crucial insights into the phylogenetic relationships within the data. This is detailed in Algorithm 2 of Appendix J. The findings from this validation approach are presented in Experiment 5 of the results chapter.

4.7. Visualizing Large-Scale Phylogenetic Trees

Identifying relationships within a complex phylogenetic tree, especially one encompassing 103,883 leaves, is challenging. Simplifying this tree through strategic assumptions is essential for clearer insights into the inter- and intra-family dynamics among malware samples. In this chapter, we will explain how with certain simplifications we can make useful observations from our complex tree.

4.7.1. Inter and Intra Family Analysis

The previous analysis on time divergence underscored the importance of identifying the most recent common ancestor (MRCA) to comprehend the evolutionary relationships among leaves within a family. This concept extends to discerning the connections between different families by aiming to identify the MRCA shared between them. We utilize the median distance from the leaves of each family to this MRCA as a metric to infer their evolutionary relationship. The median distance is chosen for its robustness to outliers, providing a reliable measure of typical evolutionary divergence between families.

To infer relationships between two families based on the median distance to the MRCA, we employ a second-order assumption: if the median distance from the MRCA to Family A is less than to Family B, it suggests Family A diverged earlier than Family B. Conversely, a smaller distance to Family B indicates that Family B diverged earlier. For a practical example, consider the phylogenetic tree shown in Figure 4.7, featuring two distinct families, Family A and Family C, each with three samples. The MRCA for both families' leaves is identified as the tree's root. The median distances from the leaves to the root are calculated as follows:

$$\text{Median Distance}_{\text{Family A}} = L2 + LP$$
$$\text{Median Distance}_{\text{Family C}} = L5 + LT$$

These distances imply that if $L2 + LP < L5 + LT$, then Family A diverged before Family C. This inference is based on our second order assumption, derived from comparing median distances to the MRCA. This approach offers a structured way to examine the sequence of divergence events to identify inter-familial relationships using the phylogenetic tree.

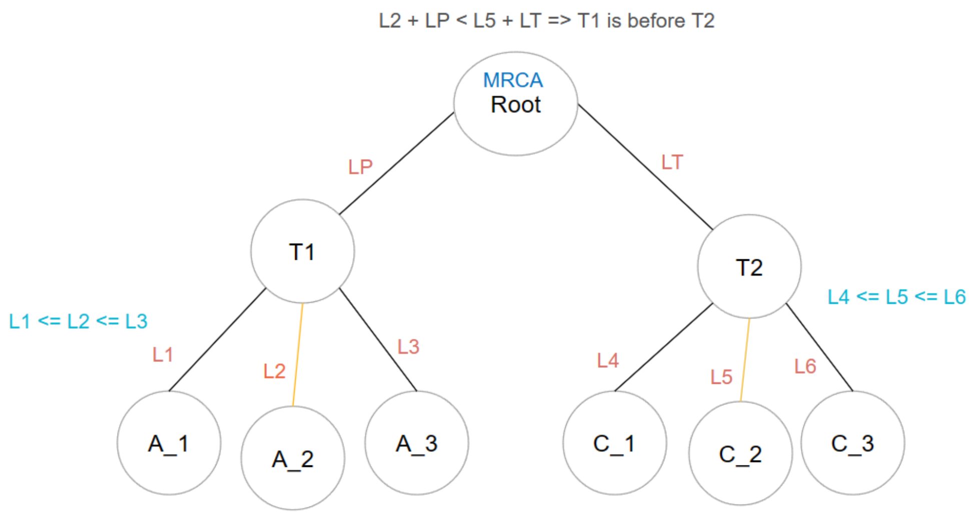


Figure 4.7: Median Distance to MRCA Analysis

Implementation

We start by loading the phylogenetic tree using the `ete3` library. After loading the tree, we iterate over the leaves and group them based on their respective family names.

Once organized by family, we then identify the most recent common ancestor for each pair of families using `ete3`'s capabilities .After identifying the MRCA, we calculate the median distance from this ancestral node to each family's leaves. These calculations help assess evolutionary depth and relatedness among the families involved.

To visually represent these relationships, we use the `networkx` library to construct directed graphs where nodes represent families and edges denote evolutionary distances to the MRCA. The directionality of edges is determined by relative distances; shorter distance results in an edge pointing from a family with a shorter distance towards one with a longer distance. Finally, we simplify the graph to only focus on significant evolutionary paths. To do we iterate through each node and select the node with with the shortest edge out of all the outgoing edges. This is detailed in algorithm 3 of appendix J.

Median Thesholding

The process of identifying the Most Recent Common Ancestor (MRCA) in phylogenetic trees is important for understanding evolutionary relationships as highlighted in the aforementioned method. However, this identification can be significantly affected by the inclusion of outliers in the dataset. For instance, consider a scenario where we have three families, A, C, and D, within a phylogenetic tree. Family C includes a potential outlier, C_4. If C_4 is included in the analysis, the MRCA for families A and

C is positioned at the root of the tree. If C_4 is excluded, the MRCA shifts to a lower node, T_P, potentially altering the evolutionary interpretations derived from the tree. This example underscores how outliers can influence the tree's topology and the resultant distances to the MRCA, as illustrated in Figure 4.8.

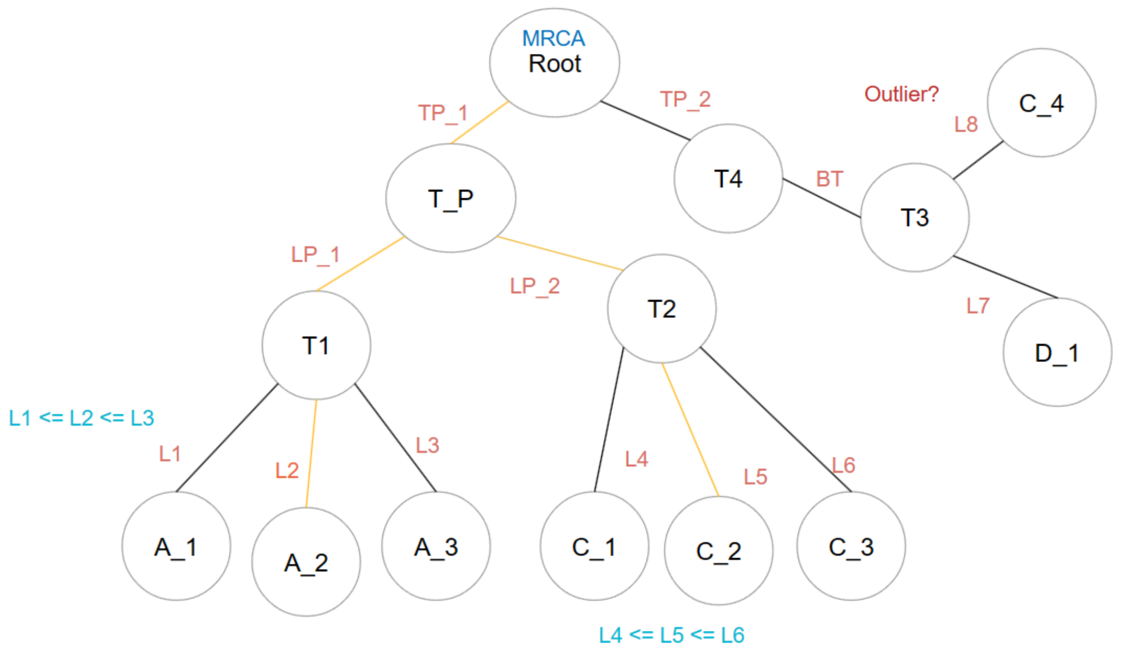


Figure 4.8: Median With Thresholding

From a theoretical standpoint, determining whether a sample like C_4 qualifies as an outlier involves assessing how significantly it deviates from the cluster formed by the majority of leaves within its family. In the context of malware research, this is particularly relevant as samples deviating sharply from others may indicate artifacts in data collection. To evaluate this, leaf-to-leaf distances within the family are analyzed. These distances are calculated by summing the individual paths from each leaf to their nearest shared ancestor and back to one another.

For example, the distance between leaves A_1 and A_2 would encompass the total length of paths $L1$ and $L2$, routed through their nearest shared ancestor. A significant difference in the distances from leaves C_1, C_2, and C_3 to C_4, compared to the distances among C_1, C_2, and C_3 themselves, might suggest that C_4 is an outlier. In literature, typically we use an IQR based approach based on boxplots where a outlier is defined as if it exceeds 1.5 times the interquartile range (IQR)[184]. We adopt the same method and consider a leaf as an outlier if its distance to the other leaves exceeds 1.5 times the IQR of median lateral(leaf to leaf) distance within each family. We adopt our previous implementation where prior to identifying mrca for each family pair, we remove the outliers using median thresholding method. This is detailed in algorithm 4 of of appendix J.

Influence of Outliers on Tree Topology

It would be intriguing to investigate whether including outliers significantly affects the topology of phylogenetic trees, indicating that many mrcas are influences by these outliers.

To determine the impact of outliers on phylogenetic tree topology, we will create two versions of the trees: one with outliers and one without, using our median thresholding method in NJ. We will then analyze both trees to assess variations in depth, breadth by depth, average degree, minimum degree, and maximum degree as indicators of tree complexity and connectivity. These metrics will help us understand how outliers may influence tree topology. Additionally, we will compare the isomorphism between the trees or specific sections. If the trees are isomorphic, it suggests that including outliers does not fundamentally alter evolutionary relationships. However if isomorphic subtrees are found within differing trees, it could indicate that certain evolutionary paths remain unaffected by outliers despite broader structural changes. The outcome of this experiment is detailed under Experiment 7 of the results chapter.

Correlation of Interfamily Analysis with Public Cybersecurity Insights and Malware Attributes

We have outlined the technical methodologies used in our study, but it remains essential to validate these approaches against manual reverse engineering and public sources. To this end, we constructed a phylogenetic tree using the NJ method with median thresholding and selected six malware families for analysis: Mirai, SmokeLoader, BotenaGo, DiscordTokenStealer, and SundownEk. We then compare the inter-family relationships derived from the tree with publicly available insights and examined the psuedo-static and dynamic features of each malware to corroborate our findings. The outcome of this experiment is presented in Experiment 8 of the results chapter.

Similarities between the findings of UPGMA and NJ

We also assess whether relationships validated using public sources with a tree constructed via the neighbor-joining (NJ) method are also observable in a tree built using the UPGMA algorithm. This serves as a further verification step to determine which method—NJ or UPGMA—produces more accurate phylogenetic trees. We achieve this by analyzing the relationships derived from case studies involving the malware families Mirai, SmokeLoader, BotenaGo, DiscordTokenStealer, and SundownEk, first identified using the NJ tree and then verifying if the same relationships are evident in the UPGMA tree. The detailed findings of this comparison are presented in Experiment 9 of the results chapter.

Alignment of Malware Clustering and Intra-family lateral leaf to leaf(lateral) Distances

In intrafamily analysis, we assess the lateral (leaf-to-leaf) distances within a family, employing methods similar to those used in inter-family analysis for outlier detection, defined here as distances exceeding 1.5 times the interquartile range (IQR). We visualize these distances with histograms and boxplots. As an experiment, we examine the correlation between these distributions and clustering patterns from dimensionality reductions to two dimensions via t-SNE, UMAP, and PCA, implemented using scikit-learn[185]. To make conclusions about the correlation we only use t-SNE, but if there are discrepancies we use UMAP and PCA to aid the analysis for additional insights.

To determine the optimal hyperparameters for these dimensionality reduction techniques, we performed a grid search using KNN with k = 1 in a supervised manner, utilizing the labels as the ground truth. For grid search parameters refer to Appenidx I and the final hyperparameters used are detailed in Table 4.7. The outcome of this experiment is discussed under Experiment 6 of the results chapter.

Table 4.7: Comparison of t-SNE, UMAP, and PCA Hyperparameters

Parameter	t-SNE Value	UMAP Value	PCA Value
Number of Components	2	2	2
Perplexity	100.0	–	–
Number of Iterations	3000	–	–
Random State	42	42	42
Learning Rate	100	–	–
Number of Jobs	-1	–	–
Number of Neighbors	–	15	–
Minimum Distance	–	0.5	–
Metric	–	Euclidean	–
Power Iteration Normalization	–	–	auto
Number of Oversamples	–	–	10
SVD Solver	–	–	auto

5 Results

First, we will review the results of experiments that validate pseudo-static, dynamic, image, and combined embeddings. Next, we will examine how effective these embeddings are for tasks for which they were not originally trained. We will then assess how well the trees constructed using temporal timestamps from Virustotal. Following this, we'll delve into identifying interesting patterns in intra-family analysis. Finally, we will discuss whether the relationships derived from our analysis align with public sources, and explore what common pseudo-static and behavioral features underpin these relationships.

5.1. Embedding Validation

In the first section, we will analyze the validation results of the embeddings. Our focus will be on determining which type—pseudo-static, behavioral, or image—performs the best and whether combining these embeddings results in better embeddings.

5.1.1. Experiment 1: Which analysis type yields the best embeddings and does combining them improve performance?

Table 5.1: Accuracy Metrics for Malware Family Classification

Embedding Type	Family Classification Accuracy (avg ± std)	Comparison
Image	**94.91 ± 0.16%**	Best Performance
Pseudo Static (Numerical)	86.39 ± 0.28%	Better than Random Guessing
Pseudo Static (Complex)	87.51 ± 0.48%	Better than Random Guessing
Dynamic	87.23 ± 0.19%	Better than Random Guessing
Combined	93.79 ± 0.19%	Second Best Performance
	Baseline Model	
Random Guessing	19.36 ± 0.17%	Baseline

The bold result indicates the best embedding compared to all other embedding types, with a statistically significant difference as determined by the Mann-Whitney U test ($\alpha < 0.05$).

Experiment 1 aimed to assess if embeddings from pseudo-static, image, and dynamic analyses yield significantly better results in terms of malware classification accuracy compared to a baseline dummy model. Table 5.1 shows the average accuracy of different embeddings trained with a Logistic Regression Model, as well as their comparison to the baseline dummy model.

The image embeddings were developed using a ResNet 50 architecture, initially initialized with `ImageNet`, and trained on 3 publicly available malware datasets. They were then further finetuned on executables derived from memory dumps tailored specifically to our dataset. The Pseudo Static embeddings utilize static analysis features applied to memory dumps, divided into two types: `Numerical`, which consists solely of numerical arrays, and `Complex`, which includes numerical arrays, lists, and dictionaries that contain both numbers and strings. On the other hand,dynamic embeddings were generated from behavioral analysis features. All embeddings were trained using a logistic regression. For benchmarking, random guessing was employed, which predicts based on the most frequent class, ignoring the input features. We anticipated that our embeddings would perform adequately when used with a linear classifier compared to random guessing.As anticipated, Table 5.1 confirms that all embeddings, trained with logistic regression, significantly outperform random guessing. This underscores their ability to identify unique features essential for differentiating between malware families.

Looking back at Table 5.1 we see that image embeddings perform best, Contrary to our initial assumption that dynamic embeddings would perform better. The improved performance of image embeddings may be attributed to two key factors. Firstly, these embeddings may benefit from pre-training on three public image malware datasets: Malimg, MaleVis, and MalwareNet. This extensive pre-training may have equiped them with capabilities for robust feature recognition. Secondly, we further refined the model by training it on executables obtained from memory dumps in our dataset. These memory dumps are likely unpacked as the malware is already loaded into the system's memory. This approach may have allowed the network to detect features specific to executable files more accurately, typically mitigating challenges posed by obfuscation techniques.

Lastly, to assess the effectiveness of combined embeddings, we analyzed how merged embeddings compare to individual ones. The combined embeddings were produced by a neural network that processed merged embeddings consisting of 6560 dimensions from pseudo-static, dynamic, and image embeddings. These were then reduced to a 1000-dimensional representation using a hidden layer of 1000 neurons. The network was trained using classification loss across malware families to achieve this condensed representation. Table 5.1 illustrates that while combining embeddings significantly enhances performance compared to pseudo-static and dynamic embeddings alone, they still do not surpass the performance achieved with image embeddings. Although the difference in performance between combined and image embeddings is small, the standard deviation clearly shows that image embeddings perform substantially better. Further analysis of accuracy differences relative to image embeddings, as shown in G.1, reinforces that image embeddings are much better than the combined embeddings. This result was rather unexpected because we expected combined embeddings to improve feature learning. We suspect that this limitation may be due to reducing the dimensions to 1000, which could have led to a loss of accuracy. We believe that maintaining a higher latent dimension during the reduction may lead to better combined embeddings that outperform image embeddings. Ideally, the dimensionality of the reduction should be considered a tunable hyperparameter.

5.2. Extending Embeddings to New Tasks

5.2.1. Experiment 2: Are embeddings, merged or individual, useful for downstream tasks?

Experiment 2 evaluates the effectiveness of embeddings for downstream tasks, focusing on two specific goals. Firstly, we examine the usefulness of combined embeddings in detecting temporal shifts over the years in experiment 2.1. Secondly, in Experiment 2.2, we examine the effectiveness of only image embeddings in differentiating between benign and malicious malware. Considering that image embeddings demonstrated the best performance, and that generating images from memory dumps is not only simpler but also potentially more cost-effective compared to the extensive processing required by large language models for behavioral and pseudo-static analyses, we opted to assess their efficacy for malware detection.

Experiment 2.1: Are embeddings capable of discerning temporal shifts across years?

In this experiment, we are investigating whether embeddings can be used to determine the age of malware. We have categorized the malware by year and trained a logistic model using the embeddings as input and the years as output. Our anticipation is that the logistic model will predict the years with significantly better accuracy compared to a dummy model, which predicts based on the most frequent class. This is evident in Table 5.2, where the logistic model significantly outperforms the dummy model, highlighting its effectiveness in predicting temporal shifts.

Table 5.2: Summary of Model Performances for Temporal Shifts in Years

Model	Accuracy (avg ± std)	Mean Absolute Error (MAE) in years (avg ± std)
Dummy Model	0.0740 ± 0.0000	5.0674 ± 0.0004
Logistic Model	0.6679 ± 0.0035	1.0023 ± 0.0167

Logistic Model indicate a statistically significant difference compared to dummy model, as determined by the Mann-Whitney U test ($\alpha < 0.05$).

Experiment 2.2: Are image embedding capable of classifying benign and malicious malware? In this experiment, we assess the effectiveness of using image embeddings to classify malware as benign or malicious by training a logistic model. The benign images were sourced from Androzoo benign dataset, with a similar encoding to ours. Given the strong performance of image embeddings in previous assessments, we expect them to perform well in differentiating between benign and malicious malware. As demonstrated in Table 5.3, this hypothesis holds true. The logistic model achieves notably higher accuracy, AUC score and F1 score compared to the dummy model.

Table 5.3: Summary of Model Performance for benign/malicious classification using Image embeddings

Model	Accuracy (avg $\pm std$)	AUC Score (avg $\pm std$)	F1 Score (avg $\pm std$)
Dummy Model	0.6523 ± 0.0001	0.5000 ± 0.0001	0.4900 ± 0.0001
Logistic Model	0.9999 ± 0.0001	1.0000 ± 0.0001	0.9990 ± 0.0001

Logistic Model indicate a statistically significant difference compared to dummy model, as determined by the Mann-Whitney U test ($\alpha < 0.05$).

5.2.2. Experiment 3: How does our approach on classifying malware based on images compare to previous work?

In our analysis, we examined the comparative efficacy of our image classification approach against other established architectures. The results, as detailed in Table 5.4, indicate that the CNN model using grayscale images had the weakest performance as expected since grayscale mapping, as used by Natarajan et al.[78]as grayscale mapping captures less information than RGB mapping.

Moreover, We anticipated that our more complex architecture, which includes significantly more layers than the other two models, would capture more robust features. Table 5.4 confirms this hypothesis, showing that our model significantly outperforms the CNNs in processing both grayscale and RGB images.

Table 5.4: Comparison of Architectures

	Accuracy (avg $\pm$ std)
Ours	**94%** $\pm$ **0.160**
CNN (gray scale**)**	81% $\pm$ 0.053
CNN (rgb)	89% $\pm$ 0.447

Our model indicates a statistically significant difference compared to gray scale and RGB, as determined by the Mann-Whitney U test ($\alpha < 0.05$).

5.3. Tree Validation

In this section, we will focus on validating the phylogenetic trees. First, we will compare the accuracy of trees constructed using two methods—UPGMA and NJ—when validated against VirusTotal timestamps. Next, we will explore how embedding drift analysis can be utilized to validate the phylogenetic trees generated from both UPGMA and NJ methods.

5.3.1. Experiment 4: Which phylogenetic tree construction method using distances produces the most accurate representation of malware evolution using VirusTotal timestamps?

In this experiment, we investigate which phylogenetic tree construction method, UPGMA or Neighbor-Joining (NJ), provides the most accurate representation of malware family evolution. This comparison is validated using timestamps from VirusTotal, which serve as a benchmark for assessing the temporal accuracy of the tree constructions.

Table 5.5: Summary of Validation Results using Year Timestamps

	Nominal (Not rooted)	Outgroup	Midpoint
NJ (Year)	0.811	**0.871**	0.631
UPGMA (Year, Rooted)	0.860	0.601	0.562

The metric employed is the ratio of correct comparisons to total comparisons, where a higher value indicates better representation.

Table 5.6: Summary of Validation Results using Month Timestamps

	Nominal (Not rooted)	Outgroup	Midpoint
NJ (Month)	0.793	0.853	0.597
UPGMA (Month, Rooted)	0.569	0.553	0.507

The metric employed is the ratio of correct comparisons to total comparisons, where a higher value indicates better representation.

The metric presented in the tables 5.6 and 5.5 represents the ratio of correct comparisons to total comparisons, indicating a higher value as a more accurate representation. We hypothesize that this ratio will increase from month to year for all tree rooting methods, as rooting provides a necessary perspective for interpreting the evolutionary relationships among the entities represented in the tree. This trend is apparent in both tables 5.5 and 5.6.We also hypothesize that UPGMA, which inherently generates a rooted tree, will not show improved accuracy with additional rooting. This hypothesis is supported by the data in tables 5.5 and 5.6, where UPGMA achieves its best results with unrooted rooting for both month and year granularities.

Conversely, we hypothesize that the NJ method, which does not naturally produce a rooted tree, will show more accurate results when rooting methods are employed. This is confirmed as outgroup rooting presents higher ratios than nominal for both month and year granularities in tables 5.5 and 5.6. We also hypothesize that midpoint rooting will not yield more accurate results for either tree type due to its potential to oversimplify evolutionary relationships. Midpoint rooting assumes that the longest distance between any two leaves in the tree is across the 'midpoint' of the tree. This assumption can lead to arbitrary root placement, which might not accurately reflect the evolutionary history of the taxa involved. This is evident for both month and years in tables 5.5 and 5.6 that irrespective of the tree building algorithm midpoint rooting does not give better results.

Finally, we hypothesized that NJ would generally yield more accurate results than UPGMA because it doesn't assume a constant rate of evolution. This is supported by the data in tables 5.5 and 5.6, where outgroup-rooted NJ outperforms outgroup-rooted UPGMA across both monthly and yearly granularities. Although UPGMA shows better results in unrooted scenarios compared to unrooted NJ, it does not surpass the performance of outgroup-rooted NJ overall.

5.3.2. Experiment 5: How can embedding drift analysis be employed as an alternative method for validating phylogenetic trees?

In this experiment, we explore how embedding drift analysis can also be used in validating the accuracy of UPGMA and NJ constuction methods. Given that UPGMA assumes a constant rate of evolution, we test this assumption by analyzing the rates of change in embeddings for different malware families. Specifically, we categorize embeddings of malware variants into different years based on VirusTotal timestamps and then calculate the Euclidean distances between these embeddings sequentially over the years, recording the minimum and maximum rates.

The findings, illustrated in Figure5.1, display the maximum and minimum mutation rates for ten selected families. Notably, the families Bashlite and SynonGlobal exhibit significantly higher mutation rates than others, challenging the validity of the constant evolution assumption upheld by UPGMA.

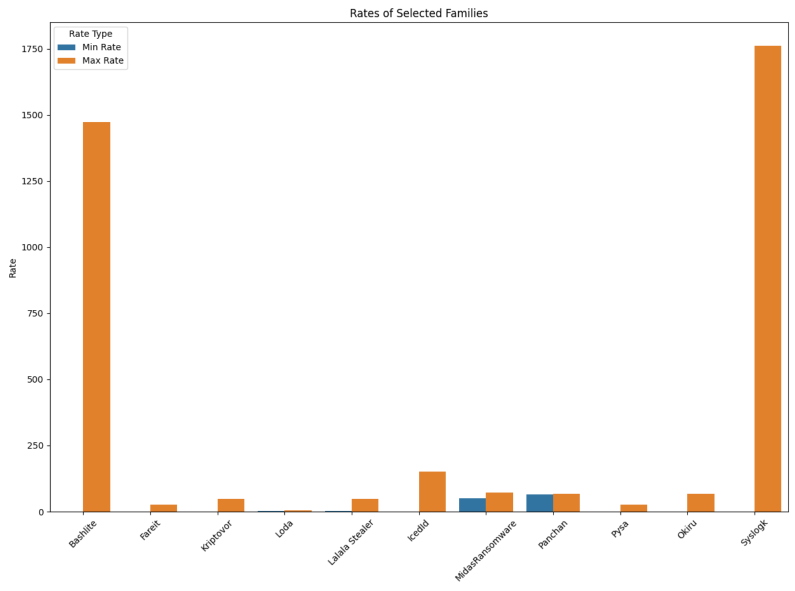


Figure 5.1: Mutation Rates(euclidean distance over year) Barchart: This shows varying rates between families.

These observations suggest that UPGMA's assumption is significantly violated as illustrated by Figure 5.1. This may be another reason why NJ performs much better than UPGMA.

5.4. Patterns and Topology

This section explores the different patterns and topologies that arise from analyses within and between families. First, we will explore if clusters visualized using t-SNE, UMAP, or PCA align with lateral distances in the phylogenetic tree. Then, we'll investigate how outliers may impact the topology of the tree by altering the position of the most recent common ancestor.

5.4.1. Experiment 6: How do clusters formed by visualizing malware embeddings with t-SNE, UMAP, and PCA align with lateral(leaf to leaf) distances in a phylogenetic tree built with NJ method?

This experiment explores the patterns exposed by mapping malware embeddings into two dimensions. We hypothesise that if samples from the same family cluster together in this 2D space, this clustering should correspond to specific patterns in the frequency distribution of lateral distances (leaf-to-leaf or sample-to-sample) within the phylogenetic tree. To test this hypothesis, we will examine these distances across 20 malware families using a tree made from Neighbour joining method, specifically selected because they each consist of over 1,000 samples. This analysis aims to determine whether the clusters evident in the t-SNE embeddings correspond to intra-family variations as indicated by the phylogenetic relationships. Additionally, to explore any discrepancies that emerge from the t-SNE results, we will employ UMAP and PCA for further examination.

Global Structure

In Figures 5.2, 5.3, and 5.4, the ability of various dimensionality reduction techniques to segregate malware families into discernible groups is clearly demonstrated. It can be seen that PCA in general performs very poorly and therefore struggles with complex, non-linear realtionships. On the other hand, both t-SNE and UMAP excel in clearly defining clusters for families like 7ev3n, Bashlite, NetWireRAT, BazarBackdoor, and KrBanker. However, other families such as Upatre and Revil exhibit overlapping embeddings in these plots, suggesting shared characteristics. This overlap may not be directly relevant to the question at hand but may indicate the presence of common traits among different malware families.

Outliers

When examining clusters, it's crucial to consider outliers as well. We will delve deeper into whether the outliers identified in the intra-family analysis of the phylogenetic tree correspond with those seen in the embedding projections. Figure K.4 provides a detailed overview of outlier counts for particular malware families. It's important to highlight that only families with outliers are represented in this diagram.

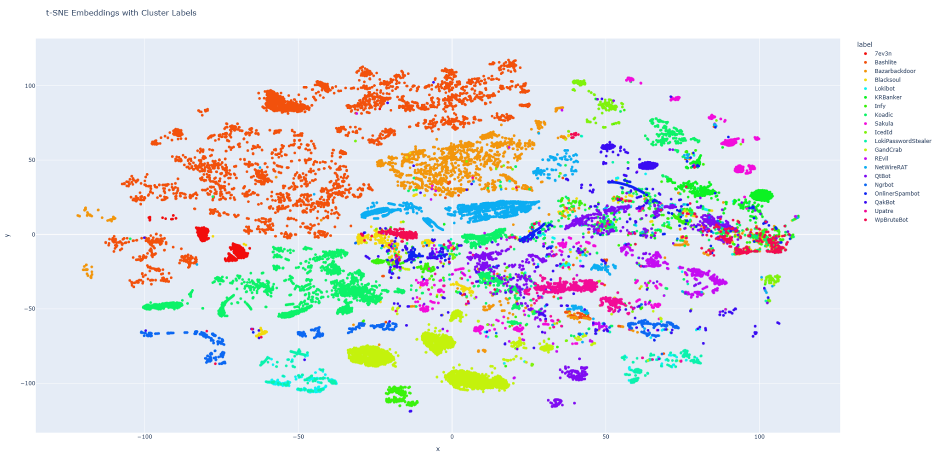


Figure 5.2: Embeddings Projected to 2D using t-SNE shows that embeddings from same family cluster together

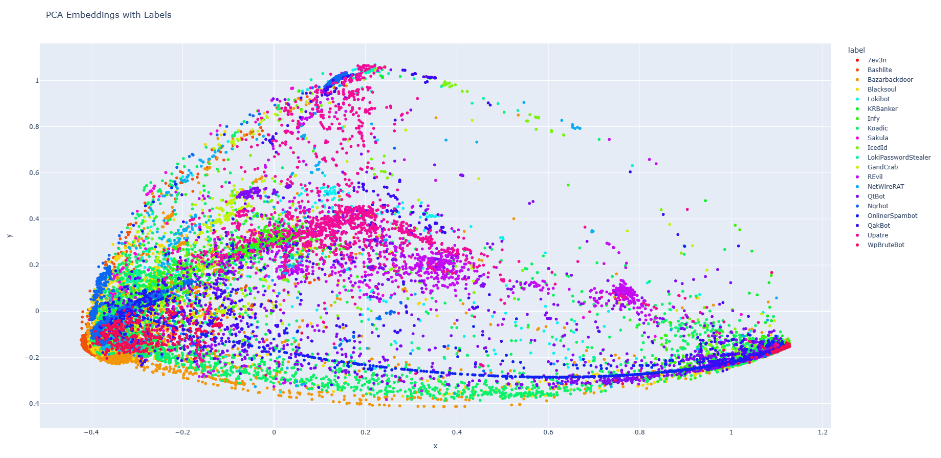


Figure 5.3: Embeddings Projected to 2D using PCA shows that PCA is unable to capture non-linear relationships

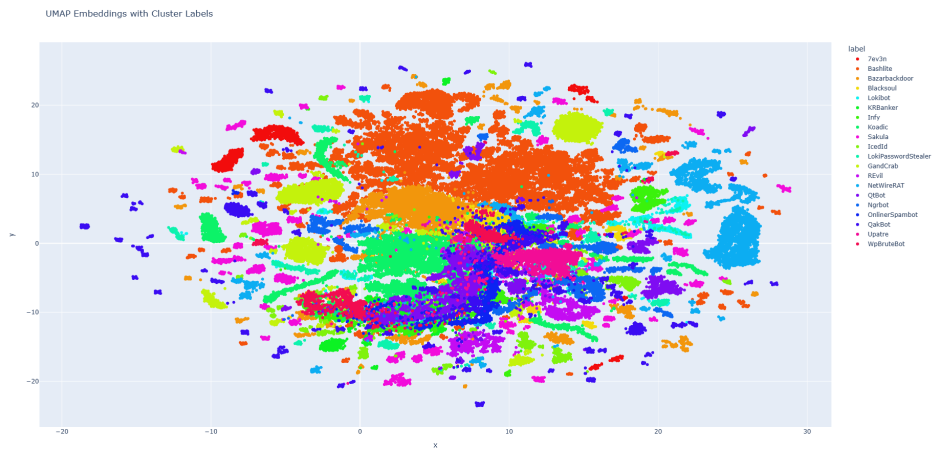


Figure 5.4: Embeddings Projected to 2D using UMAP shows that embeddings from same family cluster together

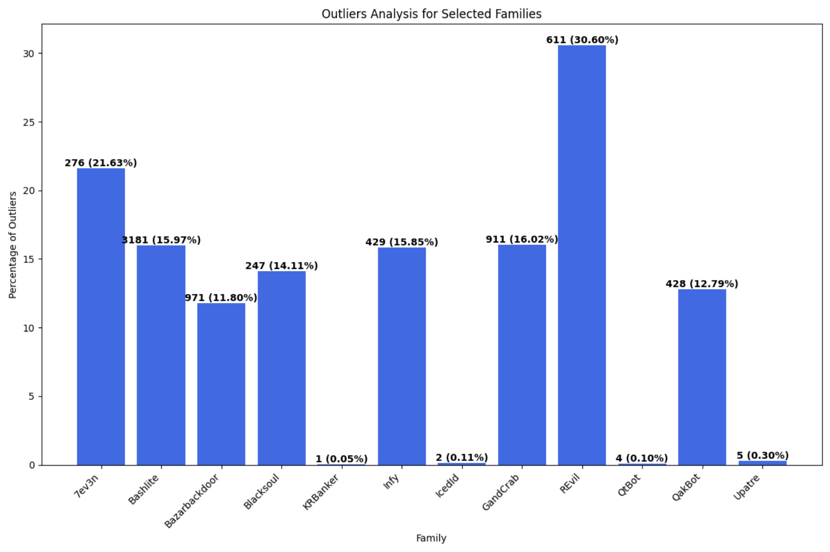


Figure 5.5: Outliers based on lateral distance of the Tree

Table 5.7: Clustering and Mode Analysis Across Malware Families

Family	Observed Clusters (t-SNE)	Modes in Lateral Distribution
7Ev3n	**2**	**1**
Bashlite	1	1
Bazarbackdoor	2	2
Blacksoul	1	1
GandCrab	3	3
IcedId	**2**	**3**
Infy	2	2
Koadic	3	3
KRBanker	2	2
Lokibot	2	2
LokiPasswordStealer	2	2
Sakula	4	4
NetWireRAT	3	3
Ngrbot	3	3
OnlineSpamBot	3	3
QakBot	2	2
QtBot	2	2
Revil	2	2
Upatre	1	1
WpBruteBot	**2**	**3**

There is significant correlation between Observed Clusters and Modes according to Spearman and Kendall's tests. The bold results are shown as discrepancies between observered clusters and modes in Lateral Distribution

From Table 5.7, it is evident that there is a significant correlation between the observed clusters in t-SNE projections and the modes in lateral distributions. This correlation generally supports our hypothesis, suggesting a consistency between clustering in reduced dimensions and actual genetic proximity as measured by lateral distances. However, it is crucial to acknowledge that the identification of observed clusters can be somewhat subjective, influenced by visual interpretation. Therefore, while the results are promising, they do not conclusively validate all aspects of our hypothesis. More rigorous statistical methods and additional analyses are required to further substantiate these findings with domain knowledge.

Table 5.7 highlights discrepancies between observed clustering via t-SNE and lateral distance distributions for the 7ev3n, WpBruteBot, and IcedId malware families. For 7ev3n, both the lateral histogram and UMAP indicate a single, closely related cluster, consistent with the Neighbour Joining analysis. In contrast, while the lateral distribution for WpBruteBot reveals three to four modes, t-SNE shows only two main clusters, and UMAP suggests two clusters along with additional small outlier groups, suggesting possible misclassification of outliers by the Neighbour Joining method. The IcedId family displays similar inconsistencies, with three to four modes observed in the lateral histogram, but only 2 to 3 clusters are apparent in t-SNE and UMAP, with outlier data points potentially misclassified as clusters by the Neighbour Joining algorithm. In the following subsection we will how we derive these insights of for WpBruteBot, for more example of analysis form Table5.7 refer to appendix K.

5.4.2. WpBruteBot

The WpBruteBot family, as depicted in Figure 5.7, demonstrates three to four modes in the lateral distance distribution. However, the t-SNE visualization in Figure 5.7 only reveals two prominent clusters. In contrast, the UMAP plot in Figure 5.8 suggests the presence of two main clusters, with additional densities at -6 and 20 that could potentially be interpreted as separate clusters. Nevertheless, we suspect these are merely outliers, especially since the PCA visualization in Figure 5.9 predominantly shows two dense regions. This discrepancy might be attributed to the neighbor joining method not classifying these points as outliers in its own feature space.

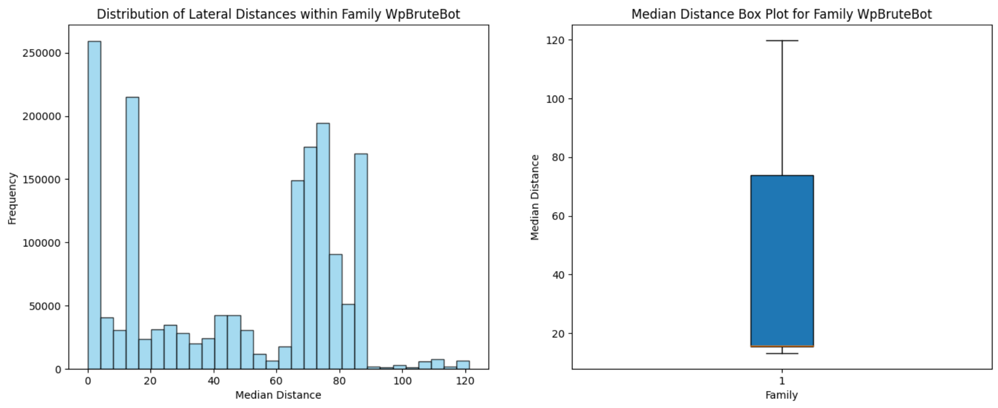


Figure 5.6: Distribution of Lateral Distances for WpBruteBot shows 3 to 4 main modes with not many outliers

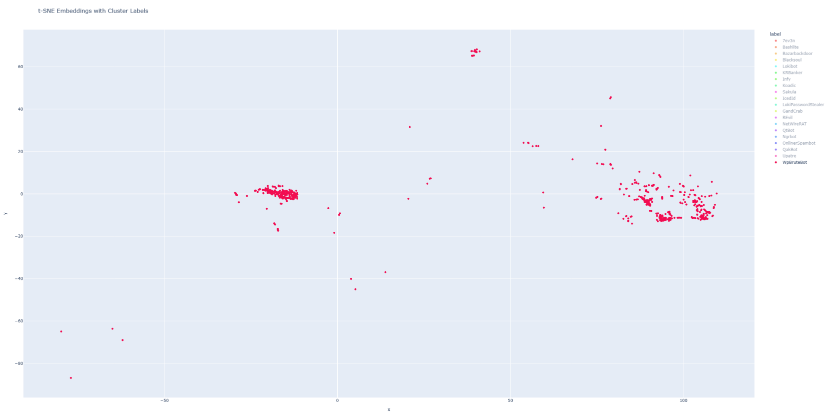


Figure 5.7: t-SNE Plot for WpBruteBot shows mainly 2 prominent clusters

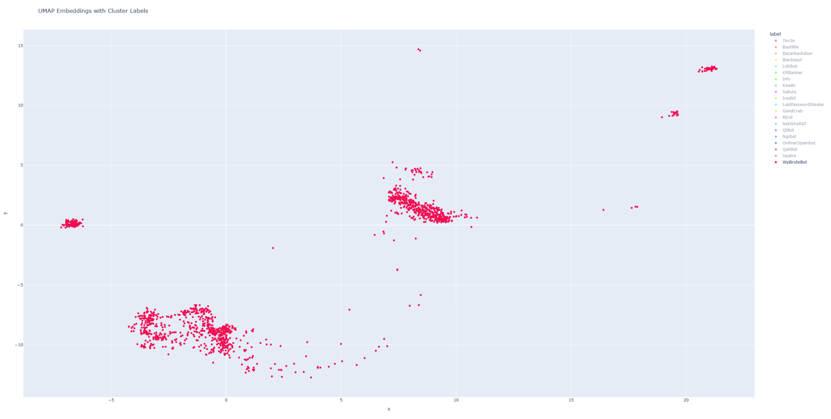


Figure 5.8: UMAP Plot for WpBruteBot shows 3 to 4 clusters

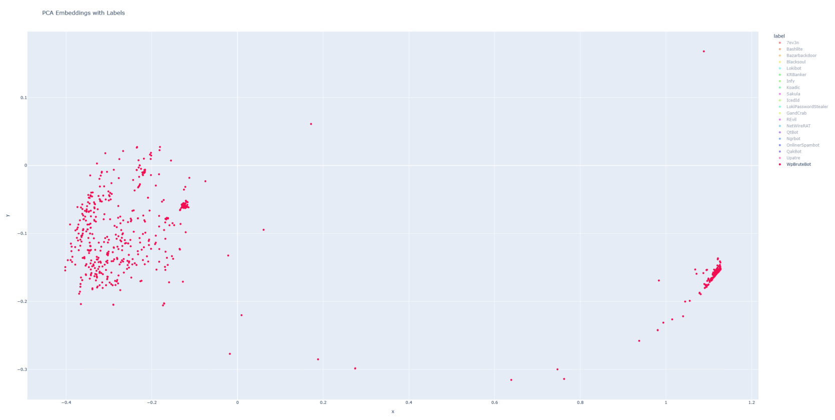


Figure 5.9: PCA Plot for WpBruteBot mainly shows 2 clusters

5.4.3. Experiment 7 : Do outliers alter the topology of a phylogenetic tree constructed using the Neighbour Joining method by changing the Most Recent Common Ancestor (MRCA)?

We hypothesize that the inclusion of outliers significantly affects the topology of phylogenetic trees, introducing increased depth and complexity that could alter evolutionary interpretations. Here, outliers are defined as leaves that are more than 1.5 times the interquartile range (IQR) of the median distance between all leaf-to-leaf(lateral) distances across families.This hypothesis is examined by comparing tree characteristics with and without outliers.

Table 5.8: Comparison of Tree Characteristics

Characteristic	**No Outliers**	**Outliers**
Depth	4	8
Breadth by Depth	{0: 1, 1: 1, 2: 1, 3: 1, 4: 1}	{0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1}
Average Degree	1.996	1.996
Minimum Degree	1	1
Maximum Degree	100	116
Isomorphic	No	
Isomorphic Subtrees Found	No	

As shown in Table 5.8, the depth of the tree significantly increases from 4 to 8 when outliers are included, confirming our hypothesis that outliers contribute to a deeper and more complex tree structure. The breadth by depth data further supports this, illustrating additional complexity through increased layers when outliers are considered.

The consistency of the average degree between the two scenarios suggests that the overall connectivity per node remains stable, despite the structural changes induced by outliers. However, the increase in maximum degree from 100 to 116 when outliers are included indicates that specific nodes, potentially representing outliers, integrate more connections, thereby influencing the overall tree topology. The absence of isomorphism and isomorphic subtrees in both scenarios highlights that the structural core of the trees is altered by the inclusion of outliers. This disruption in conventional topology is indicative of the unique evolutionary paths introduced by outliers.

In summary, the evidence provided in Table 5.8 robustly supports our hypothesis, demonstrating the significant impact of outliers on the topology of phylogenetic trees. This finding underscores the need for careful consideration of outliers in phylogenetic analyses to ensure accurate evolutionary interpretations.

5.5. Inter-Family Analysis

In this section, we'll examine whether the connections identified through inter-family analysis using NJ correspond with public cybersecurity insights. Additionally, we will investigate if the relationships identified through UPGMA align with those determined by NJ.

5.5.1. Experiment 8: Do the relationships identified through inter-family analysis using NJ method correlate with public cybersecurity insights and with the psuedo-static and dynamic features of the malware?

To address this question, we will undertake a speculative analysis of six malware case studies: Mirai, SmokeLoader, BotenaGo, DiscordTokenStealer, and SundownEk. We will focus in-depth on two primary studies—Mirai and SmokeLoader—with the remaining subjects detailed in Appendix L. This analysis involves reviewing various sources to determine potential connections between each malware family. We will explore both pseudo-static and behavioral characteristics, with specific examples detailed in Appendix M.

Our research, based on publicly available data, explores relationships between several malware families. We have confirmed connections from Mirai to Gafgyt, Moobot, Okiru, EnemyBot, and Bashlite, as well as connections from SmokeLoader to IcedID, Racoon Stealer, and Async Rat.However, interfamuly analysis suggests possible connections between Mirai and other malware families such as Turla, Conti, and Rdat, marked by high edge weights (over 25). The lack of supporting evidence from existing literature renders these links unlikely. This doesn't imply that the graph is inaccurate; instead, these high-edge-weight connections are considered with very low confidence in our inter-family analysis.

Additionally, we identified connections between BotenaGo and FritzFrog, DiscordTokenStealers and AkiraRansomware, and SundownEk and Kronos, which also agree with public sources as detailed in other studies found in Appendix L. Overall, our analysis suggests plausible, albeit speculative, relationships within our network, consistent with public sources.It's crucial to remember that these findings are speculative and do not establish definitive relationships.

Case Study: Mirai

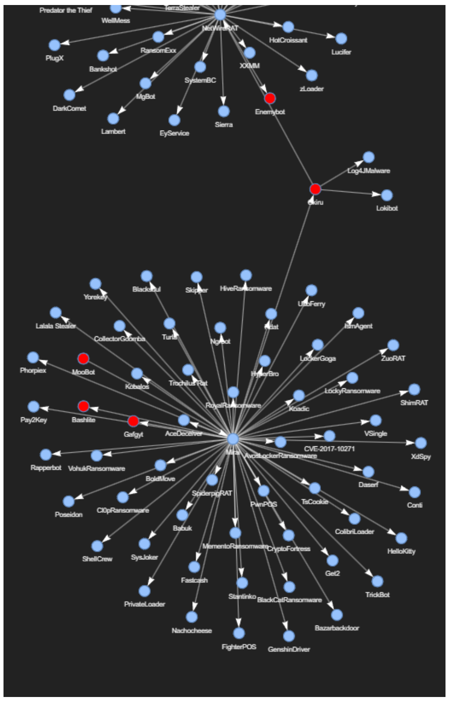


Figure 5.10: Inter-family analysis of Mirai shows that Mirai is the progenitor of many families.

Figure 5.10 shows an inter-family analysis of Mirai, indicating its role as a progenitor to several malware families such as Bashlite, Okiru, Moobot, EnemyBot, and Gafgyt. These related families, highlighted in red in Figure 5.10, will be the focus of our discussion in this section. According to Avast[186], Mirai is a type of malware that targets Internet of Things (IoT) devices, such as routers and cameras, which run on the Linux operating system. It exploits default usernames and passwords on these devices to launch Distributed Denial of Service (DDoS) attacks. The botnet became notorious following its involvement in some of the largest DDoS attacks, including the attack against DNS provider Dyn in October 2016 that temporarily rendered major websites like Twitter, Netflix, and Reddit inaccessible [187].

The pseudo-static analysis features of Mirai, reveals several operational functions. These include generic networking functions such as `socket`, `connect`, `bind`, `listen`, `accept`, `send`, `recv`, `recvfrom`, `sendto`, and `setsockopt`. These functions are essential for establishing and managing network communications, enabling Mirai to control a vast array of compromised devices across the network. Additionally, Mirai utilizes a variety of process and memory management functions including `fork`, `exec`, `malloc`, `calloc`, `realloc`, and `free` for memory and process management. However, Mirai also uses more uncommon control functions such as `signal`, `kill`, `exit`, `getpid`, `getppid`, and `prctl` . These functions allow Mirai to initiate or terminate processes as necessary.

When analyzing Mirai's dynamic behavior, it is evident that the malware extensively interacts with system files, a critical aspect of its ability to gather information and manipulate the environment. It accesses key configuration files such as `/etc/group` and `/etc/mtab`, providing essential details about system settings and current mount points that could be exploited to identify vulnerabilities or optimal points for maintaining persistence. Additionally, the deletion of files like `/tmp/cutie.x86_64` demonstrates Mirai's efforts to conceal its presence, potentially to evade forensic detection and eliminate conflicts with other processes.

In terms of command executions, Mirai actively uses shell commands to inspect system processes, exemplified by commands such as `ls -l /proc/1/status`. This behavior is typically aimed at monitoring and controlling running processes to prevent any interference from competing malware or security software, thereby ensuring sustained control over compromised devices. Moreover, Mirai utilizes various techniques cataloged in the MITRE ATT&CK framework, underscoring its sophistication and the potential breadth of its impact. Notable techniques include executing commands via a shell command-line interpreter for system manipulation, creating hidden files, links, or directories to conceal its presence, and employing the `uname` system call to gather information about the kernel version. These actions are indicative of Mirai's strategic approach to evasion and reconnaissance, allowing it to customize its operations based on the specific characteristics of the environment it infects.

Gafgyt

Numerous sources, including reports from Trend Micro [188], New Jersey Cybersecurity [189], and Threatpost [190], indicate that Gafgyt is a variant of Mirai. Similar to Mirai, Gafgyt targets IoT devices but employs distinct methods by exploiting specific vulnerabilities instead of merely relying on default credentials. Devices infected with Gafgyt are used to launch DDoS attacks, and over time, Gafgyt has evolved into more sophisticated variants with enhanced capabilities to exploit a broader range of devices and software vulnerabilities.

Gafgyt also demonstrates the ability to eliminate other malware and processes on compromised devices, asserting its dominance. In terms of thread and process management, Gafgyt utilizes functions such as *pthread_start_thread*, *pthread_kill_all_threads*, *pthread_reap_children*, and *pthread_onexit* to manage and terminate threads effectively. Additionally, it incorporates process lifecycle and control functions like restart, suspend, and *pthread_handle_sigrestart*, as well as queue management and synchronization functions such as enqueue and *remove_from_queue*, which are vital for coordinating tasks like DDoS attacks or spreading to additional devices.

The dynamic analysis features of Mirai and Gafgyt shows some similarities, particularly in their use of basic networking functions like *connect*, *recvfrom*, and *socket*. Mirai is observed extensively probing various system processes by accessing command line information from the */proc/[pid]/cmdline* directory, suggesting an aggressive strategy to monitor and manipulate running processes. In contrast, Gafgyt focuses more on network-related information, as seen from its activity in the */proc/net/route* directory, indicating an interest in manipulating the device's network behavior or scanning for other devices to infect within the network.

Both families involve terminating processes, as evidenced by targeted files like '/tmp/EB93A6/996E.elf' being eliminated by process termination commands in both cases. This suggests a common operational tactic likely aimed at stopping specific security processes or competing malware to ensure their dominance on the compromised system. While these commonalities do not conclusively prove that Gafgyt originated from Mirai, multiple sources supporting the claim strengthen the notion of a connection between the two, as identified in inter-family analysis. This linkage suggests a possible evolutionary relationship where Gafgyt may have derived certain tactics or operational strategies from Mirai.

Okiru

The cybersecurity group MalwareMustDie has identified a new variant of the Mirai malware, known as Okiru, which targets IoT devices equipped with Argonaut RISC Core (ARC) processors [191]. Similar to Mirai, Okiru scans for devices with Telnet access using default passwords, reflecting commonalities in their approach to accessing vulnerable systems. Okiru is notable for being the first piece of malware specifically designed to exploit ARC processors [192], which are found in a diverse range of system-on-chip applications including wearable health monitors, smart home appliances, energy management systems, and automotive and industrial controls. The shared Linux-based software development environment of these IoT devices makes them susceptible to such malware attacks, as highlighted by TechTarget [192].

In examining the pseudo-static features, Mirai and Okiru display many shared characteristics. Both malware families make extensive use of core networking functions, including *socket*, *connect*, *recv*, *send*, and *bind*. They also incorporate generic memory and thread management functions such as *fork*, *exec*, *malloc*, *free*, and *kill*. Additionally, they utilize system control functions like *prctl*, *setsockopt*, *getsockopt*, and *ioctl* to modify operating system behavior. Other Miscellaneous utilities, mirroring those used by Mirai, such as *time*, *getpid*, *sleep*, and *opendir*, further assist in managing operational timing and processes.

Okiru's dynamic analysis of imports and exports reveals specific activities. It generates network traffic on non-standard TCP and UDP ports, possibly as a strategy to evade detection by network security systems that typically do not scrutinize non-standard ports as closely. Additionally, it uses network protocols on non-standard ports, potentially masking its communication within legitimate-looking traffic to avoid standard port-based network filters. Okiru also accesses network configuration files such as */proc/net/tcp*, indicating efforts to monitor or manipulate crucial network communications for its operations and identifying potential targets within the network. While the observed similarities do not conclusively prove Okiru's origin from Mirai, multiple sources linking Okiru to Mirai support the idea of a connection between the two. This suggests a potential evolutionary relationship where Okiru may have adopted certain tactics and operational strategies from Mirai.

MooBot

According to reports from The Hacker News[193] and SOC Prime[194], MooBot is a derivative of the foundational Mirai botnet, but it incorporates advanced functionalities and adopts a more focused strategy. MooBot's primary objective is to exploit security vulnerabilities in IoT devices, including routers, digital video recorders, and surveillance systems, to gain unauthorized access and control. Once compromised, these devices are utilized as part of a botnet to carry out Distributed Denial of Service (DDoS) attacks. The pseudo-static analysis of MooBot reveals that it imports and exports several critical functions that demonstrate its refined capabilities.

For process and error management, MooBot imports functions such as *_cxa_begin_cleanup*, *_cxa_call_unexpected*, and *_pthread_unwind*. These functions may be vital for manipulating system processes and handling exceptions, enabling the malware to manage the lifecycle of processes smoothly. In terms of network communication control, MooBot exports functions like *_sys_recv*, *_sys_send*, *_sys_connect*, and *_sys_accept*. These are pivotal in controlling network interactions and mirror Mirai's strategy to manipulate traffic and maintain control over compromised devices. Additionally, MooBot's exports, such as
attack_method_udpgeneric and *attack_udp_ovhhex*, highlight its specialized DDoS capabilities. These features reflect tactical similarities with Mirai in executing network-based attacks.

Further insights are provided by the dynamic analysis features of MooBot, which shows extensive file operations and command executions. The malware accesses and modifies critical system configuration files, such as */etc/mtab* and */etc/group*, to understand the system's configuration and identify vulnerabilities or optimal points for persistence. It also deletes files like */tmp/sample* or */tmp/base*, likely to eliminate traces that could facilitate forensic analysis or interfere with its activities, mirroring Mirai's tendency to clean up after itself to avoid detection. Like Mirai, MooBot executes a sequence of shell commands to inspect system processes, indicative of its attempts to oversee and potentially control running processes to ensure no competing malware or security processes are active, which helps maintain its stealthy presence and control over the compromised device. The employment of techniques categorized under the MITRE ATT&CK framework, such as executing commands through a shell command-line interpreter, creating hidden files, and querying system information via the *uname* system call, underscores MooBot's sophisticated operational profile aimed at system manipulation, persistence, and evasion. These tactics align closely with those observed in Mirai, highlighting a similar strategic approach to maintaining a high infection rate while laying low within infected systems.

Although these observations imply that MooBot might be a variant of Mirai, this association remains speculative. Evidence suggesting a link between MooBot and Mirai supports the idea of a connection; however, the graph intriguingly shows the edge originating from MooBot to Mirai, rather than vice versa but this likely stems from inaccuracies in estimating evolutionary distances.

EnermyBot

According to reports from The Hacker News[193] and SOC Prime[194], Enemybot is identified as a derivative of the Mirai malware, although Figure 5.10 does not establish a direct connection from Mirai to Enemybot. Instead, it shows a transitive connection via Okiru and Netwire RAT. This indirect linkage suggests nuanced evolutionary paths in the malware landscape, reflecting sophisticated adaptations over time.

Enemybot demonstrates advanced network communication capabilities similar to Mirai, as evidenced by functions such as *setup_connection* and *port80_setup_connection*. These functionalities may indicate Enemybot's ability to establish connections for command and control (C&C) operations or to facilitate the spread to additional systems. Furthermore, Enemybot's handling of string and memory management through functions such as *strcpy*, *_GI_memchr*, and *bcmp* is crucial for data manipulation within infected devices, mirroring strategies employed by Mirai.

In terms of security tactics, Enemybot incorporates anti-debugging measures with functions such as *anti_debug_shit*, reflecting a defensive mechanism against reverse engineering similar to those observed in polymorphic malware. This strategy is complemented by its ability to manage credentials and user interactions, utilizing functions such as *consume_user_prompt* and *consume_pass_prompt*. These features automate the theft or input of credentials, closely resembling Mirai's approach to exploiting devices through default credentials. Moreover, Enemybot employs obfuscation techniques, demonstrated by the *deobf* function, to perhaps conceal its operational code and evade detection, a tactic also prevalent in Mirai's operational framework.

Dynamic analysis features reveal Enemybot's persistent and evasive maneuvers. It modifies */etc/crontab* to insert cron jobs, ensuring regular execution to maintain persistence. The use of the *uname* system call likely tailors Enemybot's behavior to the host environment or aids in evading detection. Additionally, its engagement in network communications on non-standard ports is designed to circumvent typical firewall configurations, a strategy akin to Mirai's. The malware's interaction with critical system files such as */proc/sys/vm/mmap_min_addr* and access to device watchdog mechanisms through */dev/misc/watchdog* and */dev/watchdog* suggests a sophisticated approach to system reconnaissance and attempts to disable system watchdogs. Furthermore, Enemybot's execution control functions, such as *_do_global_ctor_aux* and *_do_global_dtors_aux*, reflect a meticulous process for initializing and cleaning up its environment on new devices, paralleling Mirai's thorough preparation and cleanup methods. Its ability to manage processes through *_GI_execve* and *waitpid*, along with the network address resolution capabilities provided by *gaih_inet* and *inet_pton4*, equip Enemybot with the necessary tools for effective C&C communication and target acquisition, underscoring its similarity to Mirai in operational tactics and technical sophistication.

Bashlite

Later in our analysis, we found that Gafygt and bashlite was the same according to Stamus [195], but in our dataset there were given different names.However, rather interestingly Figure5.10 does not show an connection between Gafgyt and Bashlite but only to its parents. This suggests that our approach focuses more on linking these entities to their antecedents rather than to each other, emphasizing the tracking of lineage rather than linking those that are very similar together.

5.5.2. Speculative Assessment of Potential Connections Between Mirai and Other Malware Families

We could not find any public evidence to support connections between certain malware families, such as Conti, Turla, and RDAT, originating from Mirai. Here, we speculate on the likelihood of each of these families being derived from Mirai based on Figure5.10.

Table 5.9: Speculative Analysis of Other Families

Malware Family	**Derived from Mirai**	**Reason**
Conti	Highly unlikely	Targets large organizations through ransomware, contrasting with Mirai's IoT device DDoS attacks.
Turla	Highly unlikely	State-sponsored group using APTs, different from Mirai's disruptive IoT exploitation.
RDAT	Highly unlikely	Used for data exfiltration or remote access, not aligning with Mirai's IoT attacks.
Lallal Stealer	Unlikely	Covertly collects sensitive information, contrasting with Mirai's DDoS focus.
ZuorRat	Unlikely	Targeted RAT capabilities contrast with Mirai's broad device exploitation.
NgrBot	Probable	Shares botnet use for DDoS with Mirai, but different in targeting and spreading.
Royal Ransomware	Highly unlikely	Goals of encryption for extortion differ from Mirai's network disruption.
XdSpy	Unlikely	Espionage focus on stealth and access contrasts with Mirai's disruptive nature.
Trickbot	Probable	Both use a DDoS approach

From Table 5.9, it is evident that the connections between four main families—Conti, Turla, Rdat, and Royal Ransomware—are considered very unlikely, yet they still appear as relationships in our inter-family analysis. This indicates that the tree has imperfections, largely because it is based on a lot of assumptions. However, the confidence levels for these families are also quite low, with edge weights of 30, 35, 40, and 35 respectively. This suggests that while connections are noted, they are not strongly supported.

Case Study: SmokeLoader

SmokeLoader is a sophisticated malware loader recognized for its advanced evasion techniques [196]. Since its discovery in 2011, SmokeLoader has been actively developed, employing a variety of methods to evade detection, manage execution, and ensure persistence. This malware is particularly noted for its stealth capabilities, utilizing advanced obfuscation and encryption techniques to circumvent detection by antivirus programs and malware scanners.

SmokeLoader ensures its persistence by implementing mechanisms that maintain its activity even after system restarts. Its modularity is a key feature, allowing it to download and execute various plugins and payloads from its command and control (C&C) servers, thereby adapting its functionality to meet the dynamic needs of its operators. The range of malicious payloads that SmokeLoader can deliver is broad and varied, encompassing banking Trojans designed to steal financial information from the affected users. Additionally, it has the capability to deploy ransomware that encrypts victims' data and demands ransom payments for decryption.

Furthermore, SmokeLoader can install cryptojacking software that covertly uses the resources of infected machines to mine cryptocurrency. It is also capable of loading botnet software to incorporate infected devices into networks that are used for Distributed Denial of Service (DDoS) attacks or spam campaigns. Operationally, SmokeLoader typically infiltrates systems via phishing emails or through compromised websites that are part of exploit kits. Once established on a device, it communicates with a C&C server to download additional malware tailored to the specific objectives of its attackers.

5.5.3. High-Level Overview of SmokeLoader's Operation

SmokeLoader's operations are systematically divided into several stages, each characterized by specific techniques aimed at compromising systems, evading detection, and executing malicious activities with stealth. The detailed reverse engineering analysis can be found on Elshinbary's blog[197]. Here we present a short summary.

Smokeloader Stages

In the initial stage, SmokeLoader utilizes `LocalAlloc()` for memory allocation to minimize detection risks, a less monitored method compared to `VirtualAlloc()`. It adjusts memory permissions to `PAGE_EXECUTE_READWRITE` using `VirtualProtect()`, which facilitates the direct execution of shellcode. Moreover, SmokeLoader employs a sophisticated hashing algorithm to dynamically resolve API addresses, effectively masking its dependencies and evasive intentions. In its second stage, SmokeLoader uses process hollowing techniques, creating a new process in a suspended state to inject malicious code, which then runs under the guise of legitimate processes. It implements complex conditional operations known as opaque predicates and includes anti-analysis techniques that obfuscate the actual execution flow, complicating static analysis. Additionally, it incorporates anti-debugging measures, checking for OS version and debugging status through the Process Environment Block (PEB) to adjust its behavior in the presence of debugging tools. Communication with command and control servers is encrypted, obscuring network activities from straightforward monitoring. Lastly, SmokeLoader re-encrypts functions post-execution to revert to a less detectable state, thus maintaining persistence and evading detection. It employs anti-hooking strategies by using its own versions of system DLLs loaded from alternate locations to evade monitoring tools that hook standard API calls. Furthermore, SmokeLoader applies custom techniques for resolving imports and detecting virtualized environments, using specialized hashing functions to complicate direct API calls, enhancing its ability to operate undetected in a wide range of environments.

5.5.4. Pseudo-static and Dynamic analysis

Integrating the pseudo-static and dynamic analysis details of SmokeLoader with the staged operational techniques, we can present a more connected and comprehensive narrative that aligns the specific activities with each stage of the malware's execution process.

SmokeLoader, an older malware originally based on DOS, presents certain challenges in analysis. Specifically, Lief was unable to extract import or export data from DOS formats. Nevertheless, it is still possible to discern potential correlations between certain opcodes and SmokeLoader's operational techniques. Arithmetic and data operations such as the opcodes `add`, `sub`, and `mov` are frequently utilized for tasks like loading and decrypting payloads, crucial for the malware's ability to manipulate and prepare data. Stack management operations using `push` and `pop` are vital for managing function calls and local variables, particularly during the process hollowing phase, where the malware configures its operational environment within another host process. System interactions through `int` and `call` opcodes are fundamental for making software interrupts and function calls, crucial in SmokeLoader's anti-debugging tactics where it interacts directly with the operating system to detect and evade debugging tools. The `xor` opcode probably facilitates straightforward encryption or decryption operations, aiding in the secure handling of the malware's encrypted functions and communications.

For anti-debugging techniques, the `sidt` instruction, which stores the interrupt descriptor table register's contents, potentially serves as an anti-debugging measure to identify changes in system interrupt configurations often altered by debugging environments. Security operations employing `xor` and `and` instructions are commonly used in cryptographic tasks to securely manipulate data, aligning closely with how SmokeLoader manages its encrypted functions and command and control communications.

Dynamic analysis of SmokeLoader, helps correlate some behaviors to stages of operational tactics as discussed on n1ght-w0lf's blog[197]. The behavior of SmokeLoader involves opening, writing, and sometimes deleting critical files across the system, accessing crucial system DLLs like `ntdll.dll`, which is essential for a variety of system-level functions, including those required for process manipulation and API hooking. This is consistent with the Initial Compromise and Shellcode Execution stage, where SmokeLoader allocates memory and adjusts permissions to execute shellcode.

In the Evading Detection and Securing Persistence stage, SmokeLoader's capability to write executable files in user directories and startup folders is particularly notable. It creates links and executables in paths such as *%APPDATA\\microsoft\\windows\\start menu\\programs\\startup* , which facilitates autostart capabilities essential for maintaining persistence. This aligns with the process hollowing techniques observed, where new processes are created and existing ones hijacked to inject malicious code, running it under the guise of legitimate operations.

Moreover, the deletion of registry keys related to startup tasks and browser configurations, along with the execution of commands that initiate processes with administrative privileges, highlights SmokeLoader's efforts to modify system settings to its advantage or potentially disable security measures that could hinder its operations. These actions support the Anti-Debugging and Anti-VM techniques identified in the reverse engineering analysis, helping SmokeLoader ascertain the environment and adapt its behavior to avoid detection tools and virtualized analysis environments. Changes in file attributes and the manipulation of file locations to less monitored directories underscore the Encryption of Communication and Re-encryption of Functions tactics, enhancing its stealth and making static and network-based analysis more challenging.

5.5.5. Inter-family analysis

Now that have we established a solid understanding of this malware family. We can have a look at the result of inter-family analysis with smoke loader. This is presented in Figure5.11.

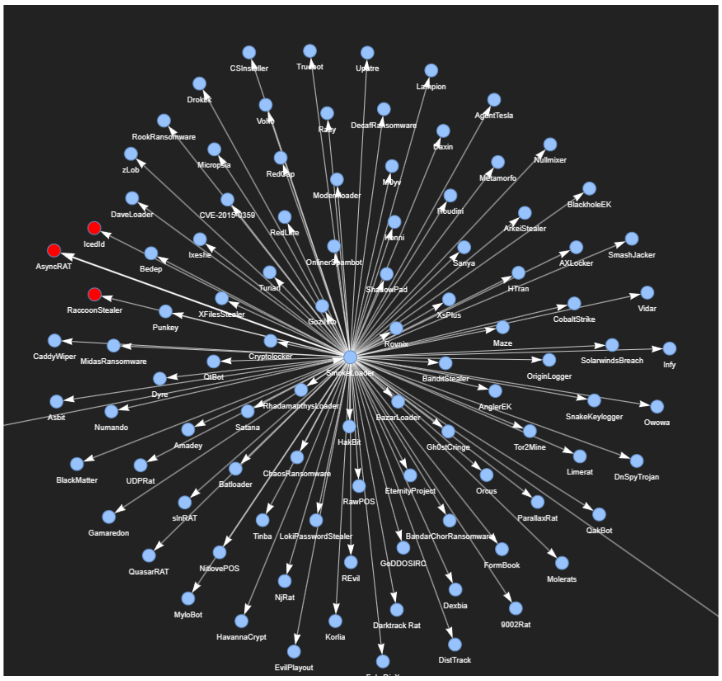


Figure 5.11: Interfamily analysis of smokeloader shows that it is the progenitor of a many families

SmokeLoader appears to be the progenitor of various malware families, likely due to its design as a versatile loader capable of downloading and executing a diverse array of malware payloads. This flexibility makes it a preferred tool for cybercriminals, allowing them to tailor their attacks based on the target environment or specific objectives. As a gateway for further exploits, SmokeLoader sets the stage for subsequent infections. We will focus our analysis on three such families: IcedID, RacoonStealer, and AsyncRAT highlighted in red in Figure 5.11.

IcedID

According to an article on Medium by Walmart Global Tech[198], IcedID employs SmokeLoader as an effective private loader. Given SmokeLoader's function as a generic loader, it's more practical to focus on dynamic analysis features rather than comparing pseudo-static features between them. The observed behaviors of IcedID align closely with those expected from SmokeLoader, suggesting its potential role in deploying various malware payloads. IcedID actively engages with both system and user directories, accessing vital DLLs and writing to directories like `\APPDATA`. These actions may reflect SmokeLoader's methods to discreetly deploy payloads by manipulating the file system, aiming to load and execute IcedID without drawing attention. Furthermore, IcedID's operations involving writing to temporary directories and modifying user profile areas are crucial for staging the malware, ensuring it is ready for execution upon system restart or user login, a tactic commonly employed by SmokeLoader.

Moreover, IcedID's strategy includes terminating and manipulating system processes, such as `svchost.exe` and `wmiadap.exe`. This manipulation potentially disables defenses or modifies system behaviors to facilitate malware activities, which likely mirrors SmokeLoader's tactics for controlling processes. IcedID also creates processes in a suspended state, setting the stage for subsequent code injection. This method closely aligns with SmokeLoader's techniques of injecting and running malware within legitimate processes to enhance stealth and evade detection by security systems. In terms of registry and system security manipulation, the use of sophisticated encryption and encoding methods like RC4 and XOR, along with dynamic function linking by IcedID, indicates a concerted effort to secure communications and functionalities. These traits are potentially essential for maintaining the stealthy operations of SmokeLoader. Also, IcedID's modifications to Windows certificates and system policies suggest attempts to deepen system integration and ensure persistence, clearly reflecting SmokeLoader's capability to conFigureand manipulate system environments to support undetected operations of malware.

IcedID's utilization of HTTPS for secure command and control communications and conducting DNS lookups to manage network interactions demonstrate its capability for sophisticated network manipulation, a characteristic shared with SmokeLoader. Moreover, evasion tactics such as detecting virtualization, employing long sleep cycles, and dynamically adjusting behaviors to counter analysis tools align with SmokeLoader's advanced evasion methodologies. These strategies are crucial to ensuring IcedID's operation remains undetected which may imply SmokeLoader as the private loader.

Raccoon Stealer

Raccoon Stealer, identified as a type of malware known as an information stealer, was first observed in April 2019 and quickly gained notoriety for its effectiveness and ease of use. Distributed typically as a service in underground forums, Raccoon Stealer follows a Malware-as-a-Service (MaaS) model, allowing even those cybercriminals with limited technical expertise to deploy the malware in exchange for a fee [199]. According to VMRA[199], Raccoon Stealer also utilizes SmokeLoader as a private loader.

The dynamic analysis features of Raccoon Stealer reveals its significant interactions with system files, specifically engineered to access and manipulate sensitive user data. For instance, the malware opens critical data files such as `%LOCALAPPDATA%\google\chrome\user data\default\login data` and `%APPDATA%\mozilla\firefox\profiles\*.default\cookies.sqlite`, indicative of its intent to harvest credentials and cookies. This behavior mirrors SmokeLoader's methods, which typically modify or access files to deploy or update malicious payloads covertly.Furthermore, Raccoon Stealer executes strategic manipulation of processes, such as terminating essential system processes like `%windir%\System32\svchost.exe -k WerSvcGroup`, and creating new processes in a suspended state for subsequent code injection. This aligns with SmokeLoader's process control tactics, aiming to disable defenses or modify system behavior to facilitate malware activities unimpeded.

