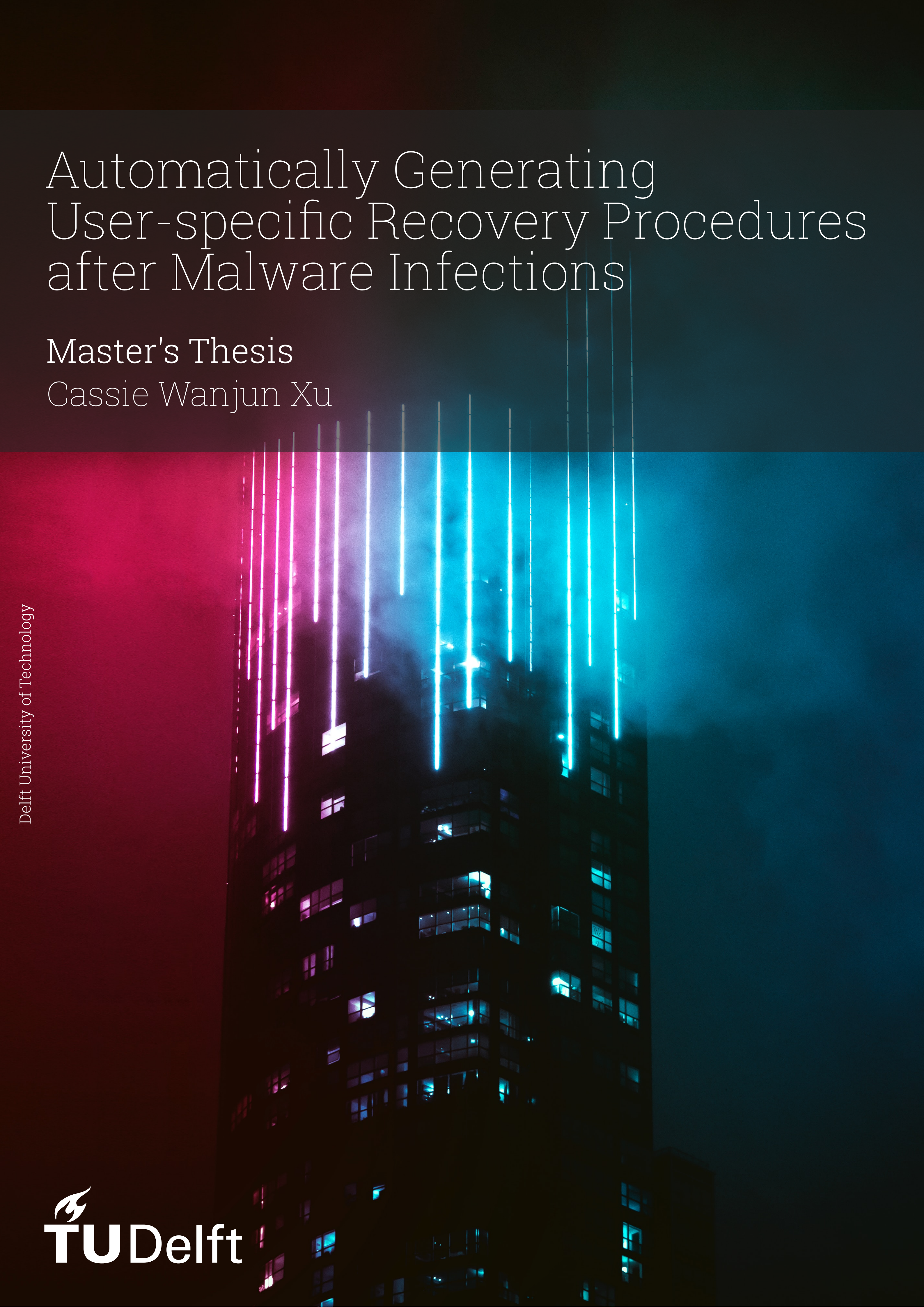
Automatically Generating User-specific Recovery Procedures after Malware Infections

Master's Thesis

Cassie Wanjun Xu



Delft University of Technology

TU Delft

Automatically Generating User-specific Recovery Procedures after Malware Infections

by

Cassie Wanjun Xu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 5, 2023 at 10:00 AM.

Student number:	5263832	
Project duration:	November 2022 – June 2023	
Thesis committee:	Dr. Andrea Continella	University Twente, daily supervisor
	Dr. Sicco Verwer	TU Delft, thesis advisor
	Dr. Thomas Durieux	TU Delft, committee member
	Jerre Starink	University Twente, daily supervisor

Cover image by Victor Rodriguez

Preface

I am very happy and proud to present you with this master's thesis. After one year of following the 4TU cybersecurity program, I realized my interest does not lie in the theoretical and mathematical security fields, but in aspects regarding system security and exploitation. Although studying at a different university, I reached out to Dr. Andrea Continella, whom I am very grateful for accepting me as a thesis student, helping me explore my research interests, and together we formed the idea for this thesis project. After 10 months, I am very satisfied with the result it turned out and surprised by how much I have learned through the process.

Besides Andrea, I would also like to thank my daily supervisor Jerre Starink for all the discussions during our biweekly meetings, always knowledgeable insights about the problems I ran into, detailed feedback for my writing, and encouragement and support through the project. Furthermore, I would like to thank Dr. Sicco Verwer for being my TU Delft supervisor and facilitating all the administrative procedures, and Dr. Thomas Durieux for being part of the thesis committee.

Finally, I am very grateful for all the fellow master students I have met at TU Delft, for all the enjoyable group projects and inspiring discussions, all the support and companionship through the past two years. The completion of this thesis also means the completion of my student life, and I'm very much looking forward to the next chapter in life.

Cassie Wanjun Xu
Den Haag, June 2023

Summary

Malware poses a serious security risk in today's digital environment. The defense against malware mainly relies on proactive detection. However, antivirus products often fail to detect new malware when the signature is not yet available. In the event of a malware infection, the common remediation strategy is reinstalling the system. However, the user loses their personal data, and thus it is not an ideal solution.

The academic works on malware remediation focus on system replay and recovery-oriented computing, which relies on heavy monitoring and is not suitable for a normal user's personal computer. The work from Paleari et al. [31] proposed a remediation methodology that can be used entirely after the infection. They run the malware sample in the sandbox to observe the behavior and generate a revert operation for each action that modifies the system state. However, the limitation of such an approach is unable to deal with the potentially different behaviors in the sandbox and on the real hosts.

In this work, we propose a system that can generate user-specific recovery procedures, without the need of any monitoring in advance. We extend the work from Paleari et al. [31] by combining information from the infected machine. We first extract the environment configuration from the infected computer and configure the same context to the sandbox virtual machine, in order to eliminate the environmental influence on the malware's behavior. After getting the behavior from the sandbox, we combine forensic evidence to understand the exact actions that happened on the system and generate the user-specific recovery procedures.

We implement a prototype based on Windows 10 and CAPE sandbox and perform an evaluation on 894 malware samples. We are able to recover 51.3% of the changes made by malware, which doubles the recovery rate compared to directly matching the sandbox result. Additionally, our experiment result also demonstrates significantly different actual behavior from the user's machine and sandbox result. Our system design maximizes the use of information displayed in the sandbox, but the unshown behavior still leads to the biggest limitation of behavior-based recovery.

Contents

1 Introduction

Malware is no strange word for everyone using any digital device these days. Malware, a short form for ”malicious software”, is a term used to describe any software which is intentionally designed to disrupt the normal functioning of a computer system or gain unauthorized access to information. Malware can take many forms, including viruses, ransomware, trojans, spyware, etc. It can cause damage to the computer system, steal sensitive information, or even render the computer unusable. Malware is often spread through email attachments and downloads from untrusted websites. Over the last decades, the total number of malware infections on the Windows operating system has increased by over 87% [47].