Advanced evasion techniques employed by Raccoon Stealer include efforts to detect virtualization through RDTSC time measurements and the implementation of long sleep intervals (>= 3 min), designed to thwart automated analysis tools and virtualized environments. These methods are characteristic of SmokeLoader's advanced evasion capabilities. Additionally, Raccoon Stealer's use of HTTPS for secure command and control communications and conducting DNS lookups to manage network interactions reflects sophisticated network manipulation strategies, akin to those of SmokeLoader, enhancing the malware's ability to operate undetected. These observations may indicate that SmokeLoader is used to deliver Raccoon Stealer.

Async Rat

According to Threathunt[200], there is evidence suggesting that AsyncRAT also utilizes SmokeLoader. Dynamic analysis features reveal several critical behaviors that align closely with those typically exhibited by SmokeLoader.

AsyncRAT engages in significant file and directory manipulation, writing executable files such as `cloud.exe` in user-specific directories like `AppData` and `Start Menu`. This behavior is consistent with SmokeLoader's method of dropping payloads into such locations to establish persistent access and execution capabilities. Furthermore, AsyncRAT manipulates files across various system and user directories, including attribute modifications and file deletions post-actions, indicative of SmokeLoader's techniques for discreetly deploying and activating secondary payloads like AsyncRAT. Process manipulation and command execution are also central to AsyncRAT's operations. It executes and injects into processes such as `msbuild.exe` and `cmd.exe`, a tactic for maintaining persistence and execution without arousing suspicion that mirrors SmokeLoader's use of process hollowing and other injection techniques. Additionally, AsyncRAT's use of system utilities like `timeout.exe` and `cmd.exe` suggests a manipulation of system processes to perform tasks or maintain stealth, a strategy often initiated by SmokeLoader to conFigureits environment or execute further malicious payloads.

Registry and system configuration manipulation by AsyncRAT involves deleting specific registry keys related to Internet settings, potentially to modify security settings or clear tracks. This behavior, common to SmokeLoader, aids in persistence and facilitates uninterrupted command and control (C2) communication. Such registry modifications are crucial for maintaining a stealthy presence and ensuring the malware's operations are not easily detected or disrupted. Moreover, AsyncRAT modifies network configurations and utilizes advanced evasion techniques, including running in suspended modes and creating guard pages, along with employing tactics to evade detection, such as using encrypted or packed data. These actions are known characteristics of SmokeLoader, which similarly uses evasion techniques to avoid detection, including environmental awareness and potentially delivering payloads in encrypted forms. SmokeLoader's configuration of systems to communicate securely with C2 servers aligns with AsyncRAT's use of HTTPS for secure communication.Both AsyncRAT and SmokeLoader share several tactics that align with the MITRE ATT&CK framework, such as Process Injection (T1055), Scripting (T1064), System Information Discovery (T1082), and Query Registry (T1012). These activities are crucial for adapting their payloads based on the environment, with Remote File Copy (T1105) inferred from AsyncRAT's behavior to write files across the system, potentially facilitated by SmokeLoader as part of its payload deployment strategy.

Experiment 9: Are there any similarities between the inter-family relationships of UPGMA and NJ

Table 5.10: Analysis of Malware Family Relationships According to UPGMA

Parent	Child Families	Likelihood	Agrees with NJ?
Ramdo	Bashlite	Unlikely	No
Zloader	Smokeloader	Likely	No
FritzFrog	BotenaGo	Likely	Yes (but order reversed)
SundownEK	Neshta	Unlikely	No
Mirai	Bashlite	Likely	Yes
Zloader	Okiru	Unlikely	No
Lokibot	Gafgyt	Unlikely	No
DiscordTokenStealer	BianLianRansomware	Unlikely	No

In this experiment, we investigate the similarities between relationships exhibited in UPGMA and NJ for the Mirai, SmokeLoader, BotenaGo, Bashlite, SundownEK and DiscordTokenStealer families. As shown in Table5.10, it is evident that most relationships from parent to child families of UPGMA do not align with NJ.

6
Discussion

This study aimed to show how deep learning and phylogenetic trees can be used the tracing of evolution of malware. Traditional methods, like reverse engineering, usually take months to years to uncover these relationships. With the rise in wide-spread malware such as Mirai, this research seeks to substantially shorten this timeframe.

We will start our discussion by summarizing the key findings from the previous chapter. We will then reflect on the questions posed in the introduction and discuss their implications for cybersecurity research. Following this, we will examine the weaknesses identified in our experiments and consider the factors that may have influenced our results. Next, we will discuss the necessary steps to transition our research into a real-world application. Finally, we will outline potential future research directions that emerged from our discussions.

6.1. Key Results

Our key findings reveal that image analysis substantially outperforms traditional static and dynamic analysis methods in both malware classification and detection. In our efforts to construct phylogenetic trees, the Neighbor-Joining algorithm has emerged as the most effective as opposed to Unweighted Pair Group Method with Arithmetic Mean(UPGMA). Furthermore, combining deep learning with phylogenetic analysis has proven to be remarkably scalable, allowing us to discern relationships among 546 malware families within a mere month. The resulting phylogenetic tree is consistent with the majority of our case studies and aligns with most of the VirusTotal timestamps.

6.2. Reflection

In this section, we reflect on the main results by answering the research questions posed in the introduction and then highlights its implications in cybersecurity research.

Q1.Which analysis type yields the best embeddings and does combining them improve performance?
Surprisingly, image embeddings significantly outperform both pseudo-static and dynamic features in malware classification. This finding was unexpected, particularly since dynamic features, which involve analyzing malware behavior, were anticipated to provide a more accurate assessment of malicious activities. However, the literature confirms that such results are not unprecedented but rather common. For instance, Gilbert et al. [103] and Kumar et al. [105] reported an accuracy of over 95% using CNNs on the Malimg dataset, indicating that images can be effectively used for malware detection or classification.

The reasons why images are effective remain very unclear. Firstly, these images are resized using interpolation methods, which alter the original encoding of the data. Despite this, the mechanisms by which these models maintain high accuracy are not well understood but are frequently discussed in the literature. Secondly, our approach involves initializing our architecture with ImageNet weights, despite further training on a public dataset. The direct applicability of ImageNet features to malware detection is questionable, yet this methodology is widely adopted in the field, as evidenced by works such as Rustam et al. [106] and Kumar et al. [105], who report high accuracies using similar strategies. Furthermore, some studies, including those by Kumar et al. [105] and Gilbert et al. [103], do not address the issue of packing in their methodologies, yet they still achieve high accuracy. Kumar et al. [105] specifically argue for the robustness of their models against packing, having tested this by packing samples with UPX and maintaining high performance, without resorting to techniques such as extracting images from memory dumps.

There have been efforts to enhance the interpretability of models; Yakura et al. [104], for instance, incorporated an attention layer in their CNN. This layer highlights crucial regions within the images, guiding more targeted analyses such as byte sequence extraction and disassembly. However, the efficacy of attention mechanisms as a genuine explanatory tool remains contentious. Studies by Bibal et al. [201], Wiegreffe & Pinter [202], and Grimsley et al. [203] question whether attention can reliably indicate interpretive validity. Overall, the reason why images, even when resized, perform so effectively in malware classification remains elusive.

Another interesting result was that the combined embeddings, which include all three types (pseudo-static, dynamic, and image), did not perform better than just using image embeddings. We think this might be due to our choice of setting the number of hidden neurons in our supervised dimensional reduction to 1000. As we are reducing the dimensionality from 6560 to 1000, it's likely causing a significant loss of information.However, we hypothesize that by making the number of hidden neurons an adjustable hyperparameter that can be fine-tuned, the combined embeddings may potentially outperform those from image analysis alone.

Q2.Are embeddings, merged or individual, useful for downstream tasks?
The embeddings are useful, particularly the image embeddings, which have proven to be the best performing. We found that image embeddings were highly effective for malware detection, achieving near-perfect accuracy and F1 scores. This should not be surprising, given the previous discussion, as high accuracy is frequently reported in the literature [103][105] [82] [106], with Rustam et al. [106] citing a 100% accuracy for malware detection on malimg dataset.

However, relying solely on image-based techniques for malware detection or classification presents practical challenges. A significant concern is the vulnerability to adversarial attacks, where images can be manipulated to produce false classifications. For example, Chen et al.[204] demonstrated a 70% attack success rate in a black box setting using gradient-based attacks. This issue is not unique to image data; it is a broader problem that affects the entire pipeline of our systems. In scenarios where the model details are accessible (white box setting), the entire pipeline, including the accuracy of the phylogenetic tree, could be compromised. Similarly, the entire pipeline can be targeted using gradient-based inversion attacks, even in a black box setting.

Despite these challenges, there are strategies to mitigate such vulnerabilities. Adversarial training is one approach that can enhance model robustness by preparing the system to handle manipulated inputs effectively. Additionally, keeping the model details undisclosed can prevent specific types of attacks, though this strategy may not always be feasible in practice. Understanding the risks associated with using deep learning is crucial, particularly as these technologies become integral to more systems. It is important to continuously explore and implement robust security measures to safeguard against these vulnerabilities, ensuring the reliability of deep learning applications in cybersecurity.

Moreover, merged embeddings have proven effective for determining the age of malware samples using VirusTotal timestamps. This capability underscores their ability to detect temporal changes, reinforcing their robustness in detection tasks. The ability to track these time-related shifts enhances their utility in malware detection, even with evolving code changes. The use of these embeddings in constructing phylogenetic trees through similarity matrices is particularly promising because they allow for detailed visual representations of the relationships and evolutionary trajectories of various malware samples.

Q3. How does our approach on classifying malware based on images compare to previous work?
Our architecture perfomrmed better than CNN models in processing both grayscale and RGB images. This is consistent with the findings of Bhodia et al.[107], who showed that applying transfer learning with architectures like ResNet enhances feature extraction and improves classification outcomes compared to traditional CNNs. While the improved performance of ResNet can partially be attributed to its deeper network structure, which provides more robust learning capabilities, it's important to recognize that

simply having more layers does not inherently lead to better performance. ResNet addresses critical challenges such as the vanishing gradient problem through the use of residual connections. These connections allow gradients to flow through the network more effectively during training, preventing the degradation of training performance that often accompanies increased network depth. This design enables ResNet to leverage deeper architectures without the drawbacks typically associated with them in traditional CNNs [205].

Q4. Which phylogenetic tree construction method using distances produces the most accurate representation of malware evolution using VirusTotal timestamps?
Neighbor-Joining (NJ) was found to be the most accurate when rooted, primarily because it does not make any molecular clock assumption while UPGMA relies on a constant evolution rate. On the other hand, UPGMA was better than NJ when unrooted mainly because UPGMA inherently produces an unrooted tree, and introducing a root changes its topology, thereby reducing its accuracy. But overall, NJ was the better method. This suggests that malware, in general, exhibits heterogeneous mutation rates.

Q5. How can embedding drift analysis be employed as an alternative method for validating phyloge- netic trees built from UPGMA nd NJ?
Embedding drift analysis involves examining changes in embeddings over time to gauge mutation rates. Our findings indicate substantial variability in mutation rates across different malware families, reinforcing the accuracy of Neighbor-Joining (NJ) in producing more precise phylogenetic trees. This analysis further reveals that some malware families, like Mirai, mutate frequently and on a large scale, while others, such as the advanced persistent threat (APT) Turla, evolve much slower, with new variants emerging perhaps only annually. This disparity underscores the necessity for customized detection strategies tailored to the specific evolutionary rates of different malware families.

Q6. How do clusters formed by visualizing malware embeddings with t-SNE, UMAP, and PCA align with lateral(leaf to leaf) distances in a phylogenetic tree built with NJ method?
We observed that the clusters formed through 2D visualizations of malware embeddings generally align with the lateral distribution of distances in phylogenetic trees. This indicates that the evolutionary model assumes that samples clustered together in the visualization are likely to be near each other on the tree. This correlation serves as an effective sanity check, as it would be problematic if samples that appear closely grouped in the visualization were distant on the phylogenetic tree. It's important to note, however, that we are projecting from a 1000-dimensional space to only 2 dimensions, which means that not all distances in the 2D projections accurately reflect those in the higher-dimensional space.

We also encountered three out of 20 instances where there was a misalignment between the visual clusters and the tree distances. This discrepancy could be due to the Neighbor-Joining method treating certain data points, which we consider outliers, as typical during the modeling process. Another possibility is that the 2D representation itself is not accurate. While t-SNE prioritizes preserving local structure, suggesting that points close together in higher dimensions remain close in the reduced dimension, UMAP seeks a balance between maintaining fidelity for both close and distant points, aiming for a more comprehensive representation. Both dimensionality reduction techniques heavily depend on hyperparameter settings, which might in certain cases not accurately reflect the structure in higher dimensions.

Q7.Do outliers alter the topology of a phylogenetic tree constructed using the NJ method by changing the Most Recent Common Ancestor (MRCA)?
Our results demonstrated that including outliers substantially altered the topology of the phylogenetic tree, making it difficult to identify any isomorphic or isomorphic substructures. This indicates that it is crucial to consider these outliers prior to constructing the tree, as their presence can fundamentally change the tree's topology and, consequently, the interpretations derived from it. One interesting observation, however, is that the average degree remains the same for trees with and without outliers. We speculate that this is due to the nature of fully connected trees, where each node maintains connections to all other nodes, suggesting a balanced connectivity across nodes.

The average degree, a measure of the average number of connections per node, remained consistent regardless of the presence of outliers. This consistency indicates that, on average, each node maintains a similar number of connections. Such stability suggests that the overall distribution of edges per node across the tree remains relatively balanced, even when the structure undergoes modifications due to the inclusion of outliers.

Furthermore, the introduction of outliers predominantly affects the tree's depth and the extremities by adding more layers or branches. However, these changes do not necessarily alter the average connectivity per node. New nodes, whether added or connected differently due to outliers, tend to maintain an average degree similar to the original setup. Particularly, outliers that form additional connections with existing nodes do not disproportionately increase the total number of edges per node. This observation implies that although the tree's topology and depth are influenced by outliers, the integral connectivity per node remains stable, reflecting a robust underlying structure capable of withstanding significant alterations.

Q8. Do the relationships identified through inter-family analysis using the Neighbour Joining method correlate with public cybersecurity insights and with the pseudo-static and dynamic features of the malware?
Most of our case studies align with the relationships depicted in the phylogenetic tree, suggesting that, despite being somewhat speculative, Our tree accurately reflects relationships corroborated by public sources. However, in the specific case of Mirai, we observed purported relationships with unrelated entities such as APTs like Turla and RATs like RDAT. These connections appear highly improbable, and interestingly, the edge distances to these nodes are also substantial. This observation raises a critical question: At what point do we determine that certain relationships are valid while others are not? Currently, our methodology does not directly address this issue. Although we use edge distances as a form of confidence indicator, we have yet to establish a clear threshold for where to set the boundary. This ambiguity highlights the need for a more defined criteria to interpret these distances within our phylogenetic analysis as we will point out later.

Another issue with phylogenetic tree algorithms is their inherent tendency to establish relationships between families. For instance, even if a hypothetical malware family evolved completely in isolation from others, the algorithm would still attempt to connect it with some other family. Therefore, it's crucial to develop a heuristic to set a threshold for the weights between the edges, ensuring that only meaningful connections are considered in the inter-family analysis.

Q9. Are there any similarities between the inter-family relationships of UPGMA and NJ?
The comparative analysis of NJ and UPGMA methodologies in understanding malware relationships reveals significant differences that influence the perceived lineage and functionalities of various malware families. The NJ method aligns Mirai with Bashlite, indicating their shared functionalities in launching DDoS attacks and targeting IoT devices. This logical linkage contrasts sharply with UPGMA's association of Bashlite with Ramdo, which lacks empirical evidence and coherence, suggesting a critical oversight in UPGMA's approach. Furthermore, UPGMA's associations — such as linking Zloader with Okiru and Lokibot with Gafgyt — deviate from more convincing connections in NJ that link Mirai with both Gafgyt and Okiru, pointing to a potential misunderstanding of malware functionalities and historical developments by UPGMA. Additionally, their positioning of Zloader as a precursor to Smokeloader due to their similar roles in payload delivery starkly contradicts NJ's portrayal of Smokeloader as a more central figure in malware distribution. This highlights UPGMAs tendency to oversimplify or misinterpret complex malware relationships.

Moreover, specific cases such as SundownEK and DiscordTokenStealer further illustrate the inconsistencies of UPGMA. For example, UPGMA incorrectly links SundownEK, an exploit kit, with Neshta - a file infector. NJ more accurately associates SundownEK with Kronos, adhering more closely to the functional realities of these malware types. Similarly, the UPGMA link between DiscordTokenStealer and BianLianRansomware contrasts sharply with the connection made by NJ to AkiraRansomware which demonstrates UPGMA's flaw in not adequately accounting for distinct operational divergences between malware types. Despite these disparities, both NJ and UPGMA agree on one relationship between FritzFrog and BotenaGo - peer-to-peer botnets that share operational characteristics like network resilience and decentralized command structures. Interestingly, NJ connects BotenaGo to FritzFrog, while UPGMA suggests the opposite. Overall, these analyses suggest that while UPGMA frequently oversimplifies or inaccurately represents malware relationships potentially leading to erroneous conclusions about their evolution; NJ typically offers a more nuanced understanding aligning closely with operational realities of involved malware families.

6.3. Factors that could influence the results

In this section we will highlight factors that may influence the results of the experiments.

6.3.1. Packing

Firstly, in pseudo-static analysis, we utilized the approach of running the malware, capturing a memory dump, and then extracting the executable from that dump. This method hinges on the fundamental assumption that the first dump containing an executable resembling a PE (Portable Executable) or ELF (Executable and Linkable Format) file is used. However, this approach can be problematic if the malware utilizes a .dll (dynamic-link library), which may link multiple executables or objects. During our preprocessing, we eliminate these linked files, but there is a risk that the malware may not execute at all unless all components specified in the .dll are present. This scenario could significantly impact the pseudo-embeddings because, effectively, we might only be extracting features from a static analysis, without truly mitigating the effects of packing.

Moreover, our analysis does not account for runtime encryption or other dynamic activities that might conceal the malware's true behavior until it is executed. Such runtime operations can change the executable's characteristics in ways that are not apparent through static analysis, potentially leading to significant discrepancies in our findings. Similarly, during our dynamic analysis, we observed that for some samples from families such as MagicRAT and AsyncRAT, no behavioral information could be detected. To manage this, we resorted to assigning a zero vector for these families in the dynamic embeddings during test time and taking the mean embeddings for that family during runtime. But this could have influenced the results of dyamic embeddings negatively.

6.3.2. Data Splitting

In our methodology, we encountered an issue with data splitting across our pipeline, where we currently use a 70/30 train-test split to retrieve the embeddings. This approach has led to potential model overfitting and resulting in data leakage in two specific instances. First, in our image analysis process, we sequentially train models V1 and V2 using this split. After training V2, we then apply the embeddings across the entire dataset, including the test data. Since V2 has already been exposed to the data in the 70% training subset, the embeddings applied can therefore be biased. Secondly, when we merge diverse embeddings (pseudo-static, dynamic, and image), we encounter the same issue of bias. We derive these embeddings using the 70/30 split and apply them across the entire dataset. Thus, 70% of the data, which was included in the training set, has already been seen, affecting the neutrality of the embeddings. A more effective approach would have been to employ a 50/50 split for all three types of embeddings, training on the first half and applying the learned embeddings to the second half. We would then use this unbiased dataset to run the phylogenetic algorithms, ensuring a more accurate and fair evaluation.

From a pessimistic perspective, the current approach may lead to biased embeddings, which in turn could result in a skewed phylogenetic tree. However, if significant bias were present, it would likely impact the performance of merged embeddings in experiments like our time-binning test, where we categorized embeddings by year and used them to predict the corresponding years. Typically, overfitting would manifest as the model learning the labels too specifically, and when tested on tasks unrelated to those labels — such as assessing whether embeddings can capture temporal concepts — performance would decline. Contrary to this expectation, we observed robust performance in these tests. Moreover, if our phylogenetic tree were indeed compromised by data leakage, it would be surprising to find that it aligns well with external validations such as VirusTotal, and that most of the familial relationships derived from the tree correlate with those recognized by public sources. Thus from an optimistic standpoint, these observations suggest that any potential data leakage did not substantially influence the overall validity of our results.

6.3.3. Labelling

Relying heavily on labels for constructing phylogenetic trees means that the accuracy of these labels is crucial. If the labels are incorrect, the resulting phylogenetic tree will misrepresent the relationships between malware samples, potentially leading to flawed interpretations and strategies based on these inaccuracies. Although validating labels with VirusTotal is a common approach[206] since it aggregates information from multiple antivirus engines, there are still inherent risks if VirusTotal's data is incomplete or biased.

6.3.4. Approximation of Neigbour Joining Method

From our experiments, we concluded that Neighbor Joining was the best method compared to UPGMA. However, due to computational complexity in our work, we opted for an optimized or rather approximate version known as RapidNJ. Employing RapidNJ introduces certain limitations that need to be considered. While RapidNJ increases computational efficiency, this can come at the expense of reduced accuracy and detail, especially in complex datasets. This approximation may not capture all nuances that a full Neighbor Joining analysis would, potentially leading to less precise phylogenetic trees. Additionally, the simplifications made to speed up calculations might introduce systematic biases in the way distances between sequences are calculated and clustered, potentially affecting the overall quality of the phylogenetic analysis. These factors could lead to oversimplified interpretations of the relationships between malware variants, which might not be representative of their true evolutionary paths.

6.3.5. Assumptions in inter and intra-family analysis

The second-order assumptions, which suggest using the most recent common ancestor (MRCA) of leaves from two different families to determine relationships based on early divergences in the phylogenetic tree, may not always be accurate. This is particularly the case when one malware family is closely connected to the root, and another is significantly further away. During traversal up the tree, inaccuracies in branch lengths can become increasingly pronounced, leading to compounded errors. These inaccuracies can distort interpretations of both intra-family and inter-family relationships and may violate the principle of relative early divergences. One potential solution to mitigate these issues is to implement a high-confidence simplification strategy. This approach would involve limiting the traversal to a specific number of ancestral levels, denoted as `X`, from a given leaf node. If the most recent common ancestor (MRCA) is not identified within those `X` levels, it would be concluded that a relationship cannot be confidently established at that level of confidence. This method helps to contain the propagation of errors in branch length estimation and provides a clearer, more reliable framework for analyzing phylogenetic relationships.

In our analysis, we also inherently assumed that hybridization events, where a gene transfers its properties to two different genes, do not occur. This assumption does not hold in all cases, particularly in the context of malware, where hybridization can be a common phenomenon. Phylogenetic trees are inherently limited in modeling such hybrids because they are structured to represent evolutionary relationships in a bifurcating tree-like form, where each split or node traditionally represents divergence into two distinct and non-recombining lineages. This framework is excellent for tracing lineage splits but fails to adequately represent more complex scenarios where there is convergent evolution, recombination, or horizontal gene transfer — events that are analogous to hybridization in biological contexts. To more accurately represent these relationships, one should use phylogenetic networks instead of trees. Phylogenetic networks are specifically designed to capture the complexities of evolutionary histories that include recombination, gene flow, or horizontal gene transfer. These networks allow for the depiction of multiple parental nodes at specific points, effectively illustrating hybridization events. This approach provides a deeper insight into the evolutionary dynamics among malware families. However, it is important to note that phylogenetic networks are more computationally demanding than traditional tree models.

Lastly, In our analysis, both for validation and for comparing relationships within and between malware families, we often rely on the distance to the most recent common ancestor (MRCA) as a crucial metric. However, we have not thoroughly addressed scenarios where these distances are very close to each other. If distances to the MRCA among different nodes are nearly identical, drawing robust conclusions about the relationships can be challenging, and under very strict criteria, might even be deemed unreliable. Despite this, our current methodology does not account for such nuances.

To improve our approach, we could introduce a tunable hyperparameter called tolerance, which adjusts how we interpret proximity to the MRCA. This can be achieved by constructing trees with varying tolerances and selecting the one that best aligns with external timestamps. This parameter would enable us to define what constitutes a significant or meaningful difference in MRCA distances, allowing for a tailored analysis that meets the specific sensitivity requirements of the study. By incorporating this adjustment, we would improve the reliability of our conclusions.

6.4. Production

Our pipeline, beginning with the generation of embeddings, lacks full interpretability, which is a common challenge in the field of artificial intelligence. Despite efforts to clarify why methods like image classification perform well, significant gaps in understanding remain. Yet, the critical question is whether this lack of interpretability significantly impacts the utility of our approach. For antivirus vendors, whose primary goal is malware detection, our methodology is highly practical. The main advantage lies in the ability of our phylogenetic tree to generate more reliable behavioral signatures by leveraging mutation rates and branch distances between malware families, a method that offers an alternative to traditional approaches such as those proposed by Vinod et al.[144], who use opcodes to build signatures from evolutionary insights.

Additionally, for stakeholders prioritizing interpretability, our work provides a valuable structured framework that aids security analysts in manually testing relationships through rigorous reverse engineering. Without such a framework, pinpointing a starting point for detailed analysis would be much more challenging.From an artificial intelligence perspective, employing ”black box” models does not present inherent disadvantages in this context. Malware creators typically overlook the interpretability of the AI systems they utilize, often employing AI to generate malware on a large scale. Leveraging AI-based strategies is likely our most effective option for counteracting these threats, making our research a crucial early effort in developing robust defenses against evolving cyber threats.

Furthermore, our pipeline can generate relationships faster than traditional methods. However, it is not yet optimized for production use, and processing times can be extensive, depending on the executable's size. For instance, analyzing a large file like Microsoft Word could take between 4 to 6 hours at a minimum. That said, one important consideration is that in our methodology the tree currently needs to be rebuilt with each new sample. Recent advancements, such as the algorithm proposed by He et al.[207], allow for online processing of samples, which is practical as it eliminates the need to rebuild the tree with each new sample. This online processing capability significantly enhances the efficiency and applicability of our approach in a real-world setting.

6.5. Future Works

This study has highlighted several promising areas for future research that are essential for advancing the application of artificial intelligence in malware analysis. One immediate need is the practical implementation and validation of theoretical algorithms such as those proposed by He et al.[207]. These algorithms hold potential for real-world application, yet they require rigorous testing to evaluate their effectiveness in dynamic settings.

Moreover, as our understanding of the impact of different features in both pseudo-static and dynamic malware analysis deepens, it becomes imperative to conduct sensitivity analyses. Such analyses would help identify the most impactful features, thereby refining our prediction models and enhancing their accuracy.The effectiveness of image-based classification methods, particularly when preprocessing involves resizing or pretraining on datasets unrelated to malware, like ImageNet, also warrants further exploration. Investigating the resilience of these methods against common obfuscation techniques such as packing could lead to more robust malware detection systems.

Additionally, the opacity of data preparation methods in public malware datasets poses a significant challenge. Advocating for transparency and standardization in how these samples are prepared could greatly enhance the reproducibility and reliability of machine learning applications in this field.Exploring the transformation of malware samples into binary sequences opens the door to applying sophisticated phylogenetic algorithms, including maximum likelihood and Bayesian methods. These approaches are well-known for their precision and robust validation mechanisms[148] but are underutilized due to their reliance on sequences.

Finally, considering the complexity and hybrid nature of malware evolution, it is pertinent to shift from traditional phylogenetic trees to more comprehensive phylogenetic networks. Such networks could provide a more nuanced representation of intra- and inter-family relationships among malware varieties, accommodating the hybridization events that are often seen in malware development.Addressing these areas will not only help overcome current limitations but also significantly advance the use of AI and machine learning tools in cybersecurity, enhancing our defences against evolving cyber threats.

7 Conclusion

This thesis has demonstrated the potent capabilities of deep learning in generating combined embeddings from static, dynamic, and image analysis to address various aspects of malware detection and analysis. By adapting static analysis to function on memory dumps—termed pseudo-static analysis—we mitigated some effects of packing, enhancing the robustness of our models. Our results consistently showed that image analysis produces superior embeddings compared to pseudo-static or dynamic analysis, underscoring its effectiveness. Further, we explored the utility of these embeddings in tasks beyond their initial training scope, such as identifying the age of malware and detecting malware using image embeddings. In both cases, these embeddings achieved high accuracy, surpassing baseline random guesses. When compared with traditional CNN architectures processing grayscale and RGB images, our architecture significantly outperformed, highlighting its advanced capability in classifying malware.

Moreover, we successfully constructed phylogenetic trees using embeddings derived from pseudo-static, dynamic, and image analysis, with Neighbor Joining (NJ) providing the most accurate representation of malware evolution. This was validated against timestamps from VirusTotal, and supplemented by Embedding Drift Analysis, which indicated heterogeneous mutation rates across malware families—thus validating that NJ, which does not assume a constant evolutionary clock unlike Unweighted Pair Group Method with Arithmetic Mean(UPGMA), offers a more accurate model. Visualization techniques like t-SNE, UMAP and PCA were employed to analyze how clusters formed by malware embeddings align with the lateral distances in a phylogenetic tree built with the NJ method. These clusters largely corresponded with the phylogenetic tree's lateral distributions, reinforcing the validity of our methodologies. We also examined the impact of outliers on the topology of the phylogenetic tree, discovering that they significantly alter the tree's structure, often preventing the identification of isomorphic substructures.

Our inter-family analysis using the NJ method corroborated with public insights for several case studies including Mirai, SmokeLoader, and BotenaGo, confirming that our phylogenetic assessments are generally aligned with known malware relationships. Additionally, while comparing the inter-family relationships derived from UPGMA and NJ methods, few similarities were found, which suggests distinct differences in how these methods conceptualize evolutionary relationships. In conclusion, employing deep learning and phylogenetic trees to trace malware evolution proves to be an effective strategy. Although our approach lacks inherent interpretability, it scales well and provides a structured framework for validating malware relationships. Furthermore, the generated phylogenetic trees offer valuable insights that can be used to develop evolutionary-based signatures for malware detection, offering a novel toolset for cybersecurity professionals in combating malware threats.

References

[1] Ashu Sharma and Sanjay Kumar Sahay. “Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey”. In: *CoRR* abs/1406.7061 (2014). arXiv: 1406.7061. URL: http://arxiv.org/abs/1406.7061.

[2] Zahid Akhtar. *Malware Detection and Analysis: Challenges and Research Opportunities*. Jan. 2021. DOI: 10.48550/arxiv.2101.08429. URL: https://arxiv.org/pdf/2101.08429v1.pdf.

[3] Andreas Moser, Christopher Kruegel, and Engin Kirda. “Limits of Static Analysis for Malware Detection”. In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. 2007, pp. 421–430. DOI: 10.1109/ACSAC.2007.21.

[4] Efstratios Chatzoglou et al. “Bypassing antivirus detection: old-school malware, new tricks”. In: *Proceedings of the 18th International Conference on Availability, Reliability and Security*. ARES ’23. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400707728. DOI: 10.1145/3600160.3605010.

[5] Qasem Abu Al-Haija, Ammar Odeh, and Hazem Qattous. “PDF Malware Detection Based on Optimizable Decision Trees”. In: *Electronics* 11.19 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11193142. URL: https://www.mdpi.com/2079-9292/11/19/3142.

[6] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. “Static PE Malware Detection Using Gradient Boosting Decision Trees Algorithm”. In: *Future Data and Security Engineering*. Ed. by Tran Khanh Dang et al. Cham: Springer International Publishing, 2018, pp. 228–236. ISBN: 978-3-030-03192-3.

[7] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. “Survey of machine learning techniques for malware analysis”. In: *Computers Security* 81 (2019), pp. 123–147. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0167404818303808.

[8] Charles LeDoux and Arun Lakhotia. “Malware and Machine Learning”. In: *Intelligent Methods for Cyber Warfare*. Ed. by Ronald R. Yager, Marek Z. Reformat, and Naif Alajlan. Cham: Springer International Publishing, 2015, pp. 1–42. ISBN: 978-3-319-08624-8. DOI: 10.1007/978-3-319-08624-8_1. URL: https://doi.org/10.1007/978-3-319-08624-8_1.

[9] Xin Su et al. “DroidDeep: using Deep Belief Network to characterize and detect android malware”. In: *Soft Computing* 24.8 (Jan. 2020), pp. 6017–6030. DOI: 10.1007/s00500-019-04589-w.

[10] Yuxin Ding, Sheng Chen, and Jun Xu. “Application of Deep Belief Networks for opcode based malware detection”. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 3901–3908. DOI: 10.1109/IJCNN.2016.7727705.

[11] Zhangjie Fu, Yongjie Ding, and Musaazi Godfrey. “An LSTM-based malware detection using transfer learning”. In: *Journal of Cybersecurity* 3.1 (2021), p. 11.

[12] Andre Karamanian. “The Application of Computer Vision to Detect Malware”. In: *European Conference on Cyber Warfare and Security*. Academic Conferences International Limited. 2018, pp. 240–243.

[13] Ahmet Selman Bozkir et al. “Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision”. In: *Computers Security* 103 (2021), p. 102166. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.102166. URL: https://www.sciencedirect.com/science/article/pii/S0167404820304399.

[14] Martin Lutz. *How hackers use Artificial Intelligence to create malware: Unveiling the new threat of wormgpt*. July 2023. URL: https://www.linkedin.com/pulse/how-hackers-use-artificial-intelligence-create-malware-martin-lutz/.

[15] Veronica Chierzi Threat Researcher. *A closer look at CHATGPT's role in Automated Malware Creation*. Nov. 2023. URL: `https://www.trendmicro.com/en_us/research/23/k/a-closer-look-at-chatgpt-s-role-in-automated-malware-creation.html`.

[16] Alex Blake. *Hackers are using AI to create vicious malware, says FBI*. July 2023. URL: `https://www.digitaltrends.com/computing/hackers-using-ai-chatgpt-to-create-malware/`.

[17] URL: `https://www.impactmybiz.com/blog/how-ai-generated-malware-is-changing-cybersecurity/`.

[18] Victor Tangermann. *Researchers create AI-powered malware that spreads on its own*. Mar. 2024. URL: `https://futurism.com/researchers-create-ai-malware`.

[19] Matt Burgess. *Here come the ai worms*. Mar. 2024. URL: `https://www.wired.com/story/here-come-the-ai-worms/`.

[20] Yair Herling. *From CHATGPT to Redline Stealer: The Dark Side of OpenAI and google bard*. Dec. 2023. URL: `https://veriti.ai/blog/veriti-research/from-chatgpt-to-redline-stealer-the-dark-side-of-openai-and-google-bard/`.

[21] Zvelo. Feb. 2024. URL: `https://zvelo.com/malicious-ai-the-rise-of-dark-llms/`.

[22] Contributing Writer Elizabeth Montalbano. *"Darkbert" GPT-based malware trains up on the entire dark web*. Dec. 2023. URL: `https://www.darkreading.com/application-security/gpt-based-malware-trains-dark-web`.

[23] URL: `https://foxdata.com/en/blogs/gptpowered-malware-darkbert-exploring-the-depths-of-the-dark-web/`.

[24] Andrew Walenstein et al. "The design space of metamorphic malware". In: *2nd International Conference on i-Warfare and Security (ICIW 2007). 2nd International Conference on i-Warfare and Security (ICIW 2007)(2007)*. 2007, pp. 241–248.

[25] Raj Badhwar. "Polymorphic and Metamorphic Malware". In: *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms*. Cham: Springer International Publishing, 2021, pp. 279–285. ISBN: 978-3-030-75354-2. DOI: `10.1007/978-3-030-75354-2_35`. URL: `https://doi.org/10.1007/978-3-030-75354-2_35`.

[26] Hamid Darabian et al. "An opcode-based technique for polymorphic Internet of Things malware detection". In: *Concurrency and Computation: Practice and Experience* 32.6 (2020), e5173.

[27] Huanran Wang et al. "An evolutionary study of IoT malware". In: *IEEE Internet of Things Journal* 8.20 (2021), pp. 15422–15440.

[28] Anastasiia Yevdokimova. *New mirai botnet variant detection: Moobot sample targets D-link routers*. Sept. 2022. URL: `https://socprime.com/blog/new-mirai-botnet-variant-detection-moobot-sample-targets-d-link-routers/`.

[29] Qinghua Zhang and Douglas S Reeves. "Metaaware: Identifying metamorphic malware". In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE. 2007, pp. 411–420.

[30] Ashu Sharma and Sanjay Kumar Sahay. "Evolution and detection of polymorphic and metamorphic malwares: A survey". In: *arXiv preprint arXiv:1406.7061* (2014).

[31] Alan Mills and Phil Legg. "Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques". In: *Journal of Cybersecurity and Privacy* 1.1 (2020), pp. 19–39.

[32] Oleg Boyarchuk et al. "Keeping up with the emotets: Tracking a multi-infrastructure botnet". In: *Digital Threats: Research and Practice* 4.3 (2023), pp. 1–29.

[33] Vyom Kulshreshtha, Deepak Motwani, and Pankaj Sharma. "A Study of Crypto-ransomware Using Detection Techniques for Defense Research". In: *Congress on Intelligent Systems*. Springer. 2022, pp. 127–146.

[34] Karl-Bridge-Microsoft. *PE format - win32 apps*. URL: `https://learn.microsoft.com/en-us/windows/win32/debug/pe-format`.

[35] URL: `https://wiki.osdev.org/ELF`.

[36] Solveig Badillo et al. “An introduction to machine learning”. In: *Clinical pharmacology & therapeutics* 107.4 (2020), pp. 871–885.

[37] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O’Reilly Media, Inc., 2019. ISBN: 1492032646.

[38] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. “Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation”. In: *Int. J. Intell. Technol. Appl. Stat* 11.2 (2018), pp. 105–111.

[39] Azal Ahmad Khan. “Balanced Split: A new train-test data splitting strategy for imbalanced datasets”. In: *arXiv preprint arXiv:2212.11116* (2022).

[40] Xinchuan Zeng and Tony R Martinez. “Distribution-balanced stratified cross-validation for accuracy estimation”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 12.1 (2000), pp. 1–12.

[41] Petro Liashchynskyi and Pavlo Liashchynskyi. “Grid search, random search, genetic algorithm: a big comparison for NAS”. In: *arXiv preprint arXiv:1912.06059* (2019).

[42] Michael P LaValley. “Logistic regression”. In: *Circulation* 117.18 (2008), pp. 2395–2399.

[43] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[44] Geetika Munjal, Madasu Hanmandlu, and Sangeet Srivastava. “Phylogenetics algorithms and applications”. In: *Ambient Communications and Computer Systems: RACCCS-2018*. Springer. 2019, pp. 187–194.

[45] Maurice Roux. “A comparative study of divisive and agglomerative hierarchical clustering algorithms”. In: *Journal of Classification* 35 (2018), pp. 345–366.

[46] Angur Mahmud Jarman. “Hierarchical cluster analysis: Comparison of single linkage, complete linkage, average linkage and centroid linkage method”. In: *Georgia Southern University* 29 (2020).

[47] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[48] Tim Sainburg, Leland McInnes, and Timothy Q Gentner. “Parametric UMAP embeddings for representation and semisupervised learning”. In: *Neural Computation* 33.11 (2021), pp. 2881–2907.

[49] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: `10.1109/ICEngTechnol.2017.8308186`.

[50] Asifullah Khan et al. “A survey of the recent architectures of deep convolutional neural networks”. In: *Artificial intelligence review* 53 (2020), pp. 5455–5516.

[51] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[52] S Zargar. “Introduction to sequence learning models: RNN, LSTM, GRU”. In: *Department of Mechanical and Aerospace Engineering, North Carolina State University* (2021).

[53] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *North American Chapter of the Association for Computational Linguistics*. 2019. URL: `https://api.semanticscholar.org/CorpusID:52967399`.

[54] Arvind Neelakantan et al. “Text and Code Embeddings by Contrastive Pre-Training”. In: ().

[55] Marcus Nachtigall, Lisa Nguyen Quang Do, and Eric Bodden. “Explaining Static Analysis - A Perspective”. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. 2019, pp. 29–32. DOI: `10.1109/ASEW.2019.00023`.

[56] Siti Rahayu Selamat, Thiam Tet Ng, et al. “An Automated Tool for Malware Analysis and Classification”. In: *Journal of Advanced Computing Technology and Application (JACTA)* 1.1 (2019), pp. 33–39.

[57] Nitin Naik et al. “Embedded YARA rules: strengthening YARA rules utilising fuzzy hashing and fuzzy rules for malware analysis”. In: *Complex & Intelligent Systems* 7 (2021), pp. 687–702.

[58] Mansour Ahmadi et al. “Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification”. In: Mar. 2016. DOI: `10.1145/2857705.2857713`.

[59] Mamoun Alazab et al. “Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures”. In: vol. 121. Dec. 2011.

[60] Samuel Kim. “PE header analysis for malware detection”. In: (2018).

[61] Igor Santos et al. “Opcode sequences as representation of executables for data-mining-based unknown malware detection”. In: *Information Sciences* 231 (2013). Data Mining for Information Security, pp. 64–82. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/j.ins.2011.08.020`. URL: `https://www.sciencedirect.com/science/article/pii/S0020025511004336`.

[62] Ronghua Tian, Lynn Batten, and S.C. Versteeg. “Function length as a tool for malware classification”. In: Nov. 2008, pp. 69–76. DOI: `10.1109/MALWARE.2008.4690860`.

[63] Syed Khurram Jah Rizvi et al. “PROUD-MAL: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable”. In: *Complex & Intelligent Systems* (2022), pp. 1–13.

[64] Matthew G Schultz et al. “Data mining methods for detection of new malicious executables”. In: *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE. 2000, pp. 38–49.

[65] Joshua Saxe and Konstantin Berlin. “Deep neural network based malware detection using two dimensional binary program features”. In: *2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE. 2015, pp. 11–20.

[66] Muhammad Zubair Shafiq et al. “A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables”. In: 2009. URL: `https://api.semanticscholar.org/CorpusID:17545434`.

[67] Karthik Raman et al. “Selecting features to classify malware”. In: *InfoSec Southwest* 2012 (2012), pp. 1–5.

[68] H Babbar et al. “Detection of Android Malware in the Internet of Things through the K-Nearest Neighbor Algorithm.” In: *Sensors (Basel, Switzerland)* 23.16 (2023), pp. 7256–7256.

[69] Justin Sahs and Latifur Khan. “A Machine Learning Approach to Android Malware Detection”. In: *2012 European Intelligence and Security Informatics Conference*.

[70] Ziyun Zhu and Tudor Dumitraş. “Featuresmith: Automatically engineering features for malware detection by mining the security literature”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 767–778.

[71] Guillermo Suarez-Tangil et al. “Dendroid: A text mining approach to analyzing and classifying code structures in android malware families”. In: *Expert Systems with Applications* 41.4 (2014), pp. 1104–1117.

[72] Richard M Kliman. *Encyclopedia of evolutionary biology*. Academic Press, 2016.

[73] Jin Xiong. *Essential bioinformatics*. Cambridge University Press, 2006.

[74] George Dahl et al. “Large-Scale Malware Classification Using Random Projections and Neural Networks”. In: *Proceedings IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. May 2013. URL: `https://www.microsoft.com/en-us/research/publication/large-scale-malware-classification-using-random-projections-and-neural-networks/`.

[75] Marek Krčál et al. “Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only”. In: (2018).

[76] Edward Raff et al. “Malware Detection by Eating a Whole EXE”. In: *stat* 1050 (2017), p. 25.

[77] Sang Ni, Quan Qian, and Rui Zhang. “Malware identification using visualization images and deep learning”. In: (2018).

[78] Lakshmanan Nataraj et al. “Malware images: visualization and automatic classification”. In: *Proceedings of the 8th international symposium on visualization for cyber security*. 2011, pp. 1–7.

[79] Kesav Kancherla and Srinivas Mukkamala. “Image visualization based malware detection”. In: *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. IEEE. 2013, pp. 40–44.

[80] Sharmila Gaikwad and Jignesh Patil. “Malware Detection in Deep Learning”. In: *Convergence of Deep Learning In Cyber-IoT Systems and Security* (2022), pp. 269–284.

[81] Sushil Kumar et al. “MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things”. In: *Future Generation Computer Systems* 125 (2021), pp. 334–351.

[82] Kyoung Soo Han et al. “Malware analysis using visualized images and entropy graphs”. In: *International Journal of Information Security* 14 (2015), pp. 1–14.

[83] Fikirte Ayalke Demmese et al. “Machine learning based fileless malware traffic classification using image visualization”. In: *Cybersecurity* 6.1 (2023), p. 32.

[84] Ahmet Selman Bozkir, Ahmet Ogulcan Cankaya, and Murat Aydos. “Utilization and comparision of convolutional neural networks in malware recognition”. In: *2019 27th signal processing and communications applications conference (SIU)*. IEEE. 2019, pp. 1–4.

[85] Duc-Ly Vu et al. “A Convolutional Transformation Network for Malware Classification”. In: *arXiv preprint arXiv:1909.07227* (2019).

[86] Ke He and Dong-Seong Kim. “Malware detection with malware images using deep learning techniques”. In: *2019 18th IEEE international conference on trust, security and privacy in computing and communications/13th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE. 2019, pp. 95–102.

[87] Anh-Duy Tran et al. “Os-independent malware detection: Applying machine learning and computer vision in memory forensics”. In: *2021 17th International Conference on Computational Intelligence and Security (CIS)*. IEEE. 2021, pp. 616–620.

[88] Islam Obaidat et al. “Jadeite: A novel image-behavior-based approach for java malware detection using deep learning”. In: *Computers & Security* 113 (2022), p. 102547.

[89] Shruti Patil et al. “Improving the robustness of ai-based malware detection using adversarial machine learning”. In: *Algorithms* 14.10 (2021), p. 297.

[90] Mohd Zamri Osman et al. “Pixel-based feature for android malware family classification using machine learning algorithms”. In: *2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)*. IEEE. 2021, pp. 552–555.

[91] ESultanik. *Esultanik/bin2png: A simple cross-platform script for encoding any binary file into a lossless PNG.* URL: `https://github.com/ESultanik/bin2png?tab=readme-ov-file`.

[92] Jongkwan Lee and Jongdeog Lee. “A Classification System for Visualized Malware Based on Multiple Autoencoder Models”. In: *IEEE Access* 9 (2021), pp. 144786–144795. DOI: `10.1109/ACCESS.2021.3122083`.

[93] Tran The Son et al. “An enhancement for image-based malware classification using machine learning with low dimension normalized input images”. In: *Journal of Information Security and Applications* 69 (2022), p. 103308.

[94] Hyun-Jong Cha et al. “Intelligent Anomaly Detection System through Malware Image Augmentation in IIoT Environment Based on Digital Twin”. In: *Applied Sciences* 13.18 (2023), p. 10196.

[95] Sara Sartoli, Yong Wei, and Shane Hampton. “Malware classification using recurrence plots and deep neural network”. In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2020, pp. 901–906.

[96] Jueun Jeon, Jong Hyuk Park, and Young-Sik Jeong. “Dynamic Analysis for IoT Malware Detection With Convolution Neural Network Model”. In: *IEEE Access* 8 (2020), pp. 96899–96911. DOI: `10.1109/ACCESS.2020.2995887`.

[97] Fangtian Zhong et al. “Malware-on-the-brain: Illuminating malware byte codes with images for malware classification”. In: *IEEE Transactions on Computers* 72.2 (2022), pp. 438–451.

[98] MDHU Sharif et al. “A deep learning based technique for the classification of malware images”. In: *Journal of Theoretical and Applied Information Technology* 101.1 (2023), pp. 135–160.

[99] S Abijah Roseline et al. “Intelligent vision-based malware detection and classification using deep random forest paradigm”. In: *IEEE Access* 8 (2020), pp. 206303–206324.

[100] Ahmet Selman Bozkir et al. “Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision”. In: *Computers & Security* 103 (2021), p. 102166.

[101] Luxin Zheng, Jian Zhang, et al. “A new malware detection method based on vmcadr in cloud environments”. In: *Security and Communication Networks* 2022 (2022).

[102] Abir Laouadi, Djamel Eddine Menacer, and Karima Benatchba. “A Machine Learning Approach for Malware Detection based on Image Conversion”. In: (2024).

[103] Daniel Gibert et al. “Using convolutional neural networks for classification of malware represented as images”. In: *Journal of Computer Virology and Hacking Techniques* 15 (2019), pp. 15–28.

[104] Hiromu Yakura et al. “Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism”. In: Nov. 2017, pp. 55–56. DOI: `10.1145/3128572.3140457`.

[105] Sanjeev Kumar and B. Janet. “DTMIC: Deep transfer learning for malware image classification”. In: *Journal of Information Security and Applications* 64 (2022), p. 103063. ISSN: 2214-2126. DOI: `https://doi.org/10.1016/j.jisa.2021.103063`. URL: `https://www.sciencedirect.com/science/article/pii/S2214212621002465`.

[106] Furqan Rustam et al. “Malware detection using image representation of malware data and transfer learning”. In: *Journal of Parallel and Distributed Computing* 172 (2023), pp. 32–50.

[107] Niket Bhodia et al. “Transfer learning for image-based malware classification”. In: *arXiv preprint arXiv:1903.11551* (2019).

[108] Michael Potuck. *Malware threat report reveals risk on Mac compared to windows and linux*. Apr. 2023. URL: `https://9to5mac.com/2023/04/27/malware-threat-report-mac-risk-vs-windows-and-linux/#:~:text=However%2C%20when%20looking%20at%20just,1%25%20being%20found%20on%20macOS.`.

[109] Abedelaziz Mohaisen and Omar Alrawi. “Unveiling zeus: automated classification of malware samples”. In: *Proceedings of the 22nd International Conference on World Wide Web*. 2013, pp. 829–832.

[110] Hisham Shehata Galal, Yousef Bassyouni Mahdy, and Mohammed Ali Atiea. “Behavior-based features model for malware detection”. In: *Journal of Computer Virology and Hacking Techniques* 12 (2016), pp. 59–67.

[111] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. “A novel approach to detect malware based on API call sequence analysis”. In: *International Journal of Distributed Sensor Networks* 11.6 (2015), p. 659101.

[112] Guanghui Liang, Jianmin Pang, and Chao Dai. “A behavior-based malware variant classification technique”. In: *International Journal of Information and Education Technology* 6.4 (2016), p. 291.

[113] Syed Zainudeen Mohd Shaid and Mohd Aizaini Maarof. “Malware behaviour visualization”. In: *Jurnal Teknologi* 70.5 (2014).

[114] Philipp Trinius et al. “Visual analysis of malware behavior using treemaps and thread graphs”. In: *2009 6th International Workshop on Visualization for Cyber Security*. IEEE. 2009, pp. 33–38.

[115] Matilda Rhode, Pete Burnap, and Kevin Jones. “Early-stage malware prediction using recurrent neural networks”. In: *computers & security* 77 (2018), pp. 578–594.

[116] Sumith Maniath et al. “Deep learning LSTM based ransomware detection”. In: *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*. IEEE. 2017, pp. 442–446.

[117] Omid E David and Nathan S Netanyahu. “Deepsign: Deep learning for automatic malware signature generation and classification”. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[118] William Hardy et al. “DL4MD: A deep learning framework for intelligent malware detection”. In: *Proceedings of the International Conference on Data Science (ICDATA)*. The Steering Committee of The World Congress in Computer Science, Computer … 2016, p. 61.

[119] URL: https://oem.avira.com/en/technology/machine-learning.

[120] URL: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[121] Microsoft Defender Security Research Team. *Seeing the big picture: Deep Learning-based fusion of behavior signals for threat detection*. May 2023. URL: https://www.microsoft.com/en-us/security/blog/2020/07/23/seeing-the-big-picture-deep-learning-based-fusion-of-behavior-signals-for-threat-detection/.

[122] Sophos Ltd. *Deal with malware detected by Deep Learning*. Jan. 2024. URL: https://docs.sophos.com/central/customer/help/en-us/ManageYourProducts/Alerts/ThreatAdvice/MalwareAdviceDeepLearning/index.html.

[123] Oct. 2023. URL: https://www.malwarebytes.com/blog/detections/malware.

[124] Azqa Nadeem et al. “Intelligent malware defenses”. In: *Security and artificial intelligence: A crossdisciplinary approach*. Springer, 2022, pp. 217–253.

[125] URL: https://www.stratosphereips.org/.

[126] URL: https://www.kaggle.com/c/malware-classification.

[127] Robert J. Joyce et al. “MOTIF: A Malware Reference Dataset with Ground Truth Family Labels”. In: *Computers Security* 124 (2023), p. 102921. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102921. URL: https://www.sciencedirect.com/science/article/pii/S0167404822003133.

[128] Hyrum S Anderson and Phil Roth. “Ember: an open dataset for training static pe malware machine learning models”. In: *arXiv preprint arXiv:1804.04637* (2018).

[129] fabrimagic72. *Fabrimagic72/malware-samples: A collection of malware samples caught by several honeypots I manage*. URL: https://github.com/fabrimagic72/malware-samples.

[130] InQuest. *Malware-samples/miscellaneous at master · inquest/malware-samples*. URL: https://github.com/InQuest/malware-samples/tree/master/miscellaneous.

[131] Jstrosch. *Malware-samples/binaries at master · JSTROSCH/malware-samples*. URL: https://github.com/jstrosch/malware-samples/tree/master/binaries.

[132] URL: https://bazaar.abuse.ch/.

[133] URL: https://vx-underground.org/.

[134] Mahmoud Kalash et al. “Malware classification with deep convolutional neural networks”. In: *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE. 2018, pp. 1–5.

[135] Musaad Darwish AlGarni et al. “An efficient convolutional neural network with transfer learning for malware classification”. In: *Wireless Communications and Mobile Computing* 2022 (2022), pp. 1–8.

[136] Ömer Aslan and Abdullah Asim Yilmaz. “A new malware classification framework based on deep learning algorithms”. In: *Ieee Access* 9 (2021), pp. 87936–87951.

[137] Aziz Makandar and Anita Patrot. “Detection and retrieval of malware using classification”. In: *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. IEEE. 2017, pp. 1–5.

[138] Wei Kitt Wong, Filbert H Juwono, and Catur Apriono. “Vision-based malware detection: A transfer learning approach using optimal ecoc-svm configuration”. In: *Ieee Access* 9 (2021), pp. 159262–159270.

[139] Safa Ben Atitallah, Maha Driss, and Iman Almomani. “A novel detection and multi-classification approach for IoT-malware using random forest voting of fine-tuning convolutional neural networks”. In: *Sensors* 22.11 (2022), p. 4302.

[140] Ismail Taha Ahmed et al. “Binary and multi-class malware threads classification”. In: *Applied Sciences* 12.24 (2022), p. 12528.

[141] Scott Freitas, Rahul Duggal, and Duen Horng Chau. “MalNet: A large-scale image database of malicious software”. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2022, pp. 3948–3952.

[142] Pratyush Panda et al. “Transfer learning for image-based malware detection for iot”. In: *Sensors* 23.6 (2023), p. 3253.

[143] Mesut GUVEN. “Leveraging deep learning and image conversion of executable files for effective malware detection: A static malware analysis approach”. In: *AIMS Mathematics* 9.6 (2024), pp. 15223–15245.

[144] P Vinod et al. “MOMENTUM: MetamOrphic malware exploration techniques using MSA signatures”. In: *2012 International Conference on Innovations in Information Technology (IIT)*. IEEE. 2012, pp. 232–237.

[145] David M Hillis, John P Huelsenbeck, and Clifford W Cunningham. “Application and accuracy of molecular phylogenies”. In: *Science* 264.5159 (1994), pp. 671–677.

[146] Tri Andriani and Mohammad Isa Irawan. “Application of unweighted pair group methods with arithmetic average (UPGMA) for identification of kinship types and spreading of ebola virus through establishment of phylogenetic tree”. In: *AIP Conference Proceedings*. Vol. 1867. 1. AIP Publishing. 2017.

[147] Andrei A Zimin, Alexandra N Karmanova, and Yinhua Lu. “UPGMA-analysis of type II CRISPR RNA-guided endonuclease Cas9 homologues from the compost metagenome”. In: *E3S web of conferences*. Vol. 265. EDP Sciences. 2021, p. 04010.

[148] Xiaomeng Wu et al. “Phylogenetic analysis using complete signature information of whole genomes and clustered Neighbour-Joining method”. In: *International journal of bioinformatics research and applications* 2.3 (2006), pp. 219–248.

[149] Peter M Hollingsworth and RA Ennos. “Neighbour joining trees, dominant markers and population genetic structure”. In: *Heredity* 92.6 (2004), pp. 490–498.

[150] Martin Simonsen, Thomas Mailund, and Christian NS Pedersen. “Inference of large phylogenies using neighbour-joining”. In: *Biomedical Engineering Systems and Technologies: Third International Joint Conference, BIOSTEC 2010, Valencia, Spain, January 20-23, 2010, Revised Selected Papers 3*. Springer. 2011, pp. 334–344.

[151] Kenneth W Cullings and Detlev R Vogler. “A 5.8 S nuclear ribosomal RNA gene sequence database: applications to ecology and evolution”. In: *Molecular Ecology* 7.7 (1998), pp. 919–923.

[152] Irena Vardić Smrzlić et al. “Molecular characterisation of Anisakidae larvae from fish in Adriatic Sea”. In: *Parasitology research* 111 (2012), pp. 2385–2391.

[153] M Thangaraj, B Valentin Bhimba, and J Meenupriya. “Phylogenetic relationship in four Aspergillus species based on the secondary structure of internal transcribed spacer region of rDNA”. In: *Journal of Advanced Bioinformatics Applications and Research ISSN* 2.3 (2011), pp. 200–205.

[154] Inaki Comas et al. “Genotyping of genetically monomorphic bacteria: DNA sequencing in Mycobacterium tuberculosis highlights the limitations of current methodologies”. In: *PloS one* 4.11 (2009), e7815.

[155] Tonny Kinene et al. “Rooting trees, methods for”. In: *Encyclopedia of Evolutionary Biology* (2016), p. 489.

[156] Sean W Graham, Richard G Olmstead, and Spencer CH Barrett. “Rooting phylogenetic trees with distant outgroups: a case study from the commelinoid monocots”. In: *Molecular biology and evolution* 19.10 (2002), pp. 1769–1781.

[157] Emanuele Cozzi et al. “The tangled genealogy of IoT malware”. In: *Proceedings of the 36th Annual Computer Security Applications Conference*. 2020, pp. 1–16.

[158] Tianxiang He et al. “A Fast Algorithm for Constructing Phylogenetic Trees with Application to IoT Malware Clustering”. In: Dec. 2019, pp. 766–778. DOI: `10.1007/978-3-030-36708-4_63`.

[159] François-Joseph Lapointe, John AW Kirsch, and Robert Bleiweiss. “Jackknifing of weighted trees: validation of phylogenies reconstructed from distance matrices”. In: *Molecular Phylogenetics and Evolution* 3.3 (1994), pp. 256–267.

[160] David R Roberts et al. “Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure”. In: *Ecography* 40.8 (2017), pp. 913–929.

[161] V Berry, O Gascuel, and G Caraux. “Choosing the tree which actually best explains the data: another look at the bootstrap in phylogenetic reconstruction”. In: *Computational Statistics & Data Analysis* 32.3-4 (2000), pp. 273–283.

[162] Md Rashidul Hasan and James Degnan. “Testing Tree-Likeness of Phylogenetic Network Data with Cross-Validation”. In: (2023).

[163] Sebastián Duchêne et al. “Cross-validation to select Bayesian hierarchical models in phylogenetics”. In: *BMC evolutionary biology* 16 (2016), pp. 1–8.

[164] Michael M Miyamoto and Walter M Fitch. “Testing species phylogenies and phylogenetic methods with congruence”. In: *Systematic Biology* 44.1 (1995), pp. 64–76.

[165] François-Joseph Lapointe and Leslie J Rissler. “Congruence, consensus, and the comparative phylogeography of codistributed species in California”. In: *The American Naturalist* 166.2 (2005), pp. 290–299.

[166] Bryan C Carstens et al. “Testing nested phylogenetic and phylogeographic hypotheses in the Plethodon vandykei species group”. In: *Systematic biology* 53.5 (2004), pp. 781–792.

[167] Aylwyn Scally and Richard Durbin. “Revising the human mutation rate: implications for understanding human evolution”. In: *Nature Reviews Genetics* 13.10 (2012), pp. 745–753.

[168] Michael Worobey et al. “Direct evidence of extensive diversity of HIV-1 in Kinshasa by 1960”. In: *Nature* 455.7213 (2008), pp. 661–664.

[169] Sebastian Duchene et al. “Temporal signal and the phylodynamic threshold of SARS-CoV-2”. In: *Virus evolution* 6.2 (2020), veaa061.

[170] Roy Halevi. *What is genetic malware analysis?* May 2022. URL: `https://intezer.com/blog/malware-analysis/defining-genetic-malware-analysis/`.

[171] Stephanie Wehner. “Analyzing worms and network traffic using compression”. In: *Journal of Computer Security* 15.3 (2007), pp. 303–320.

[172] URL: `https://openai.com/index/new-embedding-models-and-api-updates`.

[173] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists.* ” O’Reilly Media, Inc.”, 2018.

[174] Zichao Yang et al. “Hierarchical attention networks for document classification”. In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489.

[175] Amir Zadeh et al. “Tensor fusion network for multimodal sentiment analysis”. In: *arXiv preprint arXiv:1707.07250* (2017).

[176] Damien M. de Vienne, Gabriela Aguileta, and Sébastien Ollier. “Euclidean Nature of Phylogenetic Distance Matrices”. In: *Systematic Biology* 60.6 (July 2011), pp. 826–832. ISSN: 1063-5157. DOI: `10.1093/sysbio/syr066`. eprint: `https://academic.oup.com/sysbio/article-pdf/60/6/826/24559644/syr066.pdf`. URL: `https://doi.org/10.1093/sysbio/syr066`.

[177] Ahmed Mansour. “Phylip and phylogenetics”. In: *Focus on Bioinformatics: Genes, Genomes and Genomics* 3 (2009), pp. 46–49.

[178] URL: `https://mothur.org/wiki/phylip-formatted_distance_matrix/`.

[179] Ravi Kumar Yadav Dega and Gunes Ercal. "A comparative analysis of progressive multiple sequence alignment approaches using UPGMA and neighbor joining based guide trees". In: *arXiv preprint arXiv:1509.03530* (2015).

[180] Martin Simonsen, Thomas Mailund, and Christian NS Pedersen. "Rapid neighbour-joining". In: *Algorithms in Bioinformatics: 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings 8*. Springer. 2008, pp. 113–122.

[181] URL: `https://success.trendmicro.com/dcx/s/solution/1118407-emerging-threat-on-fareit-resurgence?language=en_US`.

[182] Joseph Felsenstein. "Inferring phylogenies". In: *Inferring phylogenies*. 2004, pp. 664–664.

[183] Roderick DM Page and Edward C Holmes. *Molecular evolution: a phylogenetic approach*. John Wiley & Sons, 2009.

[184] John Wilder Tukey et al. *Exploratory data analysis*. Vol. 2. Springer, 1977.

[185] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[186] Oliver Buxton. Mar. 2024. URL: `https://www.avast.com/c-mirai`.

[187] Guest Author et al. *Inside the infamous Mirai IOT Botnet: A retrospective analysis*. Sept. 2021. URL: `https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis`.

[188] URL: `https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/patch-now-new-mirai-gafgyt-variants-target-16-flaws-via-multi-exploits`.

[189] URL: `https://www.cyber.nj.gov/alerts-advisories/new-mirai-and-gafgyt-botnet-variants-target-apache-struts-and-sonicwall-gms`.

[190] Author:Tara Seals and Tara Seals. *Gafgyt botnet lifts ddos tricks from Mirai*. URL: `https://threatpost.com/gafgyt-botnet-ddos-mirai/165424/`.

[191] ByIonut Arghire. *Mirai variant targets arc CPU-based devices*. Jan. 2018. URL: `https://www.securityweek.com/mirai-variant-targets-arc-cpu-based-devices/#:~:text=January%2016%2C%202018-,A%20newly%20discovered%20variant%20of%20the%20Mirai%20Internet%https://www.securityweek.com/mirai-variant-targets-arc-cpu-based-devices/#:~:text=January%2016%2C%202018-,A%20newly%20discovered%20variant%20of%20the%20Mirai%20Internet%20of%20Things,called%20Okiru%20by%20its%20author.20of%20Things,called%20Okiru%20by%20its%20author.`.

[192] Nick Lewis. *Okiru malware: How does this mirai malware variant work?: TechTarget*. Aug. 2018. URL: `https://www.techtarget.com/searchsecurity/answer/Okiru-malware-How-does-this-Mirai-malware-variant-work#:~:text=MalwareMustDie%20reported%20that%20the%20Okiru,these%20devices%20are%20being%20targeted.`.

[193] Sept. 2022. URL: `https://thehackernews.com/2022/09/mirai-variant-moobot-botnet-exploiting.html`.

[194] Anastasiia Yevdokimova. *New mirai botnet variant detection: Moobot sample targets D-link routers*. Sept. 2022. URL: `https://socprime.com/blog/new-mirai-botnet-variant-detection-moobot-sample-targets-d-link-routers/`.

[195] Inc. Stamus Networks. *Threat detection update 27-september-2022: Stamus Networks*. URL: `https://www.stamus-networks.com/stamus-labs/detection-update-2022-09-27`.

[196] Aziz Farghly. *Smoke loader analysis*. Mar. 2024. URL: `https://medium.com/@farghly.mahmod66/smoke-loader-analysis-1f1442809802`.

[197] Abdallah Elshinbary. *Deep analysis of smokeloader*. June 2020. URL: `https://n1ght-w0lf.github.io/malware%20analysis/smokeloader/`.

[198] Jason Reaves. *IcedID leverages PrivateLoader*. Aug. 2022. URL: `https://medium.com/walmartglobaltech/icedid-leverages-privateloader-7744771bf87f`.

[199] VMRay Labs. *Malware analysis spotlight: Smoke loader*. Feb. 2024. URL: `https://www.vmray.com/malware-analysis-spotlight-smoke-loader/`.

[200] JouniMi. *Asyncrat - threat hunting with hints of incident response*. Jan. 2023. URL: `https://threathunt.blog/asyncrat/`.

[201] Adrien Bibal et al. "Is attention explanation? an introduction to the debate". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 3889–3900.

[202] Sarah Wiegreffe and Yuval Pinter. "Attention is not not explanation". In: *arXiv preprint arXiv:1908.04626* (2019).

[203] Christopher Grimsley, Elijah Mayfield, and Julia Bursten. "Why attention is not explanation: Surgical intervention and causal reasoning about neural models". In: (2020).

[204] Bingcai Chen et al. "Adversarial examples for cnn-based malware detectors". In: *IEEE Access* 7 (2019), pp. 54360–54371.

[205] Anish Shah et al. "Deep residual networks with exponential linear unit". In: *Proceedings of the third international symposium on computer vision and the internet*. 2016, pp. 59–65.

[206] Aleieldin Salem, Sebastian Banescu, and Alexander Pretschner. "Maat: Automatically analyzing virustotal for accurate labeling and effective malware detection". In: *ACM Transactions on Privacy and Security (TOPS)* 24.4 (2021), pp. 1–35.

[207] Tianxiang He et al. "Scalable and fast algorithm for constructing phylogenetic trees with application to IoT malware clustering". In: *IEEE Access* 11 (2023), pp. 8240–8253.

[208] Andy Young. *Here we botenago again!* URL: `https://www.keysight.com/blogs/en/tech/nwvs/2022/02/01/here-we-botenago-again`.

[209] Princy Victor et al. "IoT malware: An attribute-based taxonomy, detection mechanisms and challenges". In: *Peer-to-peer Networking and Applications* 16.3 (2023), pp. 1380–1431.

[210] Anna Szalay Sean Gallagher. *The 2024 sophos threat report: Cybercrime on main street*. Mar. 2024. URL: `https://news.sophos.com/en-us/2024/03/12/2024-sophos-threat-report/`.

[211] Furkan Ozturk. *Akira: Undetectable stealer unleashed*. Oct. 2023. URL: `https://threatmon.io/blog/akira-undetectable-stealer-unleashed/`.

[212] Nick Biasini. *Sundown Ek: You better take care*. Aug. 2022. URL: `https://blog.talosintelligence.com/sundown-ek/`.

[213] Malwarebytes Labs. *Inside the kronos malware - part 1: Malwarebytes labs*. Aug. 2017. URL: `https://www.malwarebytes.com/blog/news/2017/08/inside-kronos-malware`.

A
Types of Malware

Table A.1: Comprehensive Classification of Malware Based on Evasion Techniques

Type	Malware	Description
Polymorphic	Virlock	This ransomware not only encrypts files but also infects them, changing its code with each infection to evade detection.
Polymorphic	Storm Worm	Known for its widespread email campaign, this trojan horse employs polymorphic techniques to create numerous variants, complicating its detection and removal.
Polymorphic	Mariposa Botnet	Originating from Butterfly Bot, it steals financial and sensitive data, leveraging polymorphic capabilities to avoid detection.
Polymorphic	XOR.DDoS	Targeting Linux systems for DDoS attacks, XOR.DDoS uses polymorphism to evade traditional detection methods.
Polymorphic	Mirai	Known for its devastating DDoS attacks, Mirai's polymorphic variants complicate the detection process, leading to increased infection rates across IoT devices.
Polymorphic	Gafgyt	Specializes in launching DDoS attacks, continuously changing attack vectors and evolving encryption, making it hard to trace and neutralize.
Polymorphic	MooBot	As a direct offshoot of Mirai, exploits vulnerabilities quickly, adapting its scanning and exploitation techniques, infecting a wide array of devices.
Metamorphic	ZMist (ZMistfall)	Integrates into and modifies executable files using advanced metamorphic techniques to rearrange and rewrite its code.
Metamorphic	Win95/Regswap	Targets Windows 95/98 files, swaps code segments to obscure its presence, making detection challenging.
Advanced Evasive	IcedID	A banking Trojan that employs injection techniques and evasion tactics to avoid detection, spreading via malspam campaigns.
Advanced Evasive	Emotet	Initially a banking Trojan, now a sophisticated malware delivery service, known for its rapid spread and delivery of various payloads.
Advanced Evasive	GandCrab	A ransomware-as-a-service that evolved through continuous updates, utilizing both polymorphic and metamorphic techniques.

Continued on next page

Table A.1 continued from previous page

Type	Malware	Description
Advanced Evasive	Petya/NotPetya	Initially a ransomware, NotPetya was later revealed to be a wiper malware disguised as ransomware. It leveraged sophisticated spreading mechanisms, including the use of EternalBlue and additional credentials stealing techniques, to infect and spread across networks rapidly.
Advanced Evasive	Ryuk	A ransomware that typically targets enterprises, Ryuk is known for its use after initial access has been gained via other malware, such as TrickBot or Emotet. It demonstrates advanced evasion techniques by disabling security tools and encrypting network devices to maximize damage.
Advanced Evasive	Locky	A ransomware that became notorious for its widespread distribution via email campaigns and its use of polymorphic encryption techniques to evade detection.
Advanced Evasive	Dridex	Known for targeting financial information, Dridex is a banking Trojan that uses advanced evasion techniques, including polymorphic and macro-laced documents, to infect systems.
Advanced Evasive	WannaCry	Notorious for its global ransomware outbreak, WannaCry incorporates evasive maneuvers by exploiting network vulnerabilities for propagation and employing techniques that hinder static and dynamic analysis.
Advanced Evasive	Agent Tesla	An advanced spyware that exemplifies evasive malware through its polymorphic encryption mechanisms, enabling it to stealthily harvest and exfiltrate sensitive information while evading standard detection methods.

B

Executable Formats

B.1. PE Format

In this section, we highlight the composition of the entire PE file, encompassing the DOS Header, DOS Stub, PE Header proper, and the Optional Header. Each component plays a pivotal role in defining the executable file's structure, behavior, and compatibility within the Windows operating system's ecosystem as illustrated in figure B.1.

DOS Header

- **Signature ('MZ')**: Identifies the file as an executable.
- **Timestamp**: Marks when the file was created (often not used).
- **Pointer to PE Header**: Directs to the PE header.

DOS Stub

- Used when the executable is run under DOS.
- Displays a message that the program cannot run in DOS mode.

PE Header

- **Signature**: `PE\0\0` Identifies the start of the PE header.
- **Machine**: Indicates the architecture the executable is intended for.
- **Number of Sections**: Specifies how many sections are in the file.
- **TimeDateStamp**: The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a Unix timestamp), that indicates when the file was created.
- **Pointer to Symbol Table**: Used for debugging.
- **Number of Symbols**: Used with the pointer to the symbol table.
- **Size of Optional Header**: Specifies the size of the optional header.
- **Characteristics**: Indicates the characteristics of the file.

Optional Header

- **Magic**: A signature that identifies the optional header format.
- **Linker Version**: Version of the linker that produced the file.
- **Size of Code**: Size of the code (text) section, or the sum of all such sections if there are multiple text sections.
- **Size of Initialized Data**: Size of the initialized data section, or the sum of all such sections if there are multiple data sections.
- **Size of Uninitialized Data**: Size of the uninitialized data section.
- **Address of Entry Point**: Pointer to the entry point function, relative to the image base when the executable file is loaded into memory.

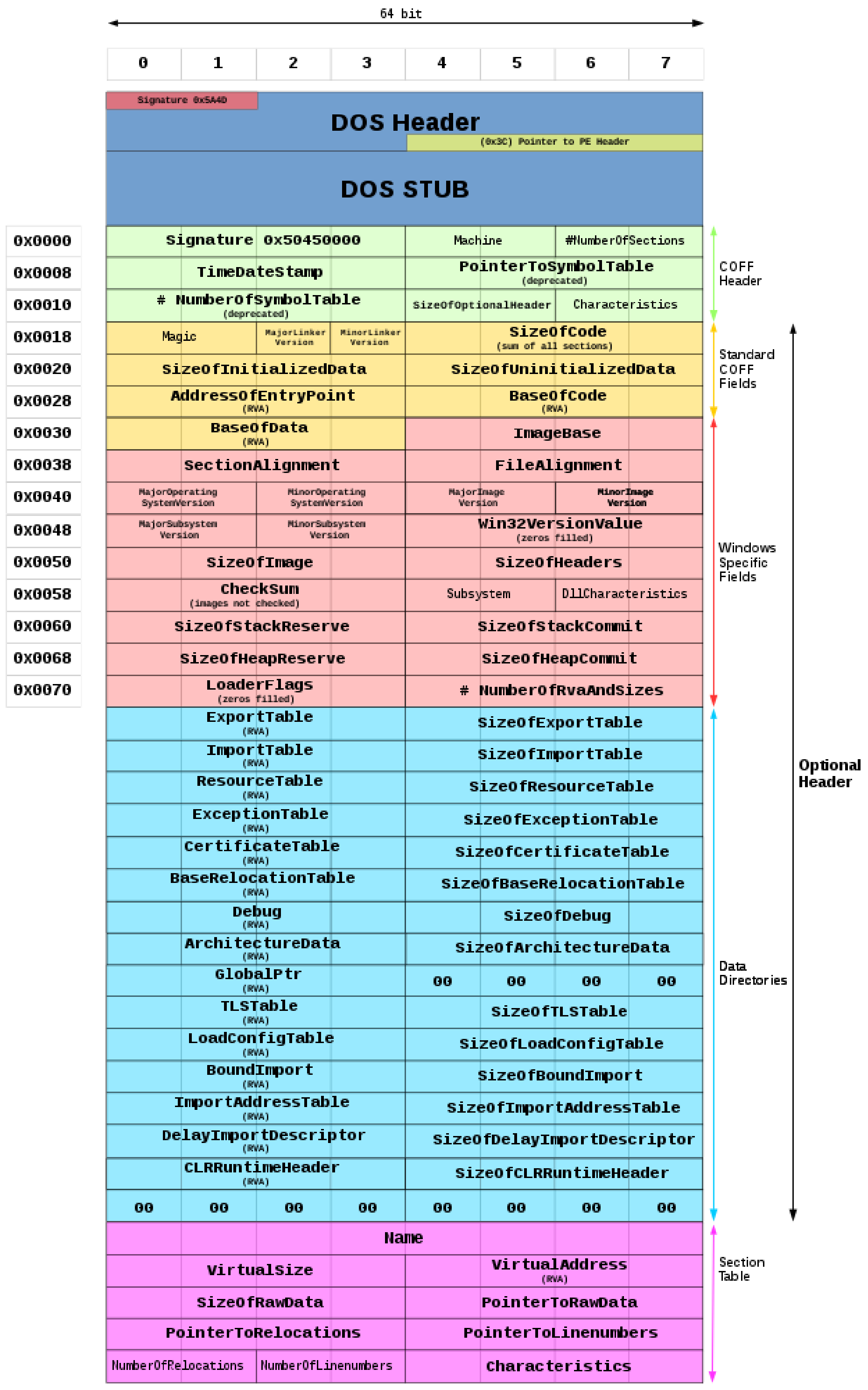


Figure B.1: PE Format

- **Base of Code**: Pointer to the beginning of the code section, relative to the image base.
- **Image Base**: Preferred address of the first byte of image when loaded into memory; must be a multiple of 64K.
- **Section Alignment**: The alignment (in bytes) of sections when they are loaded into memory. It must be greater than or equal to FileAlignment.
- **File Alignment**: The alignment factor (in bytes) that is used to align the raw data of sections in the image file.
- **Size of Image**: The size (in bytes) of the image, including all headers, as the image is loaded in memory. It must be a multiple of SectionAlignment.
- **Size of Headers**: The combined size of the MS-DOS stub, PE header, and section headers rounded up to a multiple of FileAlignment.
- **CheckSum**: The image file checksum.
- **Subsystem**: The subsystem that is required to run this image.
- **DLL Characteristics**: The flags that indicate the attributes of the DLL.
- **Size of Stack Reserve**: The size of the stack to reserve. Only SizeOfStackCommit is committed; the rest is made available one page at a time until the reserve size is reached.
- **Size of Stack Commit**: The size of the stack to commit.
- **Size of Heap Reserve**: The size of the local heap space to reserve.
- **Size of Heap Commit**: The size of the local heap space to commit.
- **Loader Flags**: The flags that indicate the attributes of the loading process.
- **Number of RVA and Sizes**: The number of data-directory entries in the remainder of the Optional Header. Each describes a location and size.

Data Directories

Data Directories provide essential configuration for the executable and are located in the Optional Header. Each directory entry points to a table or string that is crucial for the execution of the PE file. The standard Data Directories are:

- **Export Table**: Contains information about functions and variables that are available for use by other executables or dynamic-link libraries (DLLs).
- **Import Table**: Holds a list of functions and variables that the executable imports from other DLLs, which are necessary for the dynamic linking process.
- **Resource Table**: Aggregates the resource information, such as icons, menus, and dialog boxes, which are used by the executable's user interface.
- **Exception Table**: Stores the exception handling and unwinding information, used primarily in structured exception handling.
- **Certificate Table**: Contains the certificates used for authenticating the executable.
- **Base Relocation Table**: Holds the relocation information if the executable cannot be loaded at its preferred base address.
- **Debug**: Provides debug information, which is not necessary for the executable to run, but is useful during development and troubleshooting.
- **Architecture Data**: Reserved for architecture-specific data and generally not used.
- **Global Pointer**: Stores the value of the global pointer register.
- **Thread Local Storage (TLS) Table**: Contains information about thread-local storage, which is used to create thread-specific data.
- **Load Configuration Table**: Includes configuration settings related to the loading of the executable, such as security checks.
- **Bound Import**: Contains information about bound imports, which are used to speed up the loading process of the executable.
- **Import Address Table (IAT)**: Used by the loader to resolve the addresses of imported functions at runtime.

- **Delay Import Descriptor**: Holds information about DLLs that are not loaded during startup but on demand during runtime.
- **CLR Runtime Header**: Contains information relevant to the Common Language Runtime (CLR) for executables that require the .NET framework to run.
- **Reserved**: Reserved for future use.

Microsoft COFF

The Microsoft Common Object File Format (COFF) is an adaptation of the traditional COFF format, which is extensively utilized in Unix systems. It has been customized to meet the requirements of Windows operating systems for structuring binary executable files, object code, and Dynamic Link Libraries (DLLs). Below are the key features and enhancements that Microsoft COFF brings to the Windows environment:

- **Structure**: Maintains a standard layout for binary files, which includes a header followed by sections like `.text` (executable code), `.data` (initialized data), and `.bss` (uninitialized data allocated at runtime).
- **Extended Headers**: Incorporates additional headers, notably the Image Header, which contains Windows-specific flags, addresses, and sizes, supporting features unique to Windows.
- **Section Alignment**: Allows for specific alignments of sections in memory, which may differ from their alignments on disk. This feature improves access speed and efficiency in the Windows operating environment.
- **Debugging Information**: Rich debugging information is included, formatted to integrate seamlessly with Microsoft's Visual Studio development environment. This includes enhanced symbol tables and file line number information critical during the linking phase and useful for debuggers to provide source-level debugging.
- **Compatibility**: Ensures compatibility with Microsoft Visual C++ linker and loader tools, which is essential for seamless development and execution within the Windows ecosystem.
- **Linker and Loader Support**: Specifically designed to optimize the efficiency of the Windows linker and loader, supporting features like incremental linking and fast loading, which are beneficial for managing large software projects.
- **Symbol Table Management**: Features a detailed symbol table that describes function and variable names along with their attributes in an optimized manner for quick lookups during the linking stage of program compilation.

This tailored COFF version by Microsoft provides a robust framework for application development and execution within Windows environments, reflecting a significant evolution from its Unix origins to cater specifically to the complexities of modern Windows systems.

Section Headers and Sections

The section headers follow the PE header and describe the characteristics and locations of the data and code within the file. Each section can contain code, data, or both and has attributes like readable, writable, and executable. Each section header contains:

- **Name**: An 8-byte, null-padded UTF-8 encoded string. If the string is exactly 8 characters long, there is no terminating null. For longer names, this field contains a slash (/) followed by an ASCII representation of a decimal number that is an offset into the string table. Executable images do not use a string table and do not support section names longer than 8 characters.
- **Virtual Size**: The total size of the section when loaded into memory.
- **Virtual Address**: For executable images, the address of the first byte of the section relative to the image base when the section is loaded into memory.
- **Size of Raw Data**: The size of the section (for object files) or the size of the initialized data on disk (for images).
- **Pointer to Raw Data**: The file pointer to the first page of the section within the COFF file. For executable images, this is the address of the first byte before relocation is applied; for object files, this is the address of the first byte after relocation is applied.

- **Pointer to Relocations**: The file pointer to the beginning of relocation entries for the section. This is set to zero for executable images.
- **Pointer to Linenumbers**: The file pointer to the beginning of line-number entries for the section.
- **Number of Relocations**: The number of relocation entries for the section. This is set to zero for executable images.
- **Number of Linenumbers**: The number of line-number entries for the section.
- **Characteristics**: The flags that describe the characteristics of the section.

The actual sections of the executable, which contain code (.text), initialized data (.data), uninitialized data (.bss), and other data, are structured as follows:

- **.text**: Contains the executable instructions.
- **.data**: Stores initialized data, including global variables.
- **.rdata**: Contains read-only data, such as import and export directory tables.
- **.bss**: Holds uninitialized data that is zeroed when the program starts.
- **.idata**: Includes import function tables and import lookup tables.
- **.edata**: Holds the export function tables.
- **.rsrc**: Contains resource data, such as icons, cursors, and menus.
- **.reloc**: Provides relocation information.

Note: While the .rdata section typically contains the import and export directory tables, these are logically part of the Data Directories within the Optional Header. The sections themselves are allocated and aligned in memory based on the Section Alignment specified in the PE header.

B.2. ELF Format

The Executable and Linkable Format (ELF) is the primary format for executables in Unix and Linux systems. It begins with an ELF Header that supplies essential metadata to prepare the operating system for file execution. This metadata includes:

- **File Type:** Specifies whether the file is an executable, a relocatable file, or a shared object file, akin to DLLs in Windows.
- **Machine Architecture:** Indicates the hardware for which the file is compiled, crucial for determining compatibility with 32-bit or 64-bit systems.
- **Entry Point:** The memory address where program execution begins.

Distinguishing Between 32-bit and 64-bit

The difference between 32-bit and 64-bit ELF files is marked by the `e_ident[EI_CLASS]` field in the ELF Header:

- `ELFCLASS32` (`0x01`): Indicates a 32-bit ELF file.
- `ELFCLASS64` (`0x02`): Indicates a 64-bit ELF file.

Differentiating Shared Objectes from Executables

The differentiation between shared objects in Linux`(.so)` and executable files is defined by the `e_type` field in the ELF Header:

- `ET_EXEC` (`0x02`): Denotes the file as an executable.
- `ET_DYN` (`0x03`): Indicates the file is a shared object file, used by other programs during runtime.

As illustrated in Figure B.2, two critical structures are the Program Header Table and the Section Header Table, essential for the operating system to process the file correctly.

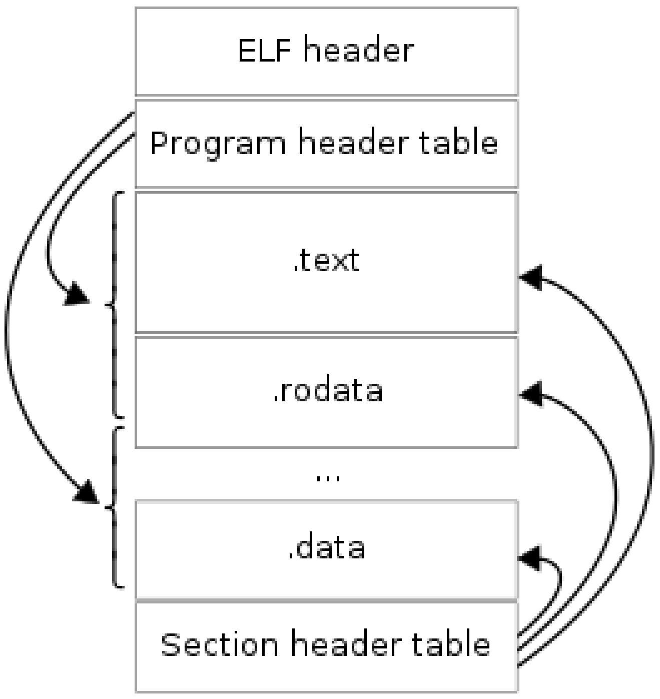


Figure B.2: Illustration of the ELF Format

Program Header Table

The Program Header Table is pivotal for execution and loading, containing entries that describe segments of the process's image in memory:

- **Code Segment:** Contains executable code of the program.
- **Data Segment:** Includes global variables and static data, storing initialized data for runtime modification.
- **Dynamic Segment:** Holds information necessary for dynamic linking, including references to shared libraries.

Section Header Table

The Section Header Table details the file's structure, instrumental in debugging and linking processes, and facilitates modular software development:

- *.text section:* Contains executable instructions of the program.
- *.data section:* Stores initialized data such as global and static variables.
- *.bss section:* Used for uninitialized data that starts with zero or null values.
- *.dynsym section:* Lists dynamic linking symbols essential for resolving symbols in shared libraries.

B.3. DOS Format

The DOS format, primarily used in legacy computing environments, represents a foundational architecture for executable files designed to operate within the DOS (Disk Operating System) system. This format is structured to efficiently handle the constrained resources typical of earlier personal computing systems. The key components of the DOS format include:

DOS Header ("MZ" Signature)

Marked by the "MZ" signature, this header is the gateway to DOS executables. It carries critical metadata necessary for the DOS operating system to correctly manage the executable. This includes the number of bytes in the last block of the program, the total number of blocks in the file, the number of relocation entries stored after the header, the size of the header itself in paragraphs, and the minimum and maximum allocation paragraphs needed by the program. This header also specifies the initial relative stack segment, which is pivotal for setting up the program's stack in memory.

Program Code

Following the header, this segment contains the executable machine code. In the DOS format, the code is often optimized for execution in the limited and direct memory access environment typical of DOS systems. It includes both the instructions that the processor will execute and the mechanism for interrupt handling, directly interacting with the hardware.

Data Sections

These segments store the static data utilized by the program, including constants, variables, and strings necessary for the program's operation. Unlike more modern executable formats, which may separate data into multiple distinct sections based on access needs and initialization status, DOS executables often contain a single data section that includes both initialized and uninitialized data.

Relocation Table

Essential for ensuring the program's flexibility, the relocation table contains pointers that adjust the program's hard-coded memory addresses. This feature allows DOS executables to adapt to different memory addresses dynamically, accommodating variations in system configuration. Each entry in the relocation table specifies parts of the code and data that must be modified when the program is loaded at an address other than its preferred starting address. This capacity is crucial for running DOS applications on a variety of hardware setups without the need for recompilation.

Overlay Management

DOS executables may also include overlays—additional code and data loaded on demand. This method is particularly useful in memory-constrained environments, as it allows applications to load additional functionality only when needed, rather than consuming memory continuously. Overlays follow the main program block and are managed by a combination of the operating system and the executable's own code.

End-of-File Marker

A specific marker that indicates the end of the executable file. In DOS, this is often a series of null bytes followed by an end-of-file character, typically used to mark the boundary of the file for both loading and execution purposes.

C
Hyperparameters

These are the hyperparameter that we use in this work. For all the hyperparameters please refer to scikit-learn [185]

Logistic Regression

- **Regularization strength (C)**: Influences the degree of regularization. Lower values indicate stronger regularization to prevent overfitting.
- **Solver**: Determines the optimization algorithm used (e.g., `newton-cg`, `lbfgs`, `liblinear`, `sag`, `saga`). Certain solvers support multinomial loss for multiclass problems.
- **Multi-class**: Specifies the approach for multiclass classification (`ovr` for One-vs-Rest, `multinomial` for multinomial logistic regression).
- **Max iterations**: Sets the maximum number of iterations for the solvers to converge.

k-Nearest Neighbors(KNN)

- **Number of Neighbors (k)**: Determines the number of nearest neighbors to consider when making classification or regression decisions.
- **Distance Metric**: Specifies the method used to calculate the distance between points (e.g., Euclidean, Manhattan).
- **Weights**: Defines how much influence each neighbor has on the final decision (e.g., uniform, distance-weighted).

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- **Perplexity**: A key hyperparameter that balances attention between local and global aspects of your data, typically chosen between 5 and 50. It can be thought of as the expected number of neighbors.
- **Learning Rate**: Determines the step size at each iteration while moving toward a minimum of the cost function. Typical values range from 10 to 1000. Setting this too high or too low can lead to poor results.
- **Number of Iterations**: The maximum number of iterations for the optimization. Sufficiently high to allow convergence.
- **Early Exaggeration**: Controls how tightly natural clusters in the original space are in the embedded space and how much space will be between them. Larger values result in more space between clusters.

Principal Component Analysis (PCA)

- **Number of Components** : Specifies the number of principal components to compute. This determines the dimensionality of the output space.
- **SVD Solver**: Specifies the method used for the singular value decomposition (SVD), which can be `auto`, `full`, `arpack`, or `randomized`.

Uniform Manifold Approximation and Projection (UMAP)

- **Number of Neighbors**: Controls how UMAP balances local versus global structure. Larger values promote a more global view of the data, while smaller values prioritize local data aspects.
- **Min Distance**: The minimum distance between points in the low-dimensional representation. Smaller values will result in tighter clusters.
- **Metric**: The metric used to measure distance in the input space, such as Euclidean, Manhattan, cosine, etc.
- **Learning Rate**: Sets the learning rate for the optimization algorithm, influencing how rapidly the model fits the data.
- **Number of Components**: Determines the number of dimensions in which to embed the data, usually 2 for visualization.

D Extraction of PE Features

1. **ByteHistogram:**
   - **Extraction Process:**
     The ByteHistogram feature extraction employs the LIEF library to methodically analyze PE files, concentrating on sections marked with the `MEM_EXECUTE` flag, indicative of executable content. This flag's presence, discerned through inspection of section headers, signifies areas within the PE file meant for execution, contrasting with those allocated for data storage or resources. Upon identifying these executable sections, their byte sequences are compiled into a collective array. Following this, a bin count is conducted using NumPy, categorizing each byte within the 0-255 range based on its occurrence.

     Targeting `MEM_EXECUTE` sections aligns the analysis with the executable components of the PE file, essential for malware analysis. This focus is pivotal, as the characteristics of the executable code often reveal malicious functionalities.

     This technique streamlines the analysis by concentrating on sections most relevant to the file's executable behavior, thereby enhancing efficiency. By examining just the executable sections, the process becomes more efficient without overlooking crucial insights into the PE file's behavior. The aggregation of bytes from these sections and subsequent bin count operation ensures a focused and efficient analysis, yielding features directly reflective of the file's executable nature.
   - **Importance:**
     This feature provides insights into the distribution of byte values within the executable sections of the file, aiding in anomaly detection, signature identification, and understanding the file's structure. It is particularly useful for identifying packed or obfuscated code, a common trait in malicious software.
   - **Final Vector:**
     The ByteHistogram results in a 256-dimensional numerical array, where each dimension represents the normalized frequency of a corresponding byte value in the executable sections. This normalization ensures comparability across PE files of different sizes by focusing on distribution patterns rather than absolute counts.
   - **Tooling:**
     - `LIEF`
     - `NumPy`

2. **ByteEntropyHistogram:**
   - **Extraction Process:**
     The ByteEntropyHistogram extraction similarly leverages the LIEF library for parsing PE files but adds an additional layer of analysis by examining the entropy within executable sections. Using NumPy, this process involves sliding a window across the bytes of each executable section and calculating the entropy for segments of bytes within that window. This method

allows for a more granular analysis of the byte distribution, capturing both the frequency of byte values and the local entropy, or randomness, of bytes in the section, which can indicate compression or encryption within the executable code.

- **Importance:**
  This feature captures the complexity and variability within executable sections, offering a nuanced view of the code's structure that can reveal attempts at obfuscation or packing. High entropy regions may suggest encrypted or compressed code, characteristics often found in malware seeking to evade detection. By analyzing the byte entropy histogram, researchers and automated systems can identify these traits, enhancing the detection of sophisticated threats.
- **Final Vector:**
  The ByteEntropyHistogram results in a two-dimensional array shaped by the entropy levels and byte values, typically sized 256x256. Each element in this matrix represents the count of byte occurrences at different entropy levels, providing a detailed landscape of the file's complexity and variability.
- **Tooling:**
  - `LIEF`
  - `NumPy`

3. **Strings:**
   - **Extraction Process:**
     The extraction of strings from a PE file involves scanning its binary content for sequences of printable characters, typically ASCII values ranging from 0x20 to 0x7f. This process not only includes the extraction of plain text strings but also seeks to identify patterns or indicators of interest, such as file paths, URLs, registry keys, and potential MZ headers within the data. The identification of these elements can hint at the file's interactions with the system, network communications, or even embedded payloads.
   - **Importance:**
     Extracting strings from executable files is crucial for several reasons. First, it can reveal direct indicators of malicious behavior or intent, such as command and control (C2) server URLs, suspicious file paths, or registry keys commonly associated with persistence mechanisms. Additionally, the presence of an unusually high number of strings, particularly those resembling code or obfuscated data, can indicate an attempt to embed malicious payloads or use steganographic techniques.
   - **Final Vector:**
     The result of the string extraction process is a multi-dimensional feature set encompassing the total number of strings, their average length, distributions of printable characters, and counts of specific string patterns (e.g., URLs, file paths). This complex feature set provides a comprehensive overview of the textual content within the PE file, serving as a basis for further analysis and classification tasks.
   - **Tooling:**
     - `NumPy`

4. **GeneralFileInfo:**
   - **Extraction Process:**
     The GeneralFileInfo feature captures fundamental attributes of the PE file, providing a high-level overview of its structure and properties. This includes the file's size, the presence of debugging information, the count of exports and imports, and indicators of security features like relocations, resources, signatures, and thread local storage (TLS). By leveraging LIEF, we parse the PE file to extract these attributes, each offering a snapshot into different aspects of the file's compilation and intended behavior.
   - **Importance:**
     The general information extracted is critical for understanding the context in which the file was created and how it is expected to interact with its environment. For example, the presence of a digital signature might suggest a semblance of legitimacy, while the number of

imports can indicate the file's dependency on external libraries, potentially revealing its functionality or malicious intent. Similarly, debug information can provide insights into the development process of the file, possibly unearthing vulnerabilities or hidden features.

- **Final Vector:**
  This feature set culminates in a compact vector encapsulating the extracted general information attributes. Each attribute contributes to a dimensional element of the vector, ranging from binary flags indicating the presence or absence of certain features to integer counts reflecting the complexity or connectivity of the file within its ecosystem.
- **Tooling:**
  - `LIEF`

5. **HeaderFileInfo:**

   - **Extraction Process:**
     The extraction of HeaderFileInfo involves a detailed analysis of the PE file's header, using LIEF to access and interpret the data. This process focuses on extracting key information from both the COFF (Common Object File Format) header and the optional PE header, which together provide critical insights into the architecture, compilation, and capabilities of the PE file. Attributes such as the compilation timestamp, target machine architecture, and system version numbers are extracted, along with characteristics defining the file's behavior, such as subsystem and DLL characteristics. This header information paints a detailed picture of the file's intended execution environment and compatibility.
   - **Importance:**
     Understanding the header information is vital for identifying the file's target architecture (e.g., x86, x64), which influences how it will execute. The compilation timestamp can indicate the file's age, potentially correlating with specific malware campaigns. System version numbers and subsystem information help in assessing compatibility issues or targeting specific operating systems, which is crucial for tailoring defensive measures. The DLL characteristics can signal advanced functionalities or defense evasion techniques employed by the file.
   - **Final Vector:**
     The resulting feature vector from the HeaderFileInfo is a mix of categorical and numerical data reflecting the extracted header attributes. Categorical data, such as the target machine architecture, are typically encoded to facilitate analysis, while numerical data are directly utilized, providing a multifaceted view of the file's header configuration.
   - **Tooling:**
     - `LIEF`

6. **SectionInfo:**

   - **Extraction Process:** The extraction of SectionInfo begins with the identification and analysis of each section within a PE file using LIEF. This process entails examining the properties of each section, such as its name, size, entropy, virtual size, and characteristics, to gather a comprehensive overview of the file's structure. For each section, these details are compiled into a structured format, highlighting aspects like section names, their sizes (both raw and virtual), and their entropy values. This detailed enumeration of section attributes provides insight into the organization, purpose, and security features of the PE file, such as which sections contain executable code, data storage, or are marked as writable or readable.
   - **Importance:** Understanding the characteristics of each section is crucial for malware analysis, as it can reveal patterns, anomalies, or signatures indicative of malicious intent. Sections with unusual sizes, high entropy, or unexpected permissions may signal attempts to hide malicious code, implement anti-debugging measures, or exploit vulnerabilities. By systematically analyzing these attributes, SectionInfo contributes to the comprehensive assessment of a PE file's potential threats and behaviors.
   - **Final Vector:** The output is a composite feature vector that encapsulates the diverse properties of all sections within the PE file. This vector includes aggregated and hashed values for section sizes, entropies, and permissions, providing a multi-dimensional representation of the file's layout and characteristics.

- **Tooling:**
  - `LIEF`

7. **ImportsInfo:**

   - **Extraction Process:** The ImportsInfo feature extraction process delves into the import address table (IAT) of a PE file, harnessing LIEF to parse and enumerate all imported libraries and their associated functions. This step is pivotal, as it reveals the external dependencies and API calls a PE file makes, which are indicative of its behavior and functionality. Each imported library, along with its imported functions, is meticulously cataloged, providing a detailed view of the file's interaction with the operating system and other binaries.
   - **Importance:**
     The analysis of imported functions and libraries is a cornerstone of malware detection, offering insights into the file's potential capabilities, such as network communication, file manipulation, or system surveillance. Anomalies or suspicious patterns in API usage can signal malicious intent, especially when considering the context of known malware signatures or behaviors. Thus, ImportsInfo plays a crucial role in the identification and classification of PE files.
   - **Final Vector:**
     The feature vector for ImportsInfo encapsulates the diversity of imported functions and libraries in a high-dimensional space, using hashing techniques to manage the variability in the number of imports. This vector, typically hashed to a fixed size, represents the unique footprint of a PE file's external interactions, serving as a significant marker for analysis.
   - **Tooling:**
     - `LIEF`

8. **ExportsInfo:**

   - **Extraction Process:** ExportsInfo focuses on identifying and cataloging the functions a PE file makes available for other binaries to use. Utilizing LIEF, the process involves parsing the PE file to access its export table, which lists all functions that the file exports. This analysis is crucial for understanding the PE file's potential to influence or interact with other processes, indicating its role within a larger software ecosystem or a malware campaign. Each exported function is recorded, providing insight into the file's capabilities and its intended interactions with other system components.
   - **Importance:**
     The exported functions of a PE file can reveal significant aspects of its functionality, such as offering services or functionalities to other processes. In the context of malware analysis, exported functions can hint at the malware's capabilities, including backdoor hooks, utility functions, or custom APIs for controlling infected systems. Recognizing unusual or suspicious exports is therefore integral to detecting and understanding malware operations.
   - **Final Vector:**
     The resulting feature vector from ExportsInfo represents the set of functions exported by the PE file, encoded into a fixed-size vector through hashing. This approach allows for a uniform representation of exports across PE files, facilitating comparison and analysis regardless of the number of exported functions.
   - **Tooling:**
     - LIEF

9. **EntryPoints:**

   - **Extraction Process:** EntryPoints analysis targets the initial execution points within a PE file, which are pivotal in determining how the program starts its execution. This process leverages LIEF to examine the PE file's headers to locate the main entry point as specified in the optional header. Additionally, it identifies entry points provided by the export table, which could serve as alternate execution paths. Special attention is given to extracting entry points from executable sections, which may indicate embedded executables or additional malicious payloads. This comprehensive examination sheds light on the various ways a PE file can be invoked, highlighting potential mechanisms for stealth execution or persistence.

- **Importance:**
  Understanding the entry points of a PE file is crucial for mapping its execution flow and predicting its behavior. For malware analysts, identifying unconventional or hidden entry points can reveal sophisticated evasion techniques or unpacking routines. Entry points can also signify the presence of embedded executables, contributing to a layered analysis of complex malware samples.
- **Final Vector:**
  The extracted entry points are synthesized into a list that captures the addresses and, where available, the names of the entry functions. This list forms the basis of a feature vector that encapsulates the PE file's execution initiation points, contributing to the behavioral fingerprint of the file.
- **Tooling:**
  - `LIEF`

10. **ExitPoints:**

    - **Extraction Process:** Identifying exit points within a PE file involves pinpointing functions that terminate the process or alter its execution flow significantly, potentially signaling the end of a malicious payload's activity. This process employs a combination of static analysis tools, including LIEF for structural examination and Capstone for disassembly, to scan the PE file for known exit functions (e.g., 'ExitProcess', 'TerminateProcess'). These tools allow for the inspection of both imported functions, which might directly call such termination functions, and the disassembled executable sections, searching for direct calls to these exit routines. This dual approach ensures a comprehensive coverage of both static and dynamic indications of process termination or alteration points.
    - **Importance:**
      Analyzing exit points is essential for understanding the cleanup or evasion techniques employed by malware. By identifying how and where a program intends to terminate its execution, analysts can infer the malware's operational strategy, including attempts to evade detection or analysis post-execution. It also aids in understanding the malware's life cycle, particularly how it ensures persistence or clears its tracks after completing its tasks.
    - **Final Vector:**
      The information gathered from the exit points analysis culminates in a feature vector comprising the names and, where possible, the addresses of the termination functions employed by the PE file. This vector provides a concise summary of the exit strategies encoded within the file, offering valuable insights into its intended operational behavior.
    - **Tooling:**
      - `LIEF`
      - `Capstone`

11. **Opcodes:**

    - **Extraction Process:** The extraction of opcodes, or operational codes, within a PE file is achieved through disassembling its executable sections. Utilizing Capstone, a robust disassembly framework, the process involves parsing the binary to identify machine-level instructions executed by the program. This step is critical, as it unveils the low-level operations that dictate the malware's behavior. The analysis targets executable sections defined by `MEM_EXECUTE` flags, similar to the ByteHistogram extraction process, ensuring a focus on the operational aspects of the binary. Through disassembly, each instruction is broken down to its opcode, the fundamental operation it performs, thus creating a comprehensive list of opcodes present in the executable code.
    - **Importance:**
      Opcodes offer a granular view of the binary's execution logic, enabling the identification of malicious patterns, algorithms, and behaviors encoded at the instruction level. By examining the opcodes, analysts can detect specific sequences indicative of malware, such as code injection, encryption routines, or evasion mechanisms. This feature is particularly valuable for signature-based detection and understanding the binary's functionality without executing it.

- **Final Vector:**

  The result of the opcode analysis is a list or array of opcodes extracted from the binary's executable sections. This collection may be further processed into a frequency vector, representing the occurrence of each opcode within the analyzed sections, or used as-is for pattern recognition and classification tasks. This vector serves as a foundational component for malware detection models, providing a detailed account of the binary's executable instructions.
- **Tooling:**
  - `Capstone`
  - `LIEF`

12. **Opcode Occurrences:**

    - **Extraction Process:** Following the disassembly and opcode extraction, the analysis advances to quantifying each opcode's occurrences within the PE file's executable sections. This procedure entails iterating through the list of opcodes previously gathered and tallying each instance. The focus remains on executable sections, where operational codes directly influence the program's behavior. By employing a dictionary or hashmap, each opcode's frequency is meticulously recorded, offering a quantitative measure of the binary's instruction use.
    - **Importance:**

      The occurrence frequency of opcodes sheds light on the binary's operational emphasis, highlighting the use of certain instructions over others. This data is instrumental in detecting anomalies or patterns distinctive to malware, such as an unusual frequency of jump instructions that may indicate obfuscation or evasion techniques. The opcode occurrences serve as a signature of the binary's execution logic, facilitating both detection and classification of malware based on instruction usage patterns.
    - **Final Vector:**

      The outcome of this analysis is a frequency vector or dictionary mapping each opcode to its occurrence count within the executable sections. This vector encapsulates the instruction profile of the PE file, enabling comparisons across binaries and identifying outliers or signatures indicative of malicious intent.
    - **Tooling:**
      - `Capstone`

13. **ImageSize:**

    - **Extraction Process:** ImageSize is determined by analyzing the PE file's total virtual size, inclusive of all headers and sections. This metric encapsulates the entire memory footprint of the PE file when loaded into memory. To compute this, the tool iterates over each section within the PE file, accumulating their sizes while also accounting for header sizes. This comprehensive summation provides a full picture of the file's size as it would appear during execution.
    - **Importance:**

      The total image size of a PE file is a fundamental characteristic, offering insights into the file's complexity and potential load on system resources. Larger files may indicate the presence of embedded resources or functionalities, whereas anomalously small sizes could suggest a file designed to be stealthy or serve as a loader for additional malicious payloads. Understanding the image size assists in gauging the file's scope and potential impact.
    - **Final Vector:**

      The result of this analysis is a single value representing the total image size of the PE file. This scalar value reflects the file's comprehensive size, combining executable code, data sections, and metadata into one metric.
    - **Tooling:**
      - `PEfile`

14. **HeaderSize:**

- **Extraction Process:** The HeaderSize feature quantifies the combined size of the PE file's headers, including the DOS, file, optional, and section headers. This is accomplished by parsing the PE file to calculate the sizes of these individual components.
- **Importance:**

  Analyzing the header size of a PE file is critical for understanding the overhead introduced by metadata and structural definitions. It can indicate the complexity of the file's structure and the presence of additional functionalities encapsulated within the headers. Large header sizes may suggest extensive use of section definitions or the presence of additional metadata, while unusually small header sizes could indicate attempts to minimize the file's footprint or manipulate header information for obfuscation purposes.
- **Final Vector:**

  This analysis yields a scalar value representing the total size of all headers within the PE file. This value is crucial for assessing the overhead and structural complexity of the file, contributing to a holistic understanding of its construction.
- **Tooling:**
  - `LIEF`
  - `PEfile`

15. **StackReserveSize:**

    - **Extraction Process:** StackReserveSize measures the amount of memory space reserved for the stack in a PE file. This feature is extracted by parsing the PE file's optional header, specifically focusing on the SizeOfStackReserve field.
    - **Importance:**

      The reserved stack size is an indicator of the anticipated memory usage for the stack during execution. It can signal the complexity and potential memory demands of the application. Abnormally high or low values may suggest unusual behaviors or potential evasion techniques designed to manipulate execution environments or evade detection.
    - **Final Vector:**

      The feature is represented as a scalar value indicating the reserved stack size in bytes. This quantitative measure is vital for assessing the PE file's memory allocation patterns and understanding its operational requirements.
    - **Tooling:**
      - `LIEF`
      - `PEfile`

16. **StackCommitSize:**

    - **Extraction Process:**

      The extraction of the *StackCommitSize* feature is performed through an analysis of the PE file's optional header, specifically targeting the size of memory committed to the stack at execution start. This process is facilitated by the `pefile` Python library, which allows for simplified access to PE file attributes, including those within the optional header where *StackCommitSize* is located.
    - **Importance:**

      The *StackCommitSize* provides critical insight into the application's memory usage patterns, focusing on stack allocation. This feature is instrumental in identifying abnormal memory allocation practices that could indicate malicious intent or exploitation attempts by malware, thus serving as a significant indicator in the distinction between benign and malicious files.
    - **Final Vector:**

      This feature produces a singular numerical value representing the memory committed for the stack, measured in bytes. It forms an integral component of a broader feature set, enriching the analysis with details on the application's memory management and execution behaviors.
    - **Tooling:**
      - `PEfile`

17. **HeapSize:**

- **Extraction Process:**

  The *HeapSize* is determined by parsing the PE file's optional header to obtain the sizes of both heap reserve and heap commit. Utilizing the `pefile` Python library, we access the `SizeOfHeapReserve` and `SizeOfHeapCommit` fields within the optional header, which indicate the total size reserved for the heap and the size of the heap space initially committed, respectively.
- **Importance:**

  Understanding the *HeapSize*, encompassing both reserve and commit sizes, is crucial for malware analysis. It offers insights into how an application plans to use memory dynamically. Unusually large heap sizes can suggest an application's potential to engage in memory-intensive operations or exploit techniques, making this feature a valuable indicator for detecting malicious activities.
- **Final Vector:**

  This feature yields two numerical values representing the reserved and initially committed heap sizes in bytes. These values contribute to a multi-dimensional feature vector, providing a nuanced view of the application's memory allocation patterns and their implications for security.
- **Tooling:**
  - `PEfile`

18. **LoaderFlags:**

    - **Extraction Process:**

      To determine the *LoaderFlags*, we examine the PE file's optional header, specifically targeting the `LoaderFlags` field. This task is accomplished using the `pefile` library, which allows for direct access to PE structure and fields. The `LoaderFlags` field provides insight into specific flags set during the loading of the executable, which can affect the behavior of the loader.
    - **Importance:**

      While *LoaderFlags* are typically set to zero in most PE files and might seem trivial, any non-standard value could indicate unusual or potentially malicious loader behavior. Analyzing these flags helps in detecting anomalies or techniques employed by malware to manipulate the loading process, contributing to a broader understanding of the executable's characteristics.
    - **Final Vector:**

      The feature results in a single numerical value representing the *LoaderFlags* set within the PE file's optional header. This value enriches the feature vector by adding a dimension that reflects the executable's loader behavior, even if it's commonly zero for most benign applications.
    - **Tooling:**
      - `PEfile`

19. **Kolmogorov Complexity:**

    - **Extraction Process:** The extraction of the Kolmogorov Complexity for a PE file involves evaluating the compressibility of the file's executable sections. each executable section identified within the PE file—through parsing with the `lief` library—is compressed. The process aims to measure the amount of redundancy in the data, which is indicative of the file's complexity.
    - **Importance:**

      Kolmogorov Complexity serves as a proxy for understanding the file's complexity and potential obfuscation levels. Highly compressed sections suggest redundancy and lower complexity, while less compressible sections might indicate sophisticated obfuscation techniques commonly employed by malware to evade detection. This feature is instrumental in distinguishing between simple, benign applications and complex, potentially malicious executables.

- **Final Vector:**
  This analysis yields a numerical value representing the compressed size of the executable sections, normalized against the original size to provide a measure of the file's Kolmogorov Complexity. This scalar value enriches the feature vector, offering insights into the executable's structural complexity.
- **Tooling:**
  - `lief`
  - `zlib`

  Through the application of these tools, the Kolmogorov Complexity feature encapsulates a critical aspect of the PE file's nature, distinguishing between varying levels of complexity and obfuscation that may hint at malicious intent.

20. **DataDirectories:**
    - **Extraction Process:**
      The extraction of *DataDirectories* involves iterating over the data directories present in a PE file, facilitated by the LIEF library. Each data directory within the PE format serves a distinct purpose, such as import tables, export tables, and resource sections.
    - **Importance:**
      Data directories are critical for understanding the structural and functional aspects of a PE file. They contain metadata and pointers to important sections of the executable, such as where to find imported functions, exported functions, and resources. Analyzing these directories helps in identifying how an executable interacts with external libraries, manages resources, and potentially, how it might behave when executed.
    - **Final Vector:**
      The feature produces a list of dictionaries for each data directory, with keys for the directory's name, size, and virtual address. This structured format preserves the detailed information about each directory, contributing to a comprehensive analysis of the PE file's structure and capabilities.
    - **Tooling:**
      - `LIEF`

E

Extraction of DOS Features

1. **ByteHistogram (DOS):**

   - **Extraction Process:** The ByteHistogram for DOS executables is obtained by analyzing the file's byte content starting from a specific offset indicative of the executable code's commencement. Initially, the DOS header is parsed with Python's `struct` module to extract relevant fields for the executable section, notably the initial stack settings ($initial_{ss}$, $initial_{sp}$), the instruction pointer ($initial_{ip}$), and the code segment ($initial_{cs}$). This parsing delineates the offset ($code_{offset}$) for analyzing executable bytes, calculated as $initial_{cs} \times 16 + initial_{ip}$. Should this offset surpass the file's size, suggesting an abnormal or unconventional file structure, the entire file is then considered for analysis. Bytes from the calculated offset to the file's conclusion are cumulatively analyzed.

     Subsequently, a bin count is performed on these aggregated bytes using NumPy, effectively mapping each byte value within the 0-255 range to its frequency of occurrence in the executable section. This nuanced approach allows for a detailed byte-level analysis of the executable part of the DOS file, providing insights into its composition and potential behaviors.
   - **Importance:** The ByteHistogram is instrumental in understanding the distribution of byte values within the DOS executable's code section. Such analysis can reveal patterns indicative of specific programming practices, compression, or obfuscation techniques. This feature becomes particularly valuable in malware analysis, where deviations from normative byte distributions can signal malicious intent or sophisticated evasion mechanisms embedded within the executable.
   - **Final Vector:** The result of this process is a 256-dimensional numerical array, with each dimension reflecting the frequency of occurrence of a corresponding byte value in the analyzed executable section. This representation captures a fundamental aspect of the DOS file's content, serving as a critical feature for subsequent analysis phases, including machine learning-based malware classification.
   - **Tooling:**
     - `struct`
     - `NumPy`

2. **ByteEntropyHistogram (DOS):**

   - **Extraction Process:** Following the initial procedures employed for ByteHistogram extraction, the ByteEntropyHistogram for DOS files begins with decoding the DOS header fields ($initial_{ss}$, $initial_{sp}$, $initial_{ip}$, $initial_{cs}$). This information aids in pinpointing the commencement of the executable content. Commencing from this determined offset, the bytes undergo analysis in designated blocks to construct a 2D entropy histogram. This histogram delineates the combined distribution of byte values and their respective entropy levels, utilizing NumPy to enhance the efficiency of these numerical operations and the structuring of data.

- **Importance:** ByteEntropyHistogram serves as a nuanced feature that encapsulates the variability and complexity of byte distributions within the DOS executable's code section. By examining both the byte values and their associated entropies, this feature provides deeper insights into the code's structure and potential obfuscation methods. It is especially useful in malware analysis for identifying packed or encrypted sections within the code, aiding in the detection of sophisticated threats.
- **Final Vector:** The output of this process is a flattened, 256-dimensional array derived from the 2D histogram, with each element representing the frequency of a specific byte-entropy pair. This representation captures the intricacies of the code's composition and entropy characteristics, offering a rich feature for subsequent analysis stages.
- **Tooling:**
  - `struct`
  - `NumPy`

  Employing these tools, the ByteEntropyHistogram is extracted, reflecting the combined distribution of byte values and their entropy within the DOS executable, thus enriching the analytical framework with a sophisticated feature.

3. **Strings (DOS):**
   - **Extraction Process:** This process is the same as that for the PE with the resulting vector.
   - **Importance:** Extracting strings provides insights into the executable's functionality, potential interactions with the system or network, and hardcoded values, which are crucial for analyzing the behavior and identifying potential threats within a DOS executable.
   - **Final Vector:** The final vector includes the number of strings, average string length, distribution of printable characters, and occurrences of specific patterns such as paths, URLs, and registry keys. This comprehensive data aids in understanding the content and context of the strings within the executable.
   - **Tooling:**
     - re
     - `NumPy`
4. **GeneralFileInfo (DOS):**
   - **Extraction Process:** The GeneralFileInfo for DOS executables is derived directly from reading the file's attributes and its DOS header. The process involves first determining the file's size through the `os.path.getsize` function. Subsequently, the DOS header is parsed to extract critical fields, including the magic number, blocks in the file, relocation entries, and initial stack and code segment values, among others. This comprehensive extraction from the DOS header provides a detailed snapshot of the file's structure and foundational attributes.
   - **Importance:** Extracting general file information is crucial for a basic yet comprehensive understanding of the DOS executable. This includes insights into the file size, layout, and initial execution settings, which are foundational for further analysis. Such information can be pivotal in identifying file integrity, compatibility, and potential malicious alterations.
   - **Final Vector:** The extracted information culminates in a structured representation encompassing various attributes of the DOS executable, including file size, header paragraphs, relocation counts, and initial execution parameters (stack and code segments). This structured data forms an essential part of the feature vector, enabling detailed analysis and comparison across executables.
   - **Tooling:**
     -
     - `struct`

     Leveraging these tools and methods, the extraction process for GeneralFileInfo effectively deciphers and structures key attributes of DOS executables, setting the stage for in-depth analysis and insights.
5. **HeaderFileInfo (DOS):**

- **Extraction Process:** The HeaderFileInfo for DOS executables focuses on extracting detailed information from the DOS header. This process involves parsing the header, which is typically the first 28 to 64 bytes of the file, to retrieve information about the file's structure and initial execution environment. Key fields such as the magic number ('MZ'), bytes on the last page, number of pages, relocations, header size in paragraphs, and initial stack and code segment settings are extracted using `struct.unpack`. This method allows for direct translation of binary data into Python data structures.
- **Importance:** Understanding the header information of DOS executables is fundamental for analyzing the file's layout, compatibility, and execution parameters. These attributes provide insights into how the operating system will load and run the file, including memory allocation and initial execution points. Such information is invaluable for both historical analysis of legacy software and the examination of potentially malicious DOS executables.
- **Final Vector:** The resulting data from the DOS header extraction process is a structured representation that encapsulates essential attributes of the executable. This includes, but is not limited to, the file's page structure, memory allocation requirements, and initial execution context. This comprehensive breakdown contributes to the overall feature set, enriching the analysis with detailed structural insights.
- **Tooling:**
  - `struct`

  Through the application of these parsing techniques, HeaderFileInfo meticulously extracts and organizes critical structural details from DOS executables, providing a foundational layer for further analysis and insights.

6. **MemoryLayout (DOS):**

   - **Extraction Process:** To extract the MemoryLayout feature for DOS executables, we delve into the DOS header to pinpoint the `Initial SP` (Stack Pointer) and `Max Allocation` values. Utilizing Python's `struct.unpack`, we navigate to the specified offsets within the header: `Initial SP` is located at offset `0x10` (16) and `Max Allocation` at offset `0x02` (2). These values are critical for understanding the executable's memory setup and its stack initiation point.
   - **Importance:** This feature provides insight into the memory blueprint and stack configuration of DOS executables, reflecting on how the program was intended to manage and utilize memory. Such insights are invaluable for understanding the executable's interaction with the DOS operating environment and hardware resources.
   - **Final Vector:** The extraction results in a vector capturing the `Initial SP` and `Max Allocation` values, enriching the analysis with details on the executable's memory layout and allocation requirements.
   - **Tooling:**
     - `struct`

     By employing these tools, we effectively uncover and interpret key memory layout parameters from DOS executables, enhancing our analytical framework with deeper insights into their operational characteristics and memory usage patterns.

7. **EntryPoints (DOS):**

   - **Extraction Process:** Entry points in DOS executables are determined by initially inspecting the DOS header for the CS (Code Segment) and IP (Instruction Pointer) fields, if the header is adequately sized. This inspection is conducted through `struct.unpack` on the bytes located at offsets 0x14 and 0x16 ($initial_{cs}$ and $initial_{ip}$, respectively), indicative of the initial execution point. The process then advances to disassembly, utilizing the `Capstone` disassembler in 16-bit mode, to detect interrupt (int) instructions within the executable's code starting from offset 0x100, signaling additional entry points critical to understanding the execution flow.
   - **Importance:** The identification of entry points is fundamental for analyzing the DOS executable's flow, especially for insights into the software's initial execution phase and interrupt-based operations. This knowledge is pivotal in legacy system analyses and malware investigation within DOS environments.

- **Final Vector:** The feature set includes the initial CS:IP entry point extracted from the DOS header and any further entry points identified through the disassembly process. This comprehensive enumeration of entry points serves as a crucial element in the feature vector, offering a deeper understanding of the executable's behavior.
- **Tooling:**
  - `struct`
  - `Capstone`

8. **EntryPoints (DOS):**
   - **Extraction Process:** The extraction process for *EntryPoints* in DOS executables begins by reading the first 28 bytes from the DOS header, aiming to uncover the initial execution point marked by the CS (Code Segment) and IP (Instruction Pointer) fields. This is executed through Python's `struct.unpack` method, which interprets the DOS header bytes to retrieve the essential fields. The specific fields extracted include e_{cp} (pages in file), e_{cblp} (bytes on the last page of the file), among others, leading to $initial_{ip}$ and $initial_{cs}$, which collectively indicate the entry point of the executable. The DOS header, being a crucial part of the executable's metadata, contains these fields at predefined offsets, facilitating the identification of the program's entry point as designated by the original developer.
   - **Importance:** *EntryPoints* is critical in DOS executable analysis as it signifies where the execution starts within the binary. This information is vital for reverse engineering, malware analysis, and understanding the executable's behavior on a DOS system. Unlike more modern PE formats, DOS executables utilize a simpler structure, where the entry point directly informs about the beginning of the program execution.
   - **Final Vector:** The feature comprises a single string encapsulating the entry point, denoted by the CS and IP values extracted from the DOS header. This singular data point enriches the feature vector with crucial execution start information, contributing to a comprehensive understanding of the DOS executable's initial behavior.
   - **Tooling:**
     `struct`

   The utilization of these tools and methodologies effectively surfaces the *EntryPoints* from DOS executables, spotlighting the starting point of the program's execution within the binary structure.

9. **ExitPoints (DOS):**
   - **Extraction Process:** Identifying *ExitPoints* within DOS executables involves utilizing the Capstone disassembly framework to analyze the binary in 16-bit mode, reflecting the DOS architecture. The analysis commences from an assumed offset (typically `0x100`, a common entry point in DOS executables), where the search for interrupt (`int`) instructions that signify exit points begins. These interrupts include `INT 20h`, signaling program termination, and `INT 21h`, which encompasses a broader range of system calls, including program exit with a return code when accompanied by the `AH=4Ch` register value. The disassembly process reveals these interrupts, capturing the instruction's address and the specific exit or system call it represents.
   - **Importance:** *ExitPoints* play a pivotal role in understanding the termination behavior of DOS executables, offering insights into how and where a program concludes its execution. This can be crucial for malware analysis, where unconventional exit points may be used as part of the malware's execution flow to evade detection or perform cleanup routines.
   - **Final Vector:** The analysis results in a collection of strings, each detailing an exit point's location (address) and the type of exit or system call it invokes. This collection enriches the feature set with specific insights into the program's termination behavior, highlighting potential areas of interest for further analysis or reverse engineering.
   - **Tooling:**
     - `Capstone`

10. **Opcodes (DOS):**

- **Extraction Process:** The extraction of *Opcodes* for DOS files employs a process to disassemble the executable's code to retrieve the opcodes. This is initiated by determining the executable's architecture to ensure accurate disassembly. The binary is examined to identify the executable sections, after which each section's content is disassembled using Capstone, a disassembly framework that supports 16-bit DOS mode. The disassembly results in a list of instructions from which opcodes are extracted and aggregated across all executable sections.
- **Importance:** *Opcodes* provide a granular view of the executable's operations, offering insights into its functionality. Analyzing the distribution and types of opcodes can reveal patterns indicative of benign or malicious intent, including the presence of common malware techniques or constructs.
- **Final Vector:** The extraction yields an array or list of opcode mnemonics, representing the low-level operations performed by the executable. This array forms part of the feature set, enabling machine learning models or analytical tools to discern patterns or anomalies within the executable code.
- **Tooling:**
  - `Capstone`
  - `pefile`

11. **Opcode Occurrences (DOS):**

    - **Extraction Process:** The process for determining *Opcode Occurrences* closely follows the approach described for *Opcodes* in DOS files, with an added layer to count the frequency of each opcode across the executable sections. After disassembling the code to identify opcodes, a mapping is created to tally occurrences of each opcode. This process involves analyzing opcode information extracted from executable sections, as outlined in the Opcodes (DOS) extraction, and aggregating their occurrences into a comprehensive dictionary.
    - **Importance:** Counting opcode occurrences offers insights into the prevalence of certain operations within the DOS executable, which can be indicative of its behavior. Patterns in opcode usage, especially those that deviate from typical benign software patterns, can signal the presence of malware or the use of obfuscation and anti-analysis techniques.
    - **Final Vector:** This process yields a dictionary or associative array where keys represent opcode mnemonics and values denote their frequency of occurrence within the executable. This structured representation enriches the feature set with quantitative insights into the opcode distribution, enhancing the analysis of the executable's characteristics.
    - **Tooling and References:** The tooling employed is identical to that utilized in the extraction of Opcodes for DOS files, with the addition of Python's `defaultdict` for efficient counting and aggregation of opcode occurrences. The methodology and tools referenced under Opcodes (DOS) are directly applicable here, emphasizing the operational consistency between these feature extraction processes.

12. **Image Size (DOS):**

    - **Extraction Process:**

      The Image Size for DOS executables is determined simply by assessing the file's total size. This straightforward approach involves utilizing the `os.path.getsize` method in Python, which returns the file size in bytes. This metric directly reflects the physical size of the DOS executable on disk, offering a basic yet critical dimension of the file's characteristics.
    - **Importance:**

      The size of an executable can provide initial hints about its complexity and the breadth of functionality it might encompass. In the context of malware analysis, unusually large or small file sizes, relative to typical DOS executables, can serve as indicators of potential malicious intent or obfuscation attempts.
    - **Final Vector:**

      The outcome of this process is a single numerical value representing the file size in bytes. This scalar value forms an essential component of the feature vector, encapsulating a fundamental attribute of the executable that aids in its overall evaluation and comparison with other files.

- **Tooling:**
  - `Struct`

13. **Memory Size (DOS):**

- **Extraction Process:** The process for determining memory size specifications in DOS files involves parsing the DOS header to extract key parameters related to the executable's memory usage, including the Initial Stack Segment (`SS`), Stack Pointer (`SP`), and memory allocation requirements ($min_{allocation}$ and $max_{allocation}$). Python's `struct.unpack` function is utilized to read these values from specific offsets within the file: `0x0E` for $initial_{ss}$, `0x10` for both $initial_{sp}$ and $max_{allocation}$, and `0x0C` for $min_{allocation}$.
- **Importance:** These memory specifications are crucial for understanding the DOS executable's memory footprint and operational requirements. They provide insights into how the executable is expected to interact with system memory, including stack initialization and memory reservation which are pertinent for both routine analysis and the identification of potential anomalies or malicious patterns.
- **Final Vector:** This analysis yields a structured representation of the executable's memory size specifications, encapsulating `initial_ss`, `initial_sp`, `min_allocation`, and `max_allocation`. These attributes contribute to a multi-dimensional feature vector that aids in the comprehensive characterization of the DOS file.
- **Tooling:**
  - `struct`

14. **Header Size (DOS):**

- **Extraction Process:** To ascertain the size of the DOS header, the process commences with the binary opening of the file, followed by the retrieval of the magic number from the first two bytes to validate the DOS ('MZ') signature. Subsequently, attention is directed to the offset at position `0x3C` (60_{10}), which points to the new executable header, indicating the end boundary of the DOS header. This offset is decoded using Python's `struct.unpack` with a format specifier of `'<H'`, denoting a little-endian unsigned short.
- **Importance:** The DOS header size is fundamental for delineating the boundary between the DOS stub and the subsequent sections of an executable. This demarcation aids in parsing and analyzing the file's structure, especially when investigating legacy or dual-mode executables that commence with a DOS header before transitioning to a more advanced format.
- **Final Vector:** The result is a singular value representing the size of the DOS header, which, when non-zero, delineates the initial segment of the executable dedicated to the DOS stub. This metric enriches the feature set with structural insights into the DOS executable.
- **Tooling:**
  - `open`
  - `struct`

15. **Block Entropy (DOS):**

- **Extraction Process:** The Block Entropy for DOS files is calculated by segmenting the file into blocks of a specified size (default is 1024 bytes) and computing the entropy for each block. This process begins with a file integrity check to ensure its presence. Following the successful opening of the file, its contents are read into memory. Each block is then individually assessed for entropy using a calculation that measures the unpredictability or randomness of the data within. The function iterates over all possible byte values (0-255), calculating the probability of each and summing their contributions to the entropy of the block. This step is crucial for identifying sections of the file that may exhibit unusual patterns, potentially indicative of obfuscation or data encoding.
- **Importance:** Understanding the entropy of different blocks within a DOS file can reveal insights into the file's complexity and areas of potential data packing or encryption. High entropy regions may indicate obfuscated or compressed code, common in malware attempting to evade detection. Conversely, low entropy might suggest regular, uncompressed data. This differentiation is pivotal in forensic and malware analysis contexts, where anomalies in file structure can lead to significant findings.

- **Final Vector:** The entropy analysis results in a set of statistics including the minimum, maximum, total, and mean entropy values across all blocks. This collection of metrics forms a composite picture of the file's entropy landscape, serving as a multi-dimensional feature within the analysis vector.
- **Tooling:**
  - Python Standard Library

16. **Kolmogorov Complexity (DOS):**
    - **Extraction Process:** The method for deriving the Kolmogorov Complexity for DOS executables is analogous to that described for PE files. It entails assessing the compressibility of the file's data to gauge its complexity.
    - **Importance:** Similar to its application in PE analysis, the Kolmogorov Complexity for DOS files provides insights into the file's complexity and obfuscation levels. It serves as an indicator of the sophistication of the code, including the presence of potential obfuscation techniques.
    - **Final Vector:** The outcome is a scalar value that reflects the Kolmogorov Complexity, offering a quantifiable measure of the file's complexity based on its compressibility.
    - **Tooling:**
    - `lief`
    - `zlib`
17. **Interrupt Info (DOS):**
    - **Extraction Process:** Interrupt information is derived by disassembling the DOS executable and identifying instances where software interrupt instructions (`int`) are used. This process begins with the entire contents of the DOS file being read into memory. Utilizing the Capstone disassembly framework set for 16-bit x86 mode, the file is disassembled starting from a predefined offset, commonly `0x100`, which is a conventional location for the start of executable code in DOS files. Each `int` instruction encountered during disassembly is recorded, capturing both its memory address and interrupt vector. This approach is crucial for understanding the software's interaction with the operating system, particularly for identifying system calls and potential interrupt-based functionalities or exploits.
    - **Importance:** Analyzing interrupts within a DOS executable provides insights into how the application interfaces with the DOS operating system, including any system services it may request. Certain interrupts can be indicative of specific behaviors or functionalities, such as file operations, I/O interactions, or termination requests. This information can be particularly valuable in malware analysis, where specific interrupt calls might suggest malicious intent or techniques.
    - **Final Vector:** The extracted interrupt information results in a list of dictionaries, each detailing an interrupt's address and its vector (`op_str`). This structured representation allows for easy integration into feature vectors, contributing to a comprehensive understanding of the DOS executable's behavior and its system-level interactions.
    - **Tooling:**
      - `Capstone`
18. **Stack Information (DOS):**
    - **Extraction Process:** The stack information, specifically the Initial Stack Pointer (SP) and Stack Segment (SS), is extracted by accessing the DOS header of the executable. Upon verifying the file as a valid DOS executable through the magic number check (`MZ`), the process seeks to the offsets where the Initial SP and SS values are stored. These values are then read and decoded, providing direct insight into the initial stack configuration upon the program's start. This method relies on the structured format of DOS headers, where specific offsets (`0x0E` for SS and `0x10` for SP) universally denote the location of these stack settings.
    - **Importance:** Understanding the initial stack settings of a DOS executable is crucial for several reasons. It aids in comprehending how the program was designed to manage its memory and provides context for the execution environment prepared by the loader. For security analysis, anomalies or peculiar configurations in these values could indicate unconventional use of the stack, which might be a sign of exploitation techniques or obfuscation.

- **Final Vector:** The outcome is a simple dictionary containing numerical values for the Initial SP and SS, reflecting the stack's initial state. This information contributes to the broader feature set, enriching the dataset with details pertinent to the executable's runtime configuration.
- **Tooling:**
  - `struct`

F

Extraction of ELF Features

1. **Byte Histogram (ELF):**

   - **Extraction Process:** The process involves iterating through each segment within an ELF binary, identifying segments marked as executable by examining their flags. Utilizing the LIEF library, the process checks for the presence of the executable flag *ELF.SEGMENT_FLAGS.X* in the segment's flag attribute. For each executable segment encountered, its content is aggregated into a cumulative byte array. This array is then subjected to a bin count operation, performed with NumPy, to quantify the frequency of each byte value, ranging from 0 to 255, within these executable segments.
   - **Importance:** The Byte Histogram feature for ELF binaries is instrumental in understanding the distribution and frequency of byte values across executable segments. This insight is valuable for malware analysis, allowing researchers to detect patterns or anomalies indicative of malicious content. Moreover, it provides a basis for comparing and contrasting binaries, assisting in the identification of similarities that could link different samples to the same family or source.
   - **Final Vector:** The output of this process is a 256-dimensional numerical array, where each index corresponds to a byte value and its value represents the frequency of that byte across the executable segments of the ELF binary. This numerical representation serves as a direct and compact summary of the byte composition within the binary's executable parts.
   - **Tooling:**
     - `LIEF`
     - `NumPy`

2. **ByteEntropyHistogram (ELF):**

   - **Extraction Process:** This procedure begins by iterating through each segment in an ELF binary, utilizing the `lief` library to check for executable segments marked by the *ELF.SEGMENT_FLAGS.X* flag. For every segment determined to be executable, its byte content is transformed into a NumPy array. The entropy of each byte block within the window size is calculated, contributing to a two-dimensional histogram that maps byte values to their respective entropy levels as illustrated below:
     - Checking each segment for the executable flag.
     - Transforming segment content into a NumPy array.
     - Calculating entropy for blocks within the defined window size.
     - Aggregating these entropy values into a 2D histogram.
   - **Importance:** The ByteEntropy Histogram is vital for identifying the level of randomness or complexity within executable segments of ELF files, providing insights into potential obfuscation or encryption that could indicate malicious intent.
   - **Final Vector:** The output is a flattened numerical array derived from the 2D entropy histogram, offering a detailed representation of entropy distribution across the executable segments of the ELF file.

- **Tooling:**
  - `lief`
  - `NumPy`

3. **Section Information(ELF):**

   - **Extraction Process:** The ELF Section Information extraction begins with verifying the presence of the ELF binary. If present, the process retrieves the entry point from the binary's header. Subsequently, it iterates through each section within the binary, collecting detailed information such as the section's name, size, entropy, file offset, and properties. These properties are determined through a custom method that interprets the section's attributes.
   - **Importance:** Understanding the specifics of each section within an ELF binary is crucial for various aspects of cybersecurity analysis. It aids in identifying the purposes of different sections, detecting anomalies, and understanding the binary's layout. The entropy value, in particular, can indicate sections that may contain encrypted or compressed data, often a characteristic of malicious software.
   - **Final Vector:** The output is a structured object containing the entry point address in hexadecimal format and a list of dictionaries. Each dictionary holds information about a specific section, providing a clear and organized overview of the binary's sections.
   - **Tooling:**
     - `lief`

4. **Segments Information(ELF):**

   - **Extraction Process:** The process initiates with the parsing of the ELF file using the LIEF library. It meticulously gathers information from each section and segment within the ELF file. For sections, details such as name, size, and virtual address are compiled into a list. Similarly, for segments, the type, physical size, and virtual address are collated. This comprehensive extraction highlights the structural components of the ELF file, delineating the sections' and segments' roles and attributes.
   - **Importance:** The information garnered on segments is vital for a nuanced understanding of the ELF file's architecture and operational blueprint. Sections detail the file's division into areas with specific functionalities, while segments describe how these sections are mapped into memory. Analyzing this information can reveal insights into the file's layout, security measures, and potential areas of vulnerability.
   - **Final Vector:** The result of this extraction is a vector containing two lists: one for sections and another for segments. Each list comprises dictionaries with key information about each segment.
   - **Tooling:**
     - `lief`

5. **Import Information(EIF):**

   - **Extraction Process:** The extraction commences with an inspection of the ELF binary for dynamic and regular symbols utilizing the LIEF library. Dynamic symbols, often indicative of external dependencies or shared libraries, are identified by their undefined section index. Similarly, regular symbols that are external (having a section index of SHN_UNDEF) are cataloged. This process distinguishes between symbols used within the binary and those that are external, highlighting the binary's external dependencies.
   - **Importance:** This feature delineates the binary's interactions with external libraries and symbols, shedding light on its dependencies and potential external calls. Such information is crucial for understanding the binary's functionality, external interactions, and potential reliance on shared libraries, which could be vectors for exploitation or indicators of functionality.
   - **Final Vector:** The outcome is a dictionary that groups symbols into "DynamicSymbols" and "SymbolTable", depending on their nature. This structured approach allows for easy identification and analysis of the binary's external symbol dependencies.
   - **Tooling:**
     - `lief`

6. **General ELF File Information:**

   - **Extraction Process:** This involves a comprehensive examination of the ELF binary to gather general information, such as file size, presence of debug symbols, counts of exported and imported functions, existence of relocations, and the total number of symbols. The LIEF library is utilized to directly access this metadata from the ELF binary. If the binary data is unavailable (indicated by `self.lief_binary` being `None`), zeroes are provided as default values for these attributes.
   - **Importance:** The collected information offers an overview of the ELF file's structural and functional attributes. It provides insight into the binary's complexity, dependencies, and interaction with the operating environment. For example, debug symbols can indicate a non-production version, while the number of imports and exports shows the level of interaction with other binaries or libraries.
   - **Final Vector:** The output is a dictionary containing the binary's general attributes, including size, virtual size, and counts of various elements like debug symbols, exports, and imports. This dictionary is part of the comprehensive feature set for analysis or machine learning applications.
   - **Tooling:**
     - `lief`

7. **ELF Header Information:**

   - **Extraction Process:** The ELF binary's header information is meticulously extracted, focusing on both the general header and the specifics of program and section headers. This procedure employs the LIEF library to access and interpret the ELF binary's header, extracting key details such as file type, entry point, machine type, and header size. Additionally, it delves into each segment and section within the binary, collecting information on their types, addresses, sizes, and, for sections, their entropy.
   - **Importance:** Analyzing the ELF header offers critical insights into the binary's architecture, purpose, and operational details. For instance, the file type can indicate whether the binary is an executable or a library, the machine type reveals the intended hardware architecture, and the entry point addresses the binary's starting point upon execution. The details of program and section headers further enrich this understanding by outlining the binary's layout and how it is mapped into memory.
   - **Final Vector:** The result is a detailed dictionary encompassing the ELF header's broad attributes alongside nested lists for both program and section headers. Each of these lists contains dictionaries describing the individual headers, providing a comprehensive profile of the binary's structure.
   - **Tooling:**
     - `lief`

8. **String Features Extraction:**

   - **Extraction Process:**

     String features are extracted by conducting a thorough analysis of the binary's raw byte content, similar to PE Strings feature extraction. This process includes searching for ASCII strings, paths, URLs, registry entries, and MZ headers using regular expressions. The length of each discovered string is calculated to determine the average string length. Furthermore, a histogram is produced to illustrate the distribution of printable ASCII characters and measure the entropy of these characters in order to assess randomness within the strings.
   - **Importance:** Extracting string features provides crucial insights into the binary's content, revealing potential indicators of its functionality or malicious intent. Paths and URLs can point to network communication or file interaction, registry entries may suggest configuration changes, and the presence of MZ headers could indicate embedded executables. The entropy measurement helps identify obfuscation or encryption within the strings, which are common in malware to evade detection.