The most common defense against malware is antivirus products installed on users’ computers. The antivirus will scan the downloaded file and alert the user if it considers the file potentially malicious. The technique behind traditional antivirus is signature-based, matching the file hash to a database where all the already-seen malware’s information is stored. However, as empirical study [45] has shown, commercial antivirus cannot provide complete protection from malware: 40% of the malware will not be detected immediately upon download. Further, more than half of all antivirus solutions will allow malware to modify important system configurations and registry keys [45]. The signature-based detection method is impotent at detecting zero-day and polymorphic malware, thus antivirus is not bulletproof for preventing infection. With malware infection being prevalent and only a proactive detection approach cannot provide full protection to the users, the capability to recover to a safe, clean, stable state after the malware infection is also crucial to a secure digital environment. Giffin [14] has named recovering after infection ”the next malware battleground”.

Within the past years, we have seen lots of academic work focused on malware detection, including static analysis, dynamic analysis, hybrid analysis, and memory analysis. Machine learning models also have been frequently leveraged to increase detection accuracy. Meanwhile, the field of malware recovery is relatively less researched and has been mainly focused on recovery-oriented computing (ROC) and untrusted program execution: detailed logs and traces are kept, to further revert and replay to recover the system after intrusion and infection. Such frameworks mainly target servers and enterprise environments. For normal individual users, we consider a fully monitored system excessive for the purpose of potential malware recovery, and the performance burden from it is impractical and unacceptable for the users. We want to explore other recovery possibilities that do not require a full, detailed monitor.

We found interesting works from Passerini et al.[32] [33] [31], where the recovery procedures are automatically generated based on behavior analysis: running the malware sample in the sandbox multiple times, abstract and generalize the behavior, and generate a revert operation on each to recover system resources. However, as also mentioned in these papers, the key limitation of their framework is unable to deal with the potentially different behaviors from the same sample. Therefore, we are going to extend their work, by adding environment profiling and forensic evidence to generate more accurate, user-specific recovery procedures.

1.1. Research Questions

The objective of this research project is to propose and implement a framework, where we are able to generate user-specific recovery procedures after malware infections. As such, we formulate the following research question:

How can we automatically generate user-specific recovery procedures after malware infections on Windows, which are able to revert the malicious changes on system resources with no extra system monitoring in place?

The research topic is further decomposed into the following sub-questions:

1. What are the current malware recovery techniques, and how effective are they?
2. What are the non-deterministic malware behaviors?
3. What are the types of forensic evidence left by malware infections, how can they be useful for malware recovery, and how can we automatically collect and process them?
4. How can we automatically generate recovery procedures, based on behavior analysis and forensic evidence?
5. How effective is our framework? What are the advantages and limitations?

1.2. Contributions

In this work, we propose a novel system for generating user-specific recoveries, without the use of any monitoring in advance and can be used entirely after the infection.

Our main contributions are as follows:

- We perform a literature review on the current malware recovery techniques. And we summarize the reasons and factors of malware's non-deterministic behaviors.
- We design an approach to combine the information from infected computers with sandbox-based recovery, to generate user-specific recovery procedures.
- We design an evaluation methodology to test malware recovery effectiveness, by utilizing system-level in-the-box monitoring and comparing the affected resources list.
- We perform an evaluation of our prototype, which demonstrates its effectiveness. Additionally, our experiment result also demonstrates significantly different behavior from the user's machine and sandbox analysis result.
- We discuss the limitation and further research directions, for our system as well as the general behavior-based recovery field.

1.3. Overview

The rest of this thesis is structured as follows: In Chapter 2 we present an overview of the background landscape and necessary information related to our work. In Chapter 3 we discuss the related works, the problem statement, and the motivation of our research. Next, we give the design of our system in Chapter 4 and the prototype implementation in Chapter 5. We then evaluate our work in Chapter 6, which includes specific case studies and large-scale testing on recovery rate. Afterward, we reflect on the limitations of our work and point to possible future works in Chapter 7. Finally, we provide a conclusion in Chapter 8, including the answers to our research questions.

2 Background

2.1. Malware

Malware, a compound word from “malicious software”, is the type of software that is intentionally designed to disrupt the normal functioning of a computer system or gain unauthorized access to information. Malware is often spread through phishing email attachments or downloaded from suspicious websites. Malware poses serious security and privacy threats to both individual users and organizations. It can steal sensitive information, for monetary, commercial competition, or political purposes. It can also steal computing resources, affect the computer’s performance significantly, or even render it completely unusable.

The amount and sophistication of malware have both been increasing continuously. According to SonicWall’s Cyber Threat Report [43], in the first half year of 2022, 2.8 billion malware hits have been recorded globally, an 11% increase over the data of the year 2021. An average of 1,501 never-seen-before variants were identified every day, and the number of total new variants discovered has skyrocketed by 2,079% since 2018. The financial consequence caused by malware to organizations has also been increasing. According to IBM’s data breach report [17], the average cost of a ransomware breach was $4.54 million in 2022. And Acronis’ cyber threat report has predicted that the damage caused by ransomware globally will exceed $30 Billion by 2023 [1].

Malware can take various forms. Based on the objective the malware is trying to achieve, the followings are the main types of malware:

- **Ransomware** In the case of a ransomware infection, the user’s files on the disk will be encrypted or locked, and they can not have regular access to their files until the ransom is paid. The criminals might also threaten to publish the victims’ data. Ransomware attacks have largely increased in recent years, partially due to the increasing popularity of cryptocurrency [23]: such anonymous payment is perfect for cyber criminals who want to receive money while hiding their identity.
- **Botnet** A botnet, short for “robot network”, is a group of internet-connected devices that are under the control of a malicious party. A botnet will leave command and control (C&C) backdoors on devices, hijack your computing resources, and uses them to perform malicious activities, typically including distributed denial-of-service (DDoS) attacks and sending spam. By using distributed devices instead of one single source, such activities become difficult to detect and block efficiently.
- **Virus** A computer virus is a type of malware that can inject itself into the host computer, replicate itself, and spread from one computer to another. With the metaphor derived from

biological viruses, a computer virus is an "infectious" malware: it can be spread from one infected computer, typically from the local network, and cause more infections to other computers.

- **Backdoor** A backdoor is a type of program that allows attackers to gain unauthorized access to the system without the victim's acknowledgment. This type of malware typically exploits vulnerabilities in legitimate software and establishes its foothold covertly. The attackers can later steal information or computing resources at their will.
- **Spyware** Spyware monitors the activities on the computer, aiming to collect information about the person or the organization. A typical form of spyware is keyloggers, which capture every keystroke you type and can therefore know your passwords, credit card information, etc. Spyware is sometimes also found in benign software when they try to collect excessive information without the user's consent.
- **Adware** Adware is the type of software that installs itself on the system and displays advertisements on the screen, most commonly in a web browser.
- **Trojan** A trojan is a type of malware that disguises itself as legitimate-use software and has malicious purposes hidden underneath. The term is derived from the deceptive Trojan Horse in the ancient Greek story.
- **Rootkit** A rootkit is a type of malware that is able to gain access as a root user (privileged account on Unix-like operating systems). Rootkits are difficult to detect and remove, since they have gained admin privilege, they often have taken measures to prevent detection and removal.

2.2. Malware Analysis

Malware analysis is the process of studying and analyzing malware to understand its behavior and capabilities, and further determine the best strategy to detect and remove the malware. There are two main categories of malware analysis, namely static analysis and dynamic analysis.

Static Analysis

The static analysis approach analyzes the malware by examining the code statically, without running it in any environment. It involves disassembling and decompiling the binary file, reading the code, and trying to understand the malware author's intention behind it, and in this way analyzing the malware's functionality and behavior [42].

Without the need for an environment to actually execute the malware, static analysis is a safe and light approach for malware analysis, and signatures (strings and code fragments from the malware) can be generated for detection. Yara [53] is the widely used pattern-matching tool for static analysis/detection. An example of Yara rules is given in Figure 2.1.

However, malware authors can use packers (to compress the code, and only decompress itself upon execution) and crypters (to cryptographically encrypt or obfuscate the code into a human-unreadable state) to evade static analysis and detection [42]. Tampering with the code readability will directly cause detection systems and human analysts incapable of understanding the program. In this situation, the complete range behavior of the malware can only be exposed in the runtime.