- **Final Vector:** The resulting feature set comprises numerical values and arrays, including the total number of strings, average string length, distribution of printable characters, total number of printable characters, entropy, and counts of paths, URLs, registry entries, and MZ headers found. This comprehensive set offers a multi-dimensional view of the binary's string usage patterns.
- **Tooling:**
  - `Python Regular Expressions (re)`
  - `NumPy`

9. **Entry Points Extraction (ELF):**

   - **Extraction Process:** The extraction of entry points from an ELF binary involves identifying the initial entry point and additional entry points presented by symbols and executable segments. The primary entry point is directly obtained from the binary's header. Furthermore, symbols of type *FUNC* (functions) are considered as potential entry points due to their executable nature. Lastly, segments flagged with execution permissions (*SEGMENT_FLAGS.X*) are scrutinized to locate additional entry points, which might indicate executable code segments.
   - **Importance:** Identifying entry points in an ELF binary is crucial for understanding the execution flow and the binary's structure. These entry points can reveal significant execution paths within the binary, including the primary starting point and functions intended for execution. This information is particularly valuable in malware analysis for uncovering potential malicious functionalities embedded within the binary.
   - **Final Vector:** The derived feature set consists of a list detailing each identified entry point, encompassing the main entry point, function-based entry points with their names and addresses, and addresses of executable segments. This collection of entry points enriches the feature vector with insights into the binary's execution structure.
   - **Tooling:**
     - `LIEF`

10. **Exit Points Extraction (ELF):**

    - **Extraction Process:** Identifying exit points within an ELF binary encompasses analyzing both dynamic symbols and executable segments. Dynamic symbols akin to import functions in PE binaries are scrutinized for common exit function names (e.g., `exit`, `_exit`, `abort`). Concurrently, executable segments flagged with `SEGMENT_FLAGS.X` are disassembled to search for calls to these exit functions, indicating potential termination points in the binary's execution flow.
    - **Importance:** Exit points are pivotal in understanding how a binary concludes its execution. In malware analysis, identifying such points can uncover how malware attempts to terminate processes, potentially to evade detection or disrupt system operations. This analysis contributes to a deeper understanding of the malware's behavior and potential impact on the infected system.
    - **Final Vector:** The resulting feature set is a list of identified exit points, categorized into dynamic symbols and specific locations within executable segments where exit functions are called. This structured approach to exit point extraction provides a detailed overview of how and where the binary may terminate its execution.
    - **Tooling:**
      - `LIEF`
      - `Capstone`

11. **Opcode Extraction (ELF):**

    - **Extraction Process:** The extraction of opcodes from an ELF file involves parsing the binary to identify executable sections, where opcodes, indicative of the binary's instruction set, reside. This task utilizes the `lief` library to parse the ELF file and determine its architecture (32-bit or 64-bit) based on the `machine_type` field. Executable sections are pinpointed for analysis, and a disassembly process is applied to each, facilitated by a disassembler. The mnemonics of the opcodes are then collected from these sections, offering a comprehensive overview of the executable instructions within the binary.

- **Importance:** Opcodes are fundamental to understanding the behavior of a binary. They represent the lowest level of instructions executed by the CPU, providing insights into the binary's functionality, potential malicious activities, and techniques used for obfuscation or evasion. Analyzing opcodes can reveal patterns or anomalies that aid in malware detection and classification.
- **Final Vector:** The resulting feature set comprises a list of all mnemonics derived from the executable sections of the ELF file. This list encapsulates the instruction set utilized by the binary, serving as a fingerprint for its operational behavior.
- **Tooling:**
  - `lief`
  - `Capstone`)

12. **Opcode Occurrences (ELF):**

    - **Extraction Process:** Counting opcode occurrences in ELF files begins with the identification of executable sections from which to extract opcodes. This process uses the `lief` library to parse the ELF file and discern its architecture, crucial for selecting the appropriate disassembly technique. For each executable section identified, the disassembly is carried out, and opcodes are extracted along with their associated addresses and parameters. This information is aggregated into a dictionary that maps each opcode to its occurrences, capturing both the frequency and context of each opcode within the binary.
    - **Importance:** Understanding the frequency and context of opcode occurrences is vital for malware analysis and classification. This data can highlight common execution patterns or rare instructions that might indicate sophisticated obfuscation or evasion mechanisms. Moreover, the detailed mapping of opcodes to their occurrences aids in constructing a comprehensive behavioral profile of the ELF binary, enriching the analysis with deeper insights into its operational mechanics.
    - **Final Vector:** The outcome is a detailed dictionary where keys are opcodes and values are lists of dictionaries detailing each occurrence's address and parameters. This structured approach retains the richness of the opcode data, facilitating nuanced analyses of the binary's behavior and characteristics.
    - **Tooling:**
      - `lief`
      - `Capstone`)

13. **Image Size(ELF):**

    - **Extraction Process:** To calculate the ELF image size, the `lief` library is employed to parse the ELF file and isolate loadable segments (`PT_LOAD`). This step is critical as loadable segments contribute directly to the image size when the binary is loaded into memory. After identifying all loadable segments, the size is approximated by summing the virtual address and the physical size of the final loadable segment, representing the end boundary of the ELF binary in memory.
    - **Importance:** Estimating the ELF image size is essential for understanding the memory footprint of the binary. It provides insights into how the binary is structured in memory, aiding in both routine analysis and security assessments. Particularly, it can help identify anomalies or features indicative of complex loading mechanisms or obfuscation tactics.
    - **Final Vector:** The process yields a single numerical value representing the estimated ELF image size. This value serves as a scalar feature within the broader set, contributing to the understanding of the binary's memory allocation and loading behavior.
    - **Tooling:**
      - `lief`

14. **Header Sizes(ELF):**

    - **Extraction Process:** The determination of ELF header sizes involves parsing the ELF binary using the `lief` library, from which the size of the ELF header is directly retrieved. Additionally, the total size of all program headers (segments) is calculated by aggregating their individual physical sizes. This process highlights the structural aspects of the ELF binary, specifically focusing on its header and the segments defined within.

- **Importance:** Knowing the sizes of the ELF header and program headers is crucial for understanding the binary layout and its allocation in memory. These metrics can provide insights into the binary's compilation and linking properties, potentially revealing optimization techniques or signs of manipulation indicative of malware.
- **Final Vector:** The result is encapsulated in a dictionary containing two key-value pairs: one for the ELF header size and another for the cumulative size of all program headers. This concise representation offers valuable information about the binary's overhead and memory usage during loading.
- **Tooling:**
  - `lief`

15. **GNU Stack Size(ELF):**
    - **Extraction Process:** The size of the GNU stack segment in an ELF binary is discerned through parsing the binary with the `lief` library. The process entails iterating over the program headers to identify the GNU_STACK segment. Once found, the segment's physical size (`physical_size`) is retrieved, providing an estimate of the stack reserve size allocated for the binary.
    - **Importance:** The GNU stack segment size is a critical attribute for understanding the memory footprint and security posture of an ELF binary. It not only reflects the memory allocation for the stack but also informs about stack execution permissions, which are pivotal for assessing the binary's vulnerability to stack-based buffer overflow attacks.
    - **Final Vector:** The outcome is a singular value representing the GNU stack segment's size within the ELF binary. This information augments the dataset with a direct measure of stack allocation, contributing to a comprehensive analysis of the binary's memory usage and security characteristics.
    - **Tooling:**
      - `lief`
16. **Heap Information(ELF):**
    - **Extraction Process:** The extraction of heap information from an ELF binary involves parsing the binary with `lief` and then applying heuristics to identify segments and sections relevant to the heap. Specifically, segments with `LOAD` type and writable (`W`) flags are considered potential candidates for heap segments, whereas sections with names containing ".heap" are considered indicative of heap sections. This method acknowledges the inherent ambiguity in precisely identifying heap-related areas due to the diverse ways heaps can be implemented or referenced within ELF binaries.
    - **Importance:** Heap information is crucial for understanding the memory layout and usage patterns of an ELF binary, especially with regards to dynamic memory allocation. Insights into heap size and configuration can reveal aspects of the binary's runtime behavior, potential for memory optimization, and susceptibility to heap-based exploits.
    - **Final Vector:** The extracted heap information results in a dictionary containing two key metrics: the size of the identified heap segment and the size of the identified heap section. This structured data provides a snapshot of heap allocation within the binary, enriching the analysis with specific memory usage insights.
    - **Tooling:**
      - `lief`
17. **Loading Flags(ELF):**
    - **Extraction Process:** The process involves parsing an ELF binary to identify the loading flags associated with each segment. Through the `lief` library, the binary's segments are enumerated, and their flags are examined to classify them based on their permissions: Read (`R`), Write (`W`), and Execute (`X`). These permissions are indicative of the segment's intended use and security implications, such as whether a segment can be written to or executed.

- **Importance:** Understanding the loading flags of ELF segments is vital for assessing the binary's security posture. Segments marked as executable but also writable could be potential vectors for code injection attacks, while read-only segments might contain critical constants or configuration data. Thus, analyzing these flags aids in identifying security risks and understanding the binary's structure.
- **Final Vector:** The extracted information culminates in a dictionary where each key represents a segment, mapped to its type and a descriptive string of its flags. This representation offers a concise overview of the segment permissions within the binary, facilitating quick security assessments and structural insights.
- **Tooling:**
  - `lief`

18. **Section Entropies(ELF):**

    - **Extraction Process:** To compute the entropy for each section in an ELF binary, the `lief` library is employed to parse the file and access its sections. Each section's content is then analyzed to calculate its entropy, a measure of randomness or complexity. This calculation is performed using a function that iterates over byte values, assessing the distribution of data within the section. The entropy for each section is computed and stored, along with statistics on the minimum, maximum, total, and mean entropy values for all sections.
    - **Importance:** Analyzing section entropies provides insights into the binary's content characteristics, such as the presence of encrypted or compressed data, which typically exhibit high entropy. This analysis aids in malware detection and understanding the binary's complexity, as sections with unusually high or low entropy may indicate obfuscation, packing, or unconventional data storage practices.
    - **Final Vector:** The output is a structured collection, typically a dictionary, where each key represents a section name, and the associated value is another dictionary detailing the entropy statistics for that section. This format encapsulates comprehensive entropy metrics for each section, enabling detailed analysis of the binary's content.
    - **Tooling:**
      - `lief`

19. **Kolmogoro Compression (ELF):**

    - **Extraction Process:** The compression of executable sections within an ELF binary is achieved by first parsing the binary using the `lief` library. This allows for the identification and extraction of sections flagged for execution. These sections are then concatenated into a single byte stream, which is subsequently compressed using the `zlib` library. The compression process aims to quantify the redundancy within the executable code, providing an indirect measure of the binary's complexity.
    - **Importance:** This compression metric offers valuable insights into the binary's nature, particularly in the context of malware analysis. Highly compressible code suggests a degree of redundancy or simplicity, while less compressible sections may indicate complex, potentially obfuscated code. This distinction is crucial for identifying sophisticated malware that employs evasion techniques.
    - **Final Vector:** The resulting compression is represented as a single value, denoted in kilobytes (KB), reflecting the compressed size of the executable sections. This scalar value contributes to the overall feature set, providing a succinct yet powerful indicator of the binary's complexity and potential obfuscation.
    - **Tooling:**
      - `lief`
      - `zlib`

G
Embedding Validation

Comapratíve Validation Of All Embeddings

Table G.1: Accuracy Differences Relative to the Combined Embeddings

Outer Fold	Pseudo-Static (Numerical)	Pseudo-Static (Complex)	Dynamic	Image
1	-6.95%	-8.04%	-7.79%	-0.20%
2	-7.57%	-8.19%	-7.79%	-0.03%
3	-7.76%	-8.33%	-7.33%	+0.20%
4	-7.07%	-8.10%	-7.84%	-0.22%
5	-6.73%	-8.77%	-7.09%	+0.52%
6	-6.90%	-7.17%	-6.82%	+0.84%
7	-7.50%	-9.35%	-7.83%	-0.24%
8	-7.25%	-7.77%	-7.13%	+0.08%
9	-7.16%	-8.03%	-7.56%	+0.33%
10	-7.53%	-8.14%	-7.56%	-0.08%
Average	**-7.34%**	**-8.38%**	**-7.60%**	**+0.03%**
Std Dev	**0.28%**	**0.64%**	**0.32%**	**0.25%**

All three embeddings except image show significant lower accuracy across folds compared to Combined embedding , as determined by the Mann-Whitney U test ($\alpha < 0.05$).

Combined Embeddings Validation

Table G.2: Validation of Concatenated Model

Outer Fold	Dummy Accuracy (%)	Concatenated Model Accuracy (%)
1	19.36	93.73
2	19.36	93.99
3	19.35	93.72
4	19.35	93.83
5	19.35	93.51
6	19.36	93.00
7	19.36	93.97
8	19.36	93.58
9	19.36	93.64
10	19.36	93.95
Average	19.36	93.84
Std Dev	0.01	0.17

Pseduo-Static(Complex) Embeddings Validation

Table G.3: Validation for Pseduo-Static(Complex) Analysis

Outer Fold	Dummy Accuracy (%)	Real Model Accuracy (%)
1	19.36	85.69
2	19.36	86.80
3	19.35	86.39
4	19.35	86.73
5	19.35	85.74
6	19.36	86.83
7	19.36	85.62
8	19.36	86.81
9	19.36	86.61
10	19.36	86.81
Average	**19.36**	**86.39**
Std Dev	**0.01**	**0.48**

Pseduo-Static(Numerical) Embeddings Validation

Table G.4: Validation for Pseudo-Static (Numerical) Analysis

Outer Fold	Dummy Accuracy (%)	Logistic Model Accuracy (%)
1	19.36	87.78
2	19.36	87.42
3	19.35	86.96
4	19.35	87.76
5	19.35	87.78
6	19.36	87.10
7	19.36	87.47
8	19.36	87.33
9	19.36	87.76
10	19.36	87.42
Average	**19.36**	**87.51**
Std Dev	**0.01**	**0.28**

Dynamic Embeddings Validation

Table G.5: Validation for Dynamic Analysis

Outer Fold	Dummy Accuracy (%)	Logistic Model Accuracy (%)
1	19.36	86.94
2	19.36	87.20
3	19.35	87.53
4	19.35	86.99
5	19.35	87.42
6	19.36	87.18
7	19.36	87.14
8	19.36	87.45
9	19.36	87.08
10	19.36	87.39
Average	19.36	87.23
Std Dev	0.01	0.19

Image Embeddings Validation

Table G.6: Validation for Image Analysis

Outer Fold	Dummy Accuracy (%)	Logistic Model Accuracy (%)
1	19.36	94.53
2	19.36	94.96
3	19.35	94.92
4	19.35	94.61
5	19.35	95.03
6	19.36	94.84
7	19.36	94.73
8	19.36	94.66
9	19.36	94.97
10	19.36	94.87
Average	19.36	94.91
Std Dev	0.01	0.16

H

Downstream Tasks

Benign/Malicious Classification

Table H.1: Detailed Accuracy Comparison Across Folds

Outer Fold	Model Accuracy (%)	Standard Model Accuracy (%)
1	99.99	65.23
2	99.99	65.23
3	99.98	65.23
4	99.99	65.22
5	100.00	65.22
6	99.99	65.22
7	99.98	65.23
8	99.99	65.23
9	100.00	65.23
10	99.99	65.23
Mean CV Accuracy	99.99	65.23
Std Dev	0.01%	0.01%

Time Divergence Analysis Validation

Table H.2: Embeddings binned into years and validated

Outer Fold	Dummy Classifier		Logistic Model	
	Accuracy	MAE	Accuracy	MAE
1	0.0740	5.0677	0.6697	0.9967
2	0.0740	5.0677	0.6661	1.0174
3	0.0740	5.0677	0.6614	1.0290
4	0.0740	5.0676	0.6712	0.9794
5	0.0740	5.0674	0.6732	0.9763
6	0.0740	5.0676	0.6681	0.9976
7	0.0740	5.0676	0.6717	0.9969
8	0.0740	5.0671	0.6640	1.0108
9	0.0741	5.0669	0.6686	1.0235
10	0.0741	5.0665	0.6654	0.9958
Average	0.0740	5.0674	0.6679	1.0023
Std Dev	0.0000	0.0004	0.0035	0.0167

H.1. VirusTotal Timestamp Validation

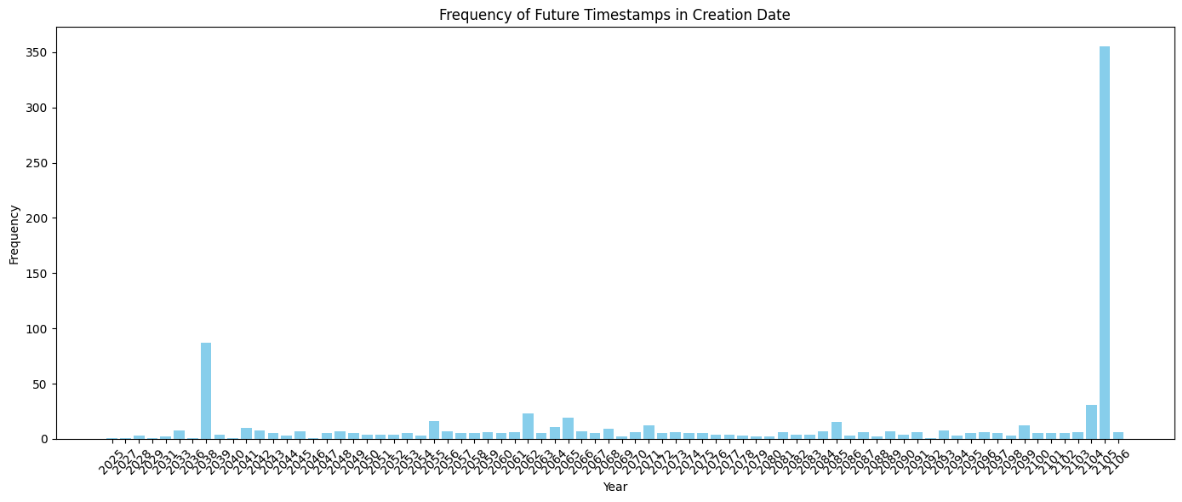


Figure H.1: Future Timestamps

I

Hyperparameters & Validation

I.1. Hyperparameter Configuration for embedding validation

Table I.1: Grid Search Parameters

Parameter Type	**Values Tested**
L1 Regularization	$0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0, 1$
L2 Regularization	$0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0, 1$
initial Learning Rate	$1, 0.1, 0.01, 0.001, 0.0001, 0.00001$

The optimal hyperparameters for various embeddings and temporal classification were identified as follows: L1 regularization and initial learning rate of 0.0001 and 0.01, respectively, for image embeddings; 0.00001 and 0.001 for pseudo-static and dynamic analysis embeddings; 0.001 and 0.01 for combined embeddings; and the same values for temporal classification using years.

I.2. Hyperparameter Configuration for t-SNE, UMAP and PCA

Table I.2: Grid Search Values for t-SNE, UMAP, and PCA Hyperparameters

Parameter	**t-SNE Values**	**UMAP Values**	**PCA Values**
Number of Components	{2}	{2}	{2}
Perplexity	{5, 30, 50, 100}	–	–
Number of Iterations	{1000, 3000, 5000}	–	–
Learning Rate	{10, 100, 200}	–	–
Number of Jobs	{-1, 1, 2, 4}	–	–
Number of Neighbors	–	{5, 10, 15, 30}	–
Minimum Distance	–	{0.1, 0.3, 0.5, 0.7}	–
Metric	–	{Euclidean, Manhattan, Cosine}	–
Power Iteration Normalization	–	–	{0, 1}
Number of Oversamples	–	–	{5, 10, 20}
SVD Solver	–	–	{auto, full, arpack}

J Phylogenetic Tree Algorithms

J.1. Validation

J.1.1. Virustotal Timestamps

Algorithm 1: Validate Chronological Order of Phylogenetic Tree Leaves

```
Input: Phylogenetic tree, timestamps associated with each leaf
Output: Proportion of correctly ordered leaf pairs
$correct\_pairs \leftarrow 0$
$total\_comparisons \leftarrow 0$
foreach $leaf$ in tree do
    $ancestor \leftarrow get\_direct\_ancestor(leaf)$
    $leaves\_from\_ancestor \leftarrow get\_leaves\_from\_ancestor(ancestor)$
    foreach $(leaf1, leaf2)$ in $leaves\_from\_ancestor$ do
        $distance1 \leftarrow get\_distance(ancestor, leaf1)$
        $distance2 \leftarrow get\_distance(ancestor, leaf2)$
        $timestamp1 \leftarrow timestamps[leaf1]$
        $timestamp2 \leftarrow timestamps[leaf2]$
        if $(distance1 < distance2$ and $timestamp1 < timestamp2)$ or $(distance1 > distance2$
          and $timestamp1 > timestamp2)$ then
            $correct\_pairs \leftarrow correct\_pairs + 1$
        end
        $total\_comparisons \leftarrow total\_comparisons + 1$
    end
end
$accuracy \leftarrow correct\_pairs / total\_comparisons$
return $accuracy$
```

J.1.2. Embedding Drift Analysis

Algorithm 2: Calculate Mutation Rates

```
Input: distances, family_variant_binning
Output: mutation_rates_results
Initialize family_min_max_rates as an empty dictionary;
Set global_min_rate to $\infty$;
Set global_max_rate to $-\infty$;
Initialize global_min_details and global_max_details to None;
for each family and its associated years_data in family_variant_binning do
    Sort family_years based on year;
    Initialize family_min_rate to $\infty$ and family_max_rate to $-\infty$;
    Initialize family_min_details and family_max_details to None;
    for each year in family_years except the last do
        for each next_year greater than current year in family_years do
            for each variant in years_data[year] do
                Create taxon1 as family_variant if variant is not ”0”, else just family;
                for each next_variant in years_data[next_year] do
                    Create taxon2 similarly as for taxon1;
                    if distance between taxon1 and taxon2 exists in distances then
                        Retrieve the distance using the Euclidean function;
                        if distance is a new min or max then
                            Update family and global min/max rates and details;
                        end
                    end
                end
            end
        end
    end
    Store family-specific min and max rates and details in family_min_max_rates;
end
Compile the results into mutation_rates_results with family and global rates and details;
return mutation_rates_results;
```

J.2. Inter-family analysis algorithms

J.2.1. Without Outlier Thresholding

Algorithm 3: Analyze Phylogenetic Tree

```
Input: Phylogenetic tree file path tree_path
Output: Graph representing evolutionary relationships
Function AnalyzeTree(tree_path):
    tree, family_to_leaves ← load_tree_and_aggregate_leaves(tree_path)
    distances ← CalculateDistances(tree, family_to_leaves)
    graph ← BuildGraph(distances)
    SimplifiedGraph ← SimplifyGraph(graph)
return
Function load_tree_and_aggregate_leaves(tree_path):
    tree ← LoadTree(tree_path)
    family_to_leaves ← {}
    foreach leaf in tree do
        family ← ExtractFamilyName(leaf)
        if not family in family_to_leaves then
            family_to_leaves[family] ← []
        end
        family_to_leaves[family].append(leaf)
    end
    return tree, family_to_leaves
return
Function CalculateDistances(tree, family_to_leaves):
    distances ← {}
    families ← GetFamilyList(family_to_leaves)
    foreach family1 in families do
        foreach family2 in families if family2 ≠ family1 then

        end
         do
            MRCA ← FindMRCA(family1, family2)
            distances1 ← [CalculateDistance(MRCA, leaf) for leaf in family_to_leaves[family1]]
            distances2 ← [CalculateDistance(MRCA, leaf) for leaf in family_to_leaves[family2]]
            median1 ← CalculateMedian(distances1)
            median2 ← CalculateMedian(distances2)
            distances[(family1, family2)] ← (median1, median2)
        end
    end
    return distances
return
Function BuildGraph(distances):
    graph ← new Graph()
    foreach (family1, family2, distance) in distances do
        if distance[0] < distance[1] then
            AddEdge(graph, family1, family2, distance[0])
        else
            AddEdge(graph, family2, family1, distance[1])
        end
    end
    return graph
return
Function SimplifyGraph(graph):
    new_graph ← new Graph()
    foreach node in graph do
        min_edge ← FindMinEdge(node)
        AddEdge(new_graph, node, min_edge.target, min_edge.weight)
    end
    return new_graph
return
Main execution block:
    tree_path ← "tree.nwk"
    tree, family_to_leaves ← load_tree_and_aggregate_leaves(tree_path)
    distances ← CalculateDistances(tree, family_to_leaves)
    G ← BuildGraph(distances)
```

J.2.2. With Outlier Thresholding

Algorithm 4: Analyze Phylogenetic Tree

```
Input: Phylogenetic tree file path tree_path
Output: Graph representing evolutionary relationships
Function AnalyzeTree(tree_path):
    tree, family_to_leaves ← load_tree_and_aggregate_leaves(tree_path)
    filtered_family_to_leaves ← remove_intra_family_outliers_iqr(tree, family_to_leaves)
    distances ← CalculateDistances(tree, filtered_family_to_leaves)
    dump_distances_to_json(distances)
    graph ← BuildGraph(distances)
    SimplifiedGraph ← SimplifyGraph(graph)
return
Function load_tree_and_aggregate_leaves(tree_path):
    tree ← LoadTree(tree_path)
    family_to_leaves ← {}
    foreach leaf in tree do
        family ← ExtractFamilyName(leaf)
        if not family in family_to_leaves then
            family_to_leaves[family] ← []
        end
        family_to_leaves[family].append(leaf)
    end
    return tree, family_to_leaves
return
Function remove_intra_family_outliers_iqr(tree, family_to_leaves):
    filtered_family_to_leaves ← {}
    foreach family, leaves in family_to_leaves do
        if len(leaves) $\geq$ 2 then
            distances ← ComputeLeafToLeafDistances(tree, leaves)
            median_distances ← Median(distances)
            IQR ← ComputeIQR(median_distances)
            filtered_leaves ← FilterLeavesUsingIQR(median_distances, IQR)
            filtered_family_to_leaves[family] ← filtered_leaves
        end
        else
            filtered_family_to_leaves[family] ← leaves
        end
    end
    return filtered_family_to_leaves
return
Main execution block:
    tree_path ← "tree.nwk"
    tree, family_to_leaves ← load_tree_and_aggregate_leaves(tree_path)
    filtered_family_to_leaves ← remove_intra_family_outliers_iqr(tree, family_to_leaves)
    distances ← CalculateDistances(tree, filtered_family_to_leaves)
    G ← BuildGraph(distances)
```

K
Correlation Analysis

K.1. Global Structure

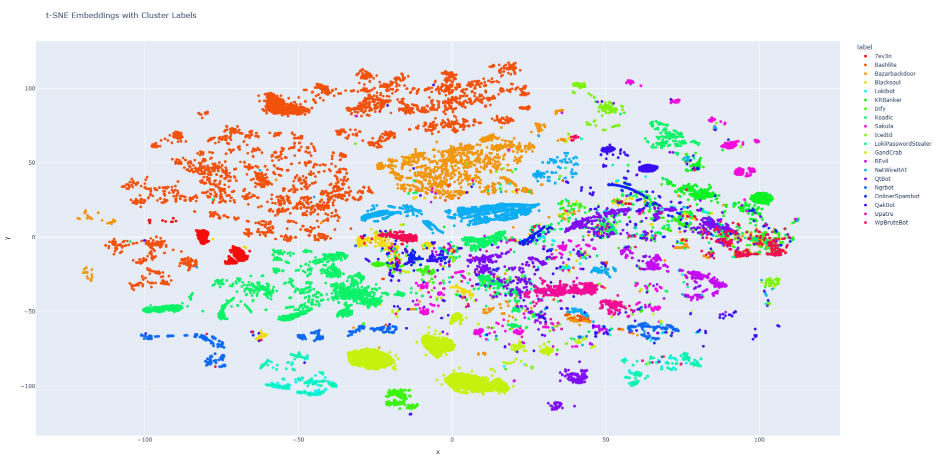


Figure K.1: Embeddings Projected to 2D using t-SNE

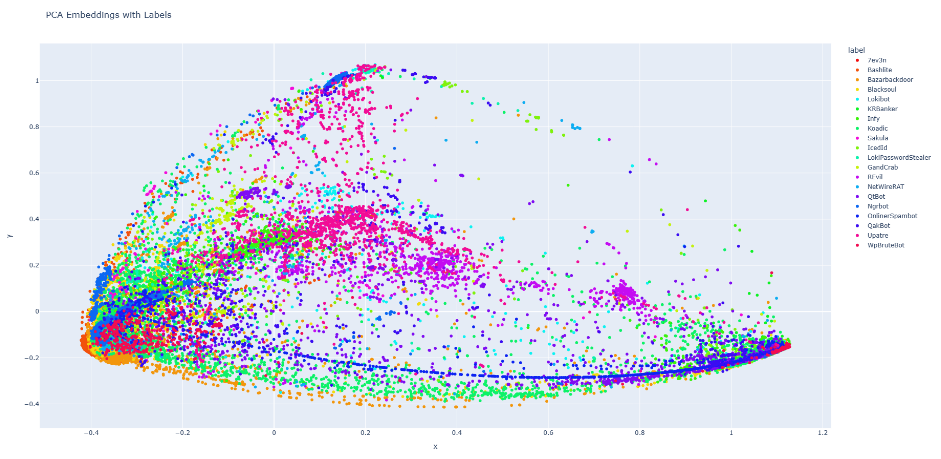


Figure K.2: Embeddings Projected to 2D using PCA

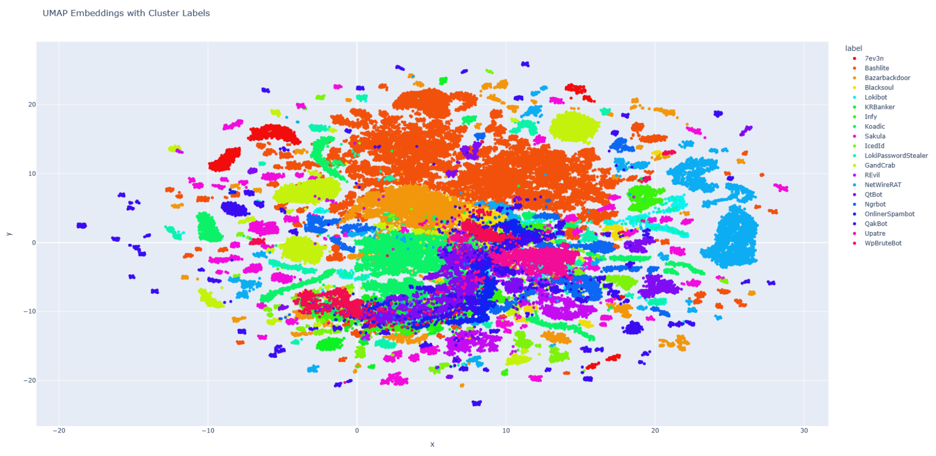


Figure K.3: Embeddings Projected to 2D using UMAP

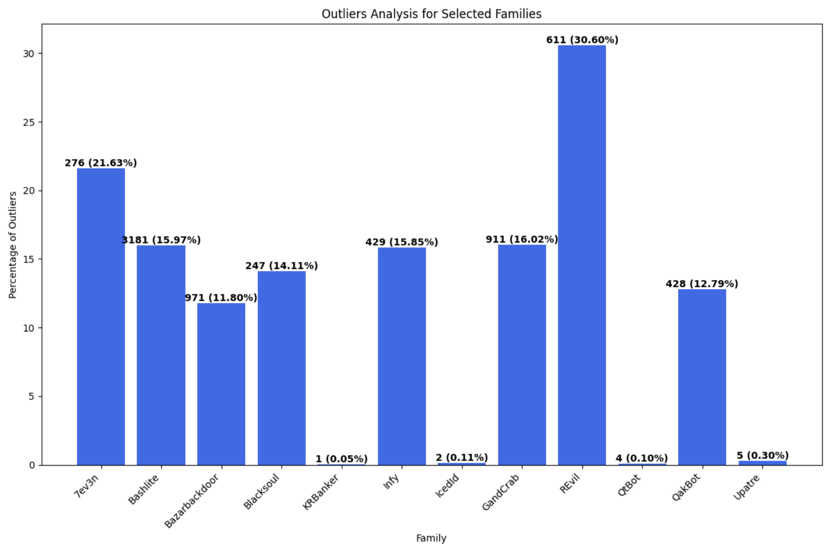


Figure K.4: Outliers based on lateral distance of the Tree highlights that some families have considerably more outliers than others

K.2. Discrepancies

K.2.1. 7ev3n

The t-SNE and UMAP visualizations demonstrate a clear clustering within the 7ev3n malware family, depicted in red, indicating a high degree of similarity across its members. Although two clusters are evident in the t-SNE plot, the intra-family lateral distance histogram in Figure K.5 indicates a single mode, possibly suggesting that the observed separation in t-SNE does not reflect significant differences in the data's higher-dimensional structure. It is important to note that while t-SNE effectively captures local proximities, it may not accurately represent more distant relationships.

In contrast, the UMAP plot in Figure K.7 shows closer proximity between clusters, consistent with the single-mode distribution observed in Figure K.5, and suggests that Neighbour Joining method used for phylogenetic analysis identifies these clusters as closely related. Moreover, when looking at the relative outliers from figure K.4, it is also apparent that 20% of these samples are considered outliers. While these outliers may not be immediately obvious in these 2D projections, a closer examination of the t-SNE plot of 7ev3n as shown in Figure K.6

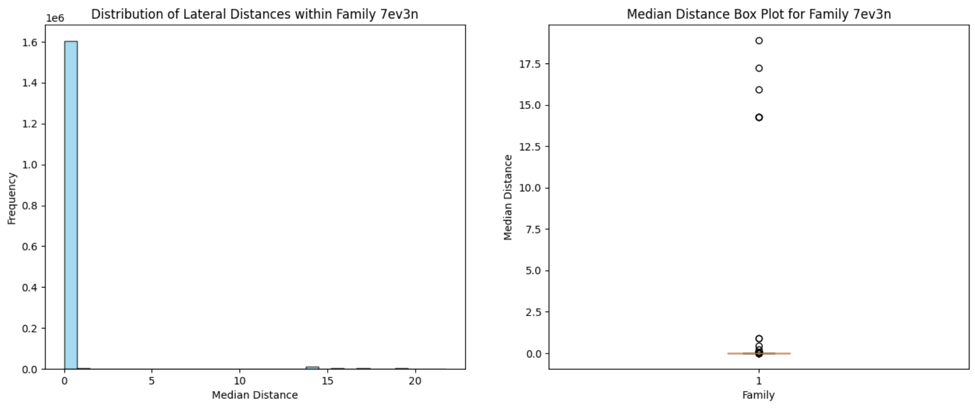


Figure K.5: Distribution of Lateral Distances for 7ev3n show one main mode

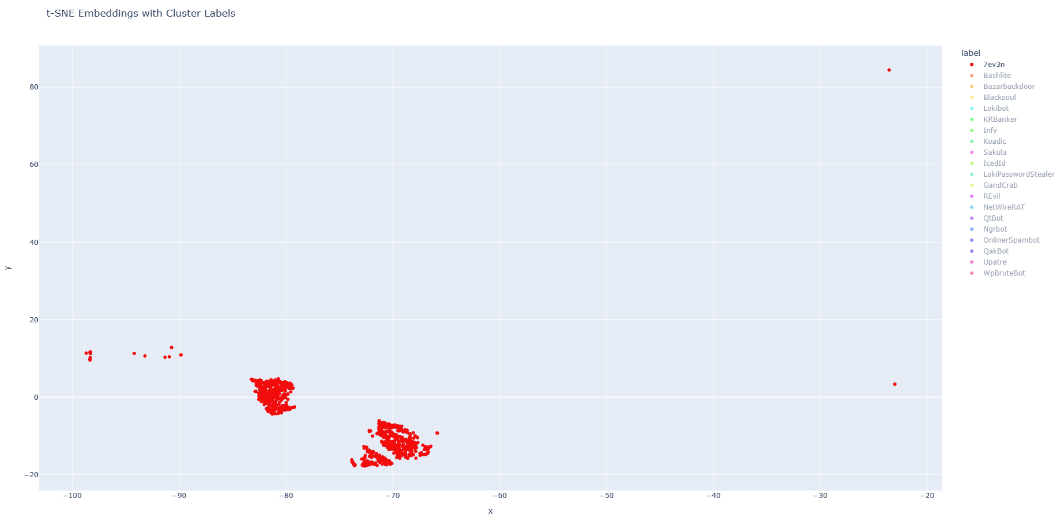


Figure K.6: t-SNE Plot for 7ev3n show two distinct clusters

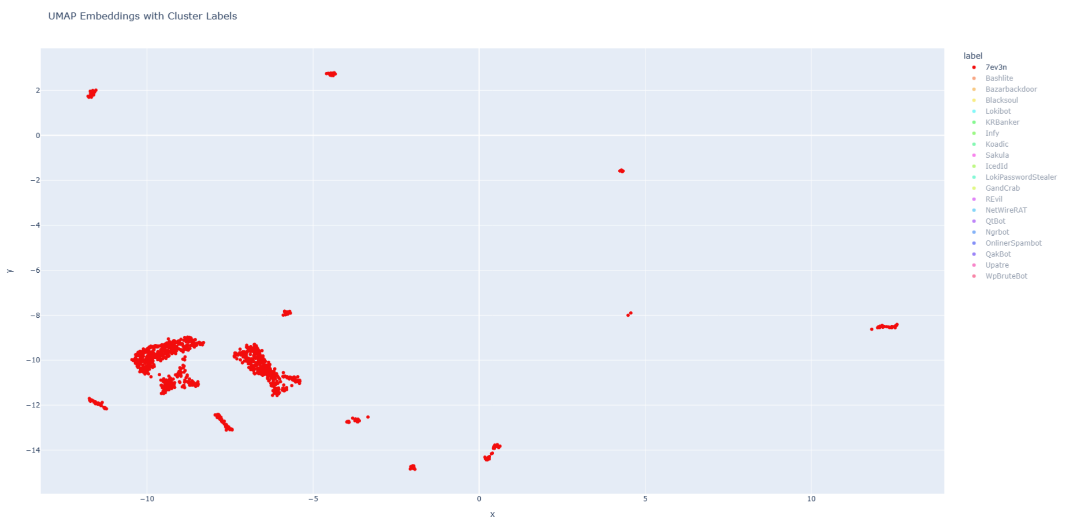


Figure K.7: UMAP Plot for 7ev3n shows 2 to 3 clusters while the rest could be interpreted as outliers

K.2.2. WpBruteBot

The WpBruteBot family, as depicted in Figure K.8, demonstrates three to four modes in the lateral distance distribution. However, the t-SNE visualization in Figure K.9 only reveals two prominent clusters. In contrast, the UMAP plot in Figure K.10 suggests the presence of two main clusters, with additional densities at -6 and 20 that could potentially be interpreted as separate clusters. Nevertheless, we suspect these are merely outliers, especially since the PCA visualization in Figure K.11 predominantly shows two dense regions. This discrepancy might be attributed to the neighbor joining method not classifying these points as outliers in its own feature space.

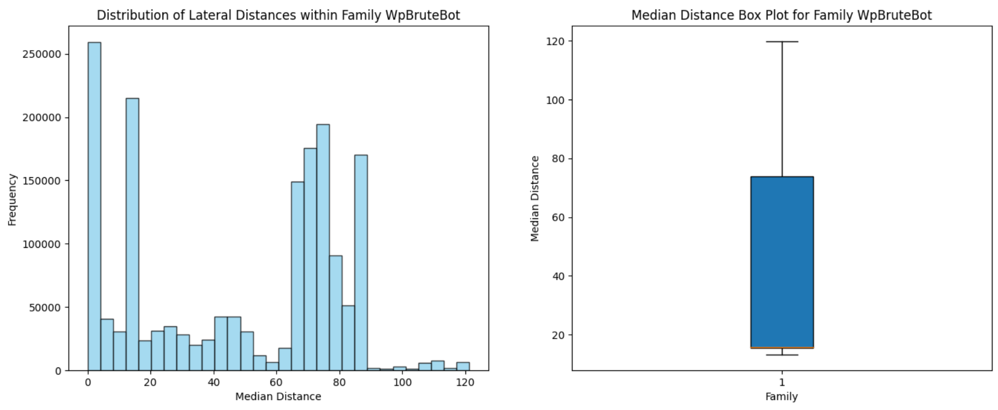


Figure K.8: Distribution of Lateral Distances for WpBruteBot shows 3 to 4 modes

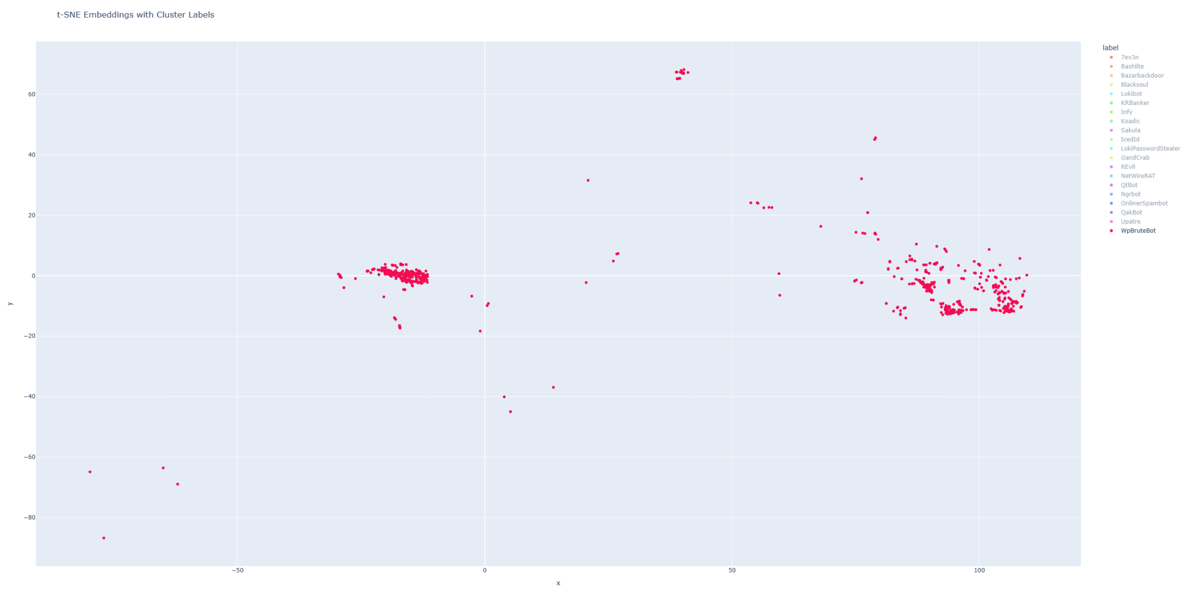


Figure K.9: t-SNE Plot for WpBruteBot show two main clusters

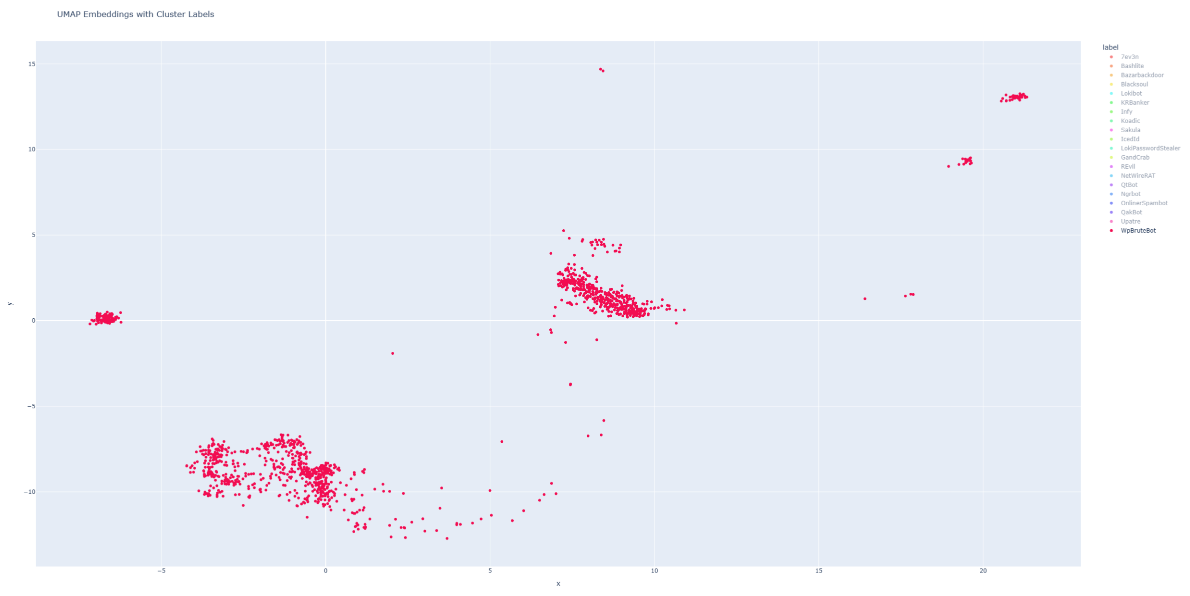


Figure K.10: UMAP Plot for WpBruteBot show two to three clusters

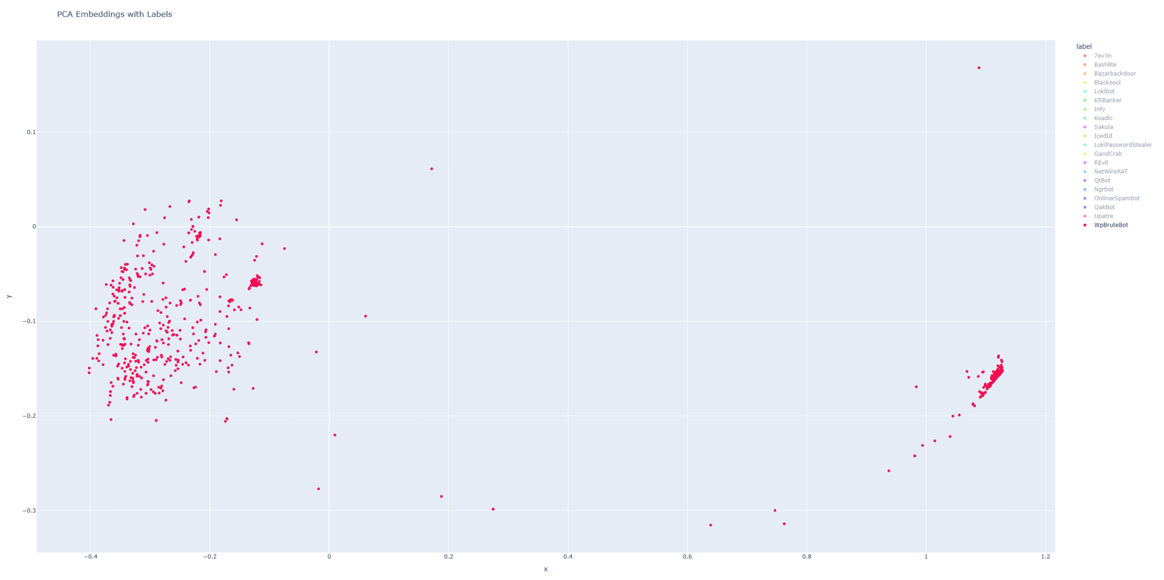


Figure K.11: PCA Plot for WpBruteBot show two clusters

K.2.3. IcedId

Similarly, the IcedId family presents a complex scenario. Figure K.12 suggests the presence of three to four modes in the lateral distance distribution. However, the t-SNE visualization (Figure K.13) primarily shows two clusters, with potential additional clusters at x-axis values of 102 and 40, though these may be outliers. The UMAP plot (Figure K.14) indicates the possibility of three clusters located at x-axis values of 15, 5, -5, and 10, yet this remains speculative. The PCA plot (Figure K.15) displays a seemingly random dispersion of points, with two denser areas possibly representing clusters at 1 on the x-axis and -0.2. These observations suggest a pattern where lateral distance modes may not align perfectly with visual clusters, possibly due to the method's treatment of outliers, underscoring a recurring theme across different visualization techniques.

It is challenging to determine if the modes in the lateral distance distributions are influenced by what might be considered outliers in the visual projections, a question that recurs across various families analyzed. This ambiguity points to the complexity of aligning high-dimensional genetic relationships with their 2D visual representations.

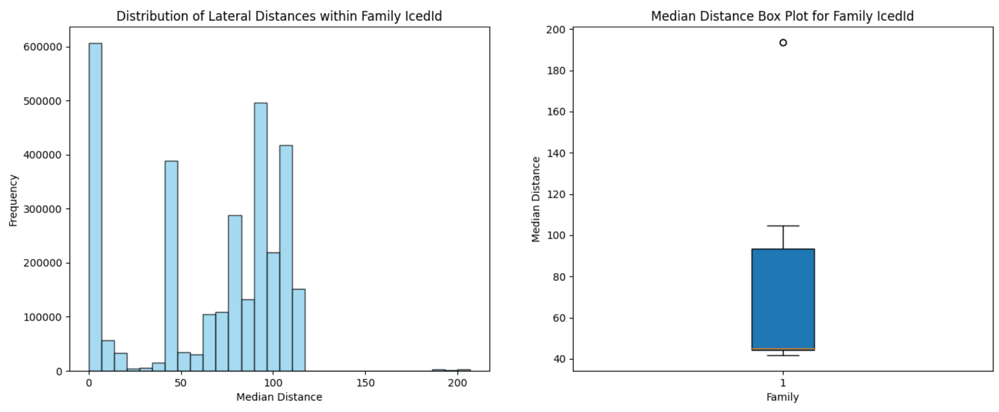


Figure K.12: t-SNE Plot for IcedId shows 3 to 4 main modes

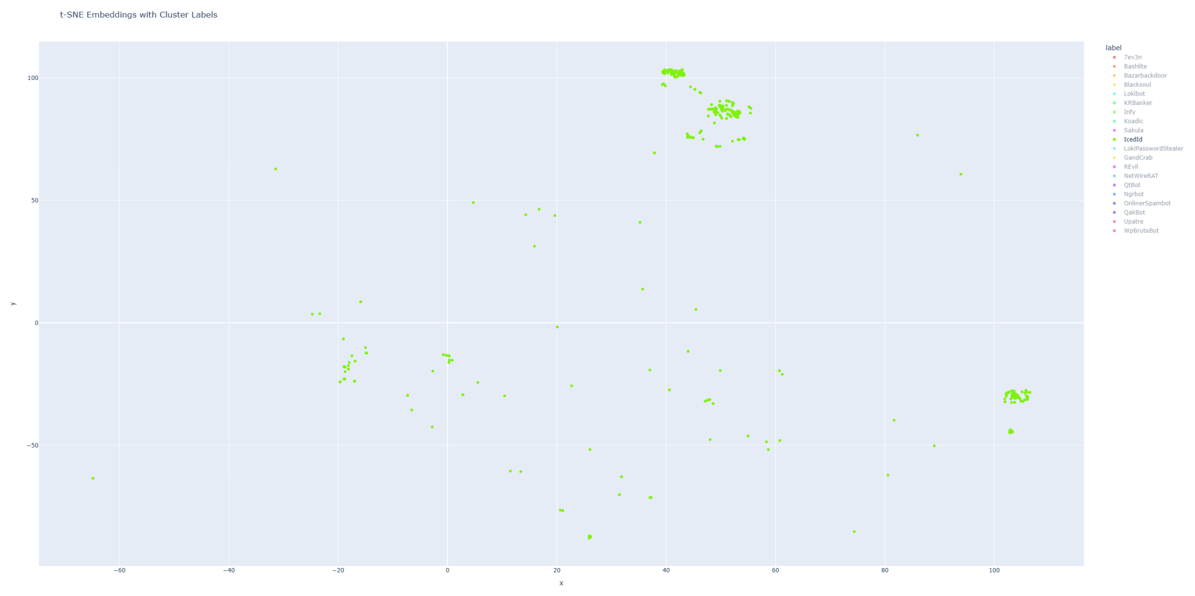


Figure K.13: t-SNE Plot for IcedId show two main clusters

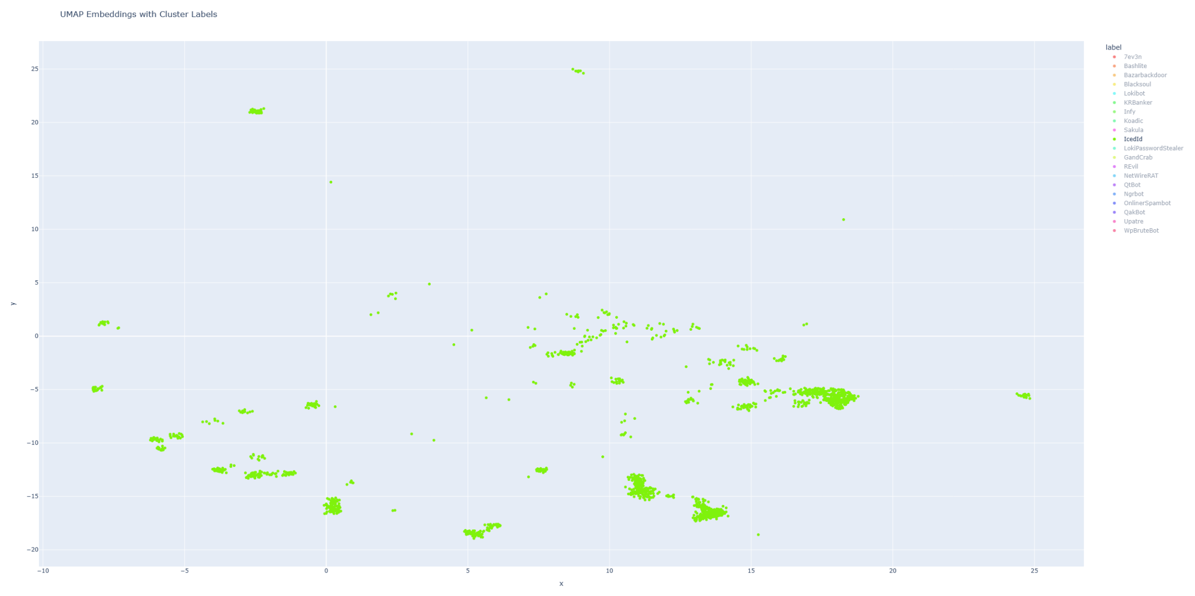


Figure K.14: t-SNE Plot for IcedId show multiple clutsters, possibly 3

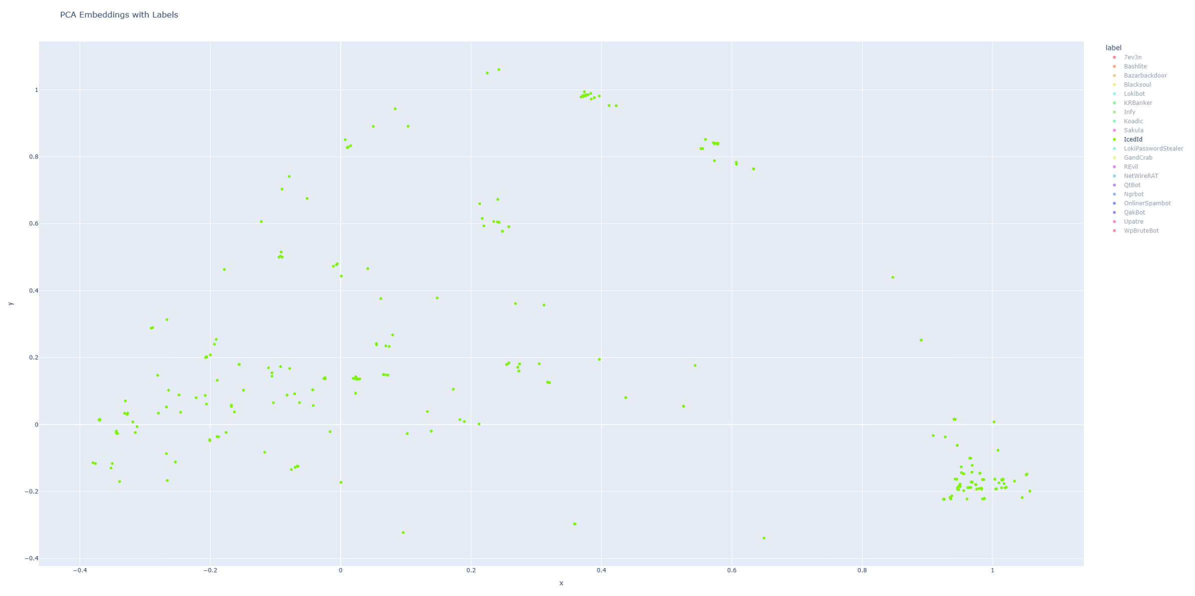


Figure K.15: PCA Plot for IcedId shows possibly 2 clusters

K.3. Tightly Clustered Families

K.3.1. Bashlite

Figure K.16 shows a concentration of lateral distances near zero for the Bashlite family, indicating a close relationship within the phylogenetic tree. This pattern is also evident in the t-SNE and UMAP visualizations, reinforcing our findings. However, the box plot reveals a significant presence of outliers, representing 15.97% of observations. A closer look at the t-SNE plot in Figure K.17 confirms that several samples are dispersed away from the core cluster.

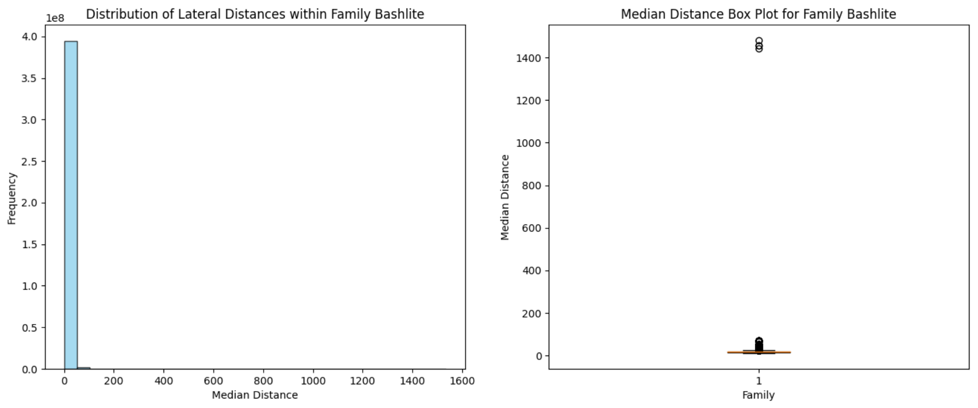


Figure K.16: Distribution of for bashlite shows one main mode

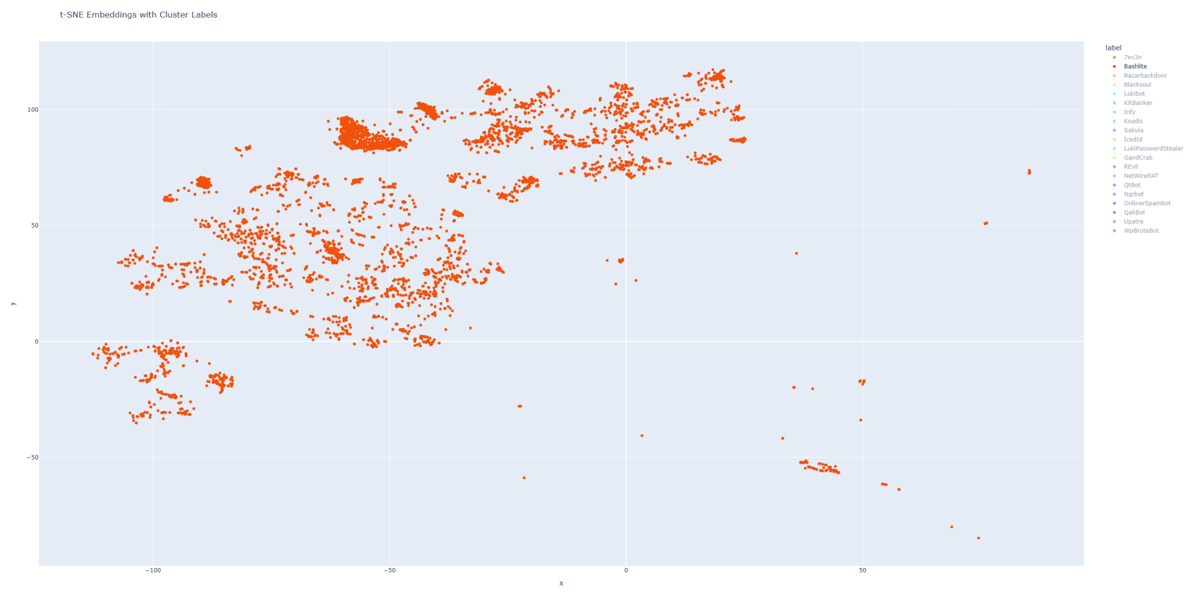


Figure K.17: t-SNE Lateral Distances for Bashlite shows one main cluster

K.4. Heterogeneous Clusters

K.4.1. Bazarbackdoor

In contrast to the families previously analyzed, BazaarBackdoor manifests a distinct clustering pattern, as seen in Figure K.18. There is a pronounced primary cluster, whereas a secondary grouping lacks clear definition within the t-SNE and UMAP frameworks (Figures K.1 and K.3). A closer look at the lateral distance histogram in Figure K.18 unveils two notable peaks: one around the value of 10, and a second, subtler peak near 83, suggesting a potential cluster with a lower density at the higher value. Delving deeper into BazaarBackdoor's specific t-SNE portrayal (as presented in Figure K.19), the presence of outliers becomes apparent. These findings coincide with the lateral distance distribution, hinting at a more intricate intra-family structure than initially apparent.

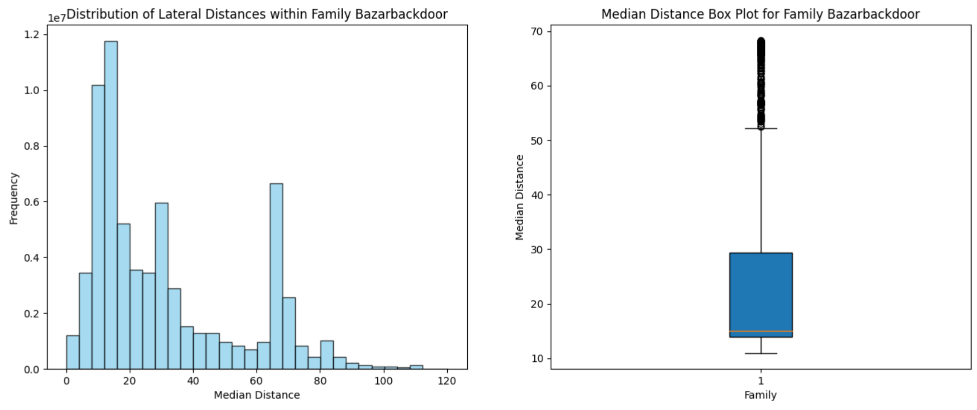


Figure K.18: Distribution of Lateral Distances for Bazarbackdoor shows two modes

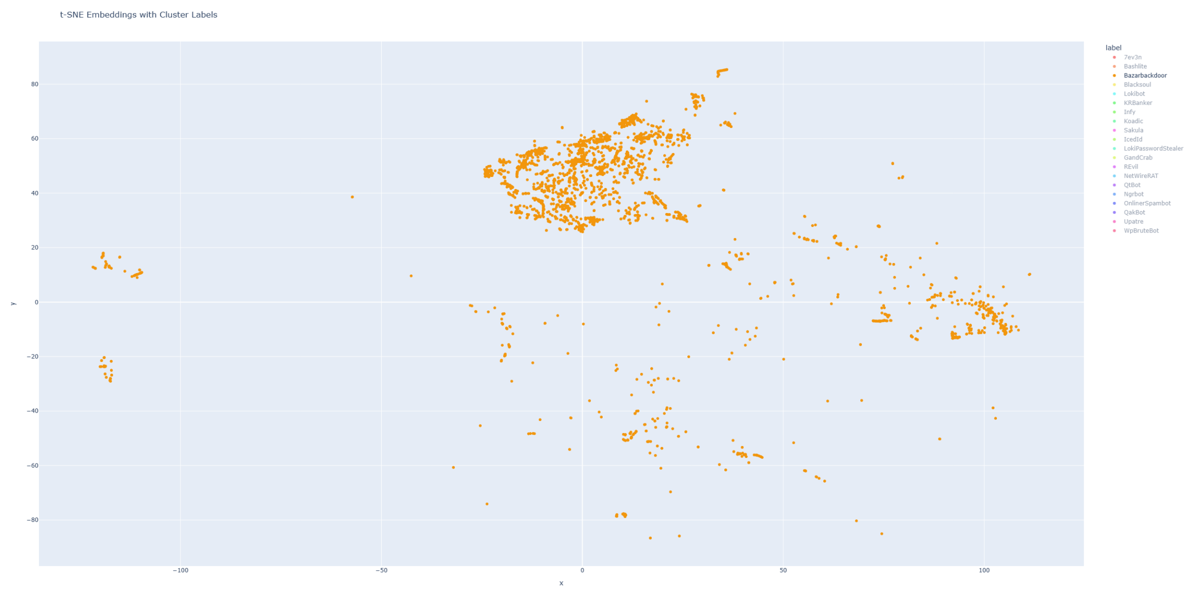


Figure K.19: t-SNE Plot for Bazarbackdoor shows two main clusters

K.4.2. KRBanker

Similarly, Inspection of the KRBanker family reveals distinct clustering patterns in Figure K.21, with a dense cluster positioned at 100 and a less dense one at 80. These observations closely correspond to the two modes identified in Figure K.20.

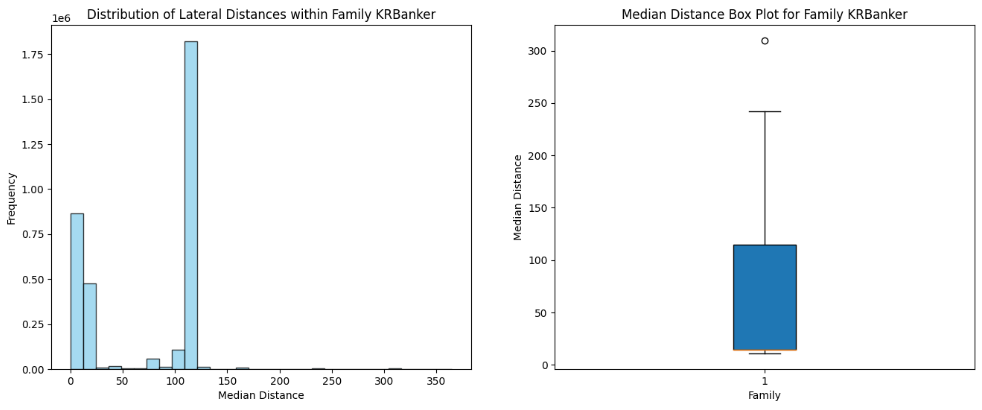


Figure K.20: Distribution of Lateral Distances for KRBanker shows two main modes

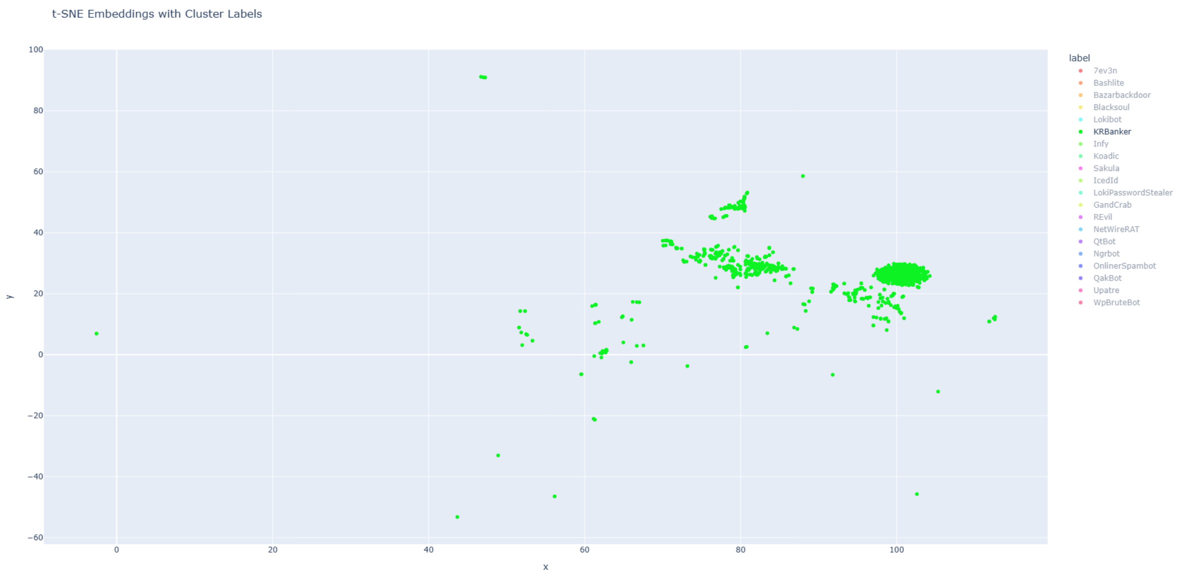


Figure K.21: t-SNE Plot for KRBanker shows two main clusters

K.4.3. GrandCrab

The GrandCrab family presents an intriguing tri-cluster formation in its t-SNE visualization, as observed in Figure K.23. Despite the clear emergence of three clusters in the visual representation, the distribution of lateral distances for GrandCrab, illustrated in Figure K.22, primarily shows two concentrated modes: one at the zero mark and another near 10. While a third cluster isn't directly discernible, the presence of points around the 40 mark on the x-axis may hint at its existence. This could account for the tri-cluster arrangement seen in the t-SNE plot, with the notable outliers identified in the corresponding boxplot (Figure K.22) possibly representing peripheral samples surrounding these clusters.

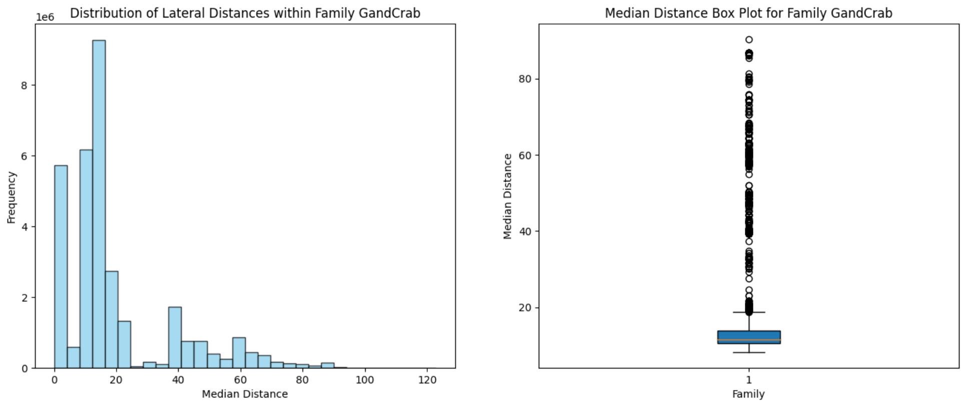


Figure K.22: Distribution of Lateral Distances for GradCrab shows 3 modes

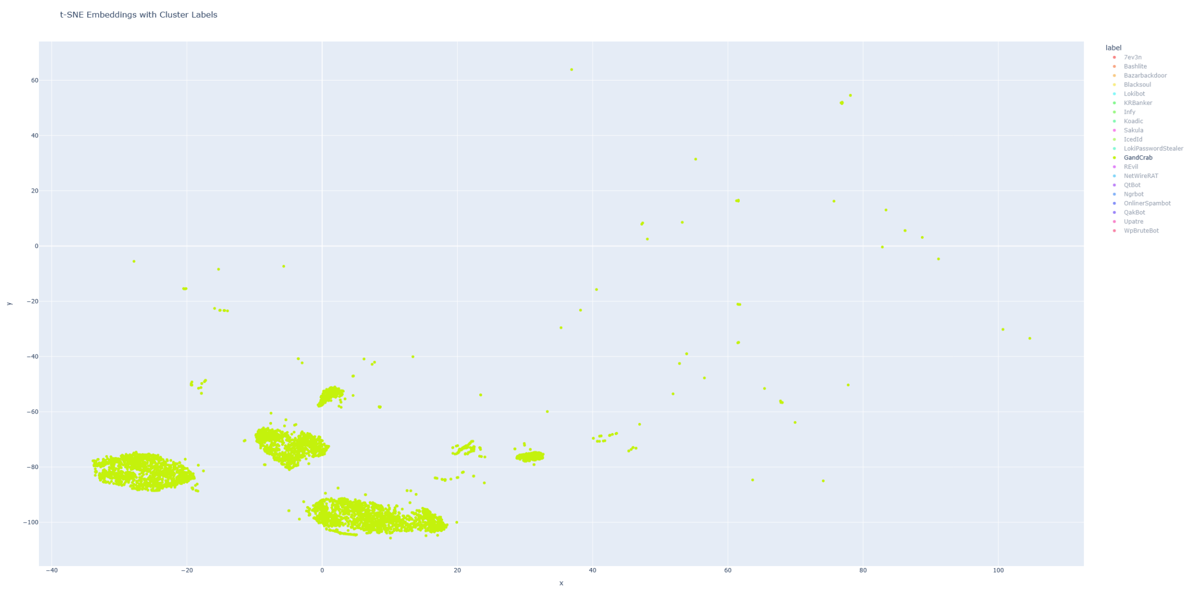


Figure K.23: t-SNE Plot for GradCrab shows 3 clusters

K.5. Wide-Spread Clusters

K.5.1. Lokibot

Lokibot exhibits similar characteristics, yet its lateral distances reveal two distinct modes, one near 0 and another around 70, as depicted in Figure K.24.