Dynamic Analysis

Dynamic analysis involves running the malware sample in an isolated and controlled environment, such as a sandbox, to observe the malware's behavior [42]. The analyst is able to observe what files the malware interacts with, what network connection is established, what system resources are modified, etc. The capability and impact of the malware are directly shown instead of interpreted from the code. This also allows the analyst to intuitively understand the

```
rule silent_banker : banker
{
    meta:
        description = "This is an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {61 40 68 00 30 00 00 68 14 80 91}
        $b = {83 49 20 25 91 83 70 27 99 60 47 59 87 99}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Figure 2.1: Yara Rules Example [54]

malware's persistence mechanism, process injection, evasion technique, etc. The API calls can be traced during dynamic analysis, and be used for automatic malware detection, commonly with machine learning models [55].

Behavior signatures can be derived from dynamic analysis and written into Sigma rules [38], a widely used rule format to query log data. Such rules related to specific malware families are produced and shared by malware analysts and can be used as indicators of compromise (IoC) in SIEM (security information and event management). An example of Sigma rules is given in Figure 2.2.

However, as running the malware is required, the risk of infecting the computer or the network is also introduced in the meantime. The environment and resources which dynamic analysis needs make it inefficient for large-scale analysis. Furthermore, some malware families are designed to evade the analysis environment: They are able to distinguish a virtual environment from a real computer's environment and then terminate themselves without showing any malicious behavior [55]. In such cases, dynamic analysis gives an incomplete picture of the program's full capabilities.

2.3. Defense against Malware

2.3.1. Detection

The most effective and cost-efficient strategy to defend against malware is being proactive: to detect malware and prevent it from executing on the computer. Antivirus is the type of software that is designed for such a purpose. The core function of an antivirus is to scan all the downloaded files and to check whether the file is known malware. The matching is mostly based on comparing the file hash to its pre-existing malware database. Heuristic comparison is sometimes also used: if part of the file content matches a malicious code snippet, it will also be flagged. The antivirus periodically updates its database to include the latest malware information. Upon detection of suspicious files, users can choose to quarantine or remove the file immediately.

The biggest drawback of antivirus software is its incapability to detect zero-day malware, which means the signature does not exist in the database yet. It is also not effective against obfuscated and polymorphic malware, which can change its code without changing its functionality. Since such malware's signature becomes different and is be included in the database, the

```
title: Execution in Outlook Temp Folder
id: "a018fdc3-46a3-44e5-9afb-2cd4af1d4b39"
status: test
description: Detects a suspicious program execution in Outlook temp
    folder
author: Florian Roth (Nextron Systems)
date: 2019/10/01
modified: 2022/10/09
tags:
    - "attack.initial_access"
    - "attack.t1566.001"
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|contains: '\Temporary Internet Files\Content.Outlook\'
    condition: selection
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Unknown
level: high
```

Figure 2.2: Sigma Rules Example [38]

antivirus will consider it benign.

For environments with higher security requirements, a traditional antivirus is not enough. Endpoint detection and response (EDR) is often used for more wide-range protection. Besides the signature-based approach, behaviors on the computer are being monitored, including running processes, file creation, registry modification, network connections, etc. Suspicious activities are determined by predefined rules set by security analysts or self-learned over time from normal baseline behaviors through artificial intelligence. Given the general public's familiarity with the name "antivirus", some security vendors also use antivirus as an umbrella term for security protection solutions provided to individual users.

As mentioned in the previous section, running samples in a sandbox to observe the behavior is an effective way to detect malware. While the antivirus database does not have the signature of zero-day malware, the sample exhibits malicious behavior in the runtime and the sandbox can use behavioral detection rules to flag the program as highly suspicious. A sandbox can be deployed in the cloud of an enterprise environment to examine suspicious files appearing in the network. However, given the storage space occupied and the time taken to examine one sample, it is rarely used in individual users' antivirus solutions.

Techniques to detect malware have been improved over time, however, malware can not always be detected and prevented from execution. Research on the effectiveness of commercial antivirus software has shown that [45]: Nearly 40% of the malware can not be detected right after appearing on the system, but require days or even weeks to be successfully identified, and on average 20% of the malware still can not be identified by the 30th day. Therefore, purely relying on detection is far away from bulletproof, and reactive defense against malware is also necessary.

2.3.2. Remediation

Malware remediation involves the process of isolating and removing malware from a computer system or network. The goal of malware remediation is to restore the system to a secure state and prevent any further harm from being done by the malware. Depending on the complexity and capability of the malware, the remediation process could be long, complex, and sometimes even impossible.

The most common approach after a malware infection is to completely reinstall the system, in which way the user can make sure his computer is in a clean state without any malware involved. But meanwhile, the user also loses their personal data located on the same drive. Another similar approach is to use previous backups/snapshots. The user will only lose the data from backup time till infection time, instead of losing all the data. The precise infection time is required to guarantee to return to a clean state, otherwise, unactivated malware still remains on the previous backup image. When using such an approach, the user should have regular backups available and needs to roll back to the version before any infection happen. Both of the above two approaches are exchanging user data for the total removal of malware changes.

Another approach is manually examining the system and remediating the infection. This requires human effort and expert knowledge. This typically happens in an enterprise environment and the security experts will contain the malware, isolate the system, and further restore the changes and remain the system's integrity. For home users' computer infections, security organizations also publish reports on how to clean malware. For example, PC Risk publishes detailed removal guides on various malware [36]. Users can follow the guides to kill malicious processes, remove the executable files, clean registry keys, and examine the system's state after cleaning. However, for people without knowledge and experience in technology, such instruction might look intimidating. It is less favorable for the users as it requires manual effort and is time-consuming.

Besides re-installing the system and manual effort, there also exist automatic remediation approaches. Most antivirus products also have the functionality to recover the system upon the detection of malware: they will remove the malware files. However, we want to point out, simply removing the malware files is not a complete remediation. The changes done to a system can leave the system in an unstable state.

Consider this toy example of a piece of malware named mal.exe. It does the following changes to the system:

1. Dropping a child file called child.exe in a system folder.
2. Set up a registry key value to automatically start itself at boot.
3. Modify the DNS resolver in order to hijack network traffic. When the user visits examplebank.com, he connects to a forgery bank website and the credentials will be stolen.
4. Turn off the system services that automatically start at boot. The malware does this by deleting the corresponding registry keys.

By looking into the remediation engine of several antivirus products, we observe that most of them simply remove the malware file and sometimes its child files, instead of performing full remediation on all the changes. Passerini et al. [33] have performed studies on the remediation capability of malware detectors. Their results show that the commercial antivirus products are good at recovering primary changes (eg. dropping a child executable file), but almost no ancillary changes (eg. the modification on the DNS resolver). For our example above, a typical antivirus product would be able to revert the first and second actions but fail to remediate the last two actions. The approach taken by antivirus is quick and light, however, there is no guarantee on the system's integrity and usability after the remediation.

Besides remediation upon the particular malware infection, there also exist system-wise cleaning tools, for example, CCleaner and BleachBit. Such software cleans potentially unwanted files and invalid Windows Registry entries. However, with such tools primarily focusing on system maintenance and cleaning, it does not actively identify malware and is not able to detect and remove stealthy malware configurations.

Within our research, we will focus on the automatic remediation of malware infections. We will further summarize the related work, and discuss our research questions and motivation in Chapter 3.

3 Motivation

In this section, we discuss the academic approaches to malware remediation and point out the limitations of state-of-art techniques. Then, we propose our research topic in detail, identify the goals and challenges, and give some related works of our proposed solution.

3.1. Automatic Malware Recovery

In section 2.3.2 we have discussed the malware remediation procedures used in practice and in commercial tools. In this section, we give a literature review on automatic malware recovery techniques in academic works. They can be categorized into log-based, snapshot-based, and behavior-based.

3.1.1. Log-based

In this category, techniques for malware recovery are equivalent to safely executing untrusted programs, as well as recovery-oriented computing (ROC) [34]. The same as transaction logs used in databases, log information is kept in order to undo unwanted operations or redo valid operations for the purpose of malware recovery.

Hsu et al. [16] proposed a framework that is able to perform automatic malware removal and system repair. They separate programs into trusted and untrusted, monitor and log all untrusted programs' operations, and can use these logs to remove the effect of untrusted programs completely and automatically when needed. However, the user not only needs to differentiate trusted and untrusted programs manually but also needs to give permission to the untrusted programs every time it attempts to write data into the system, which significantly hinders daily usability and is only suitable for high-precision systems with stringent security requirements.

Goel et al. [15] proposed the Taser intrusion recovery system. All operations made to the file system, network connection, and processes are logged. After the IDS detects an intrusion, Taser is able to determine the tainted operation related to the intrusion and only selectively revert the relevant changes, while the legitimate operations can remain in place. However, when legitimate operations depend on tainted operations, conflicts can not be solved effectively and thus affect reliability.

De Oliveira et al. [48] designed ExecRecorder, a VM-based full system replay. They checkpoint the entire system state and also log the uniprocessor architectural non-deterministic events. The mechanism imposes low-performance overhead (4%) but high storage overhead to keep the logs (5.4GB per hour).

Similar ideas have also been presented in [9] [56]. In log-based approaches, detailed monitoring and logging are set in place, which provides references for later recovery. But meanwhile, such mechanisms introduce significant runtime and storage overhead. It is mainly used for systems with high-reliability requirements and relatively low real-time performance requirements, and is not suitable for normal personal computer users.

3.1.2. Snapshot-based

Another common approach for malware recovery is snapshot-based. Instead of monitoring and logging, snapshots are periodically taken and to be rolled-back after infection happens. Typically, virtual machines and hypervisors are present in such systems.

Elbadawi and Al-Shaer [11] designed TimeVM, a framework for intrusion mitigation and recovery. Multiple “shadow” virtual machines are used to represent the virtual machine at different time points. When an infection is discovered, TimeVM is able to pick the most recent uninfected shadow VM and replay the traffic history to this VM. Then an up-to-date uninfected system state can be constructed. TimeVM focuses on restoring legitimate network connections but not data recovery.