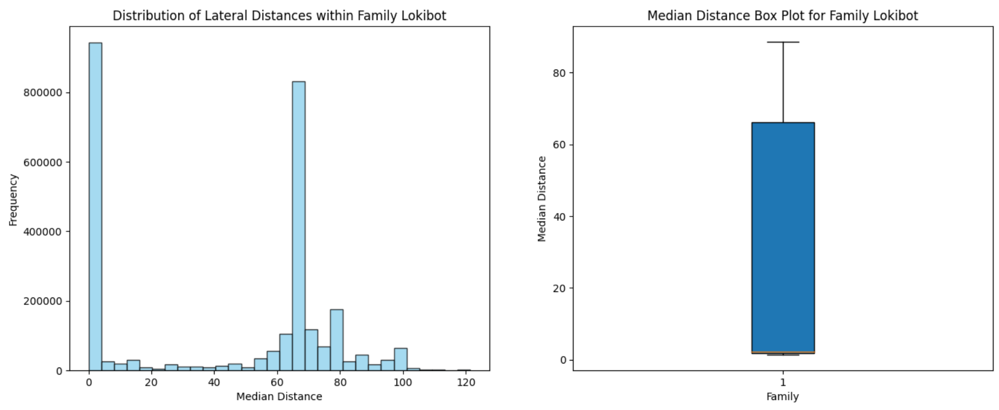


Figure K.24: Distribution of Lateral Distances for Lokibot shows 2 modes

However, these modes are not immediately apparent in the t-SNE or UMAP plots. A closer examination of the t-SNE plot in Figure K.25 provides a clearer view of the local structure, revealing two clusters positioned significantly apart at -60 and 60 on the x axis. These clusters correspond well with the modes observed in Figure K.24.

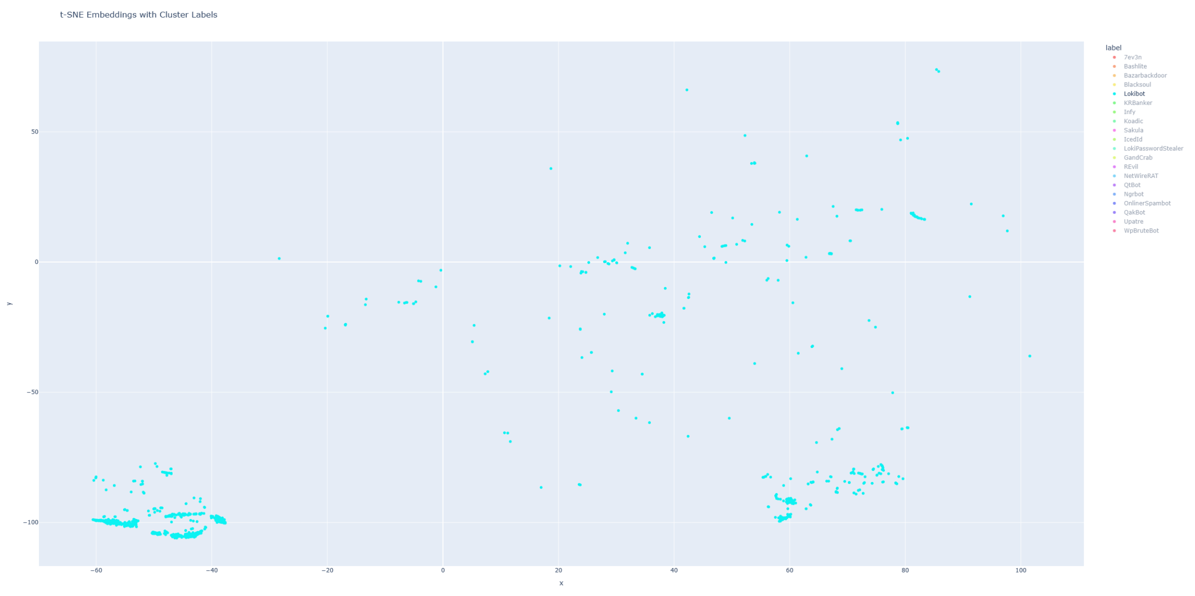


Figure K.25: t-SNE Plot for Lokibot shows 2 clusters

K.5.2. Blacksoul

Blacksoul is a malware family that does not show distinct clustering in the overall t-SNE and UMAP visualizations (Figures K.1 and K.3). However, a more detailed analysis of the t-SNE plot specifically for Blacksoul (Figure K.26) reveals a spread of data points around a main cluster at an x-axis value of -30. Although there appears to be a potential cluster near the x-axis at 100, it lacks clear definition. Conversely, the PCA plot of Blacksoul (Figure K.28) primarily shows a density gradient at -0.4 on the x-axis, with a subtle indication of another density around 1.1 x-axis. This pattern is somewhat mirrored in the lateral distance distribution (Figure K.26), where there is a predominant mode with a less pronounced secondary mode around 50, suggesting a complex intra-family structure.

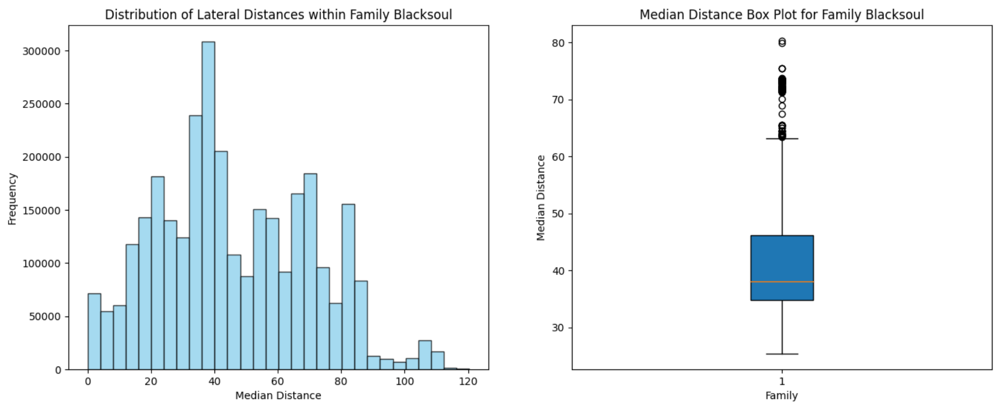


Figure K.26: Blacksoul shows one main mode

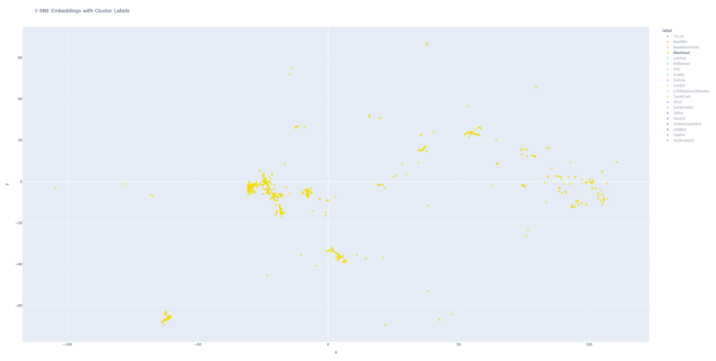


Figure K.27: t-SNE Plot for BlackSoul

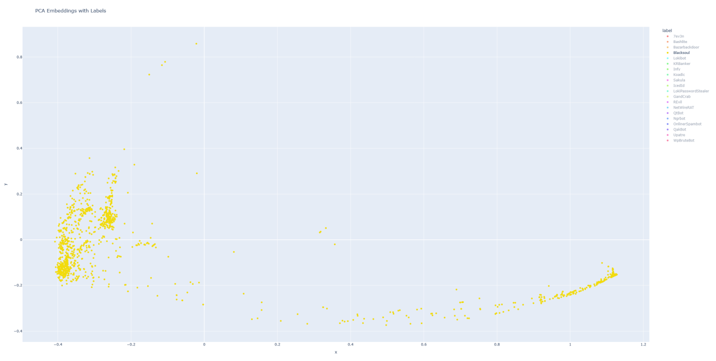


Figure K.28: PCA Plot for BlackSoul shows one main mode with possible outliers at 1

K.5.3. Online Spambot

In the t-SNE and PCA visualizations represented by Figures K.1 and K.2, the Online Spambot embeddings form a distinctive linear pattern(straight line). This regularity across both visualization methods suggests a progressive differentiation of attributes within this malware family, hinting at possible evolutionary development or systematic changes in the malware's traits over time. The lateral distance distribution for Online Spambot, while varied, displays notable peaks at intervals of 20, which could hint at a stepwise evolution in the family's characteristics. Nevertheless, this observation is tentative, and without further evidence, it is hard to draw any conclusions.

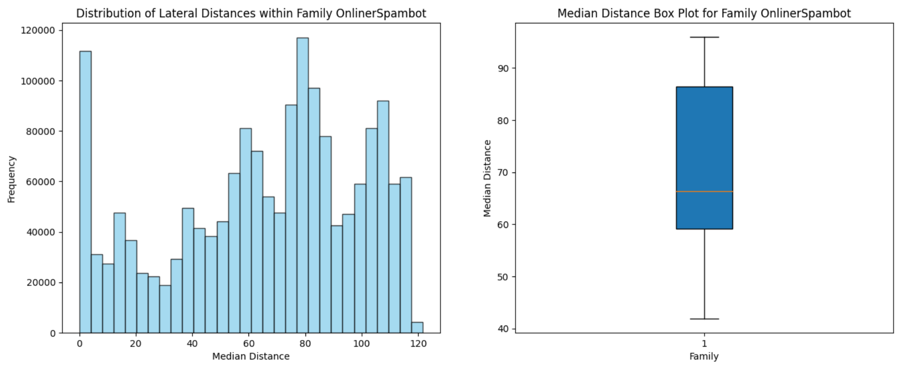


Figure K.29: Distribution of Lateral Distances for OnlinerSpambot shows increasing modes from 40 , 60 to 80

L Case Studies

Case Study: BotenaGo

BotenaGo is a relatively new malware that primarily targets Linux-based systems, exploiting vulnerabilities to launch distributed denial-of-service (DDoS) attacks [208]. Developed using the Go programming language, BotenaGo is known for its efficiency and capability to handle concurrent operations effectively, which is crucial for its role in network disruption.

A key characteristic of BotenaGo is its specific focus on Internet of Things (IoT) devices. These devices often run on Linux and are recognized for having less stringent security measures, making them prime targets for this type of malware. The malware includes dozens of different exploits for known vulnerabilities, enabling it to automatically scan for and infect vulnerable devices across the internet or within a local network. Once a device is compromised, BotenaGo can execute arbitrary code or commands as dictated by the malware operator.

One of the primary functions of BotenaGo is to conduct DDoS attacks. It is capable of sending a massive amount of traffic to targeted servers or networks, overwhelming them and potentially causing them to shut down or disrupt their services. The architecture of BotenaGo is modular, which allows its operators to easily update or modify its capabilities with new exploits or functionalities. This adaptability makes BotenaGo particularly dangerous, as it can evolve quickly in response to countermeasures or new opportunities.

After infecting an initial device, BotenaGo uses that device to further scan and propagate itself to other vulnerable devices, rapidly increasing the scale of the infection. This network propagation demonstrates the malware's sophisticated design and its potential for widespread impact on networks, particularly those incorporating IoT devices.

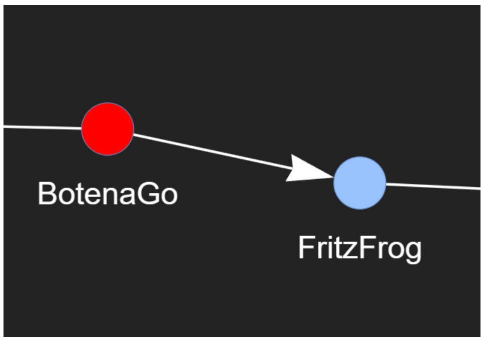
L.0.1. Pseudo-static and Dynamic analysis

The pseudo-static analysis features of the malware reveals several operational strategies based on the imported and exported functions identified, indicating sophisticated capabilities within its design. Key networking functions such as `getaddrinfo` and `freeaddrinfo` are pivotal for resolving domain names to IP addresses and managing network resources. This underlines the malware's potential to connect with command and control servers, spread across networks, or perform network reconnaissance. The dynamic generation of network traffic emphasizes the malware's adaptability to various network environments and its capability to scan for new targets.

Robust error handling is facilitated through functions like `__errno_location`, ensuring thread-safe access to system error codes. This is crucial for the malware to function reliably under diverse system configurations, enhancing its stealth and resilience. Additionally, the malware utilizes Go's runtime management capabilities, evident from exported functions such as `_cgo_panic` and `_cgo_topofstack`, essential for handling exceptions and managing memory in malicious operations. The function `crosscall2` facilitates calls between Go and C code, enabling complex tasks that require interaction with low-level system APIs, thus enhancing the malware's functional complexity.

Further analysis shows that the malware has the ability to control execution flow dynamically. Exports like `runtime.text` and `runtime.etext`, which define the executable code segment's start and

Figure L.1: FritzFrog is from BotenaGo



end points, suggest that the malware can alter its execution to inject code or modify behavior in real-time. This capability is particularly significant as it allows the malware to evade static analysis tools and adapt to operational countermeasures.

Behavioral analysis features of BotenaGO indicates extensive interactions with system files and directories, suggesting deep system integration and manipulation attempts. The malware accesses critical system configuration and log files, such as `/etc/fstab` and `/etc/gai.conf`, and engages with various `logrotate` configurations. These actions likely aim to understand system settings and manipulate log management to potentially disable logging or alter logs to hide malicious activities. Access to `/dev/mem` and firmware update configurations points to attempts to manipulate or intercept low-level device interactions or firmware behaviors, possibly embedding the malware more deeply into the system or causing harm to physical device components. Furthermore, BotenaGO's operations include deleting files in `/var/log` and temporary files in `/var/lib`, which are clear attempts to remove traces of its activities and maintain stealth within the host system. System maintenance commands such as `/sbin/fstrim` and `/usr/sbin/logrotate` suggest attempts to optimize the system to better suit its needs or manipulate system logs to conceal its presence. Additionally, using `systemctl` to query or modify the state of services like CUPS indicates potential efforts to disable security services or modify service states to prevent detection.

Performing DNS lookups as part of its operation implies that BotenaGO engages in external communications with C2 servers or scans the network to identify new targets, facilitating the spread of the malware or the execution of coordinated attacks. These behaviors highlight the sophisticated nature of BotenaGO and its capabilities for widespread impact and system manipulation.

FritzFrog

Figure **??** reveals that FristzFrog is from BotenaGo according to inter-family analysis. According to Victor et al.[209], FritzFroz is a malware written in Go that targets IoT devices, akin to BotenaGo. Despite being a DOS executable, which limits the ability to analyze imports or exports using tools like Lief, a detailed examination of the opcodes used by FritzFroz and BotenaGo offers someu insights into their operational strategies.

BotenaGo demonstrates advanced processing capabilities, utilizing SIMD instructions such as `movdqu`, `movups`, and `punpcklbw`. These instructions are crucial for optimizing the processing of large data blocks, which could be employed in network packet handling or cryptographic functions. FritzFroz, in contrast, shows less reliance on SIMD, which suggests that it may use alternative optimization strategies or focus on different functionalities within its malware operations.

Both malware families also make use of system-specific instructions, including `cpuid` and `xgetbv`, to detect the environment or enhance execution based on hardware capabilities. Additionally, BotenaGo's use of atomic and locking instructions, such as `lock cmpxchg` and `lock xadd`, highlights its sophisticated handling of concurrency issues. This capability is particularly relevant in scenarios involving network communications or multi-threaded operations.

The dynamic analysis features of FritzFrog highlights considerable similarities with BotenaGo, especially in their operational tactics concerning file manipulation, network interactions, security evasion, command execution, and the use of advanced system commands.

Both FritzFrog and BotenaGo exhibit extensive interactions with system and configuration files, signaling a concerted effort to manipulate network behavior. This includes accessing critical SSH configuration files, with FritzFrog reaching into libraries like `libpam.so.0` and BotenaGo into `libc.so.6`. Their engagement with daemon configurations, such as FritzFrog's modifications to `/etc/pam.d/sshd` and BotenaGo's adjustments in `/etc/fwupd/daemon.conf`, underscores their capability to alter essential system functionalities to facilitate their malicious activities. Network-related behaviors are prominently displayed by both malware families. They manipulate network configuration files such as `/etc/gai.conf` to affect how network addresses are resolved, indicating a deep involvement in network behavior manipulation. Additionally, their interactions with device random number generation files—`/dev/urandom` for FritzFrog and `/dev/mem` for BotenaGo—point to potential cryptographic operations. These actions, combined with SSH key interactions, are crucial for gaining network access and ensuring persistence.

In terms of security evasion and stealth, both malware families take measures to remain undetected. This includes modifying log files—FritzFrog writing to `/var/log/btmp` and BotenaGo erasing `/var/log/syslog.1`—and manipulating process management files like `/proc/self/oom_score_adj` to avoid termination. Their interaction with daemon configuration files further suggests attempts to modify log behaviors or disable certain security features, enhancing their stealth within compromised systems. Command execution is another area where FritzFrog and BotenaGo show similar strategies. Both execute system commands that affect service management and configurations, such as FritzFrog's execution of `/usr/sbin/sshd` and BotenaGo's use of `/usr/sbin/logrotate /etc/logrotate.conf`. Their use of commands to refresh or manage service configurations highlights their ability to manipulate system services effectively.

Moreover, the usage of advanced system commands and scripts for deeper system integration and functionality is evident in both malware families. They execute scripts related to log management, with FritzFrog utilizing `/usr/lib/rsyslog/rsyslog-rotate` and BotenaGo manipulating logrotate scripts. Their commands for cryptographic management, such as `/usr/bin/gpg --version` used by FritzFrog, and similar security-related commands by BotenaGo, underline their focus on securing their operations or encrypting data. Their interaction with system service controllers like `systemctl` further affects how services are loaded and managed, underscoring their sophisticated approach to system control and manipulation. These parallels between FritzFrog and BotenaGo suggest a shared methodology in their approach to compromising and controlling infected systems.

Case Study:DiscordTokenStealers

DiscordTokenStealers are a type of malware specifically designed to target users of the Discord platform, a popular chat application widely used by gamers and various online communities[210]. These malicious programs aim to steal Discord authentication tokens from infected machines. A Discord token is essentially a user's authentication key, which allows seamless access to Discord without the need to re-enter a password. If a token is stolen, it grants the attacker the same level of access to the victim's Discord account, including their personal messages, server memberships, and potentially sensitive information shared on the platform.

L.0.2. Key Characteristics of DiscordTokenStealers

The stealers targeting Discord focus on a specific attack vector: extracting Discord tokens from users' systems. These tokens are typically stored in local storage areas utilized by web browsers and the Discord desktop application. Once installed on a device, the stealer actively searches known file paths and system storage locations related to Discord to locate and extract these authentication tokens. The implementation of these stealers varies widely, ranging from simple scripts written in languages like JavaScript or Python to more sophisticated malware. These advanced versions incorporate additional functionalities designed to evade detection and ensure persistence within the host system, making them more dangerous and harder to eradicate.

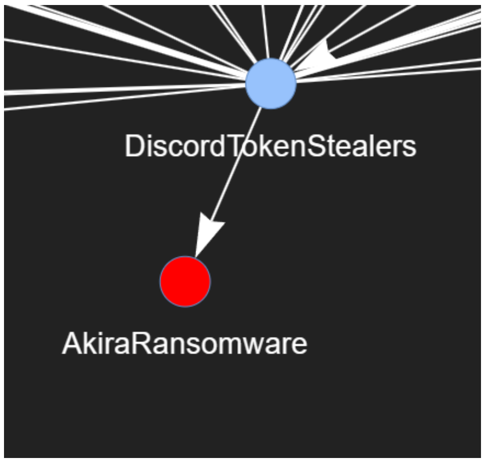
Distribution methods for these malicious tools are diverse and often insidious. They are commonly spread through phishing attacks and malicious links or as components of larger compromised software packages. Additionally, they may be distributed on hacking forums or through social engineering schemes specifically targeting Discord users, exploiting the trust within social circles to propagate the malware.The consequences of token theft are severe. Possession of a Discord token grants an attacker remote access to the victim's Discord account, allowing them to impersonate the victim, access private communications, and potentially leverage this access to spread the malware further. This can lead to significant breaches of privacy and security for the affected users and their contacts, underlining the critical nature of securing such digital tokens against theft.

To effectively analyze this malware, it's important to note that it is a DOS-based executable. Tools like Lief are unable to identify import or export features in such DOS executables, including FritzFrog. Therefore, concentrating on a behavioral analysis is crucial to comprehensively understand its impact and how it operates on infected systems. The malware extensively interacts with system files and directories, manipulating '.dll' files within `C:\Windows\assembly` and '.docx' files in user directories. Such behavior is indicative of attempts to locate and extract stored Discord tokens or to inject malicious code into executable or script files to ensure persistence and spread within the network.

Command executions such as `%windir%\Microsoft.NET\Framework\v4.0.30319\AppLaunch.exe` and `C:\Windows\SysWOW64\WerFault.exe` demonstrate the malware's exploitation of system processes to execute malicious payloads or disrupt system error reporting services, aiding its undetectability. Memory manipulation and process injections, particularly into processes like `WMIADAP.EXE`, allow the malware to operate within the security context of legitimate system processes, thus evading detection by security software and gaining elevated privileges. The malware's interactions with network-related system files, such as `wsock32.dll`, and its monitoring of network processes potentially enable it to intercept network traffic to and from the Discord application to capture authentication tokens.

Moreover, the malware employs various persistence and evasion techniques, manipulating error reporting files and directories and performing extensive operations on system and application logs to erase traces of its presence and manipulate system responses for sustained stealth. Tactics such as repeatedly deleting files, particularly within Windows Error Reporting directories, are likely aimed at preventing forensic analysis or automatic error reports that could reveal the malware's presence.Additionally, the malware interferes with essential system processes, such as terminating and manipulating `svchost.exe` and interfering with Windows Update processes (`wuapihost.exe`), to weaken the system's defenses. Frequent usage of Windows Management Instrumentation (WMI) to gather system information, check for antivirus installations, or detect virtual machines highlights the malware's efforts to assess its environment to optimize evasion strategies and payload delivery, enhancing its effectiveness and minimizing detection risk.

Figure L.2: AkiraRansomware is from DiscordTokenStealers



AkiraRansomware

Figure L.2 reveals that FristzFrog is from BotenaGo according to inter-family analysis. According to Threatmon[211], AkiraRansomware employs the same attack vector as DiscordTokenStealers, with both being DOS format malware. This necessitates a focus on dynamic analysis features for effective understanding. AkiraRansomware extensively interacts with system files, notably by creating readme files in various directories such as `c:\akira_readme.txt` and `c:\program files\akira_readme.txt`. This behavior is characteristic of ransomware, aiming to inform users of the encryption of their data and to demand a ransom. Conversely, DiscordTokenStealers specifically manipulate files associated with the Discord application to stealthily extract authentication tokens, focusing on covert operations rather than overt system disruption.

Both malware types utilize PowerShell extensively, albeit for different purposes. AkiraRansomware uses PowerShell scripts for tasks such as encryption and system reconnaissance, evidenced by files such as `\PSHost.133268678561975959.2708.DefaultAppDomain.powershell`, which suggest automated script execution within the system. Meanwhile, DiscordTokenStealers use PowerShell to search and extract stored Discord tokens from local storage and browser data, leveraging PowerShell's deep integration with Windows systems to facilitate data extraction. Command execution is another commonality; AkiraRansomware executes system-altering commands, such as `powershell.exe -Command "Get-WmiObject Win32_Shadowcopy | Remove-WmiObject"`, intended to delete shadow copies and hinder data recovery. Similarly, DiscordTokenStealers may execute commands to discreetly adjust environment settings to aid in token theft without triggering security alerts.

Behavioral patterns also highlight their strategies for evasion and persistence. AkiraRansomware may operate processes in hidden modes or manipulate system files to complicate restoration efforts. Similarly, DiscordTokenStealers engage in stealthy operations to avoid detection while harvesting and exfiltrating Discord tokens. Furthermore, both malware types implement MITRE ATT&CK techniques to enhance their effectiveness and stealth. They perform actions such as querying process information and detecting virtual environments, which demonstrate their sophisticated capability to assess and adapt to their operating environments. For instance, AkiraRansomware's queries of sensitive disk information via WMI or Win32_DiskDrive, often done to detect virtual machines, are techniques also favored by DiscordTokenStealers to ensure optimal execution conditions. These parallels may suggest that both of these malware are related.

Case Study: SundownEk

The Sundown Exploit Kit (SundownEK) is a malicious toolkit used by cybercriminals to exploit vulnerabilities in web browsers and other software to deliver malware, categorizing it among the broader range of threats known as exploit kits. These kits automatically probe for software vulnerabilities on a visitor's computer and exploit these vulnerabilities to deliver malware [212].

SundownEK typically operates through compromised or malicious websites that users might visit inadvertently. It can also spread through malvertising campaigns, where malware is disseminated via malicious advertisements placed on legitimate websites. This method of distribution enables SundownEK to reach a broad audience without requiring direct interaction from the target. The exploit kit includes a variety of exploits that target common vulnerabilities in widely used applications, such as Adobe Flash, Java, and various web browsers. SundownEK is frequently updated with new exploits as soon as they become known and available, allowing it to stay effective against unpatched systems.To evade detection by antivirus and other security software, SundownEK employs various obfuscation techniques. These may include encrypting the exploit code, using polymorphic malware, or implementing methods specifically designed to avoid triggering security mechanisms in browsers.

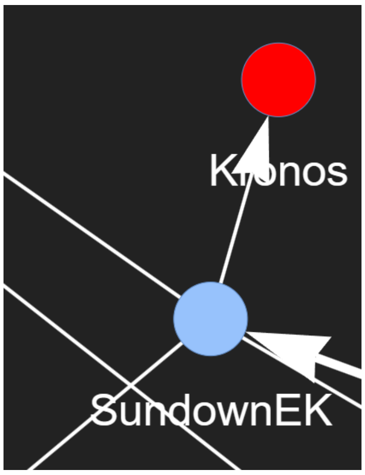
Once it successfully exploits a vulnerability, SundownEK delivers a payload, which could be any type of malware, including ransomware, spyware, or bots. The specific type of malware delivered can vary depending on the attackers' objectives and the specifics of the campaign they are running.Unlike other more well-known exploit kits such as Angler or Neutrino, SundownEK tends to maintain a lower profile. This is possibly an intentional strategy to avoid the scrutiny of security researchers and law enforcement, allowing it to operate under the radar and continue its malicious activities without significant interference.

SundownEK, recognized as a DOS-based malware, presents challenges in analyzing pseudo-static features due to its format. Consequently, a focus on dynamic analysis features is essential to uncover its operational behaviors. SundownEK extensively interacts with system and application files, indicative of exploit kits designed to manipulate or monitor system functions. It accesses significant Windows system files such as `winime32.dll`, `ws2_32.dll`, and `shell32.dll`. Moreover, it engages in behaviors such as writing to and modifying files, notably creating `Samian.dll` in a temporary directory—a tactic commonly employed by exploit kits to execute malicious payloads discreetly.The kit's exploitation of Dynamic Link Libraries (DLLs) is evident, as it opens multiple DLL files potentially to inject malicious code or exploit existing vulnerabilities. This method is typical in exploit kit operations that involve DLL hijacking or exploiting vulnerabilities.

Furthermore, SundownEK makes extensive use of temporary files and directories, a strategy exploit kits utilize to minimize detection. These locations are less monitored and provide a transient space for conducting malicious activities without leaving long-term evidence.Command executions are another critical aspect of SundownEK's behavior. It executes specific commands, such as running `996E.exe` from a temporary directory, which indicates the kit's process to activate downloaded or dropped payloads, enabling further malicious operations within the infected system.

Process interactions include terminating processes related to its operations, likely serving as a cleanup mechanism post-exploit to avoid detection and analysis or to stop processes that might interfere with its activities.Registry interactions are also significant; SundownEK modifies the system's registry, including deleting keys associated with error reporting. This behavior aims to undermine system defenses and erase traces of the kit's presence to enhance stealth and ensure persistence. Lastly, SundownEK manipulates services related to network access and routing. This manipulation is typically linked to establishing remote access capabilities or preparing the system for further exploitation and lateral movements, aligning with the typical behavior of exploit kits aimed at spreading or enabling command and control activities.

Figure L.3: Kronos is from SundownEK



Kronos

Figure L.3 reveals that FristzFrog is from BotenaGo according to inter-family analysis. The dynamic analysis features of Kronos, as suggested by Malwarebytes[213], reveals significant operational similarities with SundownEK, particularly highlighting their approaches in manipulating and exploiting system resources. Both being DOS-based malware, this analysis is crucial given the limitations in pseudo-static feature extraction. Kronos interacts extensively with critical system files such as `winime32.dll`, `ws2_32.dll`, and `shell32.dll`, integral to system operations and commonly targeted for vulnerabilities or malicious manipulations. Similarly, SundownEK exploits these system files to alter system behaviors or facilitate malware delivery, showcasing parallel tactics in exploiting system vulnerabilities.

Moreover, Kronos frequently manipulates temporary files, using locations like `C:\Documents and Settings\Administrator\Local Settings\Temp\nsc3.tmp` for staging and executing malicious payloads discreetly. This mirrors SundownEK's strategy of using temporary files as a staging area for executing malicious scripts or storing components, minimizing detection and facilitating cleanup post-execution.Command execution is another shared tactic between Kronos and SundownEK. Kronos employs system utilities such as `svchost.exe` to blend in with normal system processes or stealthily manage its operations, akin to SundownEK's use of command execution to trigger exploits or load malware following a successful exploit.

Process manipulations are evident in Kronos, where it terminates its own processes to evade detection or clear traces after executing malicious activities. This strategy reflects SundownEK's approach to terminating processes to disable security software or clear the path for its payload without interference.Stealth and evasion are also central to Kronos' operations, with changes in file attributes designed to hide its files from simple searches or make them harder to delete. SundownEK uses similar tactics, employing file obfuscation or polymorphism to complicate the detection of its components by antivirus software.

Furthermore, while Kronos may not manipulate system services as extensively as SundownEK, it interacts with services like ”RemoteAccess” to potentially adjust network settings favorably for remote control or data exfiltration. This selective interaction with system services underlines the malware's strategic use of system resources to support its malicious objectives.

M Pseudo-static and Dynamic Analysis

M.1. Mirai

M.1.1. Pseudo-static analysis features

```
section: {'entry': '0x401060', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, 'file_offset': 0, 'props': ['Type: NULL']}, {'name': '.interp', 'size': 20, 'entropy': 3.684183719779189, 'file_offset': 244, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.hash', 'size': 376, 'entropy': 1.4667777070926495, 'file_offset': 264, 'props': ['Type: HASH', 'ALLOC']}, {'name': '.dynsym', 'size': 880, 'entropy': 3.1509644214886543, 'file_offset': 640, 'props': ['Type: DYNSYM', 'ALLOC']}, {'name': '.dynstr', 'size': 420, 'entropy': 4.1612991185289365, 'file_offset': 1520, 'props': ['Type: STRTAB', 'ALLOC']}, {'name': '.rela.plt', 'size': 588, 'entropy': 2.9045781679847202, 'file_offset': 1940, 'props': ['Type: RELA', 'ALLOC']}, {'name': '.init', 'size': 48, 'entropy': 4.365601562950723, 'file_offset': 2528, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.plt', 'size': 1400, 'entropy': 4.227768835761367, 'file_offset': 2576, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.text', 'size': 34976, 'entropy': 6.696022855285639, 'file_offset': 4000, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.fini', 'size': 36, 'entropy': 4.5588138903312, 'file_offset': 38976, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.interp', 'Size': 20, 'Virtual Address': 4194548}, {'Name': '.hash', 'Size': 376, 'Virtual Address': 4194568}, {'Name': '.dynsym', 'Size': 880, 'Virtual Address': 4194944}, {'Name': '.dynstr', 'Size': 420, 'Virtual Address': 4195824}, {'Name': '.rela.plt', 'Size': 588, 'Virtual Address': 4196244}, {'Name': '.init', 'Size': 48, 'Virtual Address': 4196832}, {'Name': '.plt', 'Size': 1400, 'Virtual Address': 4196880}, {'Name': '.text', 'Size': 34976, 'Virtual Address': 4198304}, {'Name': '.fini', 'Size': 36, 'Virtual Address': 4233280}, {'Name': '.rodata', 'Size': 888, 'Virtual Address': 4233316}, {'Name': '.ctors', 'Size': 8, 'Virtual Address': 4299744}, {'Name': '.dtors', 'Size': 8, 'Virtual Address': 4299752}, {'Name': '.dynamic', 'Size': 152, 'Virtual Address': 4299764}, {'Name': '.data', 'Size': 32, 'Virtual Address': 4299916}, {'Name': '.got', 'Size': 208, 'Virtual Address': 4299948}, {'Name': '.bss', 'Size': 412, 'Virtual Address': 4300156}, {'Name': '.shstrtab', 'Size': 116, 'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.PHDR', 'Size': 192, 'Virtual Address': 4194356}, {'Type': 'SEGMENT_TYPES.INTERP', 'Size': 20, 'Virtual Address': 4194548}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 39900, 'Virtual Address': 4194304}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 412, 'Virtual Address': 4299744}, {'Type': 'SEGMENT_TYPES.DYNAMIC', 'Size': 152, 'Virtual Address': 4299764}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual Address': 0}]}
imports: {'DynamicSymbols': ['', 'ioctl', 'recv', 'connect', 'sigemptyset', 'memmove', 'getpid', 'prctl', 'memcpy', 'readlink', 'malloc', 'sleep', 'recvfrom', 'socket', 'select', 'readdir', 'sigaddset', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', 'bind', 'inet_addr', 'setsockopt', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork', '__uClibc_main', 'memset', 'getppid', 'time', 'opendir', 'getsockopt', '__errno_location', 'exit', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close', 'raise', 'free', 'sigprocmask', 'getsockname'], 'SymbolTable': ['', 'ioctl', 'recv', 'connect', 'sigemptyset', 'memmove', 'getpid', 'prctl', 'memcpy', 'readlink', 'malloc', 'sleep', 'recvfrom', 'socket', 'select', 'readdir', 'sigaddset', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', 'bind', 'inet_addr', 'setsockopt', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork', '__uClibc_main', 'memset', 'getppid', 'time', 'opendir', 'getsockopt', '__errno_location', 'exit', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close', 'raise', 'free', 'sigprocmask', 'getsockname']}
```

```
exports: ['ioctl', 'recv', 'connect', 'sigemptyset', 'memmove', 'getpid', 'prctl', 'memcpy', 'readlink', 'malloc', 'sleep', 'recvfrom', 'socket', 'select', 'readdir', 'sigaddset', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', 'bind', 'inet_addr', 'setsockopt', '_start', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork', '__uClibc_main', 'memset', 'getppid', 'time', 'opendir', 'getsockopt', '__errno_location', 'exit', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close', 'raise', 'free', 'sigprocmask', 'getsockname']
general: {'size': 41152, 'virtual_size': 0, 'has_debug': 0, 'exports': 55, 'imports': 19, 'has_relocations': 0, 'symbols': 55}
header: {'file_type': 'EXECUTABLE', 'entry_point': 4198496, 'machine_type': 'SH', 'header_size': 52, 'program_headers': [{'type': 'PHDR', 'virtual_address': 4194356, 'physical_address': 4194356, 'physical_size': 192, 'virtual_size': 192, 'flags': SEGMENT_FLAGS.???, 'alignment': 4}, {'type': 'INTERP', 'virtual_address': 4194548, 'physical_address': 4194548, 'physical_size': 20, 'virtual_size': 20, 'flags': SEGMENT_FLAGS.R, 'alignment': 1}, {'type': 'LOAD', 'virtual_address': 4194304, 'physical_address': 4194304, 'physical_size': 39900, 'virtual_size': 39900, 'flags': SEGMENT_FLAGS.???, 'alignment': 65536}, {'type': 'LOAD', 'virtual_address': 4299744, 'physical_address': 4299744, 'physical_size': 412, 'virtual_size': 824, 'flags': SEGMENT_FLAGS.???, 'alignment': 65536}, {'type': 'DYNAMIC', 'virtual_address': 4299764, 'physical_address': 4299764, 'physical_size': 152, 'virtual_size': 152, 'flags': SEGMENT_FLAGS.???, 'alignment': 4}, {'type': 'GNU_STACK', 'virtual_address': 0, 'physical_address': 0, 'physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 4}], 'section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0, 'size': 0, 'entropy': -0.0}, {'name': '.interp', 'type': 'PROGBITS', 'virtual_address': 4194548, 'size': 20, 'entropy': 3.684183719779189}, {'name': '.hash', 'type': 'HASH', 'virtual_address': 4194568, 'size': 376, 'entropy': 1.4667777070926495}, {'name': '.dynsym', 'type': 'DYNSYM', 'virtual_address': 4194944, 'size': 880, 'entropy': 3.1509644214886543}, {'name': '.dynstr', 'type': 'STRTAB', 'virtual_address': 4195824, 'size': 420, 'entropy': 4.1612991185289365}, {'name': '.rela.plt', 'type': 'RELA', 'virtual_address': 4196244, 'size': 588, 'entropy': 2.9045781679847202}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address': 4196832, 'size': 48, 'entropy': 4.365601562950723}, {'name': '.plt', 'type': 'PROGBITS', 'virtual_address': 4196880, 'size': 1400, 'entropy': 4.227768835761367}, {'name': '.text', 'type': 'PROGBITS', 'virtual_address': 4198304, 'size': 34976, 'entropy': 6.696022855285639}, {'name': '.fini', 'type': 'PROGBITS', 'virtual_address': 4233280, 'size': 36, 'entropy': 4.5588138903312}, {'name': '.rodata', 'type': 'PROGBITS', 'virtual_address': 4233316, 'size': 888, 'entropy': 5.098093887248329}, {'name': '.ctors', 'type': 'PROGBITS', 'virtual_address': 4299744, 'size': 8, 'entropy': 1.0}, {'name': '.dtors', 'type': 'PROGBITS', 'virtual_address': 4299752, 'size': 8, 'entropy': 1.0}, {'name': '.dynamic', 'type': 'DYNAMIC', 'virtual_address': 4299764, 'size': 152, 'entropy': 2.046700321855241}, {'name': '.data', 'type': 'PROGBITS', 'virtual_address': 4299916, 'size': 32, 'entropy': 2.746696364505181}, {'name': '.got', 'type': 'PROGBITS', 'virtual_address': 4299948, 'size': 208, 'entropy': 3.958263738966142}, {'name': '.bss', 'type': 'NOBITS', 'virtual_address': 4300156, 'size': 412, 'entropy': 3.1384341723299536}, {'name': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 116, 'entropy': 3.915745112170353}]}
strings: {'numstrings': 123, 'avlength': 6.195121951219512, 'printabledist': [14, 41, 42, 45, 5, 3, 10, 12, 10, 23, 6, 11, 12, 2, 3, 7, 6, 25, 11, 18, 3, 1, 1, 4, 1, 5, 7, 9, 2, 1, 0, 1, 5, 25, 15, 10, 1, 0, 2, 4, 1, 0, 2, 1, 5, 0, 0, 7, 10, 7, 4, 11, 2, 4, 3, 1, 1, 0, 5, 0, 5, 4, 4, 0, 27, 47, 26, 38, 7, 8, 7, 6, 3, 2, 7, 2, 7, 11, 3, 0, 6, 8, 4, 13, 4, 1, 1, 3, 3, 2, 6, 10, 16, 1, 0, 3], 'printables': 762, 'entropy': 5.705211162567139, 'paths': 0, 'urls': 0, 'registry': 0, 'MZ': 0}
EntryPoint: [['Main ', 4198496], ['ioctl', 4196908], ['recv', 4196936], ['conne', 4196964], ['sigem', 4196992], ['memmo', 4197020], ['getpi', 4197048], ['prctl', 4197076], ['memcp', 4197104], ['readl', 4197132], ['mallo', 4197160], ['sleep', 4197188], ['recvf', 4197216], ['socke', 4197244], ['selec', 4197272], ['readd', 4197300], ['sigad', 4197328], ['send', 4197356], ['abort', 4197384], ['accep', 4197412], ['callo', 4197440], ['write', 4197468], ['kill', 4197496], ['bind', 4197524], ['inet_', 4197552], ['setso', 4197580], ['_star', 4198496], ['signa', 4197608], ['read', 4197636], ['sendt', 4197664], ['reall', 4197692], ['liste', 4197720], ['fork', 4197748], ['__uCl', 4197776], ['memse', 4197804], ['getpp', 4197832], ['time', 4197860], ['opend', 4197888], ['getso', 4197916], ['__err', 4197944], ['exit', 4197972], ['open', 4198000], ['clock', 4198028], ['setsi', 4198056], ['close', 4198084], ['fcntl', 4198112], ['close', 4198140], ['raise', 4198168], ['free', 4198196], ['sigpr', 4198224]]
ExitPoint: [['Dynamic Symbol', 'abort'], ['Dynamic Symbol', 'exit']]
Opcodes: ['add', 'das', 'add', 'add', 'inc', 'mul', 'add', 'add', 'add', 'inc', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add',
     'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', '
    adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', '
    shr', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr
    ', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add',
    'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', '
    add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', 'inc', 'add',
    'pushfd', 'inc', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', 'inc', 'or', 'adc', '
    inc', 'add', 'pushfd', 'inc', 'add', 'add', 'add', 'add', 'shr', 'inc', 'adc', 'shr', '
    inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'add', 'shr', 'inc', 'adc', 'shr
    ', 'inc', 'or', 'adc', 'inc', 'add', 'pushfd', 'inc', 'add', 'rol', 'pushal', 'add', 'sub
    ', 'pushal', 'add']
Opcode-Occurrence: {'add': 223, 'das': 2, 'inc': 170, 'mul': 1, 'shr': 88, 'adc': 89, 'or':
    49, 'pushfd': 16, 'rol': 5, 'pushal': 10, 'sub': 10, 'in': 1, 'hlt': 1, 'popfd': 10, 'cmp
    ': 1, 'dec': 1, 'xchg': 2, 'out': 1, 'nop': 1, 'popal': 1, 'and': 2}
Image Size: 4300156
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 40676}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 412, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.PHDR', 'Flags': 'READ | EXECUTE'}, '
    Segment 1': {'Type': 'SEGMENT_TYPES.INTERP', 'Flags': 'READ'}, 'Segment 2': {'Type': '
    SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.
    LOAD', 'Flags': 'READ | WRITE'}, 'Segment 4': {'Type': 'SEGMENT_TYPES.DYNAMIC', 'Flags':
    'READ | WRITE'}, 'Segment 5': {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ | WRITE
    | EXECUTE'}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.interp':
    {'min': 3.684183719779189, 'max': 3.684183719779189, 'total': 3.684183719779189, 'count':
     1, 'mean': 3.684183719779189}, '.hash': {'min': 1.4667777070926495, 'max':
    1.4667777070926495, 'total': 1.4667777070926495, 'count': 1, 'mean': 1.4667777070926495},
     '.dynsym': {'min': 3.1509644214886543, 'max': 3.1509644214886543, 'total':
    3.1509644214886543, 'count': 1, 'mean': 3.1509644214886543}, '.dynstr': {'min':
    4.1612991185289365, 'max': 4.1612991185289365, 'total': 4.1612991185289365, 'count': 1, '
    mean': 4.1612991185289365}, '.rela.plt': {'min': 2.9045781679847202, 'max':
    2.9045781679847202, 'total': 2.9045781679847202, 'count': 1, 'mean': 2.9045781679847202},
     '.init': {'min': 4.365601562950723, 'max': 4.365601562950723, 'total':
    4.365601562950723, 'count': 1, 'mean': 4.365601562950723}, '.plt': {'min':
    4.227768835761367, 'max': 4.227768835761367, 'total': 4.227768835761367, 'count': 1, '
    mean': 4.227768835761367}, '.text': {'min': 6.696022855285639, 'max': 6.696022855285639,
    'total': 6.696022855285639, 'count': 1, 'mean': 6.696022855285639}, '.fini': {'min':
    4.5588138903312, 'max': 4.5588138903312, 'total': 4.5588138903312, 'count': 1, 'mean':
    4.5588138903312}, '.rodata': {'min': 5.098093887248329, 'max': 5.098093887248329, 'total
    ': 5.098093887248329, 'count': 1, 'mean': 5.098093887248329}, '.ctors': {'min': 1.0, 'max
    ': 1.0, 'total': 1.0, 'count': 1, 'mean': 1.0}, '.dtors': {'min': 1.0, 'max': 1.0, 'total
    ': 1.0, 'count': 1, 'mean': 1.0}, '.dynamic': {'min': 2.046700321855241, 'max':
    2.046700321855241, 'total': 2.046700321855241, 'count': 1, 'mean': 2.046700321855241}, '.
    data': {'min': 2.746696364505181, 'max': 2.746696364505181, 'total': 2.746696364505181, '
    count': 1, 'mean': 2.746696364505181}, '.got': {'min': 3.958263738966142, 'max':
    3.958263738966142, 'total': 3.958263738966142, 'count': 1, 'mean': 3.958263738966142}, '.
    bss': {'min': 3.1384341723299536, 'max': 3.1384341723299536, 'total': 3.1384341723299536,
     'count': 1, 'mean': 3.1384341723299536}, '.shstrtab': {'min': 3.915745112170353, 'max':
    3.915745112170353, 'total': 3.915745112170353, 'count': 1, 'mean': 3.915745112170353}}
Kolmogorov Complexity: 21 KB
```

M.1.2. Dynamic Analysis features

```
        "Family Name": "Mirai",
        "Behavior": {
            "files_opened": [
                "/dev/misc/watchdog",
                "/dev/watchdog",
                "/etc/UPower/UPower.conf",
                "/etc/group",
                "/etc/gtk-3.0/settings.ini",
                "/etc/locale.alias",
                "/etc/mtab",
                "/home/lonestar/.Xdefaults-buffalo",
                "/proc/",
                "/proc/3676/mounts",
                "/proc/filesystems",
                "/run/udev/data/+input:input1",
                "/run/udev/data/+input:input3",
                "/run/udev/data/+power_supply:ACAD",
                "/run/udev/data/c13:32",
                "/run/udev/data/c13:63",
                "/run/udev/data/c13:65",
                "/run/user/1000/ICEauthority",
                "/run/user/1000/gdm/Xauthority",
                "/sys/bus",
                "/sys/bus/usb/devices/",
                "/sys/class",
                "/sys/class/input/",
                "/sys/class/leds",
                "/sys/class/power_supply/",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/capabilities/sw",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/event0/../capabilities/sw
                    ",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/event0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/online",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/type",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/uevent",
                "/sys/devices/platform/i8042/serio0/input/input1/capabilities/sw",
                "/sys/devices/platform/i8042/serio0/input/input1/event1/../capabilities/sw",
                "/sys/devices/platform/i8042/serio0/input/input1/event1/uevent",
                "/sys/devices/platform/i8042/serio0/input/input1/uevent",
                "/sys/devices/platform/i8042/serio1/input/input3/capabilities/sw",
                "/sys/devices/platform/i8042/serio1/input/input3/event2/../capabilities/sw",
                "/sys/devices/platform/i8042/serio1/input/input3/event2/uevent",
                "/sys/devices/platform/i8042/serio1/input/input3/mouse0/../capabilities/sw",
                "/sys/devices/platform/i8042/serio1/input/input3/mouse0/uevent",
                "/sys/devices/platform/i8042/serio1/input/input3/uevent",
                "/sys/devices/virtual/input/mice/uevent",
                "/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache",
                "/usr/share/X11/locale/en_US.UTF-8/XLC_LOCALE",
                "/usr/share/X11/locale/locale.alias",
                "/usr/share/X11/locale/locale.dir",
                "/usr/share/icons/Adwaita/icon-theme.cache",
                "/usr/share/icons/Adwaita/index.theme",
                "/usr/share/icons/elementary-xfce-dark/icon-theme.cache"
            ],
            "files_written": [],
            "files_deleted": [
                "/tmp/cutie.x86_64"
            ],
            "command_executions": [
                "/sbin/fstrim --fstab --verbose --quiet",
                "sh -c \"ls -l /proc/2/status\"",
                "ls -l /proc/2/status",
                "sh -c \"ls -l /proc/1/status\"",
                "ls -l /proc/1/status",
                "sh -c \"ls -l /proc/3/status\"",
                "ls -l /proc/3/status",
                "sh -c \"ls -l /proc/4/status\"",
                "ls -l /proc/4/status",
                "sh -c \"ls -l /proc/5/status\"",
                "ls -l /proc/5/status",
                "sh -c \"ls -l /proc/6/status\"",
                "ls -l /proc/6/status",
                "sh -c \"ls -l /proc/7/status\"",
                "ls -l /proc/7/status",
                "sh -c \"ls -l /proc/9/status\"",
                "ls -l /proc/9/status",
                "sh -c \"ls -l /proc/10/status\"",
                "ls -l /proc/10/status",
                "sh -c \"ls -l /proc/11/status\""
            ],
            "files_attribute_changed": [],
            "processes_terminated": [],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Executes commands using a shell command-line interpreter",
                "Creates hidden files, links and/or directories",
                "Sample deletes itself",
                "Executes the \"rm\" command used to delete files or directories",
                "Uses the \"uname\" system call to query kernel version information (possible evasion)",
                "Sample reads /proc/mounts (often used for finding a writable filesystem)",
                "Detected TCP or UDP traffic on non-standard ports",
                "Performs DNS lookups",
                "Performs DNS lookups",
                "Sample tries to kill multiple processes (SIGKILL)"
            ]
        }
    }
```

M.2. Gafgyt

M.2.1. Pseudo-static analysis features

```

section: {'entry': '0x8048164', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, 'file_offset': 0, 'props': ['Type: NULL']}, {'name': '.init', 'size': 28, 'entropy': 3.6375375112660517, 'file_offset': 148, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.text', 'size': 92584, 'entropy': 6.419944555244165, 'file_offset': 176, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.fini', 'size': 23, 'entropy': 4.001822825622232, 'file_offset': 92760, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.rodata', 'size': 33187, 'entropy': 6.407619484825426, 'file_offset': 92800, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.eh_frame', 'size': 4, 'entropy': -0.0, 'file_offset': 125988, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.ctors', 'size': 12, 'entropy': 2.251629167387823, 'file_offset': 126976, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.dtors', 'size': 8, 'entropy': 1.0, 'file_offset': 126988, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.jcr', 'size': 4, 'entropy': -0.0, 'file_offset': 126996, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.got.plt', 'size': 12, 'entropy': -0.0, 'file_offset': 127000, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.init', 'Size': 28, 'Virtual Address': 134512788}, {'Name': '.text', 'Size': 92584, 'Virtual Address': 134512816}, {'Name': '.fini', 'Size': 23, 'Virtual Address': 134605400}, {'Name': '.rodata', 'Size': 33187, 'Virtual Address': 134605440}, {'Name': '.eh_frame', 'Size': 4, 'Virtual Address': 134638628}, {'Name': '.ctors', 'Size': 12, 'Virtual Address': 134639616}, {'Name': '.dtors', 'Size': 8, 'Virtual Address': 134639628}, {'Name': '.jcr', 'Size': 4, 'Virtual Address': 134639636}, {'Name': '.got.plt', 'Size': 12, 'Virtual Address': 134639640}, {'Name': '.data', 'Size': 18776, 'Virtual Address': 134639680}, {'Name': '.bss', 'Size': 35496, 'Virtual Address': 134658464}, {'Name': '.comment', 'Size': 4374, 'Virtual Address': 0}, {'Name': '.shstrtab', 'Size': 111, 'Virtual Address': 0}, {'Name': '.symtab', 'Size': 22032, 'Virtual Address': 0}, {'Name': '.strtab', 'Size': 19593, 'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.LOAD', 'Size': 125992, 'Virtual Address': 134512640}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 18840, 'Virtual Address': 134639616}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual Address': 0}]}
imports: {'SymbolTable': ['', '__register_frame_info_bases', '__deregister_frame_info_bases', '_Jv_RegisterClasses']}
exports: ['__do_global_dtors_aux', 'frame_dummy', '__do_global_ctors_aux', 'printchar', 'prints', 'printi', 'print', 'thread_self', 'pthread_kill_all_threads', 'pthread_start_thread', 'pthread_start_thread_event', 'pthread_free', 'restart', 'pthread_reap_children', 'pthread_insert_list', 'pthread_call_handlers', 'enqueue', 'remove_from_queue', '__pthread_set_own_extricate_if', 'thread_self', 'new_sem_extricate_func', 'suspend', 'pthread_null_sighandler', 'thread_self', 'pthread_sighandler_rt', 'pthread_sighandler', 'wait_node_dequeue', '__pthread_acquire', 'wait_node_free', 'restart', 'thread_self', 'suspend', 'pthread_handle_sigdebug', 'suspend', 'thread_self', 'pthread_onexit_process', 'pthread_initialize', 'pthread_handle_sigrestart', 'pthread_handle_sigcancel', 'thread_self', 'enqueue', 'remove_from_queue', '__pthread_set_own_extricate_if', 'restart', 'thread_self', '
```

```
    cond_extricate_func', 'suspend', '__pthread_set_own_extricate_if', 'thread_self', '
    join_extricate_func', connect', 'sigemptyset', 'memmove','recvfrom', 'socket', 'select']
general: {'size': 192569, 'virtual_size': 0, 'has_debug': 0, 'exports': 1377, 'imports': 0, '
    has_relocations': 0, 'symbols': 1377}
header: {'file_type': 'EXECUTABLE', 'entry_point': 134512996, 'machine_type': 'i386', '
    header_size': 52, 'program_headers': [{'type': 'LOAD', 'virtual_address': 134512640, '
    physical_address': 134512640, 'physical_size': 125992, 'virtual_size': 125992, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 134639616, '
    physical_address': 134639616, 'physical_size': 18840, 'virtual_size': 54344, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'GNU_STACK', 'virtual_address': 0, '
    physical_address': 0, 'physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???,
    'alignment': 4}], 'section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0,
    'size': 0, 'entropy': -0.0}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address':
    134512788, 'size': 28, 'entropy': 3.6375375112660517}, {'name': '.text', 'type': '
    PROGBITS', 'virtual_address': 134512816, 'size': 92584, 'entropy': 6.419944555244165}, {'
    name': '.fini', 'type': 'PROGBITS', 'virtual_address': 134605400, 'size': 23, 'entropy':
    4.001822825622232}, {'name': '.rodata', 'type': 'PROGBITS', 'virtual_address': 134605440,
     'size': 33187, 'entropy': 6.407619484825426}, {'name': '.eh_frame', 'type': 'PROGBITS',
    'virtual_address': 134638628, 'size': 4, 'entropy': -0.0}, {'name': '.ctors', 'type': '
    PROGBITS', 'virtual_address': 134639616, 'size': 12, 'entropy': 2.251629167387823}, {'
    name': '.dtors', 'type': 'PROGBITS', 'virtual_address': 134639628, 'size': 8, 'entropy':
    1.0}, {'name': '.jcr', 'type': 'PROGBITS', 'virtual_address': 134639636, 'size': 4, '
    entropy': -0.0}, {'name': '.got.plt', 'type': 'PROGBITS', 'virtual_address': 134639640, '
    size': 12, 'entropy': -0.0}, {'name': '.data', 'type': 'PROGBITS', 'virtual_address':
    134639680, 'size': 18776, 'entropy': 0.5847502356535077}, {'name': '.bss', 'type': '
    NOBITS', 'virtual_address': 134658464, 'size': 35496, 'entropy': 5.200474765192193}, {'
    name': '.comment', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 4374, 'entropy':
    3.6143694458867563}, {'name': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size
    ': 111, 'entropy': 3.9464055386973294}, {'name': '.symtab', 'type': 'SYMTAB', '
    virtual_address': 0, 'size': 22032, 'entropy': 4.332352458289653}, {'name': '.strtab', '
    type': 'STRTAB', 'virtual_address': 0, 'size': 19593, 'entropy': 4.5644808522575895}]}
strings: {'numstrings': 130, 'avlength': 5.392307692307693, 'printabledist': [10, 2, 0, 1,
    77, 11, 1, 3, 10, 3, 0, 1, 14, 1, 2, 11, 14, 0, 1, 1, 4, 3, 0, 1, 9, 7, 0, 24, 11, 1, 3,
    1, 5, 3, 0, 4, 37, 3, 2, 3, 6, 0, 1, 3, 4, 0, 1, 2, 52, 26, 13, 22, 8, 7, 16, 14, 12, 5,
    11, 12, 12, 7, 14, 8, 1, 4, 1, 5, 2, 0, 4, 0, 32, 2, 21, 0, 1, 1, 1, 1, 2, 1, 1, 14, 23,
    14, 1, 6, 6, 1, 0, 0, 9, 5, 2, 0], 'printables': 701, 'entropy': 5.4746551513671875, '
    paths': 0, 'urls': 0, 'registry': 0, 'MZ': 0}
EntryPoint: [['Main ', 134512996], ['__do_', 134512832], ['frame', 134512912], ['__do_',
    134605360], ['print', 134514183], ['print', 134514241], ['print', 134514456], ['print',
    134514749], ['threa', 134544816], ['pthre', 134545092], ['pthre', 134545154], ['pthre',
    134545361], ['pthre', 134545406], ['resta', 134545600], ['pthre', 134545613], ['pthre',
    134547532], ['pthre', 134547568], ['enque', 134547956], ['remov', 134547985], ['__pth',
    134548042], ['threa', 134548342], ['new_s', 134548405], ['suspe', 134548465], ['pthre',
    134549336], ['threa', 134549337], ['pthre', 134549400], ['pthre', 134549474], ['wait_',
    134550484], ['__pth', 134550524], ['wait_', 134550600], ['resta', 134550643], ['threa',
    134550830], ['suspe', 134550893], ['pthre', 134553159], ['suspe', 134553574], ['threa',
    134553612], ['pthre', 134553675], ['pthre', 134554093], ['pthre', 134555359], ['pthre',
    134555214], ['threa', 134556224], ['enque', 134556964], ['remov', 134556993], ['__pth',
    134557118], ['resta', 134557177], ['threa', 134557268], ['cond_', 134557331], ['suspe',
    134557391], ['__pth', 134558236], ['threa', 134558295]]
ExitPoint: []
Opcodes: ['push', 'mov', 'push', 'call', 'add', 'call', 'call', 'pop', 'pop', 'ret', 'mov', '
    ret', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop',
     'push', 'mov', 'sub', 'cmp', 'je', 'jmp', 'add', 'mov', 'call', 'mov', 'mov', 'test', '
    jne', 'mov', 'test', 'je', 'sub', 'push', 'call', 'add', 'mov', 'leave', 'ret', 'nop', '
    lea', 'push', 'mov', 'mov', 'sub', 'call', 'pop', 'add', 'test', 'je', 'push', 'push', '
    push', 'push', 'call', 'add', 'mov', 'test', 'je', 'mov', 'test', 'je', 'sub', 'push', '
    call', 'add', 'leave', 'ret', 'nop', 'nop', 'nop', 'xor', 'pop', 'mov', 'and', 'push', '
    push', 'push', 'push', 'push', 'push', 'push', 'push', 'call', 'hlt', 'nop', 'nop', 'push
    ', 'mov', 'sub', 'mov', 'mov', 'mov', 'sub', 'mov', 'mov', 'add', 'mov', 'mov', 'jmp', '
    mov', 'mov', 'sub', 'mov', 'mov', 'sub', 'mov', 'xor', 'mov', 'xor', 'xor', 'mov', 'inc',
     'cmp', 'jle', 'leave', 'ret', 'push', 'mov', 'push', 'sub', 'mov', 'mov', 'mov', 'mov',
    'inc', 'and', 'mov', 'mov', 'mov', 'mov', 'mov', 'imul', 'mov', 'imul', 'add', 'mov', '
    mul', 'add', 'mov', 'mov', 'mov', 'add', 'adc', 'mov', 'mov', 'mov', 'mov', 'mov', 'xor',
     'mov', 'mov', 'mov', 'lea', 'mov', 'mov', 'cmp', 'jae', 'inc', 'mov', 'inc', 'mov', 'mov
    ', 'mov', 'mov', 'sub', 'mov', 'mov', 'add', 'pop', 'pop', 'ret', 'push', 'mov', 'sub', '
    mov', 'sub', 'push', 'call', 'add', 'and', 'mov', 'call', 'mov', 'mov', 'not', 'and', '
    xor', 'leave', 'ret', 'push', 'mov']
Opcode-Occurrence: {'push': 1994, 'mov': 3120, 'call': 887, 'add': 984, 'pop': 55, 'ret': 49,
     'nop': 21, 'sub': 775, 'cmp': 314, 'je': 120, 'jmp': 254, 'test': 175, 'jne': 194, '
    leave': 27, 'lea': 267, 'xor': 19, 'and': 73, 'hlt': 1, 'inc': 91, 'jle': 56, 'imul': 4,
    'mul': 4, 'adc': 7, 'jae': 3, 'not': 14, 'jge': 1, 'dec': 31, 'cld': 56, 'repne scasb':
    12, 'movsx': 33, 'jl': 30, 'jg': 38, 'jns': 9, 'neg': 8, 'div': 3, 'or': 40, 'shl': 51, '
    jbe': 20, 'setae': 1, 'rep stosb': 5, 'rep stosd': 4, 'shr': 26, 'bts': 4, 'setg': 1, '
    shld': 2, 'jb': 12, 'ja': 7, 'movzx': 2, 'shrd': 2, 'repe cmpsb': 30, 'seta': 30, 'setb':
     30, 'rep movsd': 5, 'sar': 9, 'idiv': 4, 'js': 3, 'setl': 1, 'setne': 1}
Image Size: 134658456
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 144832}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 18840, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, '
    Segment 1': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 2': {'Type
    ': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ | WRITE'}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.init': {'
    min': 3.6375375112660517, 'max': 3.6375375112660517, 'total': 3.6375375112660517, 'count
    ': 1, 'mean': 3.6375375112660517}, '.text': {'min': 6.419944555244165, 'max':
    6.419944555244165, 'total': 6.419944555244165, 'count': 1, 'mean': 6.419944555244165}, '.
    fini': {'min': 4.001822825622232, 'max': 4.001822825622232, 'total': 4.001822825622232, '
    count': 1, 'mean': 4.001822825622232}, '.rodata': {'min': 6.407619484825426, 'max':
    6.407619484825426, 'total': 6.407619484825426, 'count': 1, 'mean': 6.407619484825426}, '.
    eh_frame': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.ctors': {'
    min': 2.251629167387823, 'max': 2.251629167387823, 'total': 2.251629167387823, 'count':
    1, 'mean': 2.251629167387823}, '.dtors': {'min': 1.0, 'max': 1.0, 'total': 1.0, 'count':
    1, 'mean': 1.0}, '.jcr': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0},
     '.got.plt': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.data': {'
    min': 0.5847502356535077, 'max': 0.5847502356535077, 'total': 0.5847502356535077, 'count
    ': 1, 'mean': 0.5847502356535077}, '.bss': {'min': 5.200474765192193, 'max':
    5.200474765192193, 'total': 5.200474765192193, 'count': 1, 'mean': 5.200474765192193}, '.
    comment': {'min': 3.6143694458867563, 'max': 3.6143694458867563, 'total':
    3.6143694458867563, 'count': 1, 'mean': 3.6143694458867563}, '.shstrtab': {'min':
    3.9464055386973294, 'max': 3.9464055386973294, 'total': 3.9464055386973294, 'count': 1, '
    mean': 3.9464055386973294}, '.symtab': {'min': 4.332352458289653, 'max':
    4.332352458289653, 'total': 4.332352458289653, 'count': 1, 'mean': 4.332352458289653}, '.
    strtab': {'min': 4.5644808522575895, 'max': 4.5644808522575895, 'total':
    4.5644808522575895, 'count': 1, 'mean': 4.5644808522575895}}
Kolmogorov Complexity: 46 KB
```

M.2.2. Dynamic Analysis features

```
        "Behavior": {
            "files_opened": [
                "/proc/net/route"
            ],
            "files_written": [],
            "files_deleted": [],
            "command_executions": [],
            "files_attribute_changed": [],
            "processes_terminated": [
                "/tmp/EB93A6/996E.elf"
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "get system information on Linux"
            ]
        }
```