Webster et al. [52] proposed a recovery system using the Checkpoint/Restore in Userspace (CRIU) technology in Linux containers. The main contribution of their work is that active TCP connections can remain unaffected during the recovery phase, which maintains service availability.

Similar ideas include [19] [6], where snapshot/checkpoint in VM/containers are taken for further recovery reference. The snapshot-based approaches are upgraded utilization of backups, and try to keep some of the current data in place. However, significant storage space is required to store the snapshots, and it’s yet inevitable to lose (partially) the data generated after the snapshot timestamp.

3.1.3. Behavior-based

Paleari et al. [31] proposed an automatic generation of recovery procedures based on behavior analysis. They run the malware sample in the sandbox multiple times, abstract and generalize the behavior, and generate a revert operation on each action that modifies system resources. The key limitation of their framework, which inherits from the limitation of dynamic analysis, is unable to deal with the potentially different behaviors between sandbox environments and real hosts. The recovery procedures are generated entirely after the infection, but the reliability is lower than the above two approaches, as it has no information about the specific infection and its context.

3.2. Problem Statement

The above-mentioned three categories each have their own advantages and limitations. In log-based approaches, better granularity is provided, and detailed to every single operation can be reverted. However, runtime and performance overhead is inevitable. Manual selection is also sometimes required during the recovery process. In snapshots-based approaches, multiple backup snapshots need to be taken and stored. Significant extra storage is needed. Furthermore, the data generated between the snapshot timestamp and the current often can not be restored completely, but only partially. In the behavior-based approach, no actions are needed before the infection, however, the limitation is the accuracy of the generated recovery procedures. In the case of malware behaving very differently on the infected PC and the sandbox, such recovery procedures could be invalid.

The scope of our research topic is trying to provide better recovery procedures for individual home users. We are looking for an approach that can be used after the infections, without the need for extensive monitoring beforehand. Therefore, we focus on the behavior-based approach and try to extend the current technique to improve the existing limitations.

In our scenarios, the infection has already happened and we have access to the victim's machine, which means valuable information could be retrieved and integrated. We propose the following aspects that will improve the accuracy of recovery procedures:

- **The victim's environment**: Some malware families are environment-sensitive and tend to behave differently under different environments. It is sometimes difficult for the analyst to successfully expose the targeted environment. However, in the case of post-infection, we can use the user's environment as a reference to build the analysis environment.
- **The built-in logging mechanisms in Windows**: Windows operating system, as the same as other modern operating systems, has a lot of internal logging mechanisms to help with troubleshooting and security monitoring. They provide rich resources to understand the past activities that happened in the system. Thus, without an extra monitor setup beforehand, we can utilize the already existing monitoring in Windows to make the recovery procedures more accurate. Such an idea is similar to the case that security experts examining an infected system and investigating forensic artifacts to help the recovery.

In summary, our work aims to automatically generate recovery procedures, based on the behavior displayed in the sandbox, with the integration of environment profiling and forensic artifacts.

3.3. Goals and Challenges

3.3.1. Goals

Our research topic aims to find a solution for the following common scenario: The malware has already infected the system. The antivirus products did not detect the malware successfully in time but only identified it after the infection (due to the signature database getting updated). The user is trying to get rid of the malware.

The goal of this research project is to move further with and improve the current solution. Based on behavioral analysis, we add elements that will help to give more accurate remediation: environment profiling and forensic evidence. By adding new user-specific aspects, we try to present more satisfying and reliable recovery procedures than the present solution.

The main novel difference in our recovery system is that we do not require the user to set any extra things in advance. We directly and only utilize the features that Windows operating system provides and enables by default. This property makes our system practical and useful for post-incident remediation.

Through the research process, we also explore automatic forensic analysis. We discuss the malware forensic evidence left on Windows and propose an approach to automatically collect and analyze them.

Finally, the main difficulty we are solving is the non-deterministic behaviors of malware. Through the evaluation, we provide more empirical results regarding this phenomenon.

3.3.2. Challenges

Automatic generation on malware recovery procedures is a difficult task, due to the increasing complexity and diversity of current malware, also the uncertain behaviors in a specific infection [2]. We identify the following aspects as our main challenges:

1. **Understanding malware behavior**

The number of malware families is constantly increasing and they exhibit complicated behavior and techniques. Although in our recovery scenario, we are only interested in the persistent behavior that modifies the system state, we still first need to gain a reasonable understanding of malware behaviors, and more important to our goal, how they sometimes change their behavior.

2. **The lack of existing methodology for simulating malware recovery**
   Malware research is usually done in a controlled, virtualized environment. It is very seldom performed on real hosts, due to the security risk for the entire computer network and also difficulties to analyze the result. To the best of our knowledge, we are not aware of an existing systematic methodology that is able to realistically simulate real malware infection and test remediation.
3. **The large scope regarding forensic evidence**
   There are numerous forensic artifacts inside the Windows system. This gives us a promising base to find useful evidence, but meanwhile also poses the challenge to find a needle in a haystack. We want to find the artifacts that can solve the problem, is reliable, and easy to query and parse.

3.4. Related Work

In order to answer the above research questions, there are two main areas related to completing our work: The first one is about environment-sensitive malware, and the second one is about collecting and processing forensic evidence left by malware.

3.4.1. Environment-sensitive Malware

The first problem we try to solve is dealing with the malware behaviors exhibited differently in different environments. We dig into the work regarding how malware behaves differently under different contexts and try to extract the important configurations we need to set up in the sandbox.

Avllazagaj et al. [2] performed a large-scale empirical study on variable program behaviors on malware, PUP (potentially unwanted program), and benign programs, based on execution traces collected on millions of real hosts. Their result shows that malware displays a significantly higher behavior variability compared to benign software, with the most striking difference in the file creation action and its parameters (path and file name).

Xu et al. proposed a system called GoldenEye [41] which is able to expose the malware sample's targeted victim environment. They implement API labeling on all the environment query APIs and return multiple different responses to these APIs to observe the malware's behavior. For example, if GoldenEye detects that an API querying system language is called, it will return various language values to this API call and further observe the upcoming behavior: whether the malware continues to execute, or exits without malicious behavior. By doing so iteratively, it will expose the targeted victim's environment.

A webcast given by Tamas Boczan and Brandon McCrillis from SANS Institute and VMRay [3] gave an analysis of typical context-aware malware. They pointed out the key point is to set up the analysis environment to mimic the target environment as closely as possible, and how such procedures could be integrated into the dynamic malware analysis workflow.

Miramirkhani et al. [27] pointed out that it is feasible and effective for malware to determine the non-existence of "wear and tear artifacts" to escape sandbox analysis. These artifacts, such as the browsing history, the event logs, and the DNS resolver cache, are accumulated by real users' daily usage and are not present in a fresh sandbox environment. In order to solve this problem, Liu et al. [22] designed an emulation-based system to generate system artifacts based on real user profiles. The system collects artifacts on real users' computers, builds user profiles based on the statistical data of the artifacts, and generates artificial usage traces in the sandbox.

Their experiment shows that the data generated by the system is indistinguishable from real usage data.

3.4.2. Malware Forensic Evidences

The second problem relevant to our research topic is malware forensic evidence: what kind of traces are left by the malware, what activity do they indicate, how can we collect them, and how to analyze them.

Provataki and Katos [35] presented a forensic framework for reporting malware behavior within a specific context. They run the sample in multiple differently configured analysis systems and collect the behavior from different executions. They built an extension of the Cuckoo Sandbox to correlate and analyze the different reports, to demonstrate the similarities and differences between the different execution instances.

Sainju and Atkison [40] performed an experimental analysis on the Windows event logs triggered by malware. They have run dozens of samples from famous malware families and recorded the triggered Windows log events. They suggest such event logs can be used as behavioral-based rules for malware detection.