M.3. Okiru

M.3.1. Psuedo-static analysis features

```
section: {'entry': '0x110f4', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, '
    file_offset': 0, 'props': ['Type: NULL']}, {'name': '.interp', 'size': 20, 'entropy':
    3.684183719779189, 'file_offset': 308, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.
    note.ABI-tag', 'size': 32, 'entropy': 1.498778124459133, 'file_offset': 328, 'props': ['
    Type: NOTE', 'ALLOC']}, {'name': '.hash', 'size': 372, 'entropy': 1.4538268956397733, '
    file_offset': 360, 'props': ['Type: HASH', 'ALLOC']}, {'name': '.dynsym', 'size': 864, '
    entropy': 2.4834977274409598, 'file_offset': 732, 'props': ['Type: DYNSYM', 'ALLOC']}, {'
    name': '.dynstr', 'size': 411, 'entropy': 4.173851640731175, 'file_offset': 1596, 'props
    ': ['Type: STRTAB', 'ALLOC']}, {'name': '.rela.plt', 'size': 600, 'entropy':
    2.83614668956366, 'file_offset': 2008, 'props': ['Type: RELA', 'ALLOC']}, {'name': '.init
    ', 'size': 34, 'entropy': 3.4104172527605194, 'file_offset': 2608, 'props': ['Type:
    PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.plt', 'size': 624, 'entropy':
    3.9770756122374173, 'file_offset': 2644, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR
    ']}, {'name': '.text', 'size': 26112, 'entropy': 6.781177741932852, 'file_offset': 3268,
    'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.
    interp', 'Size': 20, 'Virtual Address': 65844}, {'Name': '.note.ABI-tag', 'Size': 32, '
    Virtual Address': 65864}, {'Name': '.hash', 'Size': 372, 'Virtual Address': 65896}, {'
    Name': '.dynsym', 'Size': 864, 'Virtual Address': 66268}, {'Name': '.dynstr', 'Size':
    411, 'Virtual Address': 67132}, {'Name': '.rela.plt', 'Size': 600, 'Virtual Address':
    67544}, {'Name': '.init', 'Size': 34, 'Virtual Address': 68144}, {'Name': '.plt', 'Size':
     624, 'Virtual Address': 68180}, {'Name': '.text', 'Size': 26112, 'Virtual Address':
    68804}, {'Name': '.fini', 'Size': 22, 'Virtual Address': 94916}, {'Name': '.rodata', '
    Size': 14348, 'Virtual Address': 94940}, {'Name': '.eh_frame', 'Size': 4, 'Virtual
    Address': 109288}, {'Name': '.ctors', 'Size': 8, 'Virtual Address': 122692}, {'Name': '.
    dtors', 'Size': 8, 'Virtual Address': 122700}, {'Name': '.dynamic', 'Size': 168, 'Virtual
     Address': 122708}, {'Name': '.got.plt', 'Size': 212, 'Virtual Address': 122876}, {'Name
    ': '.data', 'Size': 448, 'Virtual Address': 123088}, {'Name': '.bss', 'Size': 196, '
    Virtual Address': 123536}, {'Name': '.comment', 'Size': 71, 'Virtual Address': 0}, {'Name
    ': '.ARC.attributes', 'Size': 50, 'Virtual Address': 0}, {'Name': '.shstrtab', 'Size':
    169, 'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.PHDR', 'Size': 256, '
    Virtual Address': 65588}, {'Type': 'SEGMENT_TYPES.INTERP', 'Size': 20, 'Virtual Address':
     65844}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 43756, 'Virtual Address': 65536}, {'Type
    ': 'SEGMENT_TYPES.LOAD', 'Size': 844, 'Virtual Address': 122692}, {'Type': 'SEGMENT_TYPES
    .DYNAMIC', 'Size': 168, 'Virtual Address': 122708}, {'Type': 'SEGMENT_TYPES.NOTE', 'Size
    ': 32, 'Virtual Address': 65864}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual
     Address': 0}, {'Type': 'SEGMENT_TYPES.GNU_RELRO', 'Size': 188, 'Virtual Address':
    122692}]}
imports: {'DynamicSymbols': ['', 'ioctl', 'sysconf', 'getdtablesize', 'recv', 'connect', '
    memmove', 'getpid', 'prctl', 'memcpy', 'readlink', 'getuid', 'malloc', 'sleep', 'recvfrom
    ', 'socket', 'select', 'readdir', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', '
    bind', 'inet_addr', 'setsockopt', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork
    ', '__uClibc_main', 'memset', 'inet_ntoa', 'getppid', 'time', 'opendir', 'getsockopt', '
    __errno_location', 'exit', 'atoi', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close
    ', 'free'], 'SymbolTable': ['', 'ioctl', 'sysconf', 'getdtablesize', 'recv', 'connect', '
    memmove', 'getpid', 'prctl', 'memcpy', 'readlink', 'getuid', 'malloc', 'sleep', 'recvfrom
    ', 'socket', 'select', 'readdir', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', '
    bind', 'inet_addr', 'setsockopt', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork
    ', '__uClibc_main', 'memset', 'inet_ntoa', 'getppid', 'time', 'opendir', 'getsockopt', '
    __errno_location', 'exit', 'atoi', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close
    ', 'free']}
exports: ['ioctl', 'sysconf', 'getdtablesize', 'recv', 'connect', 'memmove', 'getpid', 'prctl
    ', 'memcpy', 'readlink', 'getuid', 'malloc', 'sleep', 'recvfrom', 'socket', 'select', '
    readdir', 'send', 'abort', 'accept', 'calloc', 'write', 'kill', 'bind', 'inet_addr', '
    setsockopt', 'signal', 'read', 'sendto', 'realloc', 'listen', 'fork', '__uClibc_main', '
    memset', 'inet_ntoa', 'getppid', 'time', 'opendir', 'getsockopt', '__errno_location', '
    exit', 'atoi', 'open', 'clock', 'setsid', 'closedir', 'fcntl', 'close', 'free', '
    getsockname']
general: {'size': 50980, 'virtual_size': 122880, 'has_debug': 0, 'exports': 54, 'imports':
    21, 'has_relocations': 0, 'symbols': 54}
header: {'file_type': 'EXECUTABLE', 'entry_point': 69876, 'machine_type': 'ARC_COMPACT', '
    header_size': 52, 'program_headers': [{'type': 'PHDR', 'virtual_address': 65588, '
    physical_address': 65588, 'physical_size': 256, 'virtual_size': 256, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4}, {'type': 'INTERP', 'virtual_address': 65844, '
    physical_address': 65844, 'physical_size': 20, 'virtual_size': 20, 'flags': SEGMENT_FLAGS
    .R, 'alignment': 1}, {'type': 'LOAD', 'virtual_address': 65536, 'physical_address':
    65536, 'physical_size': 43756, 'virtual_size': 43756, 'flags': SEGMENT_FLAGS.???, '
    alignment': 8192}, {'type': 'LOAD', 'virtual_address': 122692, 'physical_address':
    122692, 'physical_size': 844, 'virtual_size': 1040, 'flags': SEGMENT_FLAGS.???, '
    alignment': 8192}, {'type': 'DYNAMIC', 'virtual_address': 122708, 'physical_address':
    122708, 'physical_size': 168, 'virtual_size': 168, 'flags': SEGMENT_FLAGS.???, 'alignment
    ': 4}, {'type': 'NOTE', 'virtual_address': 65864, 'physical_address': 65864, '
    physical_size': 32, 'virtual_size': 32, 'flags': SEGMENT_FLAGS.R, 'alignment': 4}, {'type
    ': 'GNU_STACK', 'virtual_address': 0, 'physical_address': 0, 'physical_size': 0, '
    virtual_size': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 16}, {'type': 'GNU_RELRO', '
    virtual_address': 122692, 'physical_address': 122692, 'physical_size': 188, 'virtual_size
    ': 188, 'flags': SEGMENT_FLAGS.R, 'alignment': 1}], 'section_headers': [{'name': '', '
    type': 'NULL', 'virtual_address': 0, 'size': 0, 'entropy': -0.0}, {'name': '.interp', '
    type': 'PROGBITS', 'virtual_address': 65844, 'size': 20, 'entropy': 3.684183719779189},
    {'name': '.note.ABI-tag', 'type': 'NOTE', 'virtual_address': 65864, 'size': 32, 'entropy
    ': 1.498778124459133}, {'name': '.hash', 'type': 'HASH', 'virtual_address': 65896, 'size
    ': 372, 'entropy': 1.4538268956397733}, {'name': '.dynsym', 'type': 'DYNSYM', '
    virtual_address': 66268, 'size': 864, 'entropy': 2.4834977274409598}, {'name': '.dynstr',
     'type': 'STRTAB', 'virtual_address': 67132, 'size': 411, 'entropy': 4.173851640731175},
    {'name': '.rela.plt', 'type': 'RELA', 'virtual_address': 67544, 'size': 600, 'entropy':
    2.83614668956366}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address': 68144, 'size
    ': 34, 'entropy': 3.4104172527605194}, {'name': '.plt', 'type': 'PROGBITS', '
    virtual_address': 68180, 'size': 624, 'entropy': 3.9770756122374173}, {'name': '.text', '
    type': 'PROGBITS', 'virtual_address': 68804, 'size': 26112, 'entropy':
    6.781177741932852}, {'name': '.fini', 'type': 'PROGBITS', 'virtual_address': 94916, 'size
    ': 22, 'entropy': 3.5726236638951634}, {'name': '.rodata', 'type': 'PROGBITS', '
    virtual_address': 94940, 'size': 14348, 'entropy': 5.3546136920220055}, {'name': '.
    eh_frame', 'type': 'PROGBITS', 'virtual_address': 109288, 'size': 4, 'entropy': -0.0}, {'
    name': '.ctors', 'type': 'PROGBITS', 'virtual_address': 122692, 'size': 8, 'entropy':
    1.0}, {'name': '.dtors', 'type': 'PROGBITS', 'virtual_address': 122700, 'size': 8, '
    entropy': 1.0}, {'name': '.dynamic', 'type': 'DYNAMIC', 'virtual_address': 122708, 'size
    ': 168, 'entropy': 2.062889341399933}, {'name': '.got.plt', 'type': 'PROGBITS', '
    virtual_address': 122876, 'size': 212, 'entropy': 2.030485858892551}, {'name': '.data', '
    type': 'PROGBITS', 'virtual_address': 123088, 'size': 448, 'entropy':
    3.5866852802054248}, {'name': '.bss', 'type': 'NOBITS', 'virtual_address': 123536, 'size
    ': 196, 'entropy': 5.288541914062246}, {'name': '.comment', 'type': 'PROGBITS', '
    virtual_address': 0, 'size': 71, 'entropy': 4.709763584891619}, {'name': '.ARC.attributes
    ', 'type': 'ARM_EXIDX', 'virtual_address': 0, 'size': 50, 'entropy': 4.578171939471838},
    {'name': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 169, 'entropy':
    4.206684723226437}]}
strings: {'numstrings': 48, 'avlength': 5.729166666666667, 'printabledist': [24, 10, 2, 6, 9,
     6, 5, 4, 12, 1, 2, 0, 2, 1, 1, 4, 4, 0, 0, 0, 0, 1, 0, 0, 0, 2, 5, 0, 0, 0, 0, 0, 18, 3,
     5, 2, 13, 5, 1, 1, 4, 0, 8, 0, 0, 0, 2, 3, 4, 1, 0, 3, 3, 4, 2, 0, 1, 0, 5, 0, 0, 0, 1,
    0, 1, 0, 2, 0, 0, 3, 0, 1, 4, 0, 4, 2, 0, 0, 1, 0, 17, 5, 4, 3, 0, 8, 5, 1, 11, 4, 3, 1,
    0, 4, 1, 0], 'printables': 275, 'entropy': 5.393893241882324, 'paths': 0, 'urls': 0, '
    registry': 0, 'MZ': 0}
EntryPoint: [['Main ', 69876], ['ioctl', 68204], ['sysco', 68216], ['getdt', 68228], ['recv',
     68240], ['conne', 68252], ['memmo', 68264], ['getpi', 68276], ['prctl', 68288], ['memcp
    ', 68300], ['readl', 68312], ['getui', 68324], ['mallo', 68336], ['sleep', 68348], ['
    recvf', 68360], ['socke', 68372], ['selec', 68384], ['readd', 68396], ['send', 68408], ['
    abort', 68420], ['accep', 68432], ['callo', 68444], ['write', 68456], ['kill', 68468], ['
    bind', 68480], ['inet_', 68492], ['setso', 68504], ['signa', 68516], ['read', 68528], ['
    sendt', 68540], ['reall', 68552], ['liste', 68564], ['fork', 68576], ['__uCl', 68588], ['
    memse', 68600], ['inet_', 68612], ['getpp', 68624], ['time', 68636], ['opend', 68648], ['
    getso', 68660], ['__err', 68672], ['exit', 68684], ['atoi', 68696], ['open', 68708], ['
    clock', 68720], ['setsi', 68732], ['close', 68744], ['fcntl', 68756], ['close', 68768],
    ['free', 68780]]
ExitPoint: [['Dynamic Symbol', 'abort'], ['Dynamic Symbol', 'exit']]
Opcodes: ['add', 'add', 'int1', 'rol', 'add', 'or', 'add', 'loopne', 'push', 'or', 'add', '
    and', 'add', 'fild', 'add', 'daa', 'int1', 'rcr', 'and', 'test', 'and', 'sub', 'push', '
    and', 'cmp', 'or', 'add', 'movups', 'or', 'add', 'mov', 'fsub', 'add', 'add', 'add', 'add
    ', 'jne', 'add', 'add', 'jno', 'add', 'add', 'xchg', 'push', 'add', 'push', 'or', 'add',
    'fst', 'add', 'test', 'add', 'pushal', 'add', 'add', 'lodsd', 'xchg', 'or', 'or', 'add',
    'fdiv', 'or', 'add', 'fdivr', 'jne', 'jno', 'jb', 'push', 'out', 'inc', 'fadd', 'fcmove',
     'jno', 'add', 'push', 'out', 'inc', 'fcom', 'loopne', 'add', 'jle', 'iretd', 'inc', 'add
    ', 'bswap', 'dec', 'jbe', 'add', 'loop', 'cmpsb', 'pxor', 'add', 'and', 'out', 'inc', '
    fcomp', 'pxor', 'add', 'salc', 'or', 'add', 'iretd', 'jmp', 'and', 'add', 'add', 'int1',
    'rol']
Opcode-Occurrence: {'add': 34, 'int1': 3, 'rol': 2, 'or': 9, 'loopne': 2, 'push': 6, 'and':
    6, 'fild': 1, 'daa': 1, 'rcr': 1, 'test': 2, 'sub': 1, 'cmp': 1, 'movups': 1, 'mov': 1, '
    fsub': 1, 'jne': 2, 'jno': 3, 'xchg': 2, 'fst': 1, 'pushal': 1, 'lodsd': 1, 'fdiv': 1, '
    fdivr': 1, 'jb': 1, 'out': 3, 'inc': 4, 'fadd': 1, 'fcmove': 1, 'fcom': 1, 'jle': 1, '
    iretd': 2, 'bswap': 1, 'dec': 1, 'jbe': 1, 'loop': 1, 'cmpsb': 1, 'pxor': 2, 'fcomp': 1,
    'salc': 1, 'jmp': 1}
Image Size: 123536
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 45264}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 844, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.PHDR', 'Flags': 'READ | EXECUTE'}, '
    Segment 1': {'Type': 'SEGMENT_TYPES.INTERP', 'Flags': 'READ'}, 'Segment 2': {'Type': '
    SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.
    LOAD', 'Flags': 'READ | WRITE'}, 'Segment 4': {'Type': 'SEGMENT_TYPES.DYNAMIC', 'Flags':
    'READ | WRITE'}, 'Segment 5': {'Type': 'SEGMENT_TYPES.NOTE', 'Flags': 'READ'}, 'Segment
    6': {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ | WRITE'}, 'Segment 7': {'Type': '
    SEGMENT_TYPES.GNU_RELRO', 'Flags': 'READ'}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.interp':
    {'min': 3.684183719779189, 'max': 3.684183719779189, 'total': 3.684183719779189, 'count':
     1, 'mean': 3.684183719779189}, '.note.ABI-tag': {'min': 1.498778124459133, 'max':
    1.498778124459133, 'total': 1.498778124459133, 'count': 1, 'mean': 1.498778124459133}, '.
    hash': {'min': 1.4538268956397733, 'max': 1.4538268956397733, 'total':
    1.4538268956397733, 'count': 1, 'mean': 1.4538268956397733}, '.dynsym': {'min':
    2.4834977274409598, 'max': 2.4834977274409598, 'total': 2.4834977274409598, 'count': 1, '
    mean': 2.4834977274409598}, '.dynstr': {'min': 4.173851640731175, 'max':
    4.173851640731175, 'total': 4.173851640731175, 'count': 1, 'mean': 4.173851640731175}, '.
    rela.plt': {'min': 2.83614668956366, 'max': 2.83614668956366, 'total': 2.83614668956366,
    'count': 1, 'mean': 2.83614668956366}, '.init': {'min': 3.4104172527605194, 'max':
    3.4104172527605194, 'total': 3.4104172527605194, 'count': 1, 'mean': 3.4104172527605194},
     '.plt': {'min': 3.9770756122374173, 'max': 3.9770756122374173, 'total':
    3.9770756122374173, 'count': 1, 'mean': 3.9770756122374173}, '.text': {'min':
    6.781177741932852, 'max': 6.781177741932852, 'total': 6.781177741932852, 'count': 1, '
    mean': 6.781177741932852}, '.fini': {'min': 3.5726236638951634, 'max':
    3.5726236638951634, 'total': 3.5726236638951634, 'count': 1, 'mean': 3.5726236638951634},
     '.rodata': {'min': 5.3546136920220055, 'max': 5.3546136920220055, 'total':
    5.3546136920220055, 'count': 1, 'mean': 5.3546136920220055}, '.eh_frame': {'min': 0.0, '
    max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.ctors': {'min': 1.0, 'max': 1.0, '
    total': 1.0, 'count': 1, 'mean': 1.0}, '.dtors': {'min': 1.0, 'max': 1.0, 'total': 1.0, '
    count': 1, 'mean': 1.0}, '.dynamic': {'min': 2.062889341399933, 'max': 2.062889341399933,
     'total': 2.062889341399933, 'count': 1, 'mean': 2.062889341399933}, '.got.plt': {'min':
    2.030485858892551, 'max': 2.030485858892551, 'total': 2.030485858892551, 'count': 1, '
    mean': 2.030485858892551}, '.data': {'min': 3.5866852802054248, 'max':
    3.5866852802054248, 'total': 3.5866852802054248, 'count': 1, 'mean': 3.5866852802054248},
     '.bss': {'min': 5.288541914062246, 'max': 5.288541914062246, 'total': 5.288541914062246,
     'count': 1, 'mean': 5.288541914062246}, '.comment': {'min': 4.709763584891619, 'max':
    4.709763584891619, 'total': 4.709763584891619, 'count': 1, 'mean': 4.709763584891619}, '.
    ARC.attributes': {'min': 4.578171939471838, 'max': 4.578171939471838, 'total':
    4.578171939471838, 'count': 1, 'mean': 4.578171939471838}, '.shstrtab': {'min':
    4.206684723226437, 'max': 4.206684723226437, 'total': 4.206684723226437, 'count': 1, '
    mean': 4.206684723226437}}
Kolmogorov Complexity: 16 KB
```

M.3.2. Dynamic Analysis Features

```

    {
        "Behavior": {
            "files_opened": [
                "/proc/net/tcp"
                "/dev/FTWDT101\\ watchdog",
                "/dev/FTWDT101_watchdog",
                "/dev/misc/watchdog",
                "/dev/watchdog"
            ],
            "files_written": [],
            "files_deleted": [],
            "command_executions": [],
            "files_attribute_changed": [],
            "processes_terminated": [
                "/tmp/EB93A6/996E.elf",
                "-sh"
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Detected TCP or UDP traffic on non-standard ports",
                "Uses network protocols on non-standard ports"
            ]
        }
    }
```

M.4. MooBot

M.4.1. Pseudo-static analysis features

```
{
section: {'entry': '0x8194', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, '
    file_offset': 0, 'props': ['Type: NULL']}, {'name': '.init', 'size': 16, 'entropy': 3.75,
     'file_offset': 212, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.text
    ', 'size': 78632, 'entropy': 6.043327442975583, 'file_offset': 240, 'props': ['Type:
    PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.fini', 'size': 16, 'entropy': 3.75, '
    file_offset': 78872, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.
    rodata', 'size': 3224, 'entropy': 4.979348101368475, 'file_offset': 78888, 'props': ['
    Type: PROGBITS', 'ALLOC']}, {'name': '.ARM.extab', 'size': 24, 'entropy':
    3.0016291673878226, 'file_offset': 82112, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name':
     '.ARM.exidx', 'size': 280, 'entropy': 4.494767067674535, 'file_offset': 82136, 'props':
    ['Type: ARM_EXIDX', 'ALLOC']}, {'name': '.eh_frame', 'size': 4, 'entropy': -0.0, '
    file_offset': 82416, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.tbss', '
    size': 8, 'entropy': 2.0, 'file_offset': 82420, 'props': ['Type: NOBITS', 'ALLOC', 'WRITE
    ']}, {'name': '.init_array', 'size': 4, 'entropy': 1.5, 'file_offset': 82420, 'props': ['
    Type: INIT_ARRAY', 'ALLOC', 'WRITE']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.
    init', 'Size': 16, 'Virtual Address': 32980}, {'Name': '.text', 'Size': 78632, 'Virtual
    Address': 33008}, {'Name': '.fini', 'Size': 16, 'Virtual Address': 111640}, {'Name': '.
    rodata', 'Size': 3224, 'Virtual Address': 111656}, {'Name': '.ARM.extab', 'Size': 24, '
    Virtual Address': 114880}, {'Name': '.ARM.exidx', 'Size': 280, 'Virtual Address':
    114904}, {'Name': '.eh_frame', 'Size': 4, 'Virtual Address': 147952}, {'Name': '.tbss', '
    Size': 8, 'Virtual Address': 147956}, {'Name': '.init_array', 'Size': 4, 'Virtual Address
    ': 147956}, {'Name': '.fini_array', 'Size': 4, 'Virtual Address': 147960}, {'Name': '.jcr
    ', 'Size': 4, 'Virtual Address': 147964}, {'Name': '.got', 'Size': 168, 'Virtual Address
    ': 147968}, {'Name': '.data', 'Size': 516, 'Virtual Address': 148136}, {'Name': '.bss', '
    Size': 12588, 'Virtual Address': 148652}, {'Name': '.comment', 'Size': 2434, 'Virtual
    Address': 0}, {'Name': '.debug_aranges', 'Size': 192, 'Virtual Address': 0}, {'Name': '.
    debug_pubnames', 'Size': 531, 'Virtual Address': 0}, {'Name': '.debug_info', 'Size':
    7459, 'Virtual Address': 0}, {'Name': '.debug_abbrev', 'Size': 1682, 'Virtual Address':
    0}, {'Name': '.debug_line', 'Size': 2503, 'Virtual Address': 0}, {'Name': '.debug_frame',
     'Size': 696, 'Virtual Address': 0}, {'Name': '.debug_str', 'Size': 2250, 'Virtual
    Address': 0}, {'Name': '.debug_loc', 'Size': 4495, 'Virtual Address': 0}, {'Name': '.
    debug_ranges', 'Size': 1368, 'Virtual Address': 0}, {'Name': '.ARM.attributes', 'Size':
    22, 'Virtual Address': 0}, {'Name': '.shstrtab', 'Size': 279, 'Virtual Address': 0}, {'
    Name': '.symtab', 'Size': 18144, 'Virtual Address': 0}, {'Name': '.strtab', 'Size': 8859,
     'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.ARM_EXIDX', 'Size': 280, '
    Virtual Address': 114904}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 82416, 'Virtual Address
    ': 32768}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 700, 'Virtual Address': 147952}, {'Type
    ': 'SEGMENT_TYPES.TLS', 'Size': 0, 'Virtual Address': 147956}, {'Type': 'SEGMENT_TYPES.
    GNU_STACK', 'Size': 0, 'Virtual Address': 0}]}
imports: {'SymbolTable': ['', '__cxa_begin_cleanup', '__cxa_call_unexpected', '
    __nptl_deallocate_tsd', '__deregister_frame_info', '__cxa_type_match', '__pthread_unwind
    ', '__nptl_nthreads', '__h_errno_location', '__gnu_Unwind_Find_exidx', '
    _Jv_RegisterClasses', '__register_frame_info']}
exports: ['__do_global_dtors_aux', 'frame_dummy', 'anti_gdb_entry', 'resolve_cnc_addr', '
    ensure_single_instance', 'setup_connection', 'add_auth_entry', '__syscall_select', '
    fd_to_DIR', '__sys_accept', '__sys_connect', '__sys_recv', '__sys_recvfrom', '__sys_send
    ', '__sys_sendto', '__malloc_largebin_index', '__malloc_trim', 'nprocessors_onln', '
    __pthread_return_0', '__check_one_fd', '__syscall_nanosleep', 'init_static_tls', '
    get_eit_entry', 'unwind_phase2_forced', 'unwind_phase2', '__gnu_unwind_pr_common', '
    ___Unwind_ForcedUnwind', '__gnu_Unwind_RaiseException', '__libc_sigaction', '
    __GI_sigaddset', '__GI_fopen', 'getrlimit', 'ioctl', '__GI_initstate_r', '__GI_sigaction
    ', '__GI_time', 'getgid', '__aeabi_read_tp', '__getpid', 'sysconf', 'random', '
    __GI_getpagesize', 'getdtablesize', 'fdopendir', 'recv', 'connect', '__GI___uClibc_fini',
     'sigemptyset', '__pthread_mutex_lock', '__sigdelset']
general: {'size': 135195, 'virtual_size': 0, 'has_debug': 0, 'exports': 1134, 'imports': 0, '
    has_relocations': 0, 'symbols': 1134}
header: {'file_type': 'EXECUTABLE', 'entry_point': 33172, 'machine_type': 'ARM', 'header_size
    ': 52, 'program_headers': [{'type': 'ARM_EXIDX', 'virtual_address': 114904, '
    physical_address': 114904, 'physical_size': 280, 'virtual_size': 280, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 4}, {'type': 'LOAD', 'virtual_address': 32768, '
    physical_address': 32768, 'physical_size': 82416, 'virtual_size': 82416, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 32768}, {'type': 'LOAD', 'virtual_address': 147952, '
    physical_address': 147952, 'physical_size': 700, 'virtual_size': 13288, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 32768}, {'type': 'TLS', 'virtual_address': 147956, '
    physical_address': 147956, 'physical_size': 0, 'virtual_size': 8, 'flags': SEGMENT_FLAGS.
    R, 'alignment': 4}, {'type': 'GNU_STACK', 'virtual_address': 0, 'physical_address': 0, '
    physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 4}], '
    section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0, 'size': 0, 'entropy
    ': -0.0}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address': 32980, 'size': 16, '
    entropy': 3.75}, {'name': '.text', 'type': 'PROGBITS', 'virtual_address': 33008, 'size':
    78632, 'entropy': 6.043327442975583}, {'name': '.fini', 'type': 'PROGBITS', '
    virtual_address': 111640, 'size': 16, 'entropy': 3.75}, {'name': '.rodata', 'type': '
    PROGBITS', 'virtual_address': 111656, 'size': 3224, 'entropy': 4.979348101368475}, {'name
    ': '.ARM.extab', 'type': 'PROGBITS', 'virtual_address': 114880, 'size': 24, 'entropy':
    3.0016291673878226}, {'name': '.ARM.exidx', 'type': 'ARM_EXIDX', 'virtual_address':
    114904, 'size': 280, 'entropy': 4.494767067674535}, {'name': '.eh_frame', 'type': '
    PROGBITS', 'virtual_address': 147952, 'size': 4, 'entropy': -0.0}, {'name': '.tbss', '
    type': 'NOBITS', 'virtual_address': 147956, 'size': 8, 'entropy': 2.0}, {'name': '.
    init_array', 'type': 'INIT_ARRAY', 'virtual_address': 147956, 'size': 4, 'entropy': 1.5},
     {'name': '.fini_array', 'type': 'FINI_ARRAY', 'virtual_address': 147960, 'size': 4, '
    entropy': 1.5}, {'name': '.jcr', 'type': 'PROGBITS', 'virtual_address': 147964, 'size':
    4, 'entropy': -0.0}, {'name': '.got', 'type': 'PROGBITS', 'virtual_address': 147968, '
    size': 168, 'entropy': 3.3050193770970253}, {'name': '.data', 'type': 'PROGBITS', '
    virtual_address': 148136, 'size': 516, 'entropy': 3.8899937364255988}, {'name': '.bss', '
    type': 'NOBITS', 'virtual_address': 148652, 'size': 12588, 'entropy': 5.516411687837259},
     {'name': '.comment', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 2434, 'entropy':
    3.74982617971519}, {'name': '.debug_aranges', 'type': 'PROGBITS', 'virtual_address': 0, '
    size': 192, 'entropy': 2.0660523192543327}, {'name': '.debug_pubnames', 'type': 'PROGBITS
    ', 'virtual_address': 0, 'size': 531, 'entropy': 4.7663720680363095}, {'name': '.
    debug_info', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 7459, 'entropy':
    5.141757006174466}, {'name': '.debug_abbrev', 'type': 'PROGBITS', 'virtual_address': 0, '
    size': 1682, 'entropy': 4.581741703113143}, {'name': '.debug_line', 'type': 'PROGBITS', '
    virtual_address': 0, 'size': 2503, 'entropy': 5.8404665183166395}, {'name': '.debug_frame
    ', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 696, 'entropy': 4.168323002102134},
    {'name': '.debug_str', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 2250, 'entropy':
     5.044492868897249}, {'name': '.debug_loc', 'type': 'PROGBITS', 'virtual_address': 0, '
    size': 4495, 'entropy': 3.2169624594073363}, {'name': '.debug_ranges', 'type': 'PROGBITS
    ', 'virtual_address': 0, 'size': 1368, 'entropy': 2.840044198252748}, {'name': '.ARM.
    attributes', 'type': 'ARM_ATTRIBUTES', 'virtual_address': 0, 'size': 22, 'entropy':
    3.259141984247901}, {'name': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size':
     279, 'entropy': 4.306943408525259}, {'name': '.symtab', 'type': 'SYMTAB', '
    virtual_address': 0, 'size': 18144, 'entropy': 3.408498530455322}, {'name': '.strtab', '
    type': 'STRTAB', 'virtual_address': 0, 'size': 8859, 'entropy': 4.546139438667893}]}
strings: {'numstrings': 1, 'avlength': 5.0, 'printabledist': [1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], '
    printables': 5, 'entropy': 2.321928024291992, 'paths': 0, 'urls': 0, 'registry': 0, 'MZ':
     0}
EntryPoint: [['Main ', 33172], ['__do_', 33008], ['frame', 33076], ['anti_', 57932], ['resol
    ', 58312], ['ensur', 57956], ['setup', 62496], ['add_a', 62696], ['__sys', 75576], ['
    fd_to', 76364], ['__sys', 77952], ['__sys', 78204], ['__sys', 78592], ['__sys', 78772],
    ['__sys', 78980], ['__sys', 79160], ['__mal', 79916], ['__mal', 83676], ['nproc', 87520],
     ['__pth', 92180], ['__che', 92372], ['__sys', 94748], ['init_', 104052], ['get_e',
    106420], ['unwin', 106964], ['unwin', 107328], ['__gnu', 108072], ['___Un', 109600], ['
    __gnu', 107532], ['__lib', 93716], ['__GI_', 79512], ['__GI_', 97396], ['getrl', 94476],
    ['ioctl', 75164], ['__GI_', 86308], ['__GI_', 93716], ['__GI_', 75980], ['getgi', 94416],
     ['__aea', 93888], ['__get', 90628], ['sysco', 87852], ['rando', 85244], ['__GI_',
    94436], ['getdt', 94332], ['fdope', 76768], ['recv', 78660], ['conne', 78272], ['__GI_',
    92248], ['sigem', 79592], ['__pth', 92180]]
ExitPoint: []
Opcodes: ['or', 'fild', 'loop', 'scasd', 'sbb', 'adc', 'jmp', 'add', 'aam', 'add', 'push', '
```

```
    jecxz', 'add', 'sbb', 'xor', 'jecxz', 'add', 'adc', 'jmp', 'adc', 'xor', 'in', 'inc', '
    mov', 'loope', 'inc', 'add', 'add', 'add', 'or', 'fild', 'loop', 'scasd', 'sbb']
Opcode-Occurrence: {'or': 2, 'fild': 2, 'loop': 2, 'scasd': 2, 'sbb': 3, 'adc': 3, 'jmp': 2,
    'add': 7, 'aam': 1, 'push': 1, 'jecxz': 2, 'xor': 2, 'in': 1, 'inc': 2, 'mov': 1, 'loope
    ': 1}
Image Size: 148652
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 83396}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 700, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.ARM_EXIDX', 'Flags': 'READ'}, 'Segment
    1': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, 'Segment 2': {'Type': '
    SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.TLS',
     'Flags': 'READ'}, 'Segment 4': {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ |
    WRITE | EXECUTE'}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.init': {'
    min': 3.75, 'max': 3.75, 'total': 3.75, 'count': 1, 'mean': 3.75}, '.text': {'min':
    6.043327442975583, 'max': 6.043327442975583, 'total': 6.043327442975583, 'count': 1, '
    mean': 6.043327442975583}, '.fini': {'min': 3.75, 'max': 3.75, 'total': 3.75, 'count': 1,
     'mean': 3.75}, '.rodata': {'min': 4.979348101368475, 'max': 4.979348101368475, 'total':
    4.979348101368475, 'count': 1, 'mean': 4.979348101368475}, '.ARM.extab': {'min':
    3.0016291673878226, 'max': 3.0016291673878226, 'total': 3.0016291673878226, 'count': 1, '
    mean': 3.0016291673878226}, '.ARM.exidx': {'min': 4.494767067674535, 'max':
    4.494767067674535, 'total': 4.494767067674535, 'count': 1, 'mean': 4.494767067674535}, '.
    eh_frame': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.tbss': {'
    min': 2.0, 'max': 2.0, 'total': 2.0, 'count': 1, 'mean': 2.0}, '.init_array': {'min':
    1.5, 'max': 1.5, 'total': 1.5, 'count': 1, 'mean': 1.5}, '.fini_array': {'min': 1.5, 'max
    ': 1.5, 'total': 1.5, 'count': 1, 'mean': 1.5}, '.jcr': {'min': 0.0, 'max': 0.0, 'total':
     0.0, 'count': 1, 'mean': 0.0}, '.got': {'min': 3.3050193770970253, 'max':
    3.3050193770970253, 'total': 3.3050193770970253, 'count': 1, 'mean': 3.3050193770970253},
     '.data': {'min': 3.8899937364255988, 'max': 3.8899937364255988, 'total':
    3.8899937364255988, 'count': 1, 'mean': 3.8899937364255988}, '.bss': {'min':
    5.516411687837259, 'max': 5.516411687837259, 'total': 5.516411687837259, 'count': 1, '
    mean': 5.516411687837259}, '.comment': {'min': 3.74982617971519, 'max': 3.74982617971519,
     'total': 3.74982617971519, 'count': 1, 'mean': 3.74982617971519}, '.debug_aranges': {'
    min': 2.0660523192543327, 'max': 2.0660523192543327, 'total': 2.0660523192543327, 'count
    ': 1, 'mean': 2.0660523192543327}, '.debug_pubnames': {'min': 4.7663720680363095, 'max':
    4.7663720680363095, 'total': 4.7663720680363095, 'count': 1, 'mean': 4.7663720680363095},
     '.debug_info': {'min': 5.141757006174466, 'max': 5.141757006174466, 'total':
    5.141757006174466, 'count': 1, 'mean': 5.141757006174466}, '.debug_abbrev': {'min':
    4.581741703113143, 'max': 4.581741703113143, 'total': 4.581741703113143, 'count': 1, '
    mean': 4.581741703113143}, '.debug_line': {'min': 5.8404665183166395, 'max':
    5.8404665183166395, 'total': 5.8404665183166395, 'count': 1, 'mean': 5.8404665183166395},
     '.debug_frame': {'min': 4.168323002102134, 'max': 4.168323002102134, 'total':
    4.168323002102134, 'count': 1, 'mean': 4.168323002102134}, '.debug_str': {'min':
    5.044492868897249, 'max': 5.044492868897249, 'total': 5.044492868897249, 'count': 1, '
    mean': 5.044492868897249}, '.debug_loc': {'min': 3.2169624594073363, 'max':
    3.2169624594073363, 'total': 3.2169624594073363, 'count': 1, 'mean': 3.2169624594073363},
     '.debug_ranges': {'min': 2.840044198252748, 'max': 2.840044198252748, 'total':
    2.840044198252748, 'count': 1, 'mean': 2.840044198252748}, '.ARM.attributes': {'min':
    3.259141984247901, 'max': 3.259141984247901, 'total': 3.259141984247901, 'count': 1, '
    mean': 3.259141984247901}, '.shstrtab': {'min': 4.306943408525259, 'max':
    4.306943408525259, 'total': 4.306943408525259, 'count': 1, 'mean': 4.306943408525259}, '.
    symtab': {'min': 3.408498530455322, 'max': 3.408498530455322, 'total': 3.408498530455322,
     'count': 1, 'mean': 3.408498530455322}, '.strtab': {'min': 4.546139438667893, 'max':
    4.546139438667893, 'total': 4.546139438667893, 'count': 1, 'mean': 4.546139438667893}}
Kolmogorov Complexity: 39 KB
}
```

M.4.2. Dynamic Analysis Features

```
    "Behavior": {
    "files_opened": [
                "/etc/mtab",
                "/proc/",
                "/proc/5318/mounts",
                "/proc/sys/vm/mmap_min_addr",
                "/tmp/base",
                "/proc/5041/mounts",
                "/tmp/sample",
                "//",
                "//lib/multipath",
                "//lib/multipath/libchecktur.so",
                "//lib/multipath/libforeign-nvme.so",
                "//lib/multipath/libprioconst.so",
                "/dev/loop-control",
                "/dev/loop0",
                "/dev/loop1",
                "/dev/loop10",
                "/dev/loop11",
                "/dev/loop12",
                "/dev/loop2",
                "/dev/loop3",
                "/dev/loop4",
                "/dev/loop5",
                "/dev/loop6",
                "/dev/loop7",
                "/dev/loop8",
                "/dev/loop9",
                "/dev/sda",
                "/dev/sda1",
                "/dev/sda2",
                "/etc//localtime",
                "/etc/adjtime",
                "/etc/apparmor.d/abstractions/base",
                "/etc/apparmor.d/abstractions/consoles",
                "/etc/apparmor.d/abstractions/dbus-strict",
                "/etc/apparmor.d/abstractions/kerberosclient",
                "/etc/apparmor.d/abstractions/ldapclient",
                "/etc/apparmor.d/abstractions/likewise",
                "/etc/apparmor.d/abstractions/mdns",
                "/etc/apparmor.d/abstractions/nameservice",
                "/etc/apparmor.d/abstractions/nis",
                "/etc/apparmor.d/abstractions/openssl",
                "/etc/apparmor.d/abstractions/perl",
                "/etc/apparmor.d/abstractions/python",
                "/etc/apparmor.d/abstractions/ssl_certs",
                "/etc/apparmor.d/abstractions/winbind",
                "/etc/apparmor.d/tunables/alias",
                "/etc/apparmor.d/tunables/global",
                "/etc/apparmor.d/tunables/home",
                "/etc/apparmor.d/tunables/home.d",
                "/etc/apparmor.d/tunables/home.d/site.local"
            ],
            "files_written": [
                "/etc/udev/rules.d/70-snap.snapd.rules.YJ6Mvq39y0DV~",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/event0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/input/input0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXPWRBN:00/wakeup/wakeup40/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/hwmon0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/power_supply/ACAD/wakeup41/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0003:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/INT0E0C:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/LNXCPU:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/LNXCPU:01/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/PNP0C02:02/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/PNP0C02:03/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/VMW0001:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:00/wakeup/wakeup0/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/APP0001:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0001:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0100:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0103:00/uevent
```

```
",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0200:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0303:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0400:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0400:01/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:00/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:01/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:02/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:03/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:04/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:05/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:06/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:07/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:08/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:09/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0a/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0b/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0c/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0d/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0e/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:0f/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:10/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:11/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:12/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:13/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:14/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:15/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:16/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:17/uevent",
                "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/PNP0A03:00/device:01/PNP0501:18/uevent"
            ],
            "files_deleted": [
                "/tmp/base",
                "/tmp/sample",
                "/etc/udev/rules.d/70-snap.core18.rules",
                "/etc/udev/rules.d/70-snap.core20.rules",
                "/etc/udev/rules.d/70-snap.lxd.rules",
                "/etc/udev/rules.d/70-snap.powershell.rules",
                "/run/snapd-snap.socket",
                "/run/snapd.socket",
                "/tmp/sanity-mountpoint-724243119",
                "/tmp/sanity-squashfs-212460176",
                "/var/lib/snapd/features/refresh-app-awareness",
                "/var/lib/snapd/maintenance.json",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.activate.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.benchmark.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.buginfo.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.check-kernel.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.daemon.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.hook.configure.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.hook.install.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.hook.remove.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.lxc-to-lxd.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.lxc.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.lxd.bin",
                "/var/lib/snapd/seccomp/bpf/snap.lxd.migrate.bin",
                "/var/lib/snapd/seccomp/bpf/snap.powershell.powershell.bin",
                "/var/lib/snapd/system-key"
            ],
            "command_executions": [
                "fusermount -u -q -z -- /run/user/1000/gvfs",
                "/usr/lib/snapd/snap-failure snapd",
                "systemctl stop snapd.socket",
                "/snap/snapd/15177/usr/lib/snapd/snapd",
                "/usr/sbin/apparmor_parser --preprocess",
                "systemctl --version",
                "systemd-detect-virt",
                "systemd-detect-virt --container",
                "mount -t squashfs /tmp/sanity-squashfs-212460176 /tmp/sanity-mountpoint-724243119",
                "umount -l /tmp/sanity-mountpoint-724243119",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp version-info",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.activate.src /var/lib/snapd/seccomp/bpf/snap.lxd.activate.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.benchmark.src /var/lib/snapd/seccomp/bpf/snap.lxd.benchmark.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.buginfo.src /var/lib/snapd/seccomp/bpf/snap.lxd.buginfo.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.check-kernel.src /var/lib/snapd/seccomp/bpf/snap.lxd.check-kernel.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.daemon.src /var/lib/snapd/seccomp/bpf/snap.lxd.daemon.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.hook.configure.src /var/lib/snapd/seccomp/bpf/snap.lxd.hook.configure.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.hook.install.src /var/lib/snapd/seccomp/bpf/snap.lxd.hook.install.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.hook.remove.src /var/lib/snapd/seccomp/bpf/snap.lxd.hook.remove.bin",
                "/snap/snapd/15177/usr/lib/snapd/snap-seccomp compile /var/lib/snapd/seccomp/bpf/snap.lxd.lxc-to-lxd.src /var/lib/snapd/seccomp/bpf/snap.lxd.lxc-to-lxd.bin"
            ],
            "files_attribute_changed": [
                "/run/mount/utab.lock"
            ],
            "processes_terminated": [],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [
                "snapd.socket"
            ],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Sample deletes itself",
                "Uses the \"uname\" system call to query kernel version information (possible evasion)",
                "Sample reads /proc/mounts (often used for finding a writable filesystem)",
                "Detected TCP or UDP traffic on non-standard ports",
                "Performs DNS lookups",
                "Performs DNS lookups",
                "Sample tries to kill multiple processes (SIGKILL)",
                "Executes the \"sed\" command used to modify input streams (typically from files or pipes)",
                "Executes commands using a shell command-line interpreter",
                "Executes the \"systemctl\" command used for controlling the systemd system and service manager",
                "Reads CPU information from /sys indicative of miner or evasive malware",
                "Reads system information from the proc file system"
            ]
        }
```

M.5. EnemyBot

M.5.1. Pseudo-static analysis features

```
section: {'entry': '0x81b0', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, 'file_offset': 0, 'props': ['Type: NULL']}, {'name': '.init', 'size': 16, 'entropy': 3.75, 'file_offset': 180, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.text', 'size': 153136, 'entropy': 5.89356430505523, 'file_offset': 208, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.fini', 'size': 16, 'entropy': 3.75, 'file_offset': 153344, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.rodata', 'size': 38212, 'entropy': 5.640213711474537, 'file_offset': 153360, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.ARM.extab', 'size': 24, 'entropy': 3.0016291673878226, 'file_offset': 191572, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.ARM.exidx', 'size': 16, 'entropy': 2.4056390622295662, 'file_offset': 191596, 'props': ['Type: ARM_EXIDX', 'ALLOC']}, {'name': '.eh_frame', 'size': 4, 'entropy': -0.0, 'file_offset': 192512, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.init_array', 'size': 4, 'entropy': 1.5, 'file_offset': 192516, 'props': ['Type: INIT_ARRAY', 'ALLOC', 'WRITE']}, {'name': '.fini_array', 'size': 4, 'entropy': 1.5, 'file_offset': 192520, 'props': ['Type: FINI_ARRAY', 'ALLOC', 'WRITE']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.init', 'Size': 16, 'Virtual Address': 32948}, {'Name': '.text', 'Size': 153136, 'Virtual Address': 32976}, {'Name': '.fini', 'Size': 16, 'Virtual Address': 186112}, {'Name': '.rodata', 'Size': 38212, 'Virtual Address': 186128}, {'Name': '.ARM.extab', 'Size': 24, 'Virtual Address': 224340}, {'Name': '.ARM.exidx', 'Size': 16, 'Virtual Address': 224364}, {'Name': '.eh_frame', 'Size': 4, 'Virtual Address': 258048}, {'Name': '.init_array', 'Size': 4, 'Virtual Address': 258052}, {'Name': '.fini_array', 'Size': 4, 'Virtual Address': 258056}, {'Name': '.jcr', 'Size': 4, 'Virtual Address': 258060}, {'Name': '.data.rel.ro', 'Size': 24, 'Virtual Address': 258064}, {'Name': '.got', 'Size': 124, 'Virtual Address': 258088}, {'Name': '.data', 'Size': 3636, 'Virtual Address': 258212}, {'Name': '.bss', 'Size': 28560, 'Virtual Address': 261848}, {'Name': '.comment', 'Size': 3694, 'Virtual Address': 0}, {'Name': '.debug_aranges', 'Size': 224, 'Virtual Address': 0}, {'Name': '.debug_info', 'Size': 1200, 'Virtual Address': 0}, {'Name': '.debug_abbrev', 'Size': 140, 'Virtual Address': 0}, {'Name': '.debug_line', 'Size': 1621, 'Virtual Address': 0}, {'Name': '.debug_frame', 'Size': 88, 'Virtual Address': 0}, {'Name': '.ARM.attributes', 'Size': 16, 'Virtual Address': 0}, {'Name': '.shstrtab', 'Size': 234, 'Virtual Address': 0}, {'Name': '.symtab', 'Size': 27104, 'Virtual Address': 0}, {'Name': '.strtab', 'Size': 13878, 'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.ARM_EXIDX', 'Size': 16, 'Virtual Address': 224364}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 191612, 'Virtual Address': 32768}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 3800, 'Virtual Address': 258048}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual Address': 0}]}
imports: {'SymbolTable': ['', '__deregister_frame_info', '__pthread_initialize_minimal', '_Jv_RegisterClasses', '__register_frame_info']}
exports: ['__do_global_dtors_aux', 'frame_dummy', 'can_consume', 'consume_iacs', 'consume_any_prompt', 'consume_shell_prompt', 'consume_user_prompt', 'consume_pass_prompt', 'deobf', 'add_auth_entry', 'random_auth_entry', 'setup_connection', 'consume_resp_prompt', 'port80_setup_connection', 'switch_socket_transport', 'anti_debug_shit', 'fd_to_DIR', '_charpad', '_fp_out_narrow', '_promoted_size', 'gaih_inet_serv', '__set_h_errno', 'gaih_inet', 'inet_pton4', 'inet_ntop4', '__malloc_largebin_index', '__malloc_trim', 'nprocessors_onln', '__pthread_return_0', '__check_one_fd', '__initbuf', 'skip_nospace', 'skip_and_NUL_space', '__read_etc_hosts_r',
    '__GI_if_freenameindex', '__GI_execve', 'ovtcp', '__libc_sigaction', 'DNSw', '
    __aeabi_dcmple', 'strcpy', 'recvLine', '__cmpdf2', 'cp', '__GI_memchr', 'bcmp', '
    adb_status', '__GI___glibc_strerror_r', 'waitpid', 'add_sock']
general: {'size': 245518, 'virtual_size': 0, 'has_debug': 0, 'exports': 1694, 'imports': 0, '
    has_relocations': 0, 'symbols': 1694}
header: {'file_type': 'EXECUTABLE', 'entry_point': 33200, 'machine_type': 'ARM', 'header_size
    ': 52, 'program_headers': [{'type': 'ARM_EXIDX', 'virtual_address': 224364, '
    physical_address': 224364, 'physical_size': 16, 'virtual_size': 16, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 4}, {'type': 'LOAD', 'virtual_address': 32768, '
    physical_address': 32768, 'physical_size': 191612, 'virtual_size': 191612, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 32768}, {'type': 'LOAD', 'virtual_address': 258048, '
    physical_address': 258048, 'physical_size': 3800, 'virtual_size': 32360, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 32768}, {'type': 'GNU_STACK', 'virtual_address': 0, '
    physical_address': 0, 'physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???,
    'alignment': 4}], 'section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0,
    'size': 0, 'entropy': -0.0}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address':
    32948, 'size': 16, 'entropy': 3.75}, {'name': '.text', 'type': 'PROGBITS', '
    virtual_address': 32976, 'size': 153136, 'entropy': 5.89356430505523}, {'name': '.fini',
    'type': 'PROGBITS', 'virtual_address': 186112, 'size': 16, 'entropy': 3.75}, {'name': '.
    rodata', 'type': 'PROGBITS', 'virtual_address': 186128, 'size': 38212, 'entropy':
    5.640213711474537}, {'name': '.ARM.extab', 'type': 'PROGBITS', 'virtual_address': 224340,
     'size': 24, 'entropy': 3.0016291673878226}, {'name': '.ARM.exidx', 'type': 'ARM_EXIDX',
    'virtual_address': 224364, 'size': 16, 'entropy': 2.4056390622295662}, {'name': '.
    eh_frame', 'type': 'PROGBITS', 'virtual_address': 258048, 'size': 4, 'entropy': -0.0}, {'
    name': '.init_array', 'type': 'INIT_ARRAY', 'virtual_address': 258052, 'size': 4, '
    entropy': 1.5}, {'name': '.fini_array', 'type': 'FINI_ARRAY', 'virtual_address': 258056,
    'size': 4, 'entropy': 1.5}, {'name': '.jcr', 'type': 'PROGBITS', 'virtual_address':
    258060, 'size': 4, 'entropy': -0.0}, {'name': '.data.rel.ro', 'type': 'PROGBITS', '
    virtual_address': 258064, 'size': 24, 'entropy': 1.5535088547976783}, {'name': '.got', '
    type': 'PROGBITS', 'virtual_address': 258088, 'size': 124, 'entropy': 3.944872193865425},
     {'name': '.data', 'type': 'PROGBITS', 'virtual_address': 258212, 'size': 3636, 'entropy
    ': 5.352623296448144}, {'name': '.bss', 'type': 'NOBITS', 'virtual_address': 261848, '
    size': 28560, 'entropy': 4.1583556590512885}, {'name': '.comment', 'type': 'PROGBITS', '
    virtual_address': 0, 'size': 3694, 'entropy': 3.711521459541648}, {'name': '.
    debug_aranges', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 224, 'entropy':
    1.9826414095147717}, {'name': '.debug_info', 'type': 'PROGBITS', 'virtual_address': 0, '
    size': 1200, 'entropy': 5.044586995237964}, {'name': '.debug_abbrev', 'type': 'PROGBITS',
     'virtual_address': 0, 'size': 140, 'entropy': 3.346439344671015}, {'name': '.debug_line
    ', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 1621, 'entropy':
    4.1989518567566115}, {'name': '.debug_frame', 'type': 'PROGBITS', 'virtual_address': 0, '
    size': 88, 'entropy': 3.316880986799484}, {'name': '.ARM.attributes', 'type': '
    ARM_ATTRIBUTES', 'virtual_address': 0, 'size': 16, 'entropy': 2.646782221599798}, {'name
    ': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 234, 'entropy':
    4.275366364838433}, {'name': '.symtab', 'type': 'SYMTAB', 'virtual_address': 0, 'size':
    27104, 'entropy': 3.5320301941545336}, {'name': '.strtab', 'type': 'STRTAB', '
    virtual_address': 0, 'size': 13878, 'entropy': 4.665672857707997}]}
strings: {'numstrings': 1, 'avlength': 5.0, 'printabledist': [1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], '
    printables': 5, 'entropy': 2.321928024291992, 'paths': 0, 'urls': 0, 'registry': 0, 'MZ':
     0}
EntryPoint: [['Main ', 33200], ['__do_', 33036], ['frame', 33104], ['can_c', 81824], ['consu
    ', 81928], ['consu', 82568], ['consu', 82868], ['consu', 83104], ['consu', 83556], ['
    deobf', 83904], ['add_a', 84276], ['rando', 84712], ['setup', 84960], ['consu', 85300],
    ['port8', 100580], ['switc', 101432], ['anti_', 117800], ['fd_to', 133696], ['_char',
    138084], ['_fp_o', 138168], ['_prom', 140728], ['gaih_', 147708], ['__set', 147904], ['
    gaih_', 147932], ['inet_', 151476], ['inet_', 152224], ['__mal', 155788], ['__mal',
    159500], ['nproc', 163612], ['__pth', 165692], ['__che', 165884], ['__ini', 176324], ['
    skip_', 181336], ['skip_', 181420], ['__rea', 185052], ['__GI_', 178744], ['__GI_',
    167428], ['ovtcp', 50784], ['__lib', 167148], ['DNSw', 64900], ['__aea', 186040], ['strcp
    ', 146516], ['recvL', 41620], ['__cmp', 185808], ['cp', 119496], ['__GI_', 175072], ['
    bcmp', 145600], ['adb_s', 101676], ['__GI_', 146980], ['waitp', 132300]]
ExitPoint: []
Opcodes: ['or', 'fild', 'loop', 'scasd', 'sbb', 'add', 'inc', 'or', 'inc', 'adc', 'inc', 'sbb
    ', 'inc', 'and', 'inc', 'sub', 'inc', 'xor', 'inc', 'cmp', 'inc', 'inc', 'inc', 'rol', '
    rol', 'rol', 'rol', 'rol', 'rol', 'rol', 'sub', 'in', 'xor', 'in', 'add', 'push', 'add',
    'sbb', 'xor', 'jecxz', 'add', 'adc', 'jmp', 'adc', 'xor', 'in', 'inc', 'mov', 'loope', '
    inc', 'add', 'add', 'add', 'lock add', 'add', 'sub', 'in', 'add', 'add', 'dec', 'loop', '
    add', 'adc', 'adc', 'das', 'adc', 'add', 'in', 'add', 'add', 'add', 'and', 'lahf', 'in',
    'add', 'pavgb', 'adc', 'or', 'fild', 'loop', 'scasd', 'sbb']
```

```
Opcode-Occurrence: {'or': 3, 'fild': 2, 'loop': 3, 'scasd': 2, 'sbb': 4, 'add': 16, 'inc':
    12, 'adc': 7, 'and': 2, 'sub': 3, 'xor': 4, 'cmp': 1, 'rol': 7, 'in': 6, 'push': 1, '
    jecxz': 1, 'jmp': 1, 'mov': 1, 'loope': 1, 'lock add': 1, 'dec': 1, 'das': 1, 'lahf': 1,
    'pavgb': 1}
Image Size: 261848
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 195428}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 3800, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.ARM_EXIDX', 'Flags': 'READ'}, 'Segment
    1': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, 'Segment 2': {'Type': '
    SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.
    GNU_STACK', 'Flags': 'READ | WRITE | EXECUTE'}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.init': {'
    min': 3.75, 'max': 3.75, 'total': 3.75, 'count': 1, 'mean': 3.75}, '.text': {'min':
    5.89356430505523, 'max': 5.89356430505523, 'total': 5.89356430505523, 'count': 1, 'mean':
     5.89356430505523}, '.fini': {'min': 3.75, 'max': 3.75, 'total': 3.75, 'count': 1, 'mean
    ': 3.75}, '.rodata': {'min': 5.640213711474537, 'max': 5.640213711474537, 'total':
    5.640213711474537, 'count': 1, 'mean': 5.640213711474537}, '.ARM.extab': {'min':
    3.0016291673878226, 'max': 3.0016291673878226, 'total': 3.0016291673878226, 'count': 1, '
    mean': 3.0016291673878226}, '.ARM.exidx': {'min': 2.4056390622295662, 'max':
    2.4056390622295662, 'total': 2.4056390622295662, 'count': 1, 'mean': 2.4056390622295662},
     '.eh_frame': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.
    init_array': {'min': 1.5, 'max': 1.5, 'total': 1.5, 'count': 1, 'mean': 1.5}, '.
    fini_array': {'min': 1.5, 'max': 1.5, 'total': 1.5, 'count': 1, 'mean': 1.5}, '.jcr': {'
    min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.data.rel.ro': {'min':
    1.5535088547976783, 'max': 1.5535088547976783, 'total': 1.5535088547976783, 'count': 1, '
    mean': 1.5535088547976783}, '.got': {'min': 3.944872193865425, 'max': 3.944872193865425,
    'total': 3.944872193865425, 'count': 1, 'mean': 3.944872193865425}, '.data': {'min':
    5.352623296448144, 'max': 5.352623296448144, 'total': 5.352623296448144, 'count': 1, '
    mean': 5.352623296448144}, '.bss': {'min': 4.1583556590512885, 'max': 4.1583556590512885,
     'total': 4.1583556590512885, 'count': 1, 'mean': 4.1583556590512885}, '.comment': {'min
    ': 3.711521459541648, 'max': 3.711521459541648, 'total': 3.711521459541648, 'count': 1, '
    mean': 3.711521459541648}, '.debug_aranges': {'min': 1.9826414095147717, 'max':
    1.9826414095147717, 'total': 1.9826414095147717, 'count': 1, 'mean': 1.9826414095147717},
     '.debug_info': {'min': 5.044586995237964, 'max': 5.044586995237964, 'total':
    5.044586995237964, 'count': 1, 'mean': 5.044586995237964}, '.debug_abbrev': {'min':
    3.346439344671015, 'max': 3.346439344671015, 'total': 3.346439344671015, 'count': 1, '
    mean': 3.346439344671015}, '.debug_line': {'min': 4.1989518567566115, 'max':
    4.1989518567566115, 'total': 4.1989518567566115, 'count': 1, 'mean': 4.1989518567566115},
     '.debug_frame': {'min': 3.316880986799484, 'max': 3.316880986799484, 'total':
    3.316880986799484, 'count': 1, 'mean': 3.316880986799484}, '.ARM.attributes': {'min':
    2.646782221599798, 'max': 2.646782221599798, 'total': 2.646782221599798, 'count': 1, '
    mean': 2.646782221599798}, '.shstrtab': {'min': 4.275366364838433, 'max':
    4.275366364838433, 'total': 4.275366364838433, 'count': 1, 'mean': 4.275366364838433}, '.
    symtab': {'min': 3.5320301941545336, 'max': 3.5320301941545336, 'total':
    3.5320301941545336, 'count': 1, 'mean': 3.5320301941545336}, '.strtab': {'min':
    4.665672857707997, 'max': 4.665672857707997, 'total': 4.665672857707997, 'count': 1, '
    mean': 4.665672857707997}}
Kolmogorov Complexity: 64 KB
```

M.5.2. Dynamic Analysis Features

```

{
        "Behavior": {
            "files_opened": [
                "/dev/misc/watchdog",
                "/dev/watchdog",
                "/etc/crontab",
                "/etc/rc.local",
                "/proc/",
                "/proc/sys/vm/mmap_min_addr",
                "/tmp/AUGXIF",
                "/tmp/sample"
            ],
            "files_written": [
                "/etc/crontab"
            ],
            "files_deleted": [],
            "command_executions": [],
            "files_attribute_changed": [],
            "processes_terminated": [],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Sample tries to persist itself using cron",
                "Uses the \"uname\" system call to query kernel version information (possible
                     evasion)",
                "Detected TCP or UDP traffic on non-standard ports"
            ]
        }
```

M.6. Netwire Rat

M.6.1. Pseudo-static analysis features

```
{
    "strings": {
      "numstrings": 4311,
      "avlength": 59.429598700997445,
      "printabledist": [
        1040, 122, 178, 163, 808, 271, 140, 157, 310, 170, 147, 148, 255, 193, 312, 150,
            13483, 13132, 13269, 13500, 13422, 13015, 13257, 12907, 13156, 13368, 559, 793,
            574, 494, 537, 567, 200, 12949, 12701, 13179, 13401, 12959, 13109, 472, 424, 425,
             127, 141, 360, 277, 280, 6862, 708, 286, 435, 706, 360, 273, 375, 714, 335, 357,
             180, 180, 342, 166, 167, 832, 191, 1135, 294, 757, 937, 2194, 591, 421, 456,
            1289, 545, 215, 981, 488, 1040, 1162, 514, 170, 1257, 1005, 1812, 639, 283, 261,
            328, 365, 164, 1195, 215, 133, 139, 146
      ],
      "printables": 256201,
      "entropy": 4.962445605417374,
      "paths": 0,
      "urls": 0,
      "registry": 0,
      "MZ": 9
    },
    "general": {
      "file_size": 1388664,
      "magic_number": "MZ",
      "bytes_in_last_block": 144,
      "blocks_in_file": 3,
      "num_relocs": 0,
      "header_paragraphs": 4,
      "min_extra_paragraphs": 0,
      "max_extra_paragraphs": 65535,
      "initial_ss": 0,
      "initial_sp": 184,
      "checksum": 0,
      "initial_ip": 0,
      "initial_cs": 0,
      "reloc_table_offset": 64,
      "overlay_number": 0
    },
    "header": {
      "e_magic": "MZ",
      "e_cblp": 144,
      "e_cp": 3,
      "e_crlc": 0,
      "e_cparhdr": 4,
      "e_minalloc": 0,
      "e_maxalloc": 65535,
      "e_ss": 0,
      "e_sp": 184,
      "e_csum": 0,
      "e_ip": 0,
      "e_cs": 0,
      "e_lfarlc": 64,
      "e_ovno": 0,
      "e_res": [0],
      "e_oemid": 0,
      "e_oeminfo": 0,
      "e_res2": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      "e_lfanew": 272
    },
    "Memory Layout": {
      "Initial SP": 184,
      "Max Allocation": 144
    },
    "EntryPoint": [
      ["Initi", "0x0:0x0"],
      ["Inter", "INT 0x21"],
      ["Inter", "INT 0x21"]
    ],
    "entry": {
      "entry": "IP: 0xb8, CS: 0x0"
    },
    "ExitPoint": [],
    "Opcodes": [
      "dec", "pop", "nop", "add", "add", "add", "add", "add", "inc", "add", "add", "add", "
          add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add",
          "add", "add", "add", "add", "add", "add", "adc", "add", "push", "pop", "mov", "mov
          ", "int", "mov", "int", "push", "push", "and", "outsw", "jb", "insw", "and", "outsb
          ", "outsb", "outsw", "je", "bound", "jb", "outsb", "and", "and", "push", "and", "or
          ", "and", "add", "add", "add", "sbb", "iret", "jb", "ret", "pushf", "jb", "ret", "
          pushf", "jb", "ret", "pushf", "xor", "and", "ret", "pushf", "in", "fld", "pushf", "
          jae", "ret", "pushf", "jg", "sbb", "inc", "jge", "pushf", "jg", "and", "ret", "
          pushf", "jg", "and", "ret", "pushf", "jnp", "inc", "pushf", "jnp", "ret", "pushf",
          "jnp", "push", "pushf", "push", "jge", "pushf", "jb", "ret", "jg", "pushf"
    ],
    "Opcode-Occurrence": {
      "dec": 1, "pop": 2, "nop": 1, "add": 30, "inc": 3, "adc": 1, "push": 6, "mov": 3, "int
          ": 2, "and": 9, "outsw": 2, "jb": 6, "insw": 1, "outsb": 3, "je": 1, "bound": 1, "
          or": 1, "sbb": 2, "iret": 1, "ret": 9, "pushf": 14, "xor": 1, "in": 1, "fld": 1, "
          jae": 1, "jg": 4, "jge": 2, "jnp": 3
    },
    "Image Size": 1388664,
    "Memory Size": {
      "Initial SS": 0,
      "Initial SP": 184,
      "Minimum Allocation": 65535,
      "Maximum Allocation": 184
    },
    "Header Size": {
      "DOS Header Size": 272
    },
    "Block Entropy": {
      "min": 0.0,
      "max": 7.864885891267435,
      "total": 7195.007462231047,
      "count": 1357,
      "mean": 5.302142566124574
    },
    "Kolmogorov Complexity": "74 KB",
    "Interrupt Info": [
      {
        "address": "0x147",
        "interrupt": "0x21"
      },
      {
        "address": "0x14c",
        "interrupt": "0x21"
      }
    ],
    "Stack Info": {
      "Initial SP": 184,
      "Initial SS": 0
  }
}
```

M.7. Bashlite

M.7.1. Psuedo-static features

```
section: {'entry': '0x8048164', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, '
    file_offset': 0, 'props': ['Type: NULL']}, {'name': '.init', 'size': 28, 'entropy':
    3.610577243331642, 'file_offset': 148, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR
    ']}, {'name': '.text', 'size': 91848, 'entropy': 6.435876216981836, 'file_offset': 176, '
    props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.fini', 'size': 23, 'entropy
    ': 4.001822825622232, 'file_offset': 92024, 'props': ['Type: PROGBITS', 'ALLOC', '
    EXECINSTR']}, {'name': '.rodata', 'size': 44451, 'entropy': 5.971768358349981, '
    file_offset': 92064, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.eh_frame', 'size':
     4, 'entropy': -0.0, 'file_offset': 136516, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name
    ': '.ctors', 'size': 12, 'entropy': 2.251629167387823, 'file_offset': 136520, 'props': ['
    Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.dtors', 'size': 8, 'entropy': 1.0, '
    file_offset': 136532, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.jcr', '
    size': 4, 'entropy': -0.0, 'file_offset': 136540, 'props': ['Type: PROGBITS', 'ALLOC', '
    WRITE']}, {'name': '.got.plt', 'size': 12, 'entropy': -0.0, 'file_offset': 136544, 'props
    ': ['Type: PROGBITS', 'ALLOC', 'WRITE']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.
    init', 'Size': 28, 'Virtual Address': 134512788}, {'Name': '.text', 'Size': 91848, '
    Virtual Address': 134512816}, {'Name': '.fini', 'Size': 23, 'Virtual Address':
    134604664}, {'Name': '.rodata', 'Size': 44451, 'Virtual Address': 134604704}, {'Name': '.
    eh_frame', 'Size': 4, 'Virtual Address': 134649156}, {'Name': '.ctors', 'Size': 12, '
    Virtual Address': 134653256}, {'Name': '.dtors', 'Size': 8, 'Virtual Address':
    134653268}, {'Name': '.jcr', 'Size': 4, 'Virtual Address': 134653276}, {'Name': '.got.plt
    ', 'Size': 12, 'Virtual Address': 134653280}, {'Name': '.data', 'Size': 24120, 'Virtual
    Address': 134653312}, {'Name': '.bss', 'Size': 35496, 'Virtual Address': 134677440}, {'
    Name': '.comment', 'Size': 4464, 'Virtual Address': 0}, {'Name': '.shstrtab', 'Size':
    111, 'Virtual Address': 0}, {'Name': '.symtab', 'Size': 22448, 'Virtual Address': 0}, {'
    Name': '.strtab', 'Size': 19935, 'Virtual Address': 0}], 'Segments': [{'Type': '
    SEGMENT_TYPES.LOAD', 'Size': 136520, 'Virtual Address': 134512640}, {'Type': '
    SEGMENT_TYPES.LOAD', 'Size': 24176, 'Virtual Address': 134653256}, {'Type': '
    SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual Address': 0}]}
imports: {'SymbolTable': ['', '__register_frame_info_bases', '__deregister_frame_info_bases',
     '_Jv_RegisterClasses']}
exports: ['__do_global_dtors_aux', 'frame_dummy', '__do_global_ctors_aux', 'printchar', '
    prints', 'printi', 'print', 'thread_self', 'pthread_kill_all_threads', '
    pthread_start_thread', 'pthread_start_thread_event', 'pthread_free', 'restart', '
    pthread_reap_children', 'pthread_insert_list', 'pthread_call_handlers', 'enqueue', '
    remove_from_queue', '__pthread_set_own_extricate_if', 'thread_self', '
    new_sem_extricate_func', 'suspend', 'pthread_null_sighandler', 'thread_self', '
    pthread_sighandler_rt', 'pthread_sighandler', 'wait_node_dequeue', '__pthread_acquire', '
    wait_node_free', 'restart', 'thread_self', 'suspend', 'pthread_handle_sigdebug', 'suspend
    ', 'thread_self', 'pthread_onexit_process', 'pthread_initialize', '
    pthread_handle_sigrestart', 'pthread_handle_sigcancel', 'thread_self', 'enqueue', '
    remove_from_queue', '__pthread_set_own_extricate_if', 'restart', 'thread_self', '
    cond_extricate_func', 'suspend', '__pthread_set_own_extricate_if', 'thread_self', '
    join_extricate_func']
general: {'size': 208295, 'virtual_size': 0, 'has_debug': 0, 'exports': 1403, 'imports': 0, '
    has_relocations': 0, 'symbols': 1403}
header: {'file_type': 'EXECUTABLE', 'entry_point': 134512996, 'machine_type': 'i386', '
    header_size': 52, 'program_headers': [{'type': 'LOAD', 'virtual_address': 134512640, '
    physical_address': 134512640, 'physical_size': 136520, 'virtual_size': 136520, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 134653256, '
    physical_address': 134653256, 'physical_size': 24176, 'virtual_size': 59680, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'GNU_STACK', 'virtual_address': 0, '
    physical_address': 0, 'physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???,
    'alignment': 4}], 'section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0,
    'size': 0, 'entropy': -0.0}, {'name': '.init', 'type': 'PROGBITS', 'virtual_address':
    134512788, 'size': 28, 'entropy': 3.610577243331642}, {'name': '.text', 'type': 'PROGBITS
    ', 'virtual_address': 134512816, 'size': 91848, 'entropy': 6.435876216981836}, {'name':
    '.fini', 'type': 'PROGBITS', 'virtual_address': 134604664, 'size': 23, 'entropy':
    4.001822825622232}, {'name': '.rodata', 'type': 'PROGBITS', 'virtual_address': 134604704,
     'size': 44451, 'entropy': 5.971768358349981}, {'name': '.eh_frame', 'type': 'PROGBITS',
    'virtual_address': 134649156, 'size': 4, 'entropy': -0.0}, {'name': '.ctors', 'type': '
    PROGBITS', 'virtual_address': 134653256, 'size': 12, 'entropy': 2.251629167387823}, {'
    name': '.dtors', 'type': 'PROGBITS', 'virtual_address': 134653268, 'size': 8, 'entropy':
    1.0}, {'name': '.jcr', 'type': 'PROGBITS', 'virtual_address': 134653276, 'size': 4, '
    entropy': -0.0}, {'name': '.got.plt', 'type': 'PROGBITS', 'virtual_address': 134653280, '
    size': 12, 'entropy': -0.0}, {'name': '.data', 'type': 'PROGBITS', 'virtual_address':
    134653312, 'size': 24120, 'entropy': 1.0846834733737656}, {'name': '.bss', 'type': '
    NOBITS', 'virtual_address': 134677440, 'size': 35496, 'entropy': 5.189375934525407}, {'
    name': '.comment', 'type': 'PROGBITS', 'virtual_address': 0, 'size': 4464, 'entropy':
    3.6143694458867563}, {'name': '.shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size
    ': 111, 'entropy': 3.9464055386973294}, {'name': '.symtab', 'type': 'SYMTAB', '
    virtual_address': 0, 'size': 22448, 'entropy': 4.325577064204675}, {'name': '.strtab', '
    type': 'STRTAB', 'virtual_address': 0, 'size': 19935, 'entropy': 4.535678049519343}]}
strings: {'numstrings': 130, 'avlength': 5.461538461538462, 'printabledist': [13, 2, 0, 1,
    75, 11, 0, 3, 9, 3, 0, 1, 12, 3, 2, 11, 3, 0, 1, 1, 5, 4, 0, 2, 10, 9, 0, 24, 11, 1, 3,
    0, 3, 3, 0, 4, 34, 14, 2, 3, 6, 0, 1, 2, 6, 1, 0, 0, 63, 30, 17, 26, 8, 5, 15, 15, 10, 5,
     11, 13, 12, 8, 16, 9, 2, 1, 0, 0, 1, 0, 6, 0, 30, 2, 21, 0, 3, 1, 1, 1, 3, 1, 1, 13, 24,
     12, 1, 6, 5, 1, 0, 0, 6, 5, 1, 0], 'printables': 710, 'entropy': 5.403498649597168, '
    paths': 0, 'urls': 0, 'registry': 0, 'MZ': 0}
EntryPoint: [['Main ', 134512996], ['__do_', 134512832], ['frame', 134512912], ['__do_',
    134604624], ['print', 134513510], ['print', 134513568], ['print', 134513783], ['print',
    134514076], ['threa', 134543636], ['pthre', 134543912], ['pthre', 134543974], ['pthre',
    134544181], ['pthre', 134544226], ['resta', 134544420], ['pthre', 134544433], ['pthre',
    134546352], ['pthre', 134546388], ['enque', 134546776], ['remov', 134546805], ['__pth',
    134546862], ['threa', 134547162], ['new_s', 134547225], ['suspe', 134547285], ['pthre',
    134548156], ['threa', 134548157], ['pthre', 134548220], ['pthre', 134548294], ['wait_',
    134549304], ['__pth', 134549344], ['wait_', 134549420], ['resta', 134549463], ['threa',
    134549650], ['suspe', 134549713], ['pthre', 134551979], ['suspe', 134552394], ['threa',
    134552432], ['pthre', 134552495], ['pthre', 134552913], ['pthre', 134554179], ['pthre',
    134554034], ['threa', 134555044], ['enque', 134555784], ['remov', 134555813], ['__pth',
    134555938], ['resta', 134555997], ['threa', 134556088], ['cond_', 134556151], ['suspe',
    134556211], ['__pth', 134557056], ['threa', 134557115]]
ExitPoint: []
Opcodes: ['push', 'mov', 'push', 'call', 'add', 'call', 'call', 'pop', 'pop', 'ret', 'mov', '
    ret', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop', 'nop',
     'push', 'mov', 'sub', 'cmp', 'je', 'jmp', 'add', 'mov', 'call', 'mov', 'mov', 'test', '
    jne', 'mov', 'test', 'je', 'sub', 'push', 'call', 'add', 'mov', 'leave', 'ret', 'nop', '
    lea', 'push', 'mov', 'mov', 'sub', 'call', 'pop', 'add', 'test', 'je', 'push', 'push', '
    push', 'push', 'call', 'add', 'mov', 'test', 'je', 'mov', 'test', 'je', 'sub', 'push', '
    call', 'add', 'leave', 'ret', 'nop', 'nop', 'nop', 'xor', 'pop', 'mov', 'and', 'push', '
    push', 'push', 'push', 'push', 'push', 'push', 'push', 'call', 'hlt', 'nop', 'nop', 'push
    ', 'mov', 'sub', 'mov', 'mov', 'mov', 'sub', 'mov', 'mov', 'add', 'mov', 'mov', 'jmp', '
    mov', 'mov', 'sub', 'mov', 'mov', 'sub', 'mov', 'xor', 'mov', 'xor', 'xor', 'mov', 'inc',
     'cmp', 'jle', 'leave', 'ret', 'push', 'mov', 'push', 'sub', 'mov', 'mov', 'mov', 'mov',
    'inc', 'and', 'mov', 'mov', 'mov', 'mov', 'mov', 'imul', 'mov', 'imul', 'add', 'mov', '
    mul', 'add', 'mov', 'mov', 'mov', 'add', 'adc', 'mov', 'mov', 'mov', 'mov', 'mov', 'xor',
     'mov', 'mov', 'mov', 'lea', 'mov', 'mov', 'cmp', 'jae', 'inc', 'mov', 'inc', 'mov', 'mov
    ', 'mov', 'mov', 'sub', 'mov', 'mov', 'add', 'pop', 'pop', 'ret', 'push', 'mov', 'push',
    'sub', 'mov', 'mov', 'mov', 'mov', 'mov', 'cld', 'mov', 'repne scasb', 'mov', 'not', 'dec
    ', 'dec', 'mov', 'jmp', 'inc', 'mov']
Opcode-Occurrence: {'push': 5002, 'mov': 8109, 'call': 2113, 'add': 1915, 'pop': 1221, 'ret':
     581, 'nop': 402, 'sub': 1430, 'cmp': 1404, 'je': 819, 'jmp': 939, 'test': 824, 'jne':
    837, 'leave': 45, 'lea': 1258, 'xor': 391, 'and': 333, 'hlt': 3, 'inc': 268, 'jle': 109,
    'imul': 37, 'mul': 4, 'adc': 9, 'jae': 32, 'cld': 68, 'repne scasb': 26, 'not': 43, 'dec
    ': 117, 'movsx': 69, 'jl': 33, 'jg': 70, 'jns': 52, 'neg': 66, 'div': 24, 'or': 274, 'shl
    ': 98, 'jbe': 136, 'rep stosb': 8, 'sar': 30, 'jge': 21, 'shr': 55, 'rep stosd': 7, 'bts
    ': 7, 'setg': 3, 'shld': 2, 'jb': 71, 'ja': 81, 'idiv': 20, 'movzx': 123, 'shrd': 4, '
    repe cmpsb': 31, 'seta': 31, 'setb': 31, 'js': 43, 'setl': 1, 'setne': 12, 'lock cmpxchg
    ': 11, 'sete': 18, 'xchg': 27, 'sbb': 21, 'int': 56, 'cdq': 16, 'fld': 19, 'fstp': 18, '
    rep movsd': 1, 'movsw': 1, 'movsb': 1, 'lodsb': 6, 'stosb': 6, 'ror': 4, 'bswap': 20, '
    rol': 1, 'bsr': 1, 'cwde': 1, 'fucom': 4, 'fnstsw': 9, 'sahf': 9, 'jp': 3, 'fldz': 2, '
    fxch': 13, 'fld1': 1, 'fdiv': 2, 'fucomp': 5, 'fchs': 1, 'fmul': 3, 'fdivrp': 1, 'fnstcw
    ': 1, 'fldcw': 2, 'fistp': 1, 'fild': 1, 'fsubp': 1, 'setge': 1, 'rep movsb': 2, 'std':
    1, 'btr': 1, 'scasb': 1, 'stosd': 3, 'stosw': 1}
Image Size: 134677432
Header Size: {'ELF Header Size': 52, 'Program Headers Total Size': 160696}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 24176, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | EXECUTE'}, '
    Segment 1': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 2': {'Type
    ': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ | WRITE'}}
```

```
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.init': {'min': 3.610577243331642, 'max': 3.610577243331642, 'total': 3.610577243331642, 'count': 1, 'mean': 3.610577243331642}, '.text': {'min': 6.435876216981836, 'max': 6.435876216981836, 'total': 6.435876216981836, 'count': 1, 'mean': 6.435876216981836}, '.fini': {'min': 4.001822825622232, 'max': 4.001822825622232, 'total': 4.001822825622232, 'count': 1, 'mean': 4.001822825622232}, '.rodata': {'min': 5.971768358349981, 'max': 5.971768358349981, 'total': 5.971768358349981, 'count': 1, 'mean': 5.971768358349981}, '.eh_frame': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.ctors': {'min': 2.251629167387823, 'max': 2.251629167387823, 'total': 2.251629167387823, 'count': 1, 'mean': 2.251629167387823}, '.dtors': {'min': 1.0, 'max': 1.0, 'total': 1.0, 'count': 1, 'mean': 1.0}, '.jcr': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.got.plt': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.data': {'min': 1.0846834733737656, 'max': 1.0846834733737656, 'total': 1.0846834733737656, 'count': 1, 'mean': 1.0846834733737656}, '.bss': {'min': 5.189375934525407, 'max': 5.189375934525407, 'total': 5.189375934525407, 'count': 1, 'mean': 5.189375934525407}, '.comment': {'min': 3.6143694458867563, 'max': 3.6143694458867563, 'total': 3.6143694458867563, 'count': 1, 'mean': 3.6143694458867563}, '.shstrtab': {'min': 3.9464055386973294, 'max': 3.9464055386973294, 'total': 3.9464055386973294, 'count': 1, 'mean': 3.9464055386973294}, '.symtab': {'min': 4.325577064204675, 'max': 4.325577064204675, 'total': 4.325577064204675, 'count': 1, 'mean': 4.325577064204675}, '.strtab': {'min': 4.535678049519343, 'max': 4.535678049519343, 'total': 4.535678049519343, 'count': 1, 'mean': 4.535678049519343}}
Kolmogorov Complexity: 46 KB
```