Dija et al. [39] described the methodology for examining traces of malware in Windows systems. They discussed both live (in memory) and offline (in disk) forensics, including kernel data structures and MFT records from the memory dump, ShimCache, USN journal, prefetch files, etc.

Đuranec et al. [10] have examined the artifacts in Windows 10 specifically related to file use, consisting of manual analysis of Prefetch, Shimcache, $UsnJrnl, Shellbag, UserAssist, etc with open-source tools. They also provide examples of how to combine different artifacts to support the investigation.

4 Design

4.1. System Requirements and Overview

Our system aims to provide recovery procedures after infections and revert the changes done by the malware. We first provide a specific example scenario based on the problem statement to make clear the system requirements:

> The user downloads an unknown software from the internet. This download does not trigger any antivirus alert. The user assumes the software to be genuine and installs it directly. However, this piece of software modifies the firewall setting and sets up a connection backdoor. Then the user's computer becomes part of a botnet and will create spam connections under the request of a command-and-control center. On the date of downloading, this malware family had not been discovered yet and no signature was existing in the antivirus database. 10 days later, after the update of the signature database, the antivirus product flags this piece of software as malware during a full system scanning. Now the user is aware that this malware has been installed on his system, and is looking for a solution to get rid of it and make sure his computer is in a safe and stable state.

Our recovery system will help him to achieve this. The following requirements are included in the scenario:

- The infection has happened. We are not doing any malware detection in our system.
- The malware has been identified by the antivirus product, and we are able to get the malware sample that caused the infection to happen.
- We have access to the user's computer to gather relevant information.
- No extra monitor is set up beforehand. The user is able to use our system after he is aware of the malware infection.

Without any monitoring beforehand, we can only rely on behavior analysis to reveal the malware's behavior. We run the malware sample in a sandbox, observe and extract its behavior, and then perform a revert action for the persistent changes done in the user's computer. This is also the idea from previous research on automatic malware recovery [31]. However, the problem with this general approach is, the information we collect from running the sample once in a controlled environment cannot accurately reflect the real infection on a user's computer. This

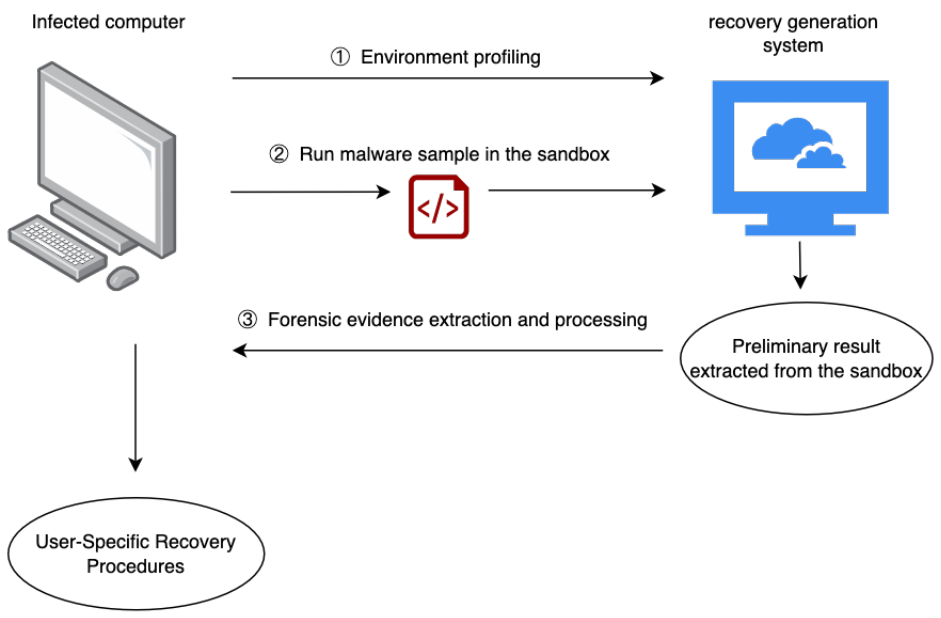

Infected computer
recovery generation system
① Environment profiling
② Run malware sample in the sandbox
③ Forensic evidence extraction and processing
Preliminary result extracted from the sandbox
User-Specific Recovery Procedures

Figure 4.1: Recovery System Overview

is also the key limitation of dynamic analysis itself. Empirical studies have shown that the behavior of one individual sample changes across different executions [2]. In malware detection, this problem is tackled by running the malware under different configurations and extracting the invariant behavior, which can be used as detection signatures. However, when our goal is doing recovery, instead of the "invariant", we are aiming at the "specific". We want to know what the malware does to this specific computer.

The reasons contributing to the different behaviors under different executions come from two aspects: some malware are environment-sensitive, and they choose to behave differently under different environments; malware is trying to do illegal operations on the computer. There are chances that certain actions do not succeed and malware wants to keep redundancy to ensure infection. More detailed explanations of the different behaviors are given in Section 4.3.1 and Section 4.4.1 when we introduce our design.

In order to understand what actually happened on the user's machine, we must utilize the information from the on-site crime scene. It provides us with the specific environment of the infection and the traces left by the infection. To utilize such information, we add two phases besides running the malware in a sandbox: building the environment and processing forensic evidence.

Therefore, our system is divided into three phases, as demonstrated in Fig 4.1:

1. **Profile environment**
   Instead of a standard virtual machine for malware analysis, we first try to build the analysis environment as similar as possible to the victim's environment and let the malware replay its behavior.
2. **Run the sample in the sandbox**
   We perform behavior analysis of the sample in a sandbox environment. The sample is executed in a controlled isolated environment and monitored using various instrumentation such as API hooking. From this, we get a behavior report of the malware sample.

3. **Combine the sandbox report and the forensic evidence from the infected system** The traces left by malware provides the most accurate information about the exact infection. We extract and process them on the user's machine, with the sandbox report as a reference to match with.

4.2. The Scope of Recovery

The first important discussion is determining the scope of recovery. Malware exhibits a wide variety of behaviors to achieve its intention. Based on the changes done to the system, we divide malware behavior into:

- **Drop child files**: This is a very common behavior of malware. Instead of executing the malicious code directly in the downloaded file, it drops child files. These files can have very small sizes and reside in a system folder, which is more covert and can remain unnoticed.
- **Modify existing files**: Instead of creating a new file, malware can also inject its malicious code into an existing file in the system. Another typical example is ransomware, which encrypts the user's data.
- **Create new process**: Malware can start a new process to run its own code.
- **Code injection**: Besides starting its own process, malware can also inject into other processes, and execute the malicious code in a stolen memory space.
- **Create network connections**: Malware can create network connections, which are used for downloading extra payloads, command & control communications, or data exfiltration.
- **Modification on the registry**: This includes 1) creating new keys: typically for achieving persistence on the system or privilege escalation, and also possibly adding payload in the registry. 2) modify/delete old keys: typically for stopping defensive system service, eg. firewall settings or logging service.

We use the system modeling for malware recovery defined from [33]: Let $S =< s_0, s_1, ..., s_n >$ be the sequence of traces (system calls) made by the malware sample during its execution. These system calls can be categorized into two groups: ones that alter the environment's state and ones that do not. For example, dropping a new child file involves writing new code to the disk and thus modifying the state of the system. Conversely, malware trying to read existing sensitive information on the computer does not cause changes to the system state. Let $S' =< s_j \in S >$, where s_j are the traces that modify the system state. We define T as state transitions. Each transition $t \in T$ represents the changes in the system resulting from the execution of a series of system calls. We further define the composition of $T = F \cup R \cup P \cup S$, where F = files, R = registry keys, P = process, S = services. A complete remediation procedure P should be able to revert all the system transitions T.

We do not recover corrupted users' own data. If the system is infected with ransomware and the user's files are encrypted, our system is not able to and does not aim to recover them. Recovering users' data requires available backup for reference and is out of the scope of this research project. Our recovery system aims at cleaning malicious files and restoring system files and settings. We remove all the malicious files and extra configurations set by the malware. And we are able to restore infected system data and settings, with a fresh, clean installation of the same operating system version as reference. We define this clean installation as the **reference system**.

We assume the active malware, including the processes and network connections, is already killed by the time the antivirus detects the malware (days after the infection). We are only interested in the persistent changes left on the computer. Services require executable files on the system, thus the malicious services are cleaned automatically after cleaning the malicious files. The actual persistent changes left on the system only regard the changes in files and registry. These changes can further be classified into create, delete, and modify.

Each of the situations and the corresponding recovery action is as follows:

Files:

- **Created file**: Malware can create one or multiple child files in the system. In the remediation procedures, we delete these files.
- **Deleted file**: Malware can delete system files, for example, legitimate DLLs. In the remediation procedures, we fetch the same file from the reference system and let the user install it.
- **Modified file**: Malware can write data into an existing system file. We perform the same remediation as deleted files, by fetching it from the reference system and letting the user replace it.

Registry keys

- **Created registry key**: Malware can create registry keys, for example for achieving persistence or privilege escalation. In the remediation procedures, we delete these registry key entries.
- **Deleted registry key**: Malware might delete registry keys, such as those related to security monitoring. We fetch the original value from the reference system and re-add them.
- **Modified registry key**: Similar to the deleted keys, malware can also modify registry keys. We fetch the corresponding registry key from the reference system and reset the value.

By now, we have a clear scope and task for the recovery procedures. In the later sections, we will discuss the different phases of our system in detail.

4.3. Customize the Sandbox

4.3.1. Environment-sensitive Malware