M.7.2. Dynamic features

```
        "Behavior": {
            "files_opened": [
                "/proc/net/route"
            ],
            "files_written": [],
            "files_deleted": [],
            "command_executions": [],
            "files_attribute_changed": [],
            "processes_terminated": [
                "/tmp/EB93A6/996E.elf",
                "/lib/systemd/systemd-udevd --daemon"
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": []
        }
    }
```

M.8. SmokeLoader

M.8.1. Psuedo-static features

```
    "strings": {
      "numstrings": 118,
      "avlength": 5.9491525423728815,
      "printabledist": [
        8, 9, 1, 8, 0, 5, 4, 16, 2, 7, 3, 4, 7, 4, 10, 4, 2, 3, 2, 1, 3, 1, 4, 4, 4, 3, 4, 8,
             14, 6, 8, 8, 6, 11, 3, 7, 4, 0, 4, 4, 9, 8, 5, 8, 10, 5, 5, 7, 7, 3, 18, 10, 8,
            3, 9, 8, 5, 3, 36, 14, 14, 3, 14, 8, 4, 9, 5, 11, 5, 14, 5, 3, 3, 13, 7, 5, 12,
            8, 10, 13, 5, 10, 12, 10, 10, 14, 4, 14, 10, 5, 10, 7, 17, 7, 10, 4
      ],
      "printables": 702,
      "entropy": 6.289097915508371,
      "paths": 0,
      "urls": 0,
      "registry": 0,
      "MZ": 2
    },
    "general": {
      "file_size": 33792,
      "magic_number": "MZ",
      "bytes_in_last_block": 128,
      "blocks_in_file": 1,
      "num_relocs": 0,
      "header_paragraphs": 4,
      "min_extra_paragraphs": 16,
      "max_extra_paragraphs": 65535,
      "initial_ss": 0,
      "initial_sp": 320,
      "checksum": 0,
      "initial_ip": 0,
      "initial_cs": 0,
      "reloc_table_offset": 64,
      "overlay_number": 0
    },
    "header": {
      "e_magic": "MZ",
      "e_cblp": 128,
      "e_cp": 1,
      "e_crlc": 0,
      "e_cparhdr": 4,
      "e_minalloc": 16,
      "e_maxalloc": 65535,
      "e_ss": 0,
      "e_sp": 320,
      "e_csum": 0,
      "e_ip": 0,
      "e_cs": 0,
      "e_lfarlc": 64,
      "e_ovno": 0,
      "e_res": [0],
      "e_oemid": 0,
      "e_oeminfo": 0,
      "e_res2": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      "e_lfanew": 128
    },
    "Memory Layout": {
      "Initial SP": 320,
      "Max Allocation": 128
    },
    "EntryPoint": [
      ["Initi", "0x0:0x0"],
      ["Inter", "INT 0x21"]
    ],
    "entry": {
      "entry": "IP: 0x140, CS: 0x0"
    },
    "ExitPoint": [],
    "Opcodes": [
      "dec", "pop", "add", "add", "add", "add", "add", "inc", "add", "add", "add", "add", "inc", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "push", "add", "and", "int", "push", "push", "and", "outsw", "jb", "insw", "and", "outsb", "outsb", "outsw", "je", "bound", "jb", "outsb", "and", "and", "push", "and", "or", "add", "add", "add", "add", "push", "inc", "add", "dec", "add", "add", "fild", "add", "add", "add", "add", "loopne", "sidt", "add", "dec", "add", "add", "add", "add", "add", "mov", "add", "adc", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "mov", "add", "add", "scasb", "and", "add", "add", "add", "adc", "add", "adc", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "js", "add", "add", "add", "adc", "add", "add", "add", "add", "add", "add", "add", "add", "add", "add", "and", "add", "add"
    ],
    "Opcode-Occurrence": {
      "dec": 3, "pop": 1, "add": 204, "inc": 3, "push": 5, "and": 8, "int": 1, "outsw": 2, "jb": 2, "insw": 1, "outsb": 3, "je": 1, "bound": 1, "or": 1, "fild": 1, "loopne": 1, "sidt": 1, "mov": 3, "adc": 4, "scasb": 1, "js": 1, "jnp": 1
    },
    "Image Size": 33792,
    "Memory Size": {
      "Initial SS": 0,
      "Initial SP": 320,
      "Minimum Allocation": 65535,
      "Maximum Allocation": 320
    },
    "Header Size": {
      "DOS Header Size": 128
    },
    "Block Entropy": {
      "min": 5.367976256687364,
      "max": 7.641454170138814,
      "total": 238.1050878964332,
      "count": 33,
      "mean": 7.215305693831309
    },
    "Kolmogorov Complexity": "29 KB",
    "Interrupt Info": [
      {
        "address": "0x14c",
        "interrupt": "0x21"
      }
    ],
    "Stack Info": {
      "Initial SP": 320,
      "Initial SS": 0
    }
  }
```

M.8.2. Dynamic Analysis Features

```

        "Behavior": {
            "files_opened": [
                "<SYSTEM32>\\ntdll.dll",
                "%HOMEPATH%\\desktop\\dashborder_120.bmp",
                "%HOMEPATH%\\desktop\\total commander 64 bit.lnk",
                "%HOMEPATH%\\desktop\\mail.ru agent.lnk",
                "%HOMEPATH%\\desktop\\168.jpg",
                "%HOMEPATH%\\desktop\\applicantform_en.doc",
                "%HOMEPATH%\\desktop\\ovp25012015.doc",
                "%HOMEPATH%\\desktop\\qip 2012.lnk",
                "C:\\WINDOWS\\system32\\winime32.dll",
                "C:\\WINDOWS\\system32\\ws2_32.dll",
                "C:\\WINDOWS\\system32\\ws2help.dll",
                "C:\\WINDOWS\\system32\\psapi.dll",
                "C:\\WINDOWS\\system32\\imm32.dll",
                "C:\\WINDOWS\\system32\\lpk.dll",
                "C:\\WINDOWS\\system32\\usp10.dll"
            ],
            "files_written": [
                "%APPDATA%\\microsoft\\windows\\jfdcrauv\\ccsrwarr.exe",
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\startup\\jfdcrauv.lnk",
                "<SYSTEM32>\\tasks\\opera scheduled autoupdate 2414526821",
                "%APPDATA%\\microsoft\\windows\\ucgbfghb\\sdutjtsr.exe",
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\startup\\ucgbfghb.lnk",
                "%APPDATA%\\microsoft\\windows\\ujbrbvrw\\rhriuaff.exe",
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\startup\\ujbrbvrw.lnk",
                "%APPDATA%\\microsoft\\windows\\bcvbaehr\\uihfraij.exe",
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\startup\\bcvbaehr.lnk",
                "%APPDATA%\\microsoft\\windows\\ihhbvewa\\ujrjfuet.exe",
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\startup\\ihhbvewa.lnk"
            ],
            "files_deleted": [
                "<PATH_SAMPLE.EXE>"
            ],
            "command_executions": [
                "C:\\Users\\Johnson\\AppData\\Local\\Temp\\23731771.exe",
                "C:\\Windows\\Explorer.EXE",
                "C:\\Users\\Johnson\\AppData\\Roaming\\Microsoft\\Windows\\wfhcuwdw\\vtahtbgc
                    .exe",
                "C:\\Users\\Johnson\\AppData\\Local\\Temp\\76
                    d9c9d7a779005f6caeaa72dbdde4nalysis_subject.exe",
                " "
            ],
            "files_attribute_changed": [
                "%APPDATA%\\microsoft\\windows\\jfdcrauv\\ccsrwarr.exe",
                "%APPDATA%\\microsoft\\windows\\ucgbfghb\\sdutjtsr.exe",
                "%APPDATA%\\microsoft\\windows\\ujbrbvrw\\rhriuaff.exe",
                "%APPDATA%\\microsoft\\windows\\bcvbaehr\\uihfraij.exe",
                "%APPDATA%\\microsoft\\windows\\ihhbvewa\\ujrjfuet.exe",
                "C:\\Users\\Johnson\\AppData\\Roaming\\Microsoft\\Windows\\wfhcuwdw",
                "C:\\Users\\Johnson\\AppData\\Roaming\\Microsoft\\Windows\\wfhcuwdw\\vtahtbgc
                    .exe",
                "C:\\Users\\Johnson\\AppData\\Roaming\\Microsoft\\Windows\\wfhcuwdw\\vtahtbgc
                    .exe\\:Zone.Identifier:$DATA"
            ],
            "processes_terminated": [
                "<PATH_SAMPLE.EXE>",
                "<SYSTEM32>\\wbem\\wmiprvse.exe"
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [
                "<HKLM>\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Schedule\\
                    CompatibilityAdapter\\Signatures\\Opera scheduled Autoupdate 2414526821.
                    job",
                "<HKLM>\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Schedule\\
                    CompatibilityAdapter\\Signatures\\Opera scheduled Autoupdate 2414526821.
                    job.fp",
                "HKLM\\SOFTWARE\\MICROSOFT\\WINDOWS\\CURRENTVERSION\\EXPLORER\\STARTPAGE\\
                    NEWSHORTCUTS"
            ],
            "mitre_attack_techniques": []
        }
    }
```

M.9. IcedID

M.9.1. Dynamic features

```
        "Behavior": {
            "files_opened": [
                "C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\",
                "C:\\Users\\user\\AppData\\LocalLow",
                "C:\\Users\\user\\AppData\\LocalLow\\Microsoft\\CryptnetUrlCache\\MetaData\\
                    AFCF8E76E06245E64045C911C7467E0F",
                "C:\\Users\\user\\AppData\\Local\\Temp\\vbxcvhb",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\uhcosofd.dll",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\uhcosofd.dll.123.Manifest",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\uhcosofd.dll.124.Manifest",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\uhcosofd.dll.2.Manifest",
                "C:\\Users\\user\\AppData\\Roaming\\jhghjdfghdfgh",
                "C:\\Users\\user\\AppData\\Roaming\\{30241F57-A566-277B-8E94-145700626C9C}\\
                    Homeca.dat",
                "C:\\Users\\user\\AppData\\Roaming\\{30241F57-A566-277B-8E94-145700626C9C}\\
                    rokoudbt64.dat",
                "C:\\Users\\user\\Desktop\\c3.dll",
                "C:\\Users\\user\\Desktop\\c3.dll.123.Manifest",
                "C:\\Users\\user\\Desktop\\c3.dll.124.Manifest",
                "C:\\Users\\user\\Desktop\\c3.dll.2.Manifest",
                "C:\\Windows\\AppPatch\\sysmain.sdb",
                "C:\\Windows\\Globalization\\Sorting\\sortdefault.nls",
                "C:\\Windows\\SYSTEM32\\AcLayers.dll",
                "C:\\Windows\\SYSTEM32\\CRYPTBASE.dll",
                "C:\\Windows\\SYSTEM32\\CRYPTSP.dll",
                "C:\\Windows\\SYSTEM32\\DNSAPI.dll",
                "C:\\Windows\\SYSTEM32\\DPAPI.DLL",
                "C:\\Windows\\SYSTEM32\\IPHLPAPI.DLL",
                "C:\\Windows\\SYSTEM32\\IPHLPAPI.dll",
                "C:\\Windows\\SYSTEM32\\NETAPI32.dll",
                "C:\\Windows\\SYSTEM32\\NETUTILS.DLL",
                "C:\\Windows\\SYSTEM32\\NTASN1.dll",
                "C:\\Windows\\SYSTEM32\\PROPSYS.dll",
                "C:\\Windows\\SYSTEM32\\Secur32.dll",
                "C:\\Windows\\SYSTEM32\\SspiCli.dll",
                "C:\\Windows\\SYSTEM32\\WINNSI.DLL",
                "C:\\Windows\\SYSTEM32\\WINSPOOL.DRV",
                "C:\\Windows\\SYSTEM32\\WKSCLI.DLL",
                "C:\\Windows\\SYSTEM32\\Winhttp.dll",
                "C:\\Windows\\SYSTEM32\\apphelp.dll",
                "C:\\Windows\\SYSTEM32\\bcrypt.dll",
                "C:\\Windows\\SYSTEM32\\cryptnet.dll",
                "C:\\Windows\\SYSTEM32\\dhcpcsvc.DLL",
                "C:\\Windows\\SYSTEM32\\dhcpcsvc6.DLL",
                "C:\\Windows\\SYSTEM32\\en-US\\regsvr32.exe.mui",
                "C:\\Windows\\SYSTEM32\\en-US\\rundll32.exe.mui",
                "C:\\Windows\\SYSTEM32\\en-US\\winnlsres.dll.mui",
                "C:\\Windows\\SYSTEM32\\gpapi.dll",
                "C:\\Windows\\SYSTEM32\\loaddll64.exe",
                "C:\\Windows\\SYSTEM32\\mskeyprotect.dll",
                "C:\\Windows\\SYSTEM32\\ncrypt.dll",
                "C:\\Windows\\SYSTEM32\\ntdll.dll",
                "C:\\Windows\\SYSTEM32\\ole32.dll",
                "C:\\Windows\\SYSTEM32\\regsvr32.exe",
                "C:\\Windows\\SYSTEM32\\rundll32.exe"
            ],
            "files_written": [
                "C:\\Users\\user\\AppData\\Local\\Temp\\vbxcvhb",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\",
                "C:\\Users\\user\\AppData\\Roaming\\jena\\Pupi3\\uhcosofd.dll",
                "C:\\Users\\user\\AppData\\Roaming\\jhghj",
                "C:\\Users\\user\\AppData\\Roaming\\jhghjdfghdfgh",
                "\\Device\\ConDrv\\\\Connect"
            ],
            "files_deleted": [],
            "command_executions": [],
            "files_attribute_changed": [],
            "processes_terminated": [
                "%windir%\\System32\\svchost.exe -k WerSvcGroup",
                "wmiadap.exe /F /T /R",
                "rundll32.exe \"%APPDATA%\\%USERNAME%\\deuldt.dll\" ,#1 --uqef=\"
                    jhghjdfghdfgh\""
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "encode data using XOR",
                "encrypt data using RC4 PRGA",
                "get common file path",
                "link function at runtime on Windows",
                "encrypt data using RC4 KSA",
                "encode data using XOR",
                "encrypt data using RC4 PRGA",
                "get common file path",
                "link function at runtime on Windows",
                "encrypt data using RC4 KSA",
                "Tries to load missing DLLs",
                "Spawns processes",
                "Creates a process in suspended mode (likely to inject code)",
                "System process connects to network (likely due to code injection)",
                "Drops files with a non matching file extension (content does not match to
                    file extension)",
                "Creates files inside the user directory",
                "Stores large binary data to the registry",
                "Adds / modifies Windows certificates",
                "May sleep (evasive loops) to hinder dynamic analysis",
                "Contains long sleeps (>= 3 min)",
                "Registers a DLL",
                "Runs a DLL by calling functions",
                "Tries to detect virtualization through RDTSC time measurements",
                "Reads the hosts file",
                "Tries to detect virtualization through RDTSC time measurements",
                "Queries the cryptographic machine GUID",
                "Reads software policies",
                "Uses HTTPS",
                "Uses HTTPS for network communication, use the SSL MITM Proxy cookbook for
                    further analysis",
                "Performs DNS lookups",
                "Uses HTTPS",
                "Performs DNS lookups"
```

M.10. RacoonStealer

M.10.1. Dynamic Analysis Features

```
        "Behavior": {
            "files_opened": [
                "%LOCALAPPDATA%\\google\\chrome\\user data\\local state",
                "%LOCALAPPDATA%\\google\\chrome\\user data\\default\\login data",
                "%LOCALAPPDATA%\\google\\chrome\\user data\\default\\cookies",
                "%LOCALAPPDATA%\\google\\chrome\\user data\\default\\web data",
                "%APPDATA%\\opera software\\opera stable\\local state",
                "%APPDATA%\\mozilla\\firefox\\profiles\\gn7ryp3k.default\\cookies.sqlite",
                "%LOCALAPPDATA%low\\wffzxleyi5ue-shm",
                "%APPDATA%\\mozilla\\firefox\\profiles\\gn7ryp3k.default\\formhistory.sqlite
                    ",
                "%APPDATA%\\thunderbird\\profiles\\wjj9aet2.default\\cookies.sqlite",
                "%LOCALAPPDATA%low\\1u510o9jd81a-shm",
                "%HOMEPATH%\\desktop\\icq.lnk",
                "%WINDIR%\\syswow64\\shdocvw.dll",
                "%HOMEPATH%\\desktop\\mail.ru agent.lnk",
                "%HOMEPATH%\\desktop\\qip 2012.lnk",
                "%HOMEPATH%\\desktop\\telegram.lnk",
                "%HOMEPATH%\\desktop\\total commander 64 bit.lnk",
                "C:\\users\\public\\desktop\\acrobat reader dc.lnk",
                "C:\\users\\public\\desktop\\google chrome.lnk",
                "C:\\users\\public\\desktop\\mirc.lnk",
                "C:\\users\\public\\desktop\\mozilla firefox.lnk",
                "C:\\users\\public\\desktop\\mozilla thunderbird.lnk",
                "C:\\users\\public\\desktop\\opera.lnk",
                "C:\\users\\public\\desktop\\steam.lnk",
                "C:\\users\\public\\desktop\\winamp.lnk",
                "%HOMEPATH%\\links\\desktop.lnk",
                "%HOMEPATH%\\links\\downloads.lnk",
                "%HOMEPATH%\\links\\recentplaces.lnk",
                "%APPDATA%\\microsoft\\windows\\recent\\activator.lnk",
                "%APPDATA%\\telegram desktop\\tdata\\90ef50e22e92cb8c0",
                "%APPDATA%\\telegram desktop\\tdata\\d877f783d5d3ef8c\\map0"
            ],
            "files_written": [
                "%LOCALAPPDATA%low\\nss3.dll",
                "%LOCALAPPDATA%low\\msvcp140.dll",
                "%LOCALAPPDATA%low\\vcruntime140.dll",
                "%LOCALAPPDATA%low\\mozglue.dll",
                "%LOCALAPPDATA%low\\freebl3.dll",
                "%LOCALAPPDATA%low\\softokn3.dll",
                "%LOCALAPPDATA%low\\sqlite3.dll",
                "%LOCALAPPDATA%low\\nssdbm3.dll",
                "%LOCALAPPDATA%low\\kd9wm9u91hx5",
                "%LOCALAPPDATA%low\\e8ri2215mw3k",
                "%LOCALAPPDATA%low\\rhn0d0bs39ia",
                "%LOCALAPPDATA%low\\wffzxleyi5ue",
                "%LOCALAPPDATA%low\\4mud8y62vphn",
                "%LOCALAPPDATA%low\\1u510o9jd81a",
                "%LOCALAPPDATA%low\\2ldoo8yukmeg",
                "%LOCALAPPDATA%low\\5vj2d6gygfdd",
                "%LOCALAPPDATA%low\\5f805gq8cypw",
                "%LOCALAPPDATA%low\\saecm9jotav4",
                "%LOCALAPPDATA%low\\e333i1sq24ng",
                "%LOCALAPPDATA%low\\vlmfc0xocol7",
                "%LOCALAPPDATA%low\\zx1u5iha1rcv",
                "%LOCALAPPDATA%low\\1r4sreeg86l8",
                "%LOCALAPPDATA%low\\o6saee1d8z4m",
                "C:\\Users\\<USER>\\AppData\\LocalLow\\nss3.dll",
                "C:\\Users\\<USER>\\AppData\\LocalLow\\msvcp140.dll",
                "C:\\Users\\<USER>\\AppData\\LocalLow\\vcruntime140.dll",
                "C:\\Users\\<USER>\\AppData\\LocalLow\\mozglue.dll",
                "C:\\Windows\\ServiceProfiles\\LocalService\\AppData\\Roaming\\Microsoft\\
                    UPnP Device Host\\upnphost\\udhisapi.dll",
                "C:\\Windows...aCollector",
                "C:\\Users\\<USER>\\AppData\\LocalLow\\freebl3.dll"
            ],
            "files_deleted": [
                "%LOCALAPPDATA%low\\kd9wm9u91hx5",
                "%LOCALAPPDATA%low\\e8ri2215mw3k",
                "%LOCALAPPDATA%low\\rhn0d0bs39ia",
                "%LOCALAPPDATA%low\\wffzxleyi5ue-shm",
                "%LOCALAPPDATA%low\\wffzxleyi5ue",
                "%LOCALAPPDATA%low\\4mud8y62vphn",
                "%LOCALAPPDATA%low\\1u510o9jd81a-shm",
                "%LOCALAPPDATA%low\\1u510o9jd81a",
                "%LOCALAPPDATA%low\\2ldoo8yukmeg",
                "%LOCALAPPDATA%low\\5vj2d6gygfdd",
                "%LOCALAPPDATA%low\\5f805gq8cypw",
                "%LOCALAPPDATA%low\\saecm9jotav4",
                "%LOCALAPPDATA%low\\e333i1sq24ng",
                "%LOCALAPPDATA%low\\vlmfc0xocol7",
                "%LOCALAPPDATA%low\\zx1u5iha1rcv",
                "%LOCALAPPDATA%low\\1r4sreeg86l8",
                "%LOCALAPPDATA%low\\o6saee1d8z4m",
                "%LOCALAPPDATA%low\\nss3.dll",
                "%LOCALAPPDATA%low\\sqlite3.dll",
                "%USERPROFILE%\\AppData\\LocalLow\\I3MH6Xv603U2",
                "%USERPROFILE%\\AppData\\LocalLow\\d4H246h0iHmp",
                "%USERPROFILE%\\AppData\\LocalLow\\f34imopd07CF",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER347.tmp.
                    WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER441.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER470.tmp.txt",
                "%USERPROFILE%\\AppData\\LocalLow\\7j0P4171y8M4",
                "%USERPROFILE%\\AppData\\LocalLow\\2rlP6Nm00dM3",
                "%USERPROFILE%\\AppData\\LocalLow\\nss3.dll",
                "%USERPROFILE%\\AppData\\LocalLow\\sqlite3.dll",
                "C:\\Windows\\System32\\spp\\store\\2.0\\cache\\cache.dat"
            ],
            "command_executions": [
                "%SAMPLEPATH%",
                "\"%SAMPLEPATH%\\ae2ba63a82ebe6a75f17a5c7a6bc9b96.exe\"",
                "\"%SAMPLEPATH%\\
                    cddc1e15fcfcb29cfcb3631f1d478640d228fd9ea38c01d347833567970d04e3.exe\"",
                "C:\\Windows\\System32\\wuapihost.exe -Embedding"
            ],
            "files_attribute_changed": [
                "%LOCALAPPDATA%\\Microsoft\\Windows\\<INETFILES>\\Content.IE5\\index.dat",
                "%APPDATA%\\Microsoft\\Windows\\Cookies\\index.dat",
                "%LOCALAPPDATA%\\Microsoft\\Windows\\History\\History.IE5\\index.dat",
                "%LOCALAPPDATA%low\\kd9wm9u91hx5",
                "%LOCALAPPDATA%low\\e8ri2215mw3k",
                "%LOCALAPPDATA%low\\rhn0d0bs39ia",
                "%LOCALAPPDATA%low\\wffzxleyi5ue",
                "%LOCALAPPDATA%low\\4mud8y62vphn",
                "%LOCALAPPDATA%low\\1u510o9jd81a",
                "%LOCALAPPDATA%low\\21doo8yukmeg",
                "%LOCALAPPDATA%low\\5vj2d6gygfdd",
                "%LOCALAPPDATA%low\\5f805gq8cypw",
                "%LOCALAPPDATA%low\\saecm9jotav4",
                "%LOCALAPPDATA%low\\e333i1sq24ng",
                "%LOCALAPPDATA%low\\vlmfc0xocol7",
                "%LOCALAPPDATA%low\\zx1u5iha1rcv",
                "%LOCALAPPDATA%low\\1r4sreeg8618"
            ],
            "processes_terminated": [
                "%windir%\\System32\\svchost.exe -k WerSvcGroup",
                "wmiadap.exe /F /T /R",
                "%windir%\\system32\\DllHost.exe /Processid:{3EB3C877-1F16-487C-9050-104
                    DBCD66683}",
                "%SAMPLEPATH%",
                "<PATH_SAMPLE.EXE>",
                "%SAMPLEPATH%\\ae2ba63a82ebe6a75f17a5c7a6bc9b96.exe",
                "%SAMPLEPATH%\\
                    cddc1e15fcfcb29cfcb3631f1d478640d228fd9ea38c01d347833567970d04e3.exe",
                "C:\\Windows\\System32\\wuapihost.exe",
                "2892 - C:\\Windows\\system32\\sc.exe start w32time task_started",
                "2940 - C:\\Windows\\system32\\rundll32.exe dfdts.dll,
                    DfdGetDefaultPolicyAndSMART",
                "2968 - taskhost.exe SYSTEM",
                "3012 - taskhost.exe $(Arg0)",
                "3032 - C:\\Windows\\system32\\schtasks.exe /delete /f /TN \"Microsoft\\
                    Windows\\Customer Experience Improvement Program\\Uploader\"",
                "C:\\Users\\user\\Desktop\\software.exe"
            ],
            "processes_killed": [],
            "processes_injected": [
                "%SAMPLEPATH%\\ae2ba63a82ebe6a75f17a5c7a6bc9b96.exe",
                "%SAMPLEPATH%\\
                    cddc1e15fcfcb29cfcb3631f1d478640d228fd9ea38c01d347833567970d04e3.exe",
                "\\\\?\\C:\\Windows\\system32\\wbem\\WMIADAP.EXE"
            ],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [
                "<HKCU>\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\ProxyBypass",
                "<HKLM>\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\ProxyBypass",
                "<HKCU>\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\IntranetName",
                "<HKLM>\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\IntranetName"
            ],
            "mitre_attack_techniques": [
                "encode data using XOR",
                "link function at runtime on Windows",
                "parse PE header",
                "reference anti-VM strings targeting Qemu",
                "Creates files inside the user directory",
                "Overwrites code with unconditional jumps - possibly settings hooks in foreign process",
                "Creates a DirectInput object (often for capturing keystrokes)",
                "May try to detect the virtual machine to hinder analysis (VM artifact strings found in memory)",
                "Tries to detect virtualization through RDTSC time measurements",
                "Queries a list of all running processes",
                "Queries the cryptographic machine GUID",
                "Reads software policies",
                "Tries to detect virtualization through RDTSC time measurements",
                "Reads the hosts file",
                "Uses HTTPS for network communication, use the SSL MITM Proxy cookbook for further analysis",
                "Uses HTTPS",
                "Posts data to webserver",
                "Performs DNS lookups",
                "Posts data to webserver",
                "C2 URLs / IPs found in malware configuration",
                "Performs DNS lookups",
                "Uses HTTPS"
            ]
        }
```

M.11. AsyncRAT

M.11.1. Dynamic Analysis Features

```
        "Behavior": {
            "files_opened": [
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe",
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe\\cloud.exe",
                "%WINDIR%\\assembly\\nativeimages_v4.0.30319_32\\mscorlib\\77f338d420d067a26b2d34f47445fc51\\mscorlib.ni.dll.aux",
                "%WINDIR%\\assembly\\nativeimages_v4.0.30319_32\\system.core\\7aa0dcace3b5d10b626540709537d280\\system.core.ni.dll.aux",
                "%WINDIR%\\assembly\\nativeimages_v4.0.30319_32\\system\\0b2f69b43a576b9edcc807a30872bd91\\system.ni.dll.aux",
                "%WINDIR%\\syswow64\\cmd.exe",
                "<SYSTEM32>\\ntdll.dll",
                "%WINDIR%\\syswow64\\ntdll.dll",
                "%WINDIR%\\assembly\\nativeimages_v4.0.30319_32\\system.xml\\bf505bb2c2b60e7a40740888cd2c3172\\system.xml.ni.dll.aux",
                "%WINDIR%\\microsoft.net\\framework\\v4.0.30319\\msbuild.exe",
                "%WINDIR%\\syswow64\\timeout.exe",
                "%WINDIR%\\assembly\\nativeimages_v4.0.30319_32\\system.configuration\\ce9750286ad44cbfb2acf176df9df0a2\\system.configuration.ni.dll.aux",
                "(6688 - r74kfb8bnx.exe) c:\\users\\xxx\\appdata\\local\\microsoft\\clr_v4.0_32\\usagelogs\\r74kfb8bnx.exe.log",
            "(6688 - r74kfb8bnx.exe) c:\\users\\xxx\\appdata\\roaming\\microsoft\\windows\\start menu\\programs\\adobe\\cloud.exe",
            "C:\\WINDOWS\\system32\\winime32.dll",
            "C:\\WINDOWS\\system32\\ws2_32.dll",
            "C:\\WINDOWS\\system32\\ws2help.dll",
            "C:\\WINDOWS\\system32\\psapi.dll",
            "C:\\WINDOWS\\system32\\mscoree.dll",
            "C:\\WINDOWS\\system32\\imm32.dll",
            "C:\\WINDOWS\\system32\\lpk.dll",
            "C:\\WINDOWS\\system32\\usp10.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\mscoreei.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v2.0.50727\\mscorwks.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\clr.dll",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996E.exe",
            "C:\\WINDOWS\\system32\\MSVCR100_CLR0400.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\Config\\machine.config",
            "C:\\WINDOWS\\assembly\\NativeImages_v4.0.30319_32\\index18.dat",
            "C:\\WINDOWS\\assembly\\NativeImages_v4.0.30319_32\\mscorlib\\cece9d0256e18427b64587ba690605d4\\mscorlib.ni.dll",
            "C:\\WINDOWS\\system32\\rpcss.dll",
            "C:\\WINDOWS\\system32\\MSCTF.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\Culture.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\locale.nlp",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\nlssorting.dll",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\SortDefault.nlp",
            "C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319\\clrjit.dll",
            "C:\\WINDOWS\\assembly\\pubpol1.dat",
            "C:\\WINDOWS\\assembly\\NativeImages_v4.0.30319_32\\System\\7169c473071af03077b1cd2a9c1dbcbe\\System.ni.dll",
            "C:\\WINDOWS\\assembly\\NativeImages_v4.0.30319_32\\System.Core\\4a9f25bff4bb74c9b6a21091923307d2\\System.Core.ni.dll",
            "C:\\WINDOWS\\system32\\shell32.dll",
            "C:\\WINDOWS\\WinSxS\\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\\comctl32.dll",
            "C:\\WINDOWS\\WindowsShell.Manifest",
            "C:\\WINDOWS\\system32\\comctl32.dll",
            "C:\\WINDOWS\\system32\\shdocvw.dll",
            "C:\\WINDOWS\\system32\\clbcatq.dll",
            "C:\\WINDOWS\\system32\\comres.dll",
            "C:\\WINDOWS\\Registration\\R000000000007.clb",
            "C:\\WINDOWS\\system32\\wininet.dll",
            "C:\\WINDOWS\\system32\\riched20.dll"
        ],
        "files_written": [
            "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe\\cloud.exe",
            "%APPDATA%\\microsoft\\windows\\start menu\\programs\\adobe\\cloud.exe",
            "(6688 - r74kfb8bnx.exe) c:\\users\\xxx\\appdata\\local\\microsoft\\clr_v4.0_32\\usagelogs\\r74kfb8bnx.exe.log",
            "(6688 - r74kfb8bnx.exe) c:\\users\\xxx\\appdata\\roaming\\microsoft\\windows\\start menu\\programs\\adobe\\cloud.exe",
            "C:\\Documents and Settings\\Administrator\\\u300c\u5f00\u59cb\u300d\u83dc\u5355\\\u7a0b\u5e8f\\adobe\\cloud.exe",
            "C:\\Users\\RDhJ0CNFevzX\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe\\cloud.exe",
            "C:\\Windows\\ServiceProfiles\\LocalService\\AppData\\Roaming\\Microsoft\\UPnP Device Host\\upnphost\\udhisapi.dll",
            "C:\\Users\\<USER>\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe",
            "C:\\Users\\<USER>\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe\\cloud.exe",
            "C:\\Users\\user\\AppData\\Local\\Microsoft\\CLR_v4.0_32\\UsageLogs\\program.exe.log",
            "C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\Caches",
            "C:\\Users\\user\\AppData\\Roaming",
            "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe",
            "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\adobe\\cloud.exe",
            "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs
```

```
                    \\adobe\\cloud.exe:Zone.Identifier",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs
                    \\adobe\\cloud.exe\\:Zone.Identifier:$DATA",
                "\\Device\\ConDrv\\\\Connect"
            ],
            "files_deleted": [
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF5AA.tmp.
                    WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF685.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF6D4.tmp.txt",
                "C:\\Windows\\System32\\spp\\store\\2.0\\cache\\cache.dat",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER292E.tmp.
                    WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER294F.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER298E.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1846.tmp.
                    WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1920.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1960.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1ECE.tmp.
                    WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1EDF.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1EF0.tmp.txt"
            ],
            "command_executions": [
                "%SAMPLEPATH%",
                "\"%ComSpec%\" ",
                "\"%ComSpec%\" /c timeout 10",
                "timeout  10",
                "\"${SamplePath}\\34669
                    d48168ec6efe33843803070cd984646e6ef4c56198278ee7f2956a4d36b.exe\" ",
                "\"C:\\Windows\\System32\\cmd.exe\" ",
                "\"C:\\Windows\\System32\\cmd.exe\" /c timeout 10",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe",
                "\"%SAMPLEPATH%\\vbc.exe\" ",
                "C:\\Windows\\System32\\wuapihost.exe -Embedding",
                "\"%SAMPLEPATH%\\34669
                    d48168ec6efe33843803070cd984646e6ef4c56198278ee7f2956a4d36b.exe\" ",
                "(5356 - conhost.exe) \\??\\C:\\Windows\\system32\\conhost.exe 0xffffffff -
                    ForceV1",
                "(8056 - msbuild.exe) C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\
                    MSBuild.exe",
                "(9092 - cmd.exe) \"C:\\Windows\\System32\\cmd.exe\" /c timeout 10",
                "(8392 - timeout.exe) timeout  10",
                "(9180 - cmd.exe) \"C:\\Windows\\System32\\cmd.exe\" ",
                "(6688 - r74kfb8bnx.exe) C:\\Users\\xxx\\AppData\\Local\\Temp\\6
                    a7267dc0e5889df7f4b88b8323fb628\\r74kFb8BNX.exe \"\"",
                "(9136 - conhost.exe) \\??\\C:\\Windows\\system32\\conhost.exe 0xffffffff -
                    ForceV1",
                " ",
                "\"C:\\WINDOWS\\system32\\cmd.exe\" "
            ],
            "files_attribute_changed": [
                "%APPDATA%\\microsoft\\windows\\start menu\\programs\\adobe\\cloud.exe"
            ],
            "processes_terminated": [
                "%windir%\\System32\\svchost.exe -k WerSvcGroup",
                "%CONHOST%
                    \"196114867320850120581458316209-930752071-12239441991204104146973047545-870432650",

                "%CONHOST%
                    \"-1776689553-199703836817677110036195869801880484998886374142-1310019265-1427296177",

                "wmiadap.exe /F /T /R",
                "%SAMPLEPATH%",
                "\"%ComSpec%\" ",
                "\"%ComSpec%\" /c timeout 10",
                "timeout  10",
                "${SamplePath}\\34669
                    d48168ec6efe33843803070cd984646e6ef4c56198278ee7f2956a4d36b.exe",
                "C:\\Windows\\SysWOW64\\cmd.exe",
                "C:\\Windows\\SysWOW64\\timeout.exe",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe",
                "%WINDIR%\\syswow64\\timeout.exe",
                "%WINDIR%\\syswow64\\cmd.exe",
                "<PATH_SAMPLE.EXE>",
                "C:\\Windows\\System32\\wuapihost.exe",
                "C:\\Windows\\System32\\conhost.exe",
                "%SAMPLEPATH%\\vbc.exe",
                "%SAMPLEPATH%\\34669d48168ec6efe33843803070cd984646e6ef4c56198278ee7f2956a4d36b.exe",
                "(8392 - timeout.exe) c:\\windows\\syswow64\\timeout.exe"
            ],
            "processes_killed": [
                "%WINDIR%\\syswow64\\cmd.exe",
                "C:\\WINDOWS\\system32\\cmd.exe"
            ],
            "processes_injected": [
                "%WINDIR%\\microsoft.net\\framework\\v4.0.30319\\msbuild.exe"
            ],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [
                "<HKCU>\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\ProxyBypass",
                "<HKLM>\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\ProxyBypass",
                "<HKCU>\\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\IntranetName",
                "<HKLM>\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ZoneMap\\IntranetName"
            ],
            "mitre_attack_techniques": [
                "invoke .NET assembly method",
                "load .NET assembly",
                "read the memory information of other process",
                "get drive type",
                "get the current computer name",
                "call encryption algorithm library",
                "create new process",
                "inject other processes by new thread",
                "create hidden child process",
                "decrypt data",
                "modify token privilege",
                "get time manytimes",
                "get the current user name",
                "get the current user name",
                "get process token",
                "modify user shell folders",
                "inject other processes by process hollowing",
                "script started by non script file",
                "run msbuild",
                "run msbuild",
                "run msbuild",
                "get window text",
                "system location discovery",
                "Creates an undocumented autostart registry key",
                "Stores files to the Windows startup directory",
                "Allocates memory in foreign processes",
                "Creates a process in suspended mode (likely to inject code)",
                "Spawns processes",
                "Injects a PE file into a foreign processes",
                "Writes to foreign memory regions",
                "Drops executable to common a third party application directory",
                "Creates files inside the user directory",
                "Icon mismatch, binary includes an Icon from a different legit application in order to fool users",
                "Creates guard pages, often used to prevent reverse engineering and debugging
```

```
                    ",
                "May sleep (evasive loops) to hinder dynamic analysis",
                "Contains long sleeps (>= 3 min)",
                "Binary may include packed or crypted data",
                "Binary may include packed or crypted data",
                ".NET source code contains potential unpacker",
                "PE file has an executable .text section which is very likely to contain
                    packed code (zlib compression ratio < 0.3)",
                "Binary contains a suspicious time stamp",
                "Queries a list of all running processes",
                "Reads ini files",
                "Queries the volume information (name, serial number etc) of a device",
                "Checks the free space of harddrives",
                "Queries the cryptographic machine GUID",
                "Reads software policies",
                "Uses HTTPS",
                "Uses HTTPS for network communication, use the SSL MITM Proxy cookbook for
                    further analysis",
                "Detected TCP or UDP traffic on non-standard ports"
            ]
        }
    }
```

M.12. BotenaGo

M.12.1. Static Analysis Features

```
section: {'entry': '0x46b2e0', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, '
    file_offset': 0, 'props': ['Type: NULL']}, {'name': '.text', 'size': 1106484, 'entropy':
    5.870752884405834, 'file_offset': 4096, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR
    ']}, {'name': '.plt', 'size': 528, 'entropy': 3.980936084806333, 'file_offset': 1110592,
    'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.rodata', 'size': 436703, '
    entropy': 4.351834860802516, 'file_offset': 1114112, 'props': ['Type: PROGBITS', 'ALLOC
    ']}, {'name': '.rela', 'size': 24, 'entropy': 1.2220198859116511, 'file_offset': 1550816,
     'props': ['Type: RELA', 'ALLOC']}, {'name': '.rela.plt', 'size': 768, 'entropy':
    1.6210579369162064, 'file_offset': 1550840, 'props': ['Type: RELA', 'ALLOC']}, {'name':
    '.gnu.version', 'size': 74, 'entropy': 1.740960896496825, 'file_offset': 1551616, 'props
    ': ['Type: GNU_VERSYM', 'ALLOC']}, {'name': '.gnu.version_r', 'size': 80, 'entropy':
    2.5252192137385356, 'file_offset': 1551712, 'props': ['Type: GNU_VERNEED', 'ALLOC']}, {'
    name': '.hash', 'size': 184, 'entropy': 1.8077229039022173, 'file_offset': 1551808, '
    props': ['Type: HASH', 'ALLOC']}, {'name': '.dynstr', 'size': 531, 'entropy':
    4.572648045349689, 'file_offset': 1552000, 'props': ['Type: STRTAB', 'ALLOC']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.
    text', 'Size': 1106484, 'Virtual Address': 4198400}, {'Name': '.plt', 'Size': 528, '
    Virtual Address': 5304896}, {'Name': '.rodata', 'Size': 436703, 'Virtual Address':
    5308416}, {'Name': '.rela', 'Size': 24, 'Virtual Address': 5745120}, {'Name': '.rela.plt
    ', 'Size': 768, 'Virtual Address': 5745144}, {'Name': '.gnu.version', 'Size': 74, '
    Virtual Address': 5745920}, {'Name': '.gnu.version_r', 'Size': 80, 'Virtual Address':
    5746016}, {'Name': '.hash', 'Size': 184, 'Virtual Address': 5746112}, {'Name': '.dynstr',
     'Size': 531, 'Virtual Address': 5746304}, {'Name': '.shstrtab', 'Size': 552, 'Virtual
    Address': 0}, {'Name': '.dynsym', 'Size': 888, 'Virtual Address': 5747424}, {'Name': '.
    typelink', 'Size': 3000, 'Virtual Address': 5748320}, {'Name': '.itablink', 'Size': 464,
    'Virtual Address': 5751320}, {'Name': '.gosymtab', 'Size': 0, 'Virtual Address':
    5751784}, {'Name': '.gopclntab', 'Size': 605612, 'Virtual Address': 5751808}, {'Name': '.
    go.buildinfo', 'Size': 32, 'Virtual Address': 6361088}, {'Name': '.got.plt', 'Size': 280,
     'Virtual Address': 6361120}, {'Name': '.dynamic', 'Size': 304, 'Virtual Address':
    6361408}, {'Name': '.got', 'Size': 8, 'Virtual Address': 6361712}, {'Name': '.noptrdata',
     'Size': 64984, 'Virtual Address': 6361728}, {'Name': '.data', 'Size': 31792, 'Virtual
    Address': 6426720}, {'Name': '.bss', 'Size': 196560, 'Virtual Address': 6458528}, {'Name
    ': '.noptrbss', 'Size': 12392, 'Virtual Address': 6655104}, {'Name': '.tbss', 'Size': 8,
    'Virtual Address': 0}, {'Name': '.zdebug_abbrev', 'Size': 281, 'Virtual Address':
    6668288}, {'Name': '.zdebug_line', 'Size': 174066, 'Virtual Address': 6668569}, {'Name':
    '.zdebug_frame', 'Size': 40738, 'Virtual Address': 6842635}, {'Name': '.zdebug_pubnames',
     'Size': 7381, 'Virtual Address': 6883373}, {'Name': '.zdebug_pubtypes', 'Size': 20287, '
    Virtual Address': 6890754}, {'Name': '.debug_gdb_scripts', 'Size': 44, 'Virtual Address':
     6911041}, {'Name': '.zdebug_info', 'Size': 303025, 'Virtual Address': 6911085}, {'Name':
     '.zdebug_loc', 'Size': 165369, 'Virtual Address': 7214110}, {'Name': '.zdebug_ranges', '
    Size': 57654, 'Virtual Address': 7379479}, {'Name': '.interp', 'Size': 28, 'Virtual
    Address': 4198372}, {'Name': '.note.go.buildid', 'Size': 100, 'Virtual Address':
    4198272}, {'Name': '.symtab', 'Size': 103464, 'Virtual Address': 0}, {'Name': '.strtab',
    'Size': 102564, 'Virtual Address': 0}], 'Segments': [{'Type': 'SEGMENT_TYPES.PHDR', 'Size
    ': 560, 'Virtual Address': 4194368}, {'Type': 'SEGMENT_TYPES.INTERP', 'Size': 28, '
    Virtual Address': 4198372}, {'Type': 'SEGMENT_TYPES.NOTE', 'Size': 100, 'Virtual Address
    ': 4198272}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 1111120, 'Virtual Address': 4194304},
     {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 1049004, 'Virtual Address': 5308416}, {'Type': '
    SEGMENT_TYPES.LOAD', 'Size': 97440, 'Virtual Address': 6361088}, {'Type': 'SEGMENT_TYPES.
    DYNAMIC', 'Size': 304, 'Virtual Address': 6361408}, {'Type': 'SEGMENT_TYPES.TLS', 'Size':
     0, 'Virtual Address': 0}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual
    Address': 0}, {'Type': 'SEGMENT_TYPES.???', 'Size': 0, 'Virtual Address': 0}]}
imports: {'DynamicSymbols': ['', '__errno_location', 'getaddrinfo', 'freeaddrinfo'], '
    SymbolTable': ['', '__errno_location', 'getaddrinfo', 'freeaddrinfo']}
exports: ['_cgo_panic', '_cgo_topofstack', 'crosscall2', 'runtime.text', 'runtime.etext']
general: {'size': 3241164, 'virtual_size': 0, 'has_debug': 0, 'exports': 4348, 'imports': 19,
     'has_relocations': 0, 'symbols': 4348}
header: {'file_type': 'EXECUTABLE', 'entry_point': 4633312, 'machine_type': 'x86_64', '
    header_size': 64, 'program_headers': [{'type': 'PHDR', 'virtual_address': 4194368, '
    physical_address': 4194368, 'physical_size': 560, 'virtual_size': 560, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 4096}, {'type': 'INTERP', 'virtual_address': 4198372, '
    physical_address': 4198372, 'physical_size': 28, 'virtual_size': 28, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 1}, {'type': 'NOTE', 'virtual_address': 4198272, '
    physical_address': 4198272, 'physical_size': 100, 'virtual_size': 100, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 4}, {'type': 'LOAD', 'virtual_address': 4194304, '
    physical_address': 4194304, 'physical_size': 1111120, 'virtual_size': 1111120, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 5308416, '
    physical_address': 5308416, 'physical_size': 1049004, 'virtual_size': 1049004, 'flags':
    SEGMENT_FLAGS.R, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 6361088, '
    physical_address': 6361088, 'physical_size': 97440, 'virtual_size': 306408, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'DYNAMIC', 'virtual_address': 6361408, '
    physical_address': 6361408, 'physical_size': 304, 'virtual_size': 304, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 8}, {'type': 'TLS', 'virtual_address': 0, '
    physical_address': 0, 'physical_size': 0, 'virtual_size': 8, 'flags': SEGMENT_FLAGS.R, '
    alignment': 8}, {'type': 'GNU_STACK', 'virtual_address': 0, 'physical_address': 0, '
    physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 8}, {'type
    ': '???', 'virtual_address': 0, 'physical_address': 0, 'physical_size': 0, 'virtual_size
    ': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 8}], 'section_headers': [{'name': '', '
    type': 'NULL', 'virtual_address': 0, 'size': 0, 'entropy': -0.0}, {'name': '.text', 'type
    ': 'PROGBITS', 'virtual_address': 4198400, 'size': 1106484, 'entropy':
    5.870752884405834}, {'name': '.plt', 'type': 'PROGBITS', 'virtual_address': 5304896, '
    size': 528, 'entropy': 3.980936084806333}, {'name': '.rodata', 'type': 'PROGBITS', '
    virtual_address': 5308416, 'size': 436703, 'entropy': 4.351834860802516}, {'name': '.rela
    ', 'type': 'RELA', 'virtual_address': 5745120, 'size': 24, 'entropy':
    1.2220198859116511}, {'name': '.rela.plt', 'type': 'RELA', 'virtual_address': 5745144, '
    size': 768, 'entropy': 1.6210579369162064}, {'name': '.gnu.version', 'type': 'GNU_VERSYM
    ', 'virtual_address': 5745920, 'size': 74, 'entropy': 1.740960896496825}, {'name': '.gnu.
    version_r', 'type': 'GNU_VERNEED', 'virtual_address': 5746016, 'size': 80, 'entropy':
    2.5252192137385356}, {'name': '.hash', 'type': 'HASH', 'virtual_address': 5746112, 'size
    ': 184, 'entropy': 1.8077229039022173}, {'name': '.dynstr', 'type': 'STRTAB', '
    virtual_address': 5746304, 'size': 531, 'entropy': 4.572648045349689}, {'name': '.
    shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 552, 'entropy':
    4.351353948202993}, {'name': '.dynsym', 'type': 'DYNSYM', 'virtual_address': 5747424, '
    size': 888, 'entropy': 1.0104030032533726}, {'name': '.typelink', 'type': 'PROGBITS', '
    virtual_address': 5748320, 'size': 3000, 'entropy': 4.352172764173542}, {'name': '.
    itablink', 'type': 'PROGBITS', 'virtual_address': 5751320, 'size': 464, 'entropy':
    2.260342786820047}, {'name': '.gosymtab', 'type': 'PROGBITS', 'virtual_address': 5751784,
     'size': 0, 'entropy': -0.0}, {'name': '.gopclntab', 'type': 'PROGBITS', 'virtual_address
    ': 5751808, 'size': 605612, 'entropy': 5.599953726591909}, {'name': '.go.buildinfo', '
    type': 'PROGBITS', 'virtual_address': 6361088, 'size': 32, 'entropy':
    3.5372301466508205}, {'name': '.got.plt', 'type': 'PROGBITS', 'virtual_address': 6361120,
     'size': 280, 'entropy': 2.18951339608244}, {'name': '.dynamic', 'type': 'DYNAMIC', '
    virtual_address': 6361408, 'size': 304, 'entropy': 1.6577221539606757}, {'name': '.got',
    'type': 'PROGBITS', 'virtual_address': 6361712, 'size': 8, 'entropy': -0.0}, {'name': '.
    noptrdata', 'type': 'PROGBITS', 'virtual_address': 6361728, 'size': 64984, 'entropy':
    5.1736147485815644}, {'name': '.data', 'type': 'PROGBITS', 'virtual_address': 6426720, '
    size': 31792, 'entropy': 1.6332834663111804}, {'name': '.bss', 'type': 'NOBITS', '
    virtual_address': 6458528, 'size': 196560, 'entropy': 7.994400722933447}, {'name': '.
    noptrbss', 'type': 'NOBITS', 'virtual_address': 6655104, 'size': 12392, 'entropy':
    7.9170020206866765}, {'name': '.tbss', 'type': 'NOBITS', 'virtual_address': 0, 'size': 8,
     'entropy': 2.75}, {'name': '.zdebug_abbrev', 'type': 'PROGBITS', 'virtual_address':
    6668288, 'size': 281, 'entropy': 7.186678878967747}, {'name': '.zdebug_line', 'type': '
    PROGBITS', 'virtual_address': 6668569, 'size': 174066, 'entropy': 7.9955651626178295}, {'
    name': '.zdebug_frame', 'type': 'PROGBITS', 'virtual_address': 6842635, 'size': 40738, '
    entropy': 7.931337274423072}, {'name': '.zdebug_pubnames', 'type': 'PROGBITS', '
    virtual_address': 6883373, 'size': 7381, 'entropy': 7.960649887021013}, {'name': '.
    zdebug_pubtypes', 'type': 'PROGBITS', 'virtual_address': 6890754, 'size': 20287, 'entropy
    ': 7.984492715530015}, {'name': '.debug_gdb_scripts', 'type': 'PROGBITS', '
    virtual_address': 6911041, 'size': 44, 'entropy': 4.220128777433187}, {'name': '.
    zdebug_info', 'type': 'PROGBITS', 'virtual_address': 6911085, 'size': 303025, 'entropy':
    7.996917784963646}, {'name': '.zdebug_loc', 'type': 'PROGBITS', 'virtual_address':
    7214110, 'size': 165369, 'entropy': 7.994350388445776}, {'name': '.zdebug_ranges', 'type
    ': 'PROGBITS', 'virtual_address': 7379479, 'size': 57654, 'entropy': 7.803982755586464},
    {'name': '.interp', 'type': 'PROGBITS', 'virtual_address': 4198372, 'size': 28, 'entropy
    ': 3.94075983254009}, {'name': '.note.go.buildid', 'type': 'NOTE', 'virtual_address':
    4198272, 'size': 100, 'entropy': 5.352271034814013}, {'name': '.symtab', 'type': 'SYMTAB
    ', 'virtual_address': 0, 'size': 103464, 'entropy': 3.2692849180450874}, {'name': '.
    strtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 102564, 'entropy':
    5.051145007242242}]}
strings: {'numstrings': 1222, 'avlength': 5.816693944353519, 'printabledist': [148, 21, 30,
    8, 856, 9, 12, 8, 135, 107, 2, 13, 8, 28, 21, 33, 227, 41, 11, 15, 30, 6, 22, 28, 151,
    639, 41, 80, 9, 19, 22, 3, 105, 75, 21, 13, 205, 30, 8, 7, 1092, 32, 51, 7, 228, 33, 11,
    8, 82, 18, 7, 15, 190, 35, 5, 3, 75, 5, 8, 10, 76, 15, 6, 9, 67, 60, 15, 91, 126, 67, 50,
     24, 66, 74, 4, 14, 116, 43, 71, 37, 113, 0, 47, 42, 157, 249, 28, 53, 69, 8, 2, 6, 101,
    6, 10, 24], 'printables': 7108, 'entropy': 5.173827171325684, 'paths': 0, 'urls': 0, '
    registry': 0, 'MZ': 2}
EntryPoint: [['Main ', 4633312], ['_cgo_', 4825536], ['_cgo_', 4627584], ['cross', 4825632],
    ['runti', 4198400]]
ExitPoint: []
Opcodes: ['push', 'push', 'push', 'push', 'mov', 'sub', 'call', 'mov', 'call', 'mov', 'mov',
    'mov', 'mov', 'mov', 'mov', 'call', 'mov', 'mov', 'call', 'sub']
Opcode-Occurrence: {'push': 85, 'mov': 102478, 'sub': 3067, 'call': 17044, 'add': 5613, 'pop
    ': 56, 'ret': 4403, 'nop': 9995, 'jmp': 10157, 'int3': 39516, 'test': 5340, 'je': 4067, '
    movaps': 8, 'lea': 16003, 'jne': 5796, 'xor': 2805, 'cmp': 12764, 'movsxd': 206, 'js': 1,
     'rep stosq': 8, 'bt': 214, 'jae': 1000, 'and': 1137, 'cmovne': 104, 'shl': 812, 'or':
    234, 'jbe': 2463, 'jge': 502, 'jl': 376, 'ja': 942, 'jb': 606, 'neg': 369, 'sar': 462, '
    movzx': 2694, 'jle': 571, 'movups': 2059, 'inc': 693, 'jg': 308, 'setb': 35, 'cpuid': 3,
    'xgetbv': 1, 'shr': 733, 'movabs': 648, 'lock cmpxchg': 134, 'sete': 281, 'xchg': 125, '
    imul': 355, 'cmove': 77, 'cmovl': 50, 'movdqu': 260, 'pcmpeqb': 20, 'pmovmskb': 17, 'bsf
    ': 23, 'seta': 11, 'bswap': 5, 'bsr': 11, 'setg': 15, 'vmovdqu': 54, 'vpcmpeqb': 11, '
    vpmovmskb': 9, 'vzeroupper': 17, 'movq': 16, 'punpcklbw': 4, 'pshufd': 2, 'popcnt': 12, '
    vpbroadcastb': 2, 'pand': 4, 'vpand': 1, 'pcmpestri': 2, 'vptest': 2, 'movss': 54, 'xorps
    ': 1421, 'ucomiss': 7, 'jnp': 18, 'movsd': 248, 'ucomisd': 48, 'setnp': 6, 'dec': 239, '
    cmova': 20, 'cmovb': 12, 'mul': 51, 'jo': 5, 'setne': 55, 'cmovg': 54, 'cvtss2sd': 23, '
    movsx': 24, 'rol': 91, 'sbb': 102, 'cmovae': 4, 'setae': 4, 'cvtsi2sd': 62, 'addsd': 26,
    'subsd': 13, 'mulsd': 30, 'cvttsd2si': 25, 'jno': 1, 'div': 33, 'lock or': 12, 'not': 41,
     'bts': 55, 'lock xadd': 158, 'divsd': 23, 'cmovle': 6, 'setl': 19, 'btr': 25, 'lock and
    ': 4, 'cqo': 7, 'idiv': 10, 'setbe': 4, 'jp': 4, 'pxor': 62, 'setge': 19, 'cdq': 3, '
    setle': 3, 'rcr': 4, 'seto': 1, 'pinsrw': 1, 'pshufhw': 1, 'movdqa': 15, 'aesenc': 111, '
    pshufb': 1, 'cld': 2, 'int': 4, 'rep movsb': 27, 'pause': 1, 'lfence': 1, 'mfence': 1, '
    rdtsc': 1, 'pinsrd': 1, 'pinsrq': 1, 'vpxor': 1, 'vmovntdq': 12, 'sfence': 3, 'rep movsq
    ': 5, 'std': 1, 'vmovdqa': 8, 'prefetchnta': 4, 'pushfq': 1, 'popfq': 1, 'syscall': 45, '
    cmovge': 3, 'cvtsd2ss': 4, 'movd': 1}
Image Size: 6458528
Header Size: {'ELF Header Size': 64, 'Program Headers Total Size': 2258556}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 97440, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.PHDR', 'Flags': 'READ'}, 'Segment 1': {'
    Type': 'SEGMENT_TYPES.INTERP', 'Flags': 'READ'}, 'Segment 2': {'Type': 'SEGMENT_TYPES.
    NOTE', 'Flags': 'READ'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ |
    EXECUTE'}, 'Segment 4': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ'}, 'Segment 5': {'
    Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 6': {'Type': '
    SEGMENT_TYPES.DYNAMIC', 'Flags': 'READ | WRITE'}, 'Segment 7': {'Type': 'SEGMENT_TYPES.
    TLS', 'Flags': 'READ'}, 'Segment 8': {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ |
     WRITE'}, 'Segment 9': {'Type': 'SEGMENT_TYPES.???', 'Flags': ''}}
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.text': {'
    min': 5.870752884405834, 'max': 5.870752884405834, 'total': 5.870752884405834, 'count':
    1, 'mean': 5.870752884405834}, '.plt': {'min': 3.980936084806333, 'max':
    3.980936084806333, 'total': 3.980936084806333, 'count': 1, 'mean': 3.980936084806333}, '.
    rodata': {'min': 4.351834860802516, 'max': 4.351834860802516, 'total': 4.351834860802516,
     'count': 1, 'mean': 4.351834860802516}, '.rela': {'min': 1.2220198859116511, 'max':
    1.2220198859116511, 'total': 1.2220198859116511, 'count': 1, 'mean': 1.2220198859116511},
     '.rela.plt': {'min': 1.6210579369162064, 'max': 1.6210579369162064, 'total':
    1.6210579369162064, 'count': 1, 'mean': 1.6210579369162064}, '.gnu.version': {'min':
    1.740960896496825, 'max': 1.740960896496825, 'total': 1.740960896496825, 'count': 1, '
```

```
mean': 1.740960896496825}, '.gnu.version_r': {'min': 2.5252192137385356, 'max':
2.5252192137385356, 'total': 2.5252192137385356, 'count': 1, 'mean': 2.5252192137385356},
 '.hash': {'min': 1.8077229039022173, 'max': 1.8077229039022173, 'total':
1.8077229039022173, 'count': 1, 'mean': 1.8077229039022173}, '.dynstr': {'min':
4.572648045349689, 'max': 4.572648045349689, 'total': 4.572648045349689, 'count': 1, '
mean': 4.572648045349689}, '.shstrtab': {'min': 4.351353948202993, 'max':
4.351353948202993, 'total': 4.351353948202993, 'count': 1, 'mean': 4.351353948202993}, '.
dynsym': {'min': 1.0104030032533726, 'max': 1.0104030032533726, 'total':
1.0104030032533726, 'count': 1, 'mean': 1.0104030032533726}, '.typelink': {'min':
4.352172764173542, 'max': 4.352172764173542, 'total': 4.352172764173542, 'count': 1, '
mean': 4.352172764173542}, '.itablink': {'min': 2.260342786820047, 'max':
2.260342786820047, 'total': 2.260342786820047, 'count': 1, 'mean': 2.260342786820047}, '.
gosymtab': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.gopclntab': {'min
': 5.599953726591909, 'max': 5.599953726591909, 'total': 5.599953726591909, 'count': 1, '
mean': 5.599953726591909}, '.go.buildinfo': {'min': 3.5372301466508205, 'max':
3.5372301466508205, 'total': 3.5372301466508205, 'count': 1, 'mean': 3.5372301466508205},
 '.got.plt': {'min': 2.18951339608244, 'max': 2.18951339608244, 'total':
2.18951339608244, 'count': 1, 'mean': 2.18951339608244}, '.dynamic': {'min':
1.6577221539606757, 'max': 1.6577221539606757, 'total': 1.6577221539606757, 'count': 1, '
mean': 1.6577221539606757}, '.got': {'min': 0.0, 'max': 0.0, 'total': 0.0, 'count': 1, '
mean': 0.0}, '.noptrdata': {'min': 5.1736147485815644, 'max': 5.1736147485815644, 'total
': 5.1736147485815644, 'count': 1, 'mean': 5.1736147485815644}, '.data': {'min':
1.6332834663111804, 'max': 1.6332834663111804, 'total': 1.6332834663111804, 'count': 1, '
mean': 1.6332834663111804}, '.bss': {'min': 7.994400722933447, 'max': 7.994400722933447,
'total': 7.994400722933447, 'count': 1, 'mean': 7.994400722933447}, '.noptrbss': {'min':
7.9170020206866765, 'max': 7.9170020206866765, 'total': 7.9170020206866765, 'count': 1, '
mean': 7.9170020206866765}, '.tbss': {'min': 2.75, 'max': 2.75, 'total': 2.75, 'count':
1, 'mean': 2.75}, '.zdebug_abbrev': {'min': 7.186678878967747, 'max': 7.186678878967747,
'total': 7.186678878967747, 'count': 1, 'mean': 7.186678878967747}, '.zdebug_line': {'min
': 7.9955651626178295, 'max': 7.9955651626178295, 'total': 7.9955651626178295, 'count':
1, 'mean': 7.9955651626178295}, '.zdebug_frame': {'min': 7.931337274423072, 'max':
7.931337274423072, 'total': 7.931337274423072, 'count': 1, 'mean': 7.931337274423072}, '.
zdebug_pubnames': {'min': 7.960649887021013, 'max': 7.960649887021013, 'total':
7.960649887021013, 'count': 1, 'mean': 7.960649887021013}, '.zdebug_pubtypes': {'min':
7.984492715530015, 'max': 7.984492715530015, 'total': 7.984492715530015, 'count': 1, '
mean': 7.984492715530015}, '.debug_gdb_scripts': {'min': 4.220128777433187, 'max':
4.220128777433187, 'total': 4.220128777433187, 'count': 1, 'mean': 4.220128777433187}, '.
zdebug_info': {'min': 7.996917784963646, 'max': 7.996917784963646, 'total':
7.996917784963646, 'count': 1, 'mean': 7.996917784963646}, '.zdebug_loc': {'min':
7.994350388445776, 'max': 7.994350388445776, 'total': 7.994350388445776, 'count': 1, '
mean': 7.994350388445776}, '.zdebug_ranges': {'min': 7.803982755586464, 'max':
7.803982755586464, 'total': 7.803982755586464, 'count': 1, 'mean': 7.803982755586464}, '.
interp': {'min': 3.94075983254009, 'max': 3.94075983254009, 'total': 3.94075983254009, '
count': 1, 'mean': 3.94075983254009}, '.note.go.buildid': {'min': 5.352271034814013, 'max
': 5.352271034814013, 'total': 5.352271034814013, 'count': 1, 'mean': 5.352271034814013},
 '.symtab': {'min': 3.2692849180450874, 'max': 3.2692849180450874, 'total':
3.2692849180450874, 'count': 1, 'mean': 3.2692849180450874}, '.strtab': {'min':
5.051145007242242, 'max': 5.051145007242242, 'total': 5.051145007242242, 'count': 1, '
mean': 5.051145007242242}}
```