Environment-sensitive malware is the type of malware that alters its behavior based on the environment or conditions in which it operates. It is also known as context-aware malware or adaptive malware.

There are two main purposes of malware being environment-sensitive. The first type is that it tries to detect virtual environments in order to hide in an analysis environment. This is a common technique for sandbox evasion (other methods include time-based evasion and anti-monitoring evasion). The malware will typically first query information to determine whether this is an actual user's computer or not. Typical values that could give away a sandbox environment include limited system resources (few CPU cores, little RAM), the presence of virtualization artifacts, successful attempts at interaction with the hypervisor, etc [5].

The second type is for target attacks. Only for the specific chosen audience, the malware executes its malicious payload. On the other computer systems that do not fall under the target, the malware exits normally without any malicious activity. An infamous example is Stuxnet. Stuxnet is a highly sophisticated malware that particularly targeted the industrial control systems used in Iran's nuclear program. It identifies machines connected to targetted programmable logic controller (PLC) devices and only infects them [12]. Malware can also utilize a known vulnerability in a specific application, and thus only be able to execute its payload after successfully detecting the presence of this application.

Besides different semantic actions, malware, just the same as benign software, has different behaviors adjusting under different operating system versions and system architectures.

In the scope of our research, we are interested in exposing the targeting environment instead of hiding the virtual environment (which is yet another topic) and trying to replicate the malware's behavior inside the sandbox. We summarize the environmental variables that contribute to the difference:

- **Operating system version** and **system architecture** (32-bit or 64-bit).
- **Installed software**: Malware can query the list of installed software on a system. It may target vulnerabilities in commonly used software at a specific version.
- **Geographic location**: Malware may check the geographic location of the infected system and only target specific regions. This can be done by checking the timezone, keyboard layouts, or installed languages.
- **Network configuration**: Malware can analyze network settings, such as IP addresses and domain names to adapt its behavior. It may target specific IP ranges, which belong to a particular organization.
- **Hardware information**: Malware may check hardware information, such as MAC address or BIOS version, for target attacks.
- **Security software presence**: Malware often checks for the presence of antivirus or security products on a system. It may modify its behavior to evade detection or disable security measures to ensure its persistence.
- **Time and date**: Malware can use time-based triggers to change its behavior. It may activate or deactivate specific functionalities at predefined dates or times to avoid detection or maximize impact.

Setting the correct context is crucial for replicating the malware's behavior on the victim's machine. Our goal is to mimic the analysis environment to the user's environment as much as possible. Based on the summarization above, we build an environment profiling of the victim's machine and configure this to the sandbox before running the sample.

4.3.2. Environment Profiling

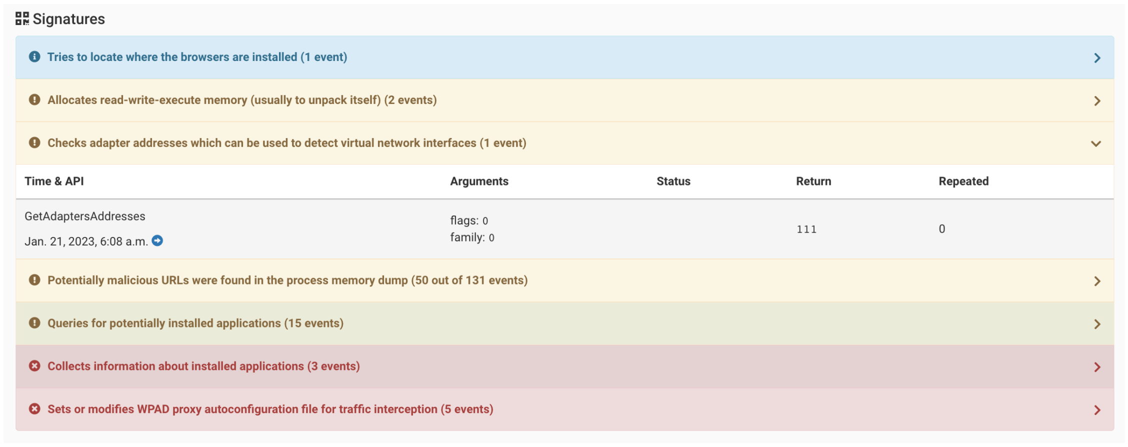
Given an environment-sensitive malware to analyze, it's usually very difficult to accurately expose the target environment in an automatic way. Such a topic has been researched in [41], but the approach still takes considerable time and system resources. However, in our scenario, since the infection has happened on the users' machine, we already have a "target environment". Instead of dynamically exposing, we set the possible environmental variables the same as in the victim's environment.

Sandbox signatures is a community-maintained database that includes lots of predefined patterns and can help analysts understand the malware's behavior more easily and quickly. For example, if the malware calls API `GetAdaptersAddress()`, the signature will identify this behavior and indicate that "the malware checks adapter addresses which can be used to detect virtual network interfaces", as the example shown in Fig 4.2. We collect the possible environmental queries from the sandbox signatures, with the addition of searching blog posts and academic papers.

We could first run the malware sample once, and record all the environment query APIs. In this way, we only set the environment variables that the malware has queried. But since this is a subset of all the possible configurations, by setting everything we also include them. And by setting all the variables directly, we avoid the risk of neglecting some stealth queries. We also save one extra round of running the sample in the sandbox, thus saving time in the whole process and being more efficient.

We include the following environmental configures in our profiling:

- **Operating system version**
- **System architecture**



Signatures

Tries to locate where the browsers are installed (1 event)

Allocates read-write-execute memory (usually to unpack itself) (2 events)

Checks adapter addresses which can be used to detect virtual network interfaces (1 event)

Time & API	Arguments	Status	Return	Repeated
GetAdaptersAddresses Jan. 21, 2023, 6:08 a.m.	flags: 0 family: 0		111	0

Potentially malicious URLs were found in the process memory dump (50 out of 131 events)

Queries for potentially installed applications (15 events)

Collects information about installed applications (3 events)

Sets or modifies WPAD proxy autoconfiguration file for traffic interception (5 events)

Figure 4.2: A sandbox signature example

- **Timezone**
- **Installed languages**
- **PC hostname**
- **Signed-in username**
- **User domain**
- **IP address**
- **MAC address**
- **UUID(universally unique identifier)**
- **Windows manufacturer**
- **System model**
- **Serial number**

The above information is collected from the user's computer and passed to the analysis sandbox. Virtual machines with different operating system versions and system architectures are prepared in the sandbox. The other variables are included in a pre-script and be executed to set up the environment before running the malware sample.

In the case of malware targeting a specific software's vulnerability, we can not discover the software automatically without additional information. We recognize this as a limitation and propose future work of integrating threat intelligence. Further discussion is given in Chapter 7.

Some information changes might affect the sandbox system, such as the IP address. The sandbox system needs to communicate with the guest VM, and thus a change of the guest IP address requires changing the relevant configuration of the sandbox. This can be trivially arranged by an extra short script changing the sandbox configuration files.

4.3.3. Other Mitigation

Besides the above-mentioned configurations, we also include some necessary mitigation to defeat sandbox evasion.

- We install commonly used software inside the guest VM, including Google Chrome, Microsoft Office, VLC media player, Microsoft Teams, etc.
- Malware might check "wear and tear artifacts" [27] to determine whether it is a real user's computer. This includes browser history, cookies, and multiple system usage logs. We use

the analysis virtual machine for normal daily activity for one month, to make sure there is enough evidence of real user usage (without including personal information or credentials, logged-in accounts are all for testing purposes).

4.4. Combining Forensic Evidence

4.4.1. Malware's Intrinsic Non-deterministic Behavior

In the previous section, we discuss the “subjective choices” that the malware chooses to take under different environments. Besides these high-level semantic behaviors, there are more non-deterministic behaviors from malware samples through different executions. When a piece of benign software needs to create a file, it creates this one file in a certain directory. However, since malware is trying to perform hostile and stealthy actions, it very likely can not be successful so easily. It will do multiple attempts to achieve its intention. The malware might try out 10 different folders to see where it has access to write into (think of a thief trying to visit all possible entrances to see which one he is able to break into). It might also drop the payload multiple times, under different names in different folders, to increase the further success rate by retaining some redundancy. An empirical study based on execution traces from real hosts shows that malware’s file creation action among different executions varies 15 times more than benign software [2].

These are intrinsic non-deterministic behaviors due to the characteristics of malware. Such uncertain behavior is impossible to accurately reconstruct. We can understand the semantic behavior by running the sample in the sandbox, but the only method to know the specific real actions is to examine the infected machine, collect evidence from the crime scene, then correspondingly build the recovery procedures. All modern operating systems have various internal logging mechanisms, which provide valuable information we can utilize. In the next section, we will discuss the forensic evidence left by malware, especially the ones related to evidence of execution and file system changes. We will categorize and summarize them, discuss the choice we make in our system, and the method to automatically process them.

4.4.2. Forensic Evidence Left by Malware

Forensic artifacts are pieces of traces left behind upon certain activities happening in the system. They can be considered digital footprints. These artifacts can include data, files, logs, and other information. Forensic artifacts play a crucial role in investigations of cyber incidents, as they can provide a wealth of information about the history of a system, such as when it was last used, which applications were running, which files were accessed, what network connections were made, and build a timeline of events related to the incident.