M.12.2. Dynamic Analysis Features

```
        "Behavior": {
            "files_opened": [
                "/etc/ld.so.cache",
                "/lib/x86_64-linux-gnu/libpthread.so.0",
                "/lib/x86_64-linux-gnu/libc.so.6",
                "//.",
                "/dev/bus/usb",
                "/dev/mei",
                "/dev/mei0",
                "/dev/mem",
                "/etc/",
                "/etc/fstab",
                "/etc/fwupd/daemon.conf",
                "/etc/fwupd/redfish.conf",
                "/etc/fwupd/remotes.d",
                "/etc/fwupd/remotes.d/dell-esrt.conf",
                "/etc/fwupd/remotes.d/lvfs-testing.conf",
                "/etc/fwupd/remotes.d/lvfs.conf",
                "/etc/fwupd/remotes.d/vendor-directory.conf",
                "/etc/fwupd/remotes.d/vendor.conf",
                "/etc/fwupd/thunderbolt.conf",
                "/etc/fwupd/upower.conf",
                "/etc/gai.conf",
                "/etc/gcrypt/hwf.deny",
                "/etc/group",
                "/etc/host.conf",
                "/etc/hosts",
                "/etc/locale.alias",
                "/etc/logrotate.conf",
                "/etc/logrotate.d",
                "/etc/logrotate.d/alternatives",
                "/etc/logrotate.d/apport",
                "/etc/logrotate.d/apt",
                "/etc/logrotate.d/bootlog",
                "/etc/logrotate.d/btmp",
                "/etc/logrotate.d/cups-daemon",
                "/etc/logrotate.d/dpkg",
                "/etc/logrotate.d/lightdm",
                "/etc/logrotate.d/ppp",
                "/etc/logrotate.d/rsyslog",
                "/etc/logrotate.d/speech-dispatcher",
                "/etc/logrotate.d/ubuntu-advantage-tools",
                "/etc/logrotate.d/ufw",
                "/etc/logrotate.d/wtmp",
                "/etc/machine-id",
                "/etc/os-release",
                "/etc/pki/fwupd",
                "/etc/pki/fwupd-metadata",
                "/etc/pki/fwupd-metadata/GPG-KEY-Linux-Foundation-Metadata",
                "/etc/pki/fwupd-metadata/GPG-KEY-Linux-Vendor-Firmware-Service",
                "/etc/pki/fwupd-metadata/LVFS-CA.pem",
                "/etc/pki/fwupd/GPG-KEY-Linux-Foundation-Firmware"
            ],
            "files_written": [
                "/var/cache/fwupd/.goutputstream-ACLH51",
                "/var/lib/fwupd/gnupg/.#lk0x0000558eb7589b80.buffalo.3589",
                "/var/lib/fwupd/gnupg/.#lk0x000055988aad3b80.buffalo.3583",
                "/var/lib/fwupd/gnupg/.#lk0x000055e78c7deb80.buffalo.3577",
                "/var/lib/fwupd/gnupg/.#lk0x0000561b847eeb80.buffalo.3587",
                "/var/lib/fwupd/gnupg/.#lk0x00005647e9e63b80.buffalo.3585",
                "/var/lib/fwupd/gnupg/.#lk0x000056527d2a9b80.buffalo.3579",
                "/var/lib/fwupd/gnupg/pubring.kbx.tmp",
                "/var/lib/fwupd/remotes.d/lvfs/metadata.xml.gz.O3VE51",
                "/var/lib/fwupd/remotes.d/lvfs/metadata.xml.gz.jcat.ML5W41",
                "/var/lib/logrotate/status.tmp",
                "/var/log/apport.log.1.gz",
                "/var/log/auth.log.1.gz",
                "/var/log/cups/access_log.1.gz",
                "/var/log/cups/error_log.1.gz",
                "/var/log/kern.log.1.gz",
                "/var/log/syslog.1.gz",
                "/var/log/ubuntu-advantage-timer.log.1.gz"
            ],
            "files_deleted": [
                "/var/lib/fwupd/gnupg/.#lk0x0000558eb7589b80.buffalo.3589",
                "/var/lib/fwupd/gnupg/.#lk0x0000558eb7589b80.buffalo.3589x",
                "/var/lib/fwupd/gnupg/.#lk0x000055988aad3b80.buffalo.3583",
                "/var/lib/fwupd/gnupg/.#lk0x000055988aad3b80.buffalo.3583x",
                "/var/lib/fwupd/gnupg/.#lk0x000055e78c7deb80.buffalo.3577",
                "/var/lib/fwupd/gnupg/.#lk0x000055e78c7deb80.buffalo.3577x",
                "/var/lib/fwupd/gnupg/.#lk0x0000561b847eeb80.buffalo.3587",
                "/var/lib/fwupd/gnupg/.#lk0x0000561b847eeb80.buffalo.3587x",
                "/var/lib/fwupd/gnupg/.#lk0x00005647e9e63b80.buffalo.3585",
                "/var/lib/fwupd/gnupg/.#lk0x00005647e9e63b80.buffalo.3585x",
                "/var/lib/fwupd/gnupg/.#lk0x000056527d2a9b80.buffalo.3579",
                "/var/lib/fwupd/gnupg/.#lk0x000056527d2a9b80.buffalo.3579x",
                "/var/lib/fwupd/gnupg/pubring.kbx.lock",
                "/var/lib/fwupd/gnupg/pubring.kbx.tmp",
                "/var/lib/logrotate/status.tmp",
                "/var/log/apport.log.1",
                "/var/log/apport.log.8.gz",
                "/var/log/auth.log.1",
                "/var/log/auth.log.5.gz",
                "/var/log/btmp.2",
                "/var/log/cups/access_log.1",
                "/var/log/cups/access_log.8.gz",
                "/var/log/cups/error_log.1",
                "/var/log/kern.log.1",
                "/var/log/kern.log.5.gz",
                "/var/log/syslog.1",
                "/var/log/syslog.8.gz",
                "/var/log/ubuntu-advantage-timer.log.1"
            ],
            "command_executions": [
                "/sbin/fstrim --fstab --verbose --quiet",
                "/usr/bin/fwupdmgr refresh",
                "/usr/sbin/logrotate /etc/logrotate.conf",
                "/bin/gzip",
                "sh -c \"\\n\\t\\tinvoke-rc.d --quiet cups restart > /dev/null\\n\" logrotate_script \"/var/log/cups/*log \"",
                "invoke-rc.d --quiet cups restart",
                "/sbin/runlevel",
                "systemctl --quiet is-enabled cups.service",
                "ls /etc/rc[S2345].d/S[0-9][0-9]cups",
                "systemctl --quiet is-active cups.service",
                "sh -c /usr/lib/rsyslog/rsyslog-rotate logrotate_script /var/log/syslog",
                "/usr/lib/rsyslog/rsyslog-rotate",
                "systemctl kill -s HUP rsyslog.service",
                "sh -c /usr/lib/rsyslog/rsyslog-rotate logrotate_script /var/log/mail.info/var/log/mail.warn/var/log/mail.err/var/log/mail.log/var/log/daemon.log/var/log/kern.log/var/log/auth.log/var/log/user.log/var/log/lpr.log/var/log/cron.log/var/log/debug/var/log/messages",
                "/usr/libexec/fwupd/fwupd",
                "/usr/bin/gpgconf --list-dirs",
                "/usr/bin/gpgconf --list-components",
                "/usr/bin/gpg --version",
                "/usr/bin/gpgsm --version",
                "/usr/bin/gpgconf --version"
            ],
            "files_attribute_changed": [
                "/var/log/apport.log.1.gz",
                "/var/log/apport.log",
                "/var/log/btmp",
                "/var/log/cups/access_log.1.gz",
                "/var/log/cups/error_log.1.gz",
                "/var/log/cups/access_log",
                "/var/log/cups/error_log",
                "/var/log/syslog.1.gz",
                "/var/log/syslog",
                "/var/log/kern.log.1.gz",
                "/var/log/auth.log.1.gz",
                "/var/log/kern.log",
                "/var/log/auth.log",
                "/var/log/ubuntu-advantage-timer.log.1.gz",
                "/var/log/ubuntu-advantage-timer.log",
                "/var/lib/logrotate/status.tmp",
                "/var/cache/fwupd/.goutputstream-ACLH51",
                "/run/motd.d/fwupd/.goutputstream-IQ9641",
                "/run/motd.d/fwupd/.goutputstream-U5OA51"
            ],
            "processes_terminated": [],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Executes commands using a shell command-line interpreter",
                "Executes the \"systemctl\" command used for controlling the systemd system and service manager",
                "Sample and/or dropped files symbols with suspicious names",
                "Creates hidden files, links and/or directories",
                "Deletes log files",
                "May try to detect the virtual machine to hinder analysis (VM artifact strings found in memory)",
                "Uses the \"uname\" system call to query kernel version information (possible evasion)",
                "Reads CPU information from /proc indicative of miner or evasive malware",
                "Reads system information from the proc file system",
                "Performs DNS lookups",
                "Performs DNS lookups"
            ]
        }
    }
```

M.13. FritzFrog

M.13.1. Pseudo-Static analysis

```
section: {'entry': '0x45d1d0', 'sections': [{'name': '', 'size': 0, 'entropy': -0.0, 'file_offset': 0, 'props': ['Type: NULL']}, {'name': '.text', 'size': 4158883, 'entropy': 5.892871008625818, 'file_offset': 4096, 'props': ['Type: PROGBITS', 'ALLOC', 'EXECINSTR']}, {'name': '.rodata', 'size': 1793920, 'entropy': 4.279657059185294, 'file_offset': 4165632, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.shstrtab', 'size': 124, 'entropy': 4.055184950639968, 'file_offset': 5959552, 'props': ['Type: STRTAB']}, {'name': '.typelink', 'size': 14072, 'entropy': 5.181381557486275, 'file_offset': 5959680, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.itablink', 'size': 3384, 'entropy': 2.530798983757644, 'file_offset': 5973752, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.gosymtab', 'size': 0, 'entropy': -0.0, 'file_offset': 5977136, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.gopclntab', 'size': 2773252, 'entropy': 5.737537728525165, 'file_offset': 5977152, 'props': ['Type: PROGBITS', 'ALLOC']}, {'name': '.noptrdata', 'size': 215808, 'entropy': 6.286892199147341, 'file_offset': 8753152, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}, {'name': '.data', 'size': 107056, 'entropy': 1.7852031614129318, 'file_offset': 8968960, 'props': ['Type: PROGBITS', 'ALLOC', 'WRITE']}]}
Segment Information: {'Sections': [{'Name': '', 'Size': 0, 'Virtual Address': 0}, {'Name': '.text', 'Size': 4158883, 'Virtual Address': 4198400}, {'Name': '.rodata', 'Size': 1793920, 'Virtual Address': 8359936}, {'Name': '.shstrtab', 'Size': 124, 'Virtual Address': 0}, {'Name': '.typelink', 'Size': 14072, 'Virtual Address': 10153984}, {'Name': '.itablink', 'Size': 3384, 'Virtual Address': 10168056}, {'Name': '.gosymtab', 'Size': 0, 'Virtual Address': 10171440}, {'Name': '.gopclntab', 'Size': 2773252, 'Virtual Address': 10171456}, {'Name': '.noptrdata', 'Size': 215808, 'Virtual Address': 12947456}, {'Name': '.data', 'Size': 107056, 'Virtual Address': 13163264}, {'Name': '.bss', 'Size': 125296, 'Virtual Address': 13270336}, {'Name': '.noptrbss', 'Size': 13336, 'Virtual Address': 13395648}, {'Name': '.note.go.buildid', 'Size': 100, 'Virtual Address': 4198300}], 'Segments': [{'Type': 'SEGMENT_TYPES.PHDR', 'Size': 392, 'Virtual Address': 4194368}, {'Type': 'SEGMENT_TYPES.NOTE', 'Size': 100, 'Virtual Address': 4198300}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 4162979, 'Virtual Address': 4194304}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 4584772, 'Virtual Address': 8359936}, {'Type': 'SEGMENT_TYPES.LOAD', 'Size': 322880, 'Virtual Address': 12947456}, {'Type': 'SEGMENT_TYPES.GNU_STACK', 'Size': 0, 'Virtual Address': 0}, {'Type': 'SEGMENT_TYPES.???', 'Size': 0, 'Virtual Address': 0}]}
imports: {}
exports: []
general: {'size': 9076032, 'virtual_size': 0, 'has_debug': 0, 'exports': 0, 'imports': 0, 'has_relocations': 0, 'symbols': 0}
header: {'file_type': 'EXECUTABLE', 'entry_point': 4575696, 'machine_type': 'x86_64', 'header_size': 64, 'program_headers': [{'type': 'PHDR', 'virtual_address': 4194368, 'physical_address': 4194368, 'physical_size': 392, 'virtual_size': 392, 'flags': SEGMENT_FLAGS.R, 'alignment': 4096}, {'type': 'NOTE', 'virtual_address': 4198300, 'physical_address': 4198300, 'physical_size': 100, 'virtual_size': 100, 'flags': SEGMENT_FLAGS.R, 'alignment': 4}, {'type': 'LOAD', 'virtual_address': 4194304, 'physical_address': 4194304, 'physical_size': 4162979, 'virtual_size': 4162979, 'flags': SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 8359936, 'physical_address': 8359936, 'physical_size': 4584772, 'virtual_size': 4584772, 'flags': SEGMENT_FLAGS.R, 'alignment': 4096}, {'type': 'LOAD', 'virtual_address': 12947456, '
```

```
    physical_address': 12947456, 'physical_size': 322880, 'virtual_size': 461528, 'flags':
    SEGMENT_FLAGS.???, 'alignment': 4096}, {'type': 'GNU_STACK', 'virtual_address': 0, '
    physical_address': 0, 'physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???,
    'alignment': 8}, {'type': '???', 'virtual_address': 0, 'physical_address': 0, '
    physical_size': 0, 'virtual_size': 0, 'flags': SEGMENT_FLAGS.???, 'alignment': 8}], '
    section_headers': [{'name': '', 'type': 'NULL', 'virtual_address': 0, 'size': 0, 'entropy
    ': -0.0}, {'name': '.text', 'type': 'PROGBITS', 'virtual_address': 4198400, 'size':
    4158883, 'entropy': 5.892871008625818}, {'name': '.rodata', 'type': 'PROGBITS', '
    virtual_address': 8359936, 'size': 1793920, 'entropy': 4.279657059185294}, {'name': '.
    shstrtab', 'type': 'STRTAB', 'virtual_address': 0, 'size': 124, 'entropy':
    4.055184950639968}, {'name': '.typelink', 'type': 'PROGBITS', 'virtual_address':
    10153984, 'size': 14072, 'entropy': 5.181381557486275}, {'name': '.itablink', 'type': '
    PROGBITS', 'virtual_address': 10168056, 'size': 3384, 'entropy': 2.530798983757644}, {'
    name': '.gosymtab', 'type': 'PROGBITS', 'virtual_address': 10171440, 'size': 0, 'entropy
    ': -0.0}, {'name': '.gopclntab', 'type': 'PROGBITS', 'virtual_address': 10171456, 'size':
     2773252, 'entropy': 5.737537728525165}, {'name': '.noptrdata', 'type': 'PROGBITS', '
    virtual_address': 12947456, 'size': 215808, 'entropy': 6.286892199147341}, {'name': '.
    data', 'type': 'PROGBITS', 'virtual_address': 13163264, 'size': 107056, 'entropy':
    1.7852031614129318}, {'name': '.bss', 'type': 'NOBITS', 'virtual_address': 13270336, '
    size': 125296, 'entropy': -0.0}, {'name': '.noptrbss', 'type': 'NOBITS', 'virtual_address
    ': 13395648, 'size': 13336, 'entropy': -0.0}, {'name': '.note.go.buildid', 'type': 'NOTE
    ', 'virtual_address': 4198300, 'size': 100, 'entropy': 5.297173284770744}]}
strings: {'numstrings': 4868, 'avlength': 5.942686935086278, 'printabledist': [567, 23, 35,
    21, 3733, 14, 8, 15, 484, 395, 18, 95, 38, 113, 35, 24, 624, 217, 86, 197, 65, 28, 41,
    23, 475, 3087, 91, 839, 77, 50, 36, 19, 469, 288, 56, 67, 826, 222, 73, 30, 4219, 130,
    65, 40, 967, 103, 53, 29, 424, 56, 59, 44, 859, 11, 22, 11, 371, 24, 58, 17, 351, 11, 40,
     7, 299, 217, 51, 261, 187, 308, 316, 106, 289, 215, 8, 49, 206, 77, 238, 259, 461, 5,
    173, 190, 816, 1103, 39, 98, 276, 39, 16, 2, 908, 35, 36, 71], 'printables': 28929, '
    entropy': 5.014535903930664, 'paths': 0, 'urls': 0, 'registry': 0, 'MZ': 5}
EntryPoint: [['Main ', 4575696], ['Execu', 4194304]]
ExitPoint: []
Opcodes: ['mov', 'cmp', 'jbe', 'sub', 'mov', 'lea', 'call', 'mov', 'mov', 'mov', 'mov', 'call
    ', 'mov', 'add', 'ret', 'call', 'jmp', 'int3', 'int3', 'int3', 'int3', 'int3', 'mov', '
    lea', 'cmp', 'jbe', 'sub', 'mov', 'lea', 'mov', 'mov', 'jmp', 'mov', 'mov', 'mov', 'test
    ', 'je', 'mov', 'mov', 'mov', 'mov', 'mov', 'call', 'mov', 'test', 'jge', 'xor', 'xor', '
    mov', 'cmp', 'jl', 'mov', 'cmp', 'jne', 'mov', 'mov', 'mov', 'mov', 'mov', 'mov', 'call',
     'mov', 'test', 'jl', 'cmp', 'jb', 'mov', 'cmp', 'ja', 'lea', 'mov', 'neg', 'sar', 'and',
     'mov', 'lea', 'lea', 'cmp', 'ja', 'mov', 'mov', 'sub', 'mov', 'neg', 'sar', 'and', 'lea
    ', 'cmp', 'jne', 'movzx', 'cmp', 'jne', 'mov', 'cmp', 'jne', 'movzx', 'cmp', 'jne', '
    movzx', 'cmp', 'jne', 'mov', 'xor', 'jmp', 'mov', 'lea', 'cmp', 'jge', 'mov', 'cmp', 'jbe
    ', 'mov', 'shl', 'mov', 'test', 'je', 'mov', 'mov', 'cmp', 'ja', 'jmp', 'mov', 'cmp', '
    jbe', 'movzx', 'jmp', 'mov', 'mov', 'jmp', 'mov', 'mov', 'mov', 'xor', 'jmp', 'lea', 'cmp
    ', 'jge', 'mov', 'cmp', 'jbe', 'mov', 'shl', 'mov', 'mov', 'cmp', 'jne', 'mov', 'mov', '
    mov', 'mov', 'mov', 'call', 'cmp', 'jne', 'mov', 'movzx', 'mov', 'mov', 'mov', 'jmp', '
    mov', 'mov', 'cmp', 'jbe', 'mov', 'mov', 'mov', 'cmp', 'jbe', 'movzx', 'mov', 'mov', 'mov
    ', 'jmp', 'call', 'lea', 'mov', 'mov', 'call', 'mov', 'mov', 'mov', 'mov', 'call', 'lea',
     'mov', 'mov', 'call', 'call', 'mov', 'mov', 'jmp', 'mov', 'mov', 'cmp', 'jne', 'movzx',
    'cmp', 'jne', 'movzx']
Opcode-Occurrence: {'mov': 3670, 'cmp': 540, 'jbe': 124, 'sub': 158, 'lea': 575, 'call': 703,
     'add': 295, 'ret': 262, 'jmp': 411, 'int3': 1269, 'test': 201, 'je': 193, 'jge': 14, '
    xor': 162, 'jl': 30, 'jne': 237, 'jb': 33, 'ja': 50, 'neg': 11, 'sar': 15, 'and': 38, '
    movzx': 97, 'shl': 34, 'jle': 9, 'movups': 45, 'inc': 33, 'ud2': 68, 'nop': 179, 'bt':
    20, 'setb': 21, 'jae': 22, 'cpuid': 1, 'xgetbv': 1, 'cmove': 2, 'cmovl': 2, 'movdqu': 33,
     'pcmpeqb': 20, 'pmovmskb': 17, 'bsf': 6, 'seta': 1, 'bswap': 4, 'bsr': 2, 'shr': 30, '
    setg': 1, 'sete': 26, 'vmovdqu': 18, 'vpcmpeqb': 11, 'vpmovmskb': 9, 'vzeroupper': 11, '
    movq': 4, 'punpcklbw': 4, 'pshufd': 2, 'popcnt': 6, 'vpbroadcastb': 2, 'pand': 3, 'vpand
    ': 1, 'pcmpestri': 2, 'vptest': 2, 'lock cmpxchg': 8, 'lock xadd': 1, 'xchg': 12, 'movss
    ': 16, 'xorps': 31, 'ucomiss': 5, 'jnp': 4, 'movabs': 45, 'imul': 46, 'movsd': 24, '
    ucomisd': 5, 'setnp': 6, 'or': 7, 'dec': 6, 'cmova': 4, 'mul': 1, 'jo': 1, 'jg': 3, '
    setne': 4, 'cmovg': 4, 'movsxd': 2, 'cvtss2sd': 3, 'movsx': 2, 'rol': 14, 'cmovne': 4, '
    cmovae': 2}
Image Size: 13270336
Header Size: {'ELF Header Size': 64, 'Program Headers Total Size': 9071123}
GNU Physical Size: 0
Heap Size: {'Heap Segment Size': 322880, 'Heap Section Size': 0}
Loader Flags: {'Segment 0': {'Type': 'SEGMENT_TYPES.PHDR', 'Flags': 'READ'}, 'Segment 1': {'
    Type': 'SEGMENT_TYPES.NOTE', 'Flags': 'READ'}, 'Segment 2': {'Type': 'SEGMENT_TYPES.LOAD
    ', 'Flags': 'READ | EXECUTE'}, 'Segment 3': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ
    '}, 'Segment 4': {'Type': 'SEGMENT_TYPES.LOAD', 'Flags': 'READ | WRITE'}, 'Segment 5': {'
    Type': 'SEGMENT_TYPES.GNU_STACK', 'Flags': 'READ | WRITE'}, 'Segment 6': {'Type': '
    SEGMENT_TYPES.???', 'Flags': ''}}
```

```
Section Entropy: {'': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.text': {'
    min': 5.892871008625818, 'max': 5.892871008625818, 'total': 5.892871008625818, 'count':
    1, 'mean': 5.892871008625818}, '.rodata': {'min': 4.279657059185294, 'max':
    4.279657059185294, 'total': 4.279657059185294, 'count': 1, 'mean': 4.279657059185294}, '.
    shstrtab': {'min': 4.055184950639968, 'max': 4.055184950639968, 'total':
    4.055184950639968, 'count': 1, 'mean': 4.055184950639968}, '.typelink': {'min':
    5.181381557486275, 'max': 5.181381557486275, 'total': 5.181381557486275, 'count': 1, '
    mean': 5.181381557486275}, '.itablink': {'min': 2.530798983757644, 'max':
    2.530798983757644, 'total': 2.530798983757644, 'count': 1, 'mean': 2.530798983757644}, '.
    gosymtab': {'min': 0, 'max': 0, 'total': 0, 'count': 1, 'mean': 0.0}, '.gopclntab': {'min
    ': 5.737537728525165, 'max': 5.737537728525165, 'total': 5.737537728525165, 'count': 1, '
    mean': 5.737537728525165}, '.noptrdata': {'min': 6.286892199147341, 'max':
    6.286892199147341, 'total': 6.286892199147341, 'count': 1, 'mean': 6.286892199147341}, '.
    data': {'min': 1.7852031614129318, 'max': 1.7852031614129318, 'total':
    1.7852031614129318, 'count': 1, 'mean': 1.7852031614129318}, '.bss': {'min': 0.0, 'max':
    0.0, 'total': 0.0, 'count': 1, 'mean': 0.0}, '.noptrbss': {'min': 0.0, 'max': 0.0, 'total
    ': 0.0, 'count': 1, 'mean': 0.0}, '.note.go.buildid': {'min': 5.297173284770744, 'max':
    5.297173284770744, 'total': 5.297173284770744, 'count': 1, 'mean': 5.297173284770744}}
Kolmogorov Complexity: 1597 KB
```

M.13.2. Dynamic Analysis Features

```
    "Behavior": {
        "files_opened": [
            "/proc/self/oom_score_adj",
            "/etc/ld.so.cache",
            "/lib/x86_64-linux-gnu/libwrap.so.0",
            "/lib/x86_64-linux-gnu/libpam.so.0",
            "/lib/x86_64-linux-gnu/libselinux.so.1",
            "/usr/lib/x86_64-linux-gnu/libck-connector.so.0",
            "/lib/x86_64-linux-gnu/libdbus-1.so.3",
            "/lib/x86_64-linux-gnu/libcrypto.so.1.0.0",
            "/lib/x86_64-linux-gnu/libutil.so.1",
            "/lib/x86_64-linux-gnu/libz.so.1",
            "/lib/x86_64-linux-gnu/libcrypt.so.1",
            "/usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2",
            "/usr/lib/x86_64-linux-gnu/libkrb5.so.3",
            "/lib/x86_64-linux-gnu/libcom_err.so.2",
            "/lib/x86_64-linux-gnu/libc.so.6",
            "/lib/x86_64-linux-gnu/libnsl.so.1",
            "/lib/x86_64-linux-gnu/libaudit.so.1",
            "/lib/x86_64-linux-gnu/libdl.so.2",
            "/lib/x86_64-linux-gnu/libpcre.so.3",
            "/lib/x86_64-linux-gnu/libpthread.so.0",
            "/lib/x86_64-linux-gnu/librt.so.1",
            "/usr/lib/x86_64-linux-gnu/libk5crypto.so.3",
            "/usr/lib/x86_64-linux-gnu/libkrb5support.so.0",
            "/lib/x86_64-linux-gnu/libkeyutils.so.1",
            "/lib/x86_64-linux-gnu/libresolv.so.2",
            "/proc/filesystems",
            "/dev/null",
            "/usr/lib/ssl/openssl.cnf",
            "/dev/urandom",
            "/etc/gai.conf",
            "/etc/nsswitch.conf",
            "/lib/x86_64-linux-gnu/libnss_compat.so.2",
            "/lib/x86_64-linux-gnu/libnss_nis.so.2",
            "/lib/x86_64-linux-gnu/libnss_files.so.2",
            "/etc/passwd",
            "/etc/ssh/ssh_host_rsa_key",
            "/etc/ssh/ssh_host_rsa_key.pub",
            "/etc/ssh/ssh_host_dsa_key",
            "/etc/ssh/ssh_host_dsa_key.pub",
            "/etc/ssh/ssh_host_ecdsa_key",
            "/etc/ssh/ssh_host_ecdsa_key.pub",
            "/etc/ssh/ssh_host_ed25519_key",
            "/etc/ssh/ssh_host_ed25519_key.pub",
            "/etc/protocols",
            "/etc/hosts.allow",
            "/etc/hosts.deny",
            "/etc/localtime",
            "/var/log/btmp",
            "/etc/pam.d/sshd",
            "/etc/pam.d/common-auth"
        ],
        "files_written": [
            "/proc/self/oom_score_adj",
            "/var/log/btmp",
            "/root/.ssh/authorized_keys",
            "/var/cache/fwupd/.goutputstream-U7LL31",
            "/var/lib/fwupd/gnupg/.#lk0x0000557b1bd52b80.buffalo.6124",
            "/var/lib/fwupd/gnupg/.#lk0x0000559be0816b80.buffalo.6143",
            "/var/lib/fwupd/gnupg/.#lk0x000055ac75868b80.buffalo.6138",
            "/var/lib/fwupd/gnupg/.#lk0x000055bec710eb80.buffalo.6128",
            "/var/lib/fwupd/gnupg/.#lk0x000055e6bd352b80.buffalo.6122",
            "/var/lib/fwupd/gnupg/.#lk0x000055ed080d8b80.buffalo.6159",
            "/var/lib/fwupd/gnupg/pubring.kbx.tmp",
            "/var/lib/fwupd/remotes.d/lvfs/metadata.xml.gz.L0UK31",
            "/var/lib/fwupd/remotes.d/lvfs/metadata.xml.gz.jcat.W43921",
            "/var/lib/logrotate/status.tmp",
            "/var/log/auth.log.1.gz",
            "/var/log/kern.log.1.gz",
            "/var/log/syslog.1.gz",
            "/var/log/ubuntu-advantage-timer.log.1.gz"
        ],
        "files_deleted": [
            "//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/dev/shm//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/etc//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/opt//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/root//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/run/user/1000//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619
                ",
            "/tmp/..//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/tmp//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/usr//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/cache//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/lib/fwupd/gnupg/.#lk0x0000557b1bd52b80.buffalo.6124",
            "/var/lib/fwupd/gnupg/.#lk0x0000557b1bd52b80.buffalo.6124x",
            "/var/lib/fwupd/gnupg/.#lk0x0000559be0816b80.buffalo.6143",
            "/var/lib/fwupd/gnupg/.#lk0x0000559be0816b80.buffalo.6143x",
            "/var/lib/fwupd/gnupg/.#lk0x000055ac75868b80.buffalo.6138",
            "/var/lib/fwupd/gnupg/.#lk0x000055ac75868b80.buffalo.6138x",
            "/var/lib/fwupd/gnupg/.#lk0x000055bec710eb80.buffalo.6128",
            "/var/lib/fwupd/gnupg/.#lk0x000055bec710eb80.buffalo.6128x",
            "/var/lib/fwupd/gnupg/.#lk0x000055e6bd352b80.buffalo.6122",
            "/var/lib/fwupd/gnupg/.#lk0x000055e6bd352b80.buffalo.6122x",
            "/var/lib/fwupd/gnupg/.#lk0x000055ed080d8b80.buffalo.6159",
            "/var/lib/fwupd/gnupg/.#lk0x000055ed080d8b80.buffalo.6159x",
            "/var/lib/fwupd/gnupg/pubring.kbx.lock",
            "/var/lib/fwupd/gnupg/pubring.kbx.tmp",
            "/var/lib/logrotate/status.tmp",
            "/var/local//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/log//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/log/auth.log.1",
            "/var/log/kern.log.1",
            "/var/log/syslog.1",
            "/var/log/ubuntu-advantage-timer.log.1",
            "/var/mail//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/opt//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/spool//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/tmp//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619",
            "/var/www//tmp/68b86856665f2cc0e0e71668c0b6aac8b14326c623995ba5963f22257619"
        ],
        "command_executions": [
            "/usr/sbin/sshd",
            "/usr/sbin/sshd -D -R",
            "/sbin/fstrim --fstab --verbose --quiet",
            "/usr/sbin/logrotate /etc/logrotate.conf",
            "/bin/gzip",
            "sh -c \"\\n\\t\\tinvoke-rc.d --quiet cups restart > /dev/null\\n\"
                logrotate_script \"/var/log/cups/*log \"",
            "invoke-rc.d --quiet cups restart",
            "/sbin/runlevel",
            "systemctl --quiet is-enabled cups.service",
            "ls /etc/rc[S2345].d/S[0-9][0-9]cups",
            "systemctl --quiet is-active cups.service",
            "sh -c /usr/lib/rsyslog/rsyslog-rotate logrotate_script /var/log/syslog",
            "/usr/lib/rsyslog/rsyslog-rotate",
            "systemctl kill -s HUP rsyslog.service",
            "sh -c /usr/lib/rsyslog/rsyslog-rotate logrotate_script /var/log/mail.info/var/
                log/mail.warn/var/log/mail.err/var/log/mail.log/var/log/daemon.log/var/log/
                kern.log/var/log/auth.log/var/log/user.log/var/log/lpr.log/var/log/cron.log/
                var/log/debug/var/log/messages",
            "/usr/libexec/fwupd/fwupd",
            "/usr/bin/gpgconf --list-dirs",
            "/usr/bin/gpgconf --list-components",
            "/usr/bin/gpg --version",
            "/usr/bin/gpgsm --version"
        ],
        "files_attribute_changed": [
            "/var/log/syslog.1.gz",
            "/var/log/syslog",
            "/var/log/kern.log.1.gz",
            "/var/log/auth.log.1.gz",
            "/var/log/kern.log",
            "/var/log/auth.log",
            "/var/log/ubuntu-advantage-timer.log.1.gz",
            "/var/log/ubuntu-advantage-timer.log",
            "/var/lib/logrotate/status.tmp",
            "/var/cache/fwupd/.goutputstream-U7LL31",
            "/run/motd.d/fwupd/.goutputstream-0D0D31",
            "/run/motd.d/fwupd/.goutputstream-TCWH31"
        ],
        "processes_terminated": [
            "sshd: [accepted]",
            "/usr/sbin/sshd -D -R"
        ],
        "processes_killed": [],
        "processes_injected": [],
        "services_opened": [],
        "services_created": [],
        "services_started": [],
        "services_stopped": [],
        "services_deleted": [],
        "windows_searched": [],
        "registry_keys_deleted": [],
        "mitre_attack_techniques": [
            "Executes commands using a shell command-line interpreter",
            "Executes the \"systemctl\" command used for controlling the systemd system and
                service manager",
            "Creates hidden files, links and/or directories",
            "Deletes log files",
            "Sample deletes itself",
            "Enumerates processes within the \"proc\" file system",
            "Uses the \"uname\" system call to query kernel version information (possible
                evasion)",
            "May try to detect the virtual machine to hinder analysis (VM artifact strings
                found in memory)",
            "Executes the \"uname\" command used to read OS and architecture name",
            "Reads CPU information from /proc indicative of miner or evasive malware",
            "Reads CPU information from /sys indicative of miner or evasive malware",
            "Reads system information from the proc file system",
            "Executes the \"free\" command used for querying memory usage (likely indicative
                for DDoS or mining capabilities)"
        ]
    }
}
}
```

M.14. DiscordTokenStealers

M.14.1. Dynamic Analysis Features

```
        "Behavior": {
            "files_opened": [
                "C:\\Windows\\Globalization\\Sorting\\sortdefault.nls",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\Config\\machine.config",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Management\\4dfa27fdd6a4cce26f99585e1c744f9b\\System.Management.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.IdentityModel\\9d45d2d6b426b57dc732ff567bb32dad\\System.IdentityModel.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Runteb92aa12#\\c56771a9cfb87e660d60453e232abe27\\System.Runtime.Serialization.ni.dll.aux",
                "C:\\Windows\\System32\\en-US\\tzres.dll.mui",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\Microsoft.CSharp\\dd1e55e4b87101888a94f28ce396f2ea\\Microsoft.CSharp.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Web.28b9ef5a#\\32f14fd0a5448b124076cd99f9b731dd\\System.Web.Extensions.ni.dll.aux",
                "C:\\Users\\Virtual\\Documents\\test nopqrst.docx",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Dynamic\\788fba784cfc29d8c324d66f6ee4c427\\System.Dynamic.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.ServiceModel\\f101d49ff42f71da4271bfa41dda9bd2\\System.ServiceModel.ni.dll.aux",
                "C:\\Users\\Virtual\\Desktop\\iiUDWxDuMalX.txt",
                "C:\\Users\\Virtual\\Documents\\test abcdef.docx",
                "C:\\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\mscorwks.dll",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Core\\55560c2014611e9119f99923c9ebdeef\\System.Core.ni.dll.aux",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\SortDefault.nlp",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Windows.Forms\\5aac750b35b27770dccb1a43f83cced7\\System.Windows.Forms.ni.dll.aux",
                "C:\\Users\\Virtual\\Desktop\\thoIgVqzAOWO.docm",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe.Config",
                "C:\\Windows\\assembly\\pubpol41.dat",
                "C:\\Users\\Virtual\\Documents\\test.docx",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Xml\\d86b080a37c60a872c82b912a2a63dac\\System.Xml.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Configuration\\46957030830964165644b52b0696c5d9\\System.Configuration.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Security\\11689060f7aa189e220cf9df9a97254e\\System.Security.ni.dll.aux",
                "C:\\Windows\\System32\\tzres.dll",
                "C:\\Users\\Virtual\\Desktop\\mdaxbBmTycg.txt",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Web\\14da86a7ddbf09bd27b30061ff9a4f5e\\System.Web.ni.dll.aux",
                "C:\\Users\\Virtual\\Desktop\\bGjjNflGGw.txt",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System\\52cca48930e580e3189eac47158c20be\\System.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Drawing\\646b4b01cb29986f8e076aa65c9e9753\\System.Drawing.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\mscorlib\\225759bb87c854c0fff27b1d84858c21\\mscorlib.ni.dll.aux",
                "C:\\Users\\Virtual\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\p6d36utp.default\\cookies.sqlite",
                "C:\\Users\\Virtual\\Documents\\test2.docx",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\SMDiagnostics\\4a2a848ea1fea1a74d5aa2f1c21c5ce8\\SMDiagnostics.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\WindowsBase\\32512bd09e2231f6eebb15fc17e3ad79\\WindowsBase.ni.dll.aux",
                "C:\\Users\\Virtual\\Documents\\test uvwxyz.docx",
                "C:\\Users\\Virtual\\Documents\\test ghijklm.docx",
                "C:\\Users\\Virtual\\Desktop\\ywHDbqpLmfeIaYpQq.docm",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\Presentatio5ae0f00f#\\da36abbea6ef456f432434d4d8d835c1\\PresentationFramework.ni.dll.aux",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\clr.dll",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\PresentationCore\\416ba33cb980d07643e82c4c45bd5786\\PresentationCore.ni.dll.aux",
                "C:\\Windows\\assembly\\NativeImages_v4.0.30319_32\\System.Servd1dec626#\\52
```

```
                    e9ac689c75dd011f0f7e827551e985\\System.ServiceModel.Internals.ni.dll.aux",
                "C:\\Users\\Virtual\\Desktop\\wkbGjjNflGGwcMJUy.docx",
                "C:\\Users\\Virtual\\Desktop\\UDWxDuMalX.txt",
                "C:\\Users\\Virtual\\Desktop\\tkmdaxbBmTy.txt",
                "C:\\Users\\RDhJ0CNFevzX\\Desktop\\RunGame.exe",
                "C:\\Windows\\system32\\version.dll",
                "C:\\Windows\\system32\\wsock32.dll",
                "C:\\Users\\<USER>\\Downloads\\"
            ],
            "files_written": [
                "C:\\Windows\\ServiceProfiles\\LocalService\\AppData\\Roaming\\Microsoft\\UPnP Device Host\\upnphost\\udhisapi.dll",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_RunGame.exe_2c4e7df7914dab9b98258d4557888593634d80_4dfbf061_17f1eb2b",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_RunGame.exe_2c4e7df7914dab9b98258d4557888593634d80_4dfbf061_17f1eb2b\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_RunGame.exe_4b3666d9d476bc73f9231f3895155d123532e_4dfbf061_0aba3e9a",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_RunGame.exe_4b3666d9d476bc73f9231f3895155d123532e_4dfbf061_0aba3e9a\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_RunGame.exe_41d25ef31fb8b7f36a9520329af1821a1ab50e5_4dfbf061_185e1325",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_RunGame.exe_41d25ef31fb8b7f36a9520329af1821a1ab50e5_4dfbf061_185e1325\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER122C.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER122C.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER12CA.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER12CA.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3563.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3563.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3AB3.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3AB3.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3B41.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3B41.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERCBD.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERCBD.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERDFE0.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERDFE0.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE58F.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE58F.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE64B.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE64B.tmp.xml",
                "C:\\Users\\user\\AppData\\Local\\DBG",
                "C:\\Users\\user\\AppData\\Local\\Microsoft\\CLR_v4.0_32\\UsageLogs\\AppLaunch.exe.log",
                "C:\\Users\\user\\AppData\\Local\\Yandex",
                "C:\\Users\\user\\AppData\\Local\\Yandex\\YaAddon",
                "C:\\Users\\user\\AppData\\Roaming",
                "C:\\Windows\\AppCompat\\Programs\\Amcache.hve.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_software.exe_6c6f1611ea6228f3565c493d29da453d6fa23ba_8cb75e93_161e74af",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_software.exe_6c6f1611ea6228f3565c493d29da453d6fa23ba_8cb75e93_161e74af\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_software.exe_e4bdf4e37be47550bf65bd9e5858ffd21db4b_8cb75e93_169e18f3",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportArchive\\AppCrash_software.exe_e4bdf4e37be47550bf65bd9e5858ffd21db4b_8cb75e93_169e18f3\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_software.exe_f1784d283516c80aa509045ecc0eabbf534f91_8cb75e93_159e44d5",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_software.exe_f1784d283516c80aa509045ecc0eabbf534f91_8cb75e93_159e44d5\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1182.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1182.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER39F8.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER39F8.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER418A.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER418A.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4322.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4322.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6454.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6454.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6B1B.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6B1B.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6CB2.tmp"
            ],
            "files_deleted": [
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WEREFDE.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERFEE3.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE7EF.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF628.tmp.WERInternalMetadata.xml",
                "C:\\Windows\\System32\\spp\\store\\2.0\\cache\\cache.dat",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER22F4.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER2383.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER2393.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3DFE.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3E02.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3E12.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_Microsoft.Window_dc7c09b5b22497c66105fec8cf4793ba24a83ae_676f2386_02e510af",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\ReportQueue\\AppCrash_Microsoft.Window_dc7c09b5b22497c66105fec8cf4793ba24a83ae_676f2386_02e510af\\Report.wer",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER122C.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER122C.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER12CA.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER12CA.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3563.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3563.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3AB3.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3AB3.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3B41.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER3B41.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4DA1.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4DB2.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7A31.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7A42.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA29B.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA2AC.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERCBD.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERCBD.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERDFE0.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERDFE0.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE58F.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE58F.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE64B.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERE64B.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1182.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1182.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER1784.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER17A4.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER39F8.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER39F8.tmp.dmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER418A.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER418A.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4322.tmp",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4322.tmp.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4944.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER4965.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER6454.tmp"
            ],
            "command_executions": [
                "\"%windir%\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe\"",
                "\"%SAMPLEPATH%\\RunGame.exe\" ",
                "\"C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe\"",
                "C:\\Windows\\SysWOW64\\WerFault.exe -u -p 3940 -s 380",
                "\"%SAMPLEPATH%\\1514
                    b6e4de20f86adf92c6814dce3100c08459d63249a22a3280e1df80b1569a.exe\" ",
                "C:\\Windows\\SysWOW64\\WerFault.exe -u -p 3300 -s 380",
                "C:\\Windows\\System32\\wuapihost.exe -Embedding",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe",
                "C:\\Users\\Virtual\\AppData\\Local\\Temp\\1514
                    b6e4de20f86adf92c6814dce3100c08459d63249a22a3280e1df80b1569a.exe",
                "C:\\Windows\\SysWOW64\\WerFault.exe -u -p 1136 -s 316",
                "\"C:\\Users\\RDhJ0CNFevzX\\Desktop\\RunGame.exe\" ",
                "C:\\Windows\\SysWOW64\\WerFault.exe -u -p 1136 -s 300",
                "C:\\Windows\\SysWOW64\\WerFault.exe -u -p 1136 -s 328"
            ],
            "files_attribute_changed": [],
            "processes_terminated": [
                "%windir%\\System32\\svchost.exe -k WerSvcGroup",
                "wmiadap.exe /F /T /R",
                "\"%windir%\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe\"",
                "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe",
                "C:\\Windows\\System32\\wuapihost.exe",
                "c:\\windows\\syswow64\\werfault.exe",
                "c:\\windows\\microsoft.net\\framework\\v4.0.30319\\applaunch.exe",
                "c:\\users\\rdhj0cnfevzx\\desktop\\rungame.exe",
                "2588 - \"C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe
                    \"",
                "2776 - C:\\Windows\\SysWOW64\\WerFault.exe -u -p 2588 -s 424",
                "2768 - C:\\Windows\\SysWOW64\\WerFault.exe -u -p 2372 -s 392"
            ],
            "processes_killed": [],
            "processes_injected": [
                "\\\\?\\C:\\Windows\\system32\\wbem\\WMIADAP.EXE"
            ],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "Queries process information (via WMI, Win32_Process)",
                "Queries sensitive disk information (via WMI, Win32_DiskDrive, often done to
                    detect virtual machines)",
                "Checks if Antivirus program is installed (via WMI)",
                "Queries sensitive video device information (via WMI, Win32_VideoController,
                    often done to detect virtual machines)",
                "Queries sensitive processor information (via WMI, Win32_Processor, often
                    done to detect virtual machines)",
                "Allocates memory in foreign processes",
                "Injects a PE file into a foreign processes",
                "Writes to foreign memory regions",
                "Creates a process in suspended mode (likely to inject code)",
                "Spawns processes",
                "Creates files inside the user directory",
                "Creates files inside the system directory",
                "Creates guard pages, often used to prevent reverse engineering and debugging
                    ",
                "Contains long sleeps (>= 3 min)",
                "May sleep (evasive loops) to hinder dynamic analysis",
                "Checks if the current process is being debugged",
                "Queries sensitive disk information (via WMI, Win32_DiskDrive, often done to
                    detect virtual machines)",
                "Queries sensitive video device information (via WMI, Win32_VideoController, often done to detect virtual machines)",
                "Queries sensitive processor information (via WMI, Win32_Processor, often done to detect virtual machines)",
                "Binary may include packed or crypted data",
                "PE file has section (not .text) which is very likely to contain packed code (zlib compression ratio < 0.011)",
                "Binary may include packed or crypted data",
                "Tries to harvest and steal browser information (history, passwords, etc)",
                "Checks if the current process is being debugged",
                "Queries sensitive disk information (via WMI, Win32_DiskDrive, often done to detect virtual machines)",
                "Checks if Antivirus program is installed (via WMI)",
                "Queries sensitive video device information (via WMI, Win32_VideoController, often done to detect virtual machines)",
                "Queries sensitive processor information (via WMI, Win32_Processor, often done to detect virtual machines)",
                "Tries to detect virtualization through RDTSC time measurements",
                "Queries a list of all running processes",
                "Sample monitors Window changes (e.g. starting applications), analyze the sample with the simulation cookbook",
                "Reads the hosts file",
                "Queries process information (via WMI, Win32_Process)",
                "Queries the volume information (name, serial number etc) of a device",
                "Queries the cryptographic machine GUID",
                "Reads software policies",
                "Queries sensitive disk information (via WMI, Win32_DiskDrive, often done to detect virtual machines)",
                "Tries to detect virtualization through RDTSC time measurements",
                "Tries to harvest and steal browser information (history, passwords, etc)",
                "Detected TCP or UDP traffic on non-standard ports"
            ]
        }
```

M.15. AkiraRansomware

M.15.1. Dyanmic Analysis Features

```

    {
        "Behavior": {
            "files_opened": [
                "\\PSHost.133281746551022938.812.DefaultAppDomain.powershell",
                "\\PSHost.133282719736797001.3952.DefaultAppDomain.powershell",
                "\\PSHost.133285824068170416.3588.DefaultAppDomain.powershell",
                "\\PSHost.133286932881007403.4060.DefaultAppDomain.powershell",
                "\\PSHost.133289086137316116.2424.DefaultAppDomain.powershell",
                "\\PSHost.133293195520062876.3140.DefaultAppDomain.powershell",
                "\\PSHost.133294552943440169.4088.DefaultAppDomain.powershell",
                "\\PSHost.133300645080595166.3808.DefaultAppDomain.powershell",
                "\\PSHost.133305626681675359.2564.DefaultAppDomain.powershell",
                "\\PSHost.133319168554649880.4056.DefaultAppDomain.powershell",
                "\\PSHost.133346664075337154.812.DefaultAppDomain.powershell",
                "\\PSHost.133348970128827419.2380.DefaultAppDomain.powershell",
                "\\PSHost.133414652484119965.3748.DefaultAppDomain.powershell",
                "\\PSHost.133417643899982848.3144.DefaultAppDomain.powershell",
                "\\PSHost.133424622098157219.3636.DefaultAppDomain.powershell",
                "\\PSHost.133459508022249821.3268.DefaultAppDomain.powershell",
                "\\PSHost.133510421520694168.2988.DefaultAppDomain.powershell",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\infopath.en-us\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\infopath.en-us\\infopathmui.xml",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\classlist",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\charsets.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\calendars.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\amd64\\jvm.cfg",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\dcf.en-us\\dcfmui.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\excel.en-us\\excelmui.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\groove.en-us\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\groove.en-us\\groovemui.xml",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_it.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_fr.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_ko.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_ja.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\ffjcext.zip",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_es.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_de.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_sv.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\deploy\\messages_pt_br.properties",
                "c:\\msocache\\all users\\{90160000-0011-0000-0000-0000000ff1ce}-c\\proplusww.xml",
                "c:\\msocache\\all users\\{90160000-0011-0000-0000-0000000ff1ce}-c\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\msoidclil.dll",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\excel.en-us\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\aceexch.dll",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\1033\\readme.htm",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\acecore.dll",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\localedata.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\jfxrt.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\jaccess.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\dnsns.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\sunjce_provider.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\ext\\sunec.jar"
            ],
            "files_written": [
                "c:\\program files\\windowsapps\\microsoft.net.native.framework.1.0_1.0.22929.0_x64__8wekyb3d8bbwe\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.net.native.framework.1.0_1.0.22929.0_x64__8wekyb3d8bbwe\\appxmetadata\\akira_readme.txt",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\infopath.en-us\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office setup controller\\infopath.en-us\\infopathmui.xml",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\classlist",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\charsets.jar",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\calendars.properties",
                "c:\\program files\\java\\jre1.8.0_171\\lib\\amd64\\jvm.cfg",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0_x64__8wekyb3d8bbwe\\html\\lv-lv\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0_x64__8wekyb3d8bbwe\\html\\hi-in\\resources\\css\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsmaps_4.1509.50911.0_x64__8wekyb3d8bbwe\\assets\\secondarytiles\\directions\\transit\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsmaps_4.1509.50911.0_x64__8wekyb3d8bbwe\\assets\\secondarytiles\\directions\\driving\\contrast-white\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-0044-0409-0000-0000000ff1ce}-c\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-0090-0409-0000-0000000ff1ce}-c\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-002c-0409-0000-0000000ff1ce}-c\\proof.fr\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-002c-0409-0000-0000000ff1ce}-c\\proof.en\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-002c-0409-0000-0000000ff1ce}-c\\proof.es\\akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-001b-0409-0000-0000000ff1ce}-c\\
                    akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-002c-0409-0000-0000000ff1ce}-c\\
                    akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-00a1-0409-0000-0000000ff1ce}-c\\
                    akira_readme.txt",
                "c:\\msocache\\all users\\{90160000-00ba-0409-0000-0000000ff1ce}-c\\
                    akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\gameplayassets\\tripeaks\\respacks
                    \\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\gameplayassets\\tripeaks\\
                    akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\gameplayassets\\theme\\akira_readme
                    .txt",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0
                    _x64__8wekyb3d8bbwe\\html\\hi-in\\deeplink\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0
                    _x64__8wekyb3d8bbwe\\html\\hi-in\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.zunemusic_3.6.13251.0
                    _x64__8wekyb3d8bbwe\\skipmerge\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.zunemusic_3.6.13251.0
                    _x64__8wekyb3d8bbwe\\styles\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.bingfinance_4.6.169.0
                    _x86__8wekyb3d8bbwe\\jsonresources\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.bingfinance_4.6.169.0
                    _x86__8wekyb3d8bbwe\\configuration\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowscommunicationsapps_17
                    .6308.42271.0_x64__8wekyb3d8bbwe\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowscamera_2015.1071.40.0
                    _x64__8wekyb3d8bbwe\\microsoft.system.package.metadata\\akira_readme.txt
                    ",
                "c:\\program files\\windowsapps\\microsoft.windowscamera_2015.1071.40.0
                    _x64__8wekyb3d8bbwe\\_resources\\akira_readme.txt",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office
                    setup controller\\dcf.en-us\\dcfmui.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office
                    setup controller\\excel.en-us\\excelmui.xml",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\microsoft.system.package.metadata\\akira_readme.txt
                    ",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\microsoft.msn.shell\\themes\\glyphs\\akira_readme.
                    txt",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\microsoft.msn.shell\\themes\\glyphs\\font\\
                    akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\microsoft.msn.shell\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\microsoft.msn.shell\\themes\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.bingsports_4.6.169.0
                    _x86__8wekyb3d8bbwe\\msadvertisingjs\\akira_readme.txt",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office
                    setup controller\\groove.en-us\\setup.xml",
                "c:\\program files (x86)\\common files\\microsoft shared\\office16\\office
                    setup controller\\groove.en-us\\groovemui.xml",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\buttons\\deal\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\buttons\\back\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3
                    .3.9211.0_x64__8wekyb3d8bbwe\\assets\\buttons\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0
                    _x64__8wekyb3d8bbwe\\html\\sl-si\\deeplink\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.windowsphone_10.1510.9010.0
                    _x64__8wekyb3d8bbwe\\html\\sl-si\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.messaging_1.10.22012.0
                    _x86__8wekyb3d8bbwe\\skypeapp\\view\\akira_readme.txt",
                "c:\\program files\\windowsapps\\microsoft.microsoftsolitairecollection_3.3.9211.0_x64__8wekyb3d8bbwe\\environmentintegration\\content\\audio\\akira_readme.txt"
            ],
            "files_deleted": [
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_wys3elej.cj5.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_edm40khs.wxg.psm1",
                "%USERPROFILE%",
                "%USERPROFILE%\\AppData\\Local\\akira_readme.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_sgodrupw.tgl.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_4itcgsay.sdt.psm1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_q2uvauzr.wsv.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_bfczmqkb.lis.psm1",
                "C:\\Windows\\System32\\spp\\store\\2.0\\cache\\cache.dat",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7EEF.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7F0F.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7FCA.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7FEB.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7FCB.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7FEC.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_ktw0sxk3.poi.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_matt0tpn.lke.psm1",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER9BCD.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA004.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA014.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA227.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA228.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA239.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_5yo5xgkn.uif.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_z45oeu5g.s1l.psm1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_wdofsrzu.s1s.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_ljok5cdt.q42.psm1",
                "%USERPROFILE%\\AppData\\Local\\Microsoft\\CLR_v4.0\\UsageLogs\\powershell.exe.log",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER8F1B.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER90A2.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER90B3.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_wxebrkwj.ze2.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_j1jj5zfz.wuw.psm1",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER73D3.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER748F.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER749F.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_hmhrdvgt.1ir.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_oyqfkjo1.nwc.psm1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_azesfn13.3h3.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_cltupn5y.pbn.psm1",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7A3C.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7B45.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER7B56.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_xzxhtyn1.uue.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_rkpzszdz.anv.psm1",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER8DB4.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER8FB8.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER8FD8.tmp.txt",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_b5jgzz15.stx.ps1",
                "%USERPROFILE%\\AppData\\Local\\Temp\\__PSScriptPolicyTest_kztktwwj.aot.psm1"
            ],
            "command_executions": [
                "\"%SAMPLEPATH%\\win_locker.exe\" ",
                "powershell.exe -Command \"Get-WmiObject Win32_Shadowcopy | Remove-WmiObject
                    \"",
                "\"%SAMPLEPATH%\\8631
                    ac37f605daacf47095955837ec5abbd5e98c540ffd58bb9bf873b1685a50.exe\" ",
                "C:\\Windows\\System32\\wuapihost.exe -Embedding"
            ],
            "files_attribute_changed": [],
            "processes_terminated": [
                "%windir%\\System32\\svchost.exe -k WerSvcGroup",
                "\"pwsh.exe\" -Command Get-WmiObject Win32_Shadowcopy | Remove-WmiObject",
                "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
                "C:\\Windows\\System32\\conhost.exe",
                "C:\\Windows\\SystemApps\\ShellExperienceHost_cw5n1h2txyewy\\
                    ShellExperienceHost.exe",
                "C:\\Windows\\System32\\wuapihost.exe"
            ],
            "processes_killed": [],
            "processes_injected": [],
            "services_opened": [],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [],
            "mitre_attack_techniques": [
                "contain obfuscated stackstrings",
                "encode data using Base64",
                "parse PE header",
                "get system information on Windows",
                "create new key via CryptAcquireContext",
                "encrypt or decrypt via WinCrypt",
                "enumerate processes on remote desktop session host",
                "connect to WMI namespace via WbemLocator",
                "get disk information",
                "get geographical location",
                "link function at runtime on Windows",
                "(Process #1) win_locker.exe modifies the content of multiple user files.",
                "(Process #1) win_locker.exe resolves 21 API functions by name.",
                "(Process #1) win_locker.exe changes the appearance of folder \"C:\\Program
                    Files\\Common Files\\microsoft shared\\Stationery\".",
                "(Process #4) wmiprvse.exe starts (process #6) powershell.exe with a hidden
                    window.",
                "The process attempted to dynamically load a malicious function",
                "The process tried to load dynamically one or more functions.",
                "The process has tried to detect the debugger probing the use of page guards
                    .",
                "The process attempted to detect a running debugger using common APIs",
                "It Tries to detect injection methods",
                "Queries process information (via WMI, Win32_Process)",
                "Creates processes via WMI",
                "Installs a chrome extension",
                "Stores files to the Windows startup directory",
                "Contains long sleeps (>= 3 min)",
                "Contains medium sleeps (>= 30s)",
                "May sleep (evasive loops) to hinder dynamic analysis",
                "Tries to harvest and steal browser information (history, passwords, etc)",
                "AV process strings found (often used to terminate AV products)",
                "Queries a list of all running processes",
                "Sample monitors Window changes (e.g. starting applications), analyze the
                    sample with the simulation cookbook",
                "Enumerates the file system",
                "Queries process information (via WMI, Win32_Process)",
                "Queries the volume information (name, serial number etc) of a device",
                "Reads software policies",
                "Public key (encryption) found",
                "Installs a chrome extension",
                "Tries to harvest and steal browser information (history, passwords, etc)",
                "Downloads files",
                "Found Tor onion address",
                "Writes a notice file (html or txt) to demand a ransom"
            ]
        }
    }
```

M.16. SundownEK

M.16.1. Dyanmic Analysis Features

```

        "Behavior": {
            "files_opened": [
                "C:\\WINDOWS\\system32\\winime32.dll",
                "C:\\WINDOWS\\system32\\ws2_32.dll",
                "C:\\WINDOWS\\system32\\ws2help.dll",
                "C:\\WINDOWS\\system32\\psapi.dll",
                "C:\\WINDOWS\\system32\\imm32.dll",
                "C:\\WINDOWS\\system32\\lpk.dll",
                "C:\\WINDOWS\\system32\\usp10.dll",
                "C:\\WINDOWS\\WinSxS\\x86_Microsoft.Windows.Common-
                    Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\\comctl32.dll",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996
                    E.exe",
                "C:\\WINDOWS\\system32\\shell32.dll",
                "C:\\WINDOWS\\WindowsShell.Manifest",
                "C:\\WINDOWS\\system32\\rpcss.dll",
                "C:\\WINDOWS\\system32\\MSCTF.dll",
                "C:\\WINDOWS\\system32\\shfolder.dll",
                "C:\\WINDOWS\\system32\\setupapi.dll",
                "C:\\DiskD",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsn2.tmp",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsz4.tmp",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsz4.tmp\\
                    Samian.dll",
                "C:\\Documents and Settings\\Administrator\\Application Data\\Read Me Info.
                    txt",
                "C:\\Documents and Settings\\Administrator\\Application Data\\jackajeoaer.jpg
                    ",
                "C:\\WINDOWS\\system32\\iphlpapi.dll",
                "C:\\WINDOWS\\system32\\urlmon.dll",
                "C:\\WINDOWS\\system32\\wininet.dll",
                "C:\\WINDOWS\\system32\\MSCTFIME.IME",
                "C:\\WINDOWS\\system32\\uxtheme.dll",
                "C:\\WINDOWS\\system32\\MSIMTF.dll",
                "C:\\WINDOWS\\system32\\faultrep.dll",
                "C:\\WINDOWS\\system32\\winsta.dll",
                "C:\\WINDOWS\\system32\\wtsapi32.dll",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsc2.tmp",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsj4.tmp",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsj4.tmp\\
                    Samian.dll",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nss2.tmp",
                "C:\\WINDOWS\\system32\\mprapi.dll",
                "C:\\WINDOWS\\system32\\activeds.dll",
                "C:\\WINDOWS\\system32\\adsldpc.dll",
                "C:\\WINDOWS\\system32\\atl.dll",
                "C:\\WINDOWS\\system32\\rtutils.dll",
                "C:\\WINDOWS\\system32\\samlib.dll",
                "C:\\ea67e5a16d58cd3c6c6abc1213f55caaf4a109c8f5583ad9da9df2e803be0d7e",
                "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsv2.tmp",
                "jackajeoaer.jpg",
                "C:\\Documents and Settings\\<USER>\\Application Data\\Read Me Info.txt",
                "\\\\.\\PIPE\\lsarpc",
                "\\\\.\\MountPointManager",
                "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsv1.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\bin.exe",
            "C:\\Documents and Settings\\<USER>\\Application Data\\tag(2)",
            "C:\\Documents and Settings\\<USER>\\Application Data\\jackajeoaer.jpg"
        ],
        "files_written": [
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsc3.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\bin.exe",
            "C:\\Documents and Settings\\Administrator\\Application Data\\tag(2)",
            "C:\\Documents and Settings\\Administrator\\Application Data\\Read Me Info.
                txt",
            "C:\\Documents and Settings\\Administrator\\Application Data\\jackajeoaer.jpg
                ",
            "C:\\Documents and Settings\\Administrator\\Application Data\\index(11).php",
            "C:\\Documents and Settings\\Administrator\\Application Data\\photo(5).jpg",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsz4.tmp\\
                Samian.dll",
            "C:\\WINDOWS\\wininit.ini",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsj4.tmp\\
                Samian.dll",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsx3.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsv2.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\bin.exe",
            "C:\\Documents and Settings\\<USER>\\Application Data\\tag(2)",
            "C:\\Documents and Settings\\<USER>\\Application Data\\Read Me Info.txt",
            "C:\\Documents and Settings\\<USER>\\Application Data\\jackajeoaer.jpg",
            "C:\\Documents and Settings\\<USER>\\Application Data\\index(11).php",
            "C:\\Documents and Settings\\<USER>\\Application Data\\photo(5).jpg",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsa3.tmp\\Samian.dll"
        ],
        "files_deleted": [
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsn2.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsc3.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsz4.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsz4.tmp\\
                Samian.dll",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsc2.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsj4.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsj4.tmp\\
                Samian.dll",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nss2.tmp",
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\nsx3.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsv1.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsa3.tmp",
            "C:\\DOCUME~1\\<USER>~1\\LOCALS~1\\Temp\\nsa3.tmp\\Samian.dll"
        ],
        "command_executions": [
            "\"C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6
                \\996E.exe\""
        ],
        "files_attribute_changed": [],
        "processes_terminated": [
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996
                E.exe"
        ],
        "processes_killed": [],
        "processes_injected": [
            "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996
                E.exe",
            "ea67e5a16d58cd3c6c6abc1213f55caaf4a109c8f5583ad9da9df2e803be0d7e"
        ],
        "services_opened": [
            "RemoteAccess",
            "Router"
        ],
        "services_created": [],
        "services_started": [],
        "services_stopped": [],
        "services_deleted": [],
        "windows_searched": [],
        "registry_keys_deleted": [
            "\\REGISTRY\\MACHINE\\SOFTWARE\\Microsoft\\PCHealth\\ErrorReporting\\DW\\",
                "\\REGISTRY\\MACHINE\\SOFTWARE\\Microsoft\\PCHealth\\ErrorReporting\\DW\\
                    DWFileTreeRoot"
            ],
            "mitre_attack_techniques": []
        }
    }
```

M.17. Kronos

M.17.1. Dyanmic Analysis Features

```
        "Behavior": {
            "files_opened": [
                "C:\\Users\\Admin\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\vxmlbk3x.
                    default\\user.js",
                "C:\\Users\\Admin\\Desktop\\Google Chrome.lnk",
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Internet Explorer\\Quick
                    Launch\\User Pinned\\TaskBar\\Google Chrome.lnk",
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs
                    \\Google Chrome.lnk",
                "C:\\WINDOWS\\system32\\winime32.dll",
                "C:\\WINDOWS\\system32\\ws2_32.dll",
                "C:\\WINDOWS\\system32\\ws2help.dll",
                "C:\\WINDOWS\\system32\\psapi.dll",
                "C:\\WINDOWS\\system32\\imm32.dll",
                "C:\\WINDOWS\\system32\\lpk.dll",
                "C:\\WINDOWS\\system32\\usp10.dll",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996
                    E.exe",
                "C:\\WINDOWS\\system32\\shell32.dll",
                "C:\\WINDOWS\\WinSxS\\x86_Microsoft.Windows.Common-
                    Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\\comctl32.dll",
                "C:\\WINDOWS\\WindowsShell.Manifest",
                "C:\\WINDOWS\\system32\\comctl32.dll",
                "C:\\WINDOWS\\system32\\rpcss.dll",
                "C:\\WINDOWS\\system32\\MSCTF.dll",
                "C:\\WINDOWS\\system32\\clbcatq.dll",
                "C:\\WINDOWS\\system32\\comres.dll",
                "C:\\WINDOWS\\Registration\\R000000000007.clb",
                "C:\\Documents and Settings\\Administrator\\\u684c\u9762\\\u817e\u8bafQQ.lnk
                    ",
                "C:\\WINDOWS\\system32\\linkinfo.dll",
                "C:\\WINDOWS\\system32\\ntshrui.dll",
                "C:\\WINDOWS\\system32\\atl.dll",
                "C:\\WINDOWS\\system32\\setupapi.dll",
                "C:\\DiskD",
                "C:\\Documents and Settings\\All Users\\\u684c\u9762\\Adobe Reader 9.lnk",
                "C:\\Documents and Settings\\All Users\\\u684c\u9762\\\u62db\u884c\u4e13\
                    u4e1a\u7248.lnk",
                "C:\\Documents and Settings\\All Users\\\u684c\u9762\\\u641c\u72d7\u9ad8\
                    u901f\u6d4f\u89c8\u5668.lnk",
                "C:\\WINDOWS\\system32\\svchost.exe",
                "C:\\WINDOWS\\system32\\apphelp.dll",
                "C:\\WINDOWS\\AppPatch\\sysmain.sdb",
                "C:\\WINDOWS\\system32\\ntdll.dll",
                "C:\\WINDOWS\\system32\\kernel32.dll",
                "C:\\WINDOWS\\system32\\unicode.nls",
                "C:\\WINDOWS\\system32\\locale.nls",
                "C:\\WINDOWS\\system32\\sorttbls.nls",
                "C:\\WINDOWS\\system32\\advapi32.dll",
                "C:\\WINDOWS\\system32\\rpcrt4.dll",
                "C:\\WINDOWS\\system32\\secur32.dll",
                "C:\\WINDOWS\\system32\\shimeng.dll",
                "C:\\WINDOWS\\AppPatch\\AcGenral.dll",
                "C:\\WINDOWS\\system32\\user32.dll",
                "C:\\WINDOWS\\system32\\gdi32.dll",
                "C:\\WINDOWS\\system32\\winmm.dll",
                "C:\\WINDOWS\\system32\\ole32.dll",
                "C:\\WINDOWS\\system32\\msvcrt.dll",
                "C:\\WINDOWS\\system32\\oleaut32.dll",
                "C:\\WINDOWS\\system32\\msacm32.dll"
            ],
            "files_written": [
                "C:\\Users\\Admin\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\vxmlbk3x.default\\user.js",
                "C:\\Users\\Admin\\Desktop\\Google Chrome.lnk",
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Internet Explorer\\Quick Launch\\User Pinned\\TaskBar\\Google Chrome.lnk",
                "C:\\Users\\Admin\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Google Chrome.lnk",
                "C:\\Documents and Settings\\Administrator\\Application Data\\Microsoft\\{44EC26F7-0313-49A4-A379-385505289087}\\71d77268.exe",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temporary Internet Files\\Content.IE5\\C1OS62RY\\connect[1].php",
                "C:\\Users\\<USER>\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles/zh35t979.default-release\\user.js",
                "C:\\Users\\<USER>\\AppData\\Roaming\\Microsoft\\Internet Explorer\\Quick Launch\\User Pinned\\TaskBar\\Google Chrome.lnk",
                "C:\\Users\\<USER>\\Desktop\\Google Chrome.lnk",
                "C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\Caches",
                "C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\History",
                "C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\INetCache",
                "C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\INetCookies",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{F35B4841-C554-4AE3-8110-67B8F528E7E6}",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{F35B4841-C554-4AE3-8110-67B8F528E7E6}\\bac58a5f.exe",
                "C:\\Users\\user\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\05rdgbi2.default-release\\user.js",
                "C:\\Users\\user\\Desktop\\Google Chrome.lnk",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{7A779156-93BD-4CC1-9B72-9B3E7E59022F}",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{7A779156-93BD-4CC1-9B72-9B3E7E59022F}\\bac58a5f.exe",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{46695B5A-CDF6-4BD4-8CE3-CE87E0EBF6EB}",
                "C:\\Users\\user\\AppData\\Roaming\\Microsoft\\{46695B5A-CDF6-4BD4-8CE3-CE87E0EBF6EB}\\bac58a5f.exe"
            ],
            "files_deleted": [
                "%USERPROFILE%\\Desktop\\Google Chrome.lnk",
                "%USERPROFILE%\\AppData\\Local\\Microsoft\\Windows\\Caches\\{3DA71D5A-20CC-432F-A115-DFE92379E91F}.3.ver0x0000000000000018.db",
                "C:\\Windows\\ServiceProfiles\\LocalService\\AppData\\Roaming\\Microsoft\\Crypto\\Keys\\de7cf8a7901d2ad13e5c67c29e5d1662_cbbb49d6-b7ff-44ca-aba5-8a5e250d4d42",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERA3C.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERB45.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERBA4.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERBB4.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERBB5.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERBC5.tmp.txt",
                "C:\\Windows\\System32\\spp\\store\\2.0\\cache\\cache.dat",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF26E.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF339.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WERF379.tmp.txt",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER23E.tmp.WERInternalMetadata.xml",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER24F.tmp.csv",
                "C:\\ProgramData\\Microsoft\\Windows\\WER\\Temp\\WER25F.tmp.txt",
                "C:\\aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc:Zone.Identifier",
                "C:\\Users\\<USER>\\Downloads\\aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe:Zone.Identifier",
                "C:\\Users\\Public\\Desktop\\Google Chrome.lnk",
                "C:\\Users\\user\\Desktop\\executable.exe:Zone.Identifier",
                "C:\\Users\\user\\Desktop\\software.exe:Zone.Identifier",
                "C:\\Users\\user\\Desktop\\program.exe:Zone.Identifier"
            ],
            "command_executions": [
                "\"${SamplePath}\\
                    aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe\" ",
                "\"C:\\Windows\\system32\\svchost.exe\"",
                "\"%SAMPLEPATH%\\
                    aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe\" ",
                "C:\\Windows\\System32\\wuapihost.exe -Embedding",
                "\"C:\\WINDOWS\\system32\\svchost.exe\""
            ],
            "files_attribute_changed": [
                "C:\\Documents and Settings\\Administrator\\Application Data\\Microsoft\\{44
                    EC26F7-0313-49A4-A379-385505289087}\\71d77268.exe"
            ],
            "processes_terminated": [
                "${SamplePath}\\
                    aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe",
                "C:\\Windows\\SysWOW64\\svchost.exe",
                "%SAMPLEPATH%\\
                    aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe",
                "C:\\Windows\\System32\\wuapihost.exe",
                "C:\\Documents and Settings\\Administrator\\Local Settings\\Temp\\EB93A6\\996
                    E.exe",
                "aad98f57ce0d2d2bb1494d82157d07e1f80fb6ee02dd5f95cd6a1a2dc40141bc.exe"
            ],
            "processes_killed": [],
            "processes_injected": [
                "C:\\Program Files\\Internet Explorer\\IEXPLORE.EXE",
                "C:\\WINDOWS\\system32\\VBoxService.exe",
                "C:\\WINDOWS\\system32\\svchost.exe"
            ],
            "services_opened": [
                "RASMAN"
            ],
            "services_created": [],
            "services_started": [],
            "services_stopped": [],
            "services_deleted": [],
            "windows_searched": [],
            "registry_keys_deleted": [
                "\\REGISTRY\\USER\\S-1-5-21-1482476501-1645522239-1417001333-500\\Software\\
                    Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ProxyServer",
                "\\REGISTRY\\USER\\S-1-5-21-1482476501-1645522239-1417001333-500\\Software\\
                    Microsoft\\Windows\\CurrentVersion\\Internet Settings\\ProxyOverride",
                "\\REGISTRY\\USER\\S-1-5-21-1482476501-1645522239-1417001333-500\\Software\\
                    Microsoft\\Windows\\CurrentVersion\\Internet Settings\\AutoConfigURL"
            ],
            "mitre_attack_techniques": [
                "encode data using XOR",
                "access PEB ldr_data",
                "encrypt data using blowfish",
                "acquire debug privileges",
                "set file attributes",
                "write process memory",
                "get system information on Windows",
                "link function at runtime on Windows",
                "encrypt data using RC4 PRGA",
                "get socket status",
                "log keystrokes via polling",
                "reference AES constants",
                "get common file path",
                "enumerate files on windows",
                "enumerate files recursively",
                "query environment variable",
                "get file size",
                "enumerate processes",
                "enumerate processes",
                "modify access privileges",
                "parse PE header",
                "bypass Mark of the Web",
                "check OS version",
                "get socket information",
                "get local IPv4 addresses",
                "interact with driver via control codes",
                "get token membership",
                "persist via Run registry key",
                "check if file exists",
                "reference Base64 string",
                "traverse file",
                "use com interface",
                "create lnk file",
                "create lnk file",
                "Sample uses process hollowing technique",
                "Creates an autostart registry key",
                "Creates autostart registry keys with suspicious names",
                "Maps a DLL or memory area into another process",
                "Sample uses process hollowing technique",
                "System process connects to network (likely due to code injection)",
                "Creates a process in suspended mode (likely to inject code)",
                "Spawns processes",
                "Creates files inside the user directory",
                "May sleep (evasive loops) to hinder dynamic analysis",
                "Contains long sleeps (>= 3 min)",
                "Queries disk information (often used to detect virtual machines)",
                "Hides that the sample has been downloaded from the Internet (zone.identifier)",
                "Tries to harvest and steal browser information (history, passwords, etc)",
                "Monitors registry run keys for changes",
                "Queries disk information (often used to detect virtual machines)"
            ]
        }
        }
    }
```