Users normally do not feel the existence of these artifacts. The logging and recording are happening in the background without hindering usability. And lots of the artifacts can not be directly accessed and manipulated but require specific forensic tools.

The forensic artifacts on Windows operating systems can briefly be summarized into the following categories:

- **System logs**: The operating systems generate various system logs that record important events, errors, warnings, and user activities. These logs include system event logs, security logs, application logs, and network logs.
- **Registry entries**: Windows operating systems maintain a centralized registry database that stores configuration settings, user activities, installed software, and system information.
- **File system metadata**: File system metadata includes file timestamps (created, modified, accessed), file permissions, and file attributes. The changes made to the file system are also recorded, which can be used for recovery in case of data loss or system crashes.

- **User account artifacts**: User account artifacts provide information about user activities and account information. These can include user login history, password hashes, account privileges, etc.
- **Internet browser artifacts**: Browser artifacts include browsing history, cached web pages, saved passwords, cookies, and downloads, which can help reveal users' visited websites and potential sources of malicious downloads.
- **Network artifacts**: Network artifacts include firewall logs, DHCP logs, DNS logs, etc. These logs can reveal interactions with malicious servers or hosts.
- **Memory artifacts**: A memory dump is a volatile artifact that can provide valuable insights into running processes, active network connections, and open files. Memory artifacts can help identify hidden processes, injected code, and evidence of malware presence.

Some stored information is not intended for forensic uses, but can also serve as forensic evidence, as it provides information about certain activity history. For example, Prefetch files are used by the Windows operating system to accelerate executable files' loading time, but can be used to check the last executed time of applications, which helps build the timeline in an incident investigation.

As discussed in the recovery scope in Section 4.2, we want to track down the changes done by the malware related to files and registry, both aspects include creation, deletion, and modification. With the sandbox analysis result as a reference, we want to identify the specific changes that have happened on the infected system. In order to do so, we need to find out which artifacts contain relevant information, and how can we separate the actions done by the malware from the other normal activities.

Direct evidence

The Windows Event Log is a centralized repository that stores records of events generated by the operating system, applications, and other system components. It is an extensive source to diagnose and investigate a system. A natural choice is to examine the Windows Event Log. During manual analysis, human analysts usually use their "instinct" to find out suspicious records and activities. However, in automatic analysis, we need to have specific rules to query and match.

Malware might have different names for their files, but they usually tend to disguise as legitimate processes when executing. It is unlikely that they have a randomized process name, in which case it will expose itself as soon as the user checks running processes. Thus we consider the process name a good identifier when checking into the event logs. We are able to identify process creation through Event ID 4688: Process Creation and Event ID 592: Process Creation (Scheduled Task), which are both enabled in Security Log by default [18]. However, no information about changes in the file system and registry is monitored by default in Windows Event Logs. To the extent of our knowledge, this information, changes done by a past process, is also not available in the registry or any other forensic artifacts.

A possible solution is using extra monitoring services, for example turning on file auditing or using external tools like Sysmon, but they require to be set up in advance, which does not suit our goal of generating recovery procedures entirely after the infection.

Therefore, we have realized one single source as Windows Event Logs is not able to provide the answer, we need to combine different artifacts. Our attempt turns to use correlation analysis. Event correlation is the process of analyzing and linking related events from different sources. By identifying co-existing evidence from different logs, we are able to discover more information than from any single source.

Besides the process name, the execution time is also an effective identifier to track down the malware. Timestamp is an inevitable element in most forensic artifacts. If we know the execution time of the malware, we can use it to query other artifacts and locate the changes caused by the malware.

Evidence of execution

Evidence of execution refers to records about when and how a particular program was executed. It takes different forms in multiple artifacts for different purposes. We have summarized the evidence of execution-related artifacts in Table 4.1, including their location and function.

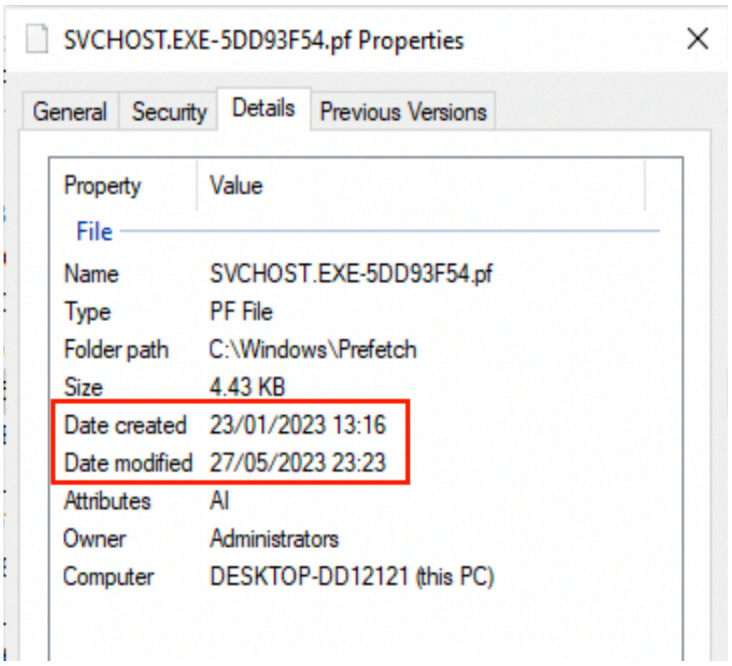
Table 4.1: Evidence of Execution Forensic Artifacts in Windows 10

Name	**Location**	**Description**
Prefetch Files	%SystemRoot%\Prefetch	Stores information about executed applications, including execution time, run counts, and file paths.
ShimCache	Registry(HKLM)	Contains details about recently executed applications, including timestamps, file paths, and execution counts.
Jump Lists	%AppData%\Microsoft \Windows\Recent\ CustomDestinations	Tracks recently accessed files and executed commands for specific applications.
Shellbags	Registry (HKCU)	Stores folder settings and metadata, recently accessed folders, and execution patterns.
AmCache	Registry (HKLM)	Contains information about executed applications, file paths, timestamps, and execution-related data.
User Assist	Registry (HKCU)	Records the use of applications by the user, including execution counts, timestamps, and paths.

After collecting and comparing the multiple options, we select Prefetch to retrieve the execution time. When a program is executed in Windows, it saves certain data to facilitate faster loading in future instances. This data is stored in Prefetch files, which locate in `C:\Windows\Prefetch` directory. In the first 10 seconds of a program's execution, the Prefetch file is generated or updated. On the first run of the application, the name and path of the file, and the first execution time (equivalent to the creation time of this Prefetch file) are stored. On every next run, the last run time and the run count are updated.

Prefetch files have the `.pf` extension and have the name format as `name.exe-xxxxxxxx.pf`, where the last 8 digits are a hash value of the executable file's path. The same executable at different locations each has its own Prefetch file. Windows 8 and above versions store a total of 1024 Prefetch files, while Windows 7 and previous versions store 128 Prefetch files. Prefetch files can be parsed and extracted from Prefetch parser PECmd from Eric Zimmerman's tools. In our case, we are only interested in the first execution time, which is a direct characteristic of the prefetch file: "CreationTime", which can be directly retrieved without parsing the Prefetch data. An example is shown in Fig 4.3.

In our recovery scenario, we have detected the malware sample. Therefore, we are aware



Property	Value
File	
Name	SVCHOST.EXE-5DD93F54.pf
Type	PF File
Folder path	C:\Windows\Prefetch
Size	4.43 KB
Date created	23/01/2023 13:16
Date modified	27/05/2023 23:23
Attributes	AI
Owner	Administrators
Computer	DESKTOP-DD12121 (this PC)

Figure 4.3: A Prefetch File Example

of the sample's name and location. We are able to find the Prefetch file of this executable and get its first execution time.

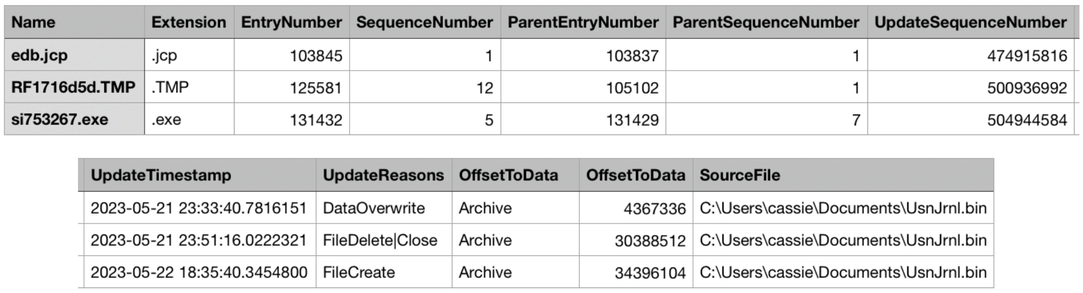
Changes in file system

The next step is tracking down the changes made to the file system. Windows operating systems use NTFS (New Technology File System). It is the successor after FAT (File Allocation Table) introduced in 1993 by Microsoft and has since then been the default file system for Windows operating systems. It has features including journaling, user-based access control, alternate data streams, volume shadow copy, etc [25].

NTFS maintains several files to help define and organize the file system, including:

- **Master File Table (MFT)** is a critical component of the NTFS file system. It is a database-like structure that contains information about all files and directories on a volume. The MFT records store the file/directory name, creation date, access permissions, size, and the location of files and directories [25]. It acts as a directory for the file system, allowing efficient and organized access to file data. The record number for a file in MFT is called FileId, similar to the Inode number in a Unix-style file system.
- The **$LOGFILE**, located in the root directory of the volume, serves as the transaction log for the file system. It maintains a record of changes made to the file system metadata, such as updates to the MFT. The $LogFile helps ensure the consistency and recoverability of the file system in the event of system crashes or unexpected power loss.
- **USN Journal (Update Sequence Number Journal)** records all the changes made to the volume. It keeps track of file system modifications, such as file creations, deletions, renames, and attribute changes. The USN journal helps improve the efficiency of file system operations, such as backup and indexing, by providing a way to quickly identify changed files.

The USN journal file particularly fits our requirement: we want to know the changes that have been made to the file system around the malware's execution time. From Windows 7, this



Name	Extension	EntryNumber	SequenceNumber	ParentEntryNumber	ParentSequenceNumber	UpdateSequenceNumber
edb.jcp	.jcp	103845	1	103837	1	474915816
RF1716d5d.TMP	.TMP	125581	12	105102	1	500936992
si753267.exe	.exe	131432	5	131429	7	504944584

UpdateTimestamp	UpdateReasons	OffsetToData	OffsetToData	SourceFile
2023-05-21 23:33:40.7816151	DataOverwrite	Archive	4367336	C:\Users\cassie\Documents\UsnJrnl.bin
2023-05-21 23:51:16.0222321	FileDelete|Close	Archive	30388512	C:\Users\cassie\Documents\UsnJrnl.bin
2023-05-22 18:35:40.3454800	FileCreate	Archive	34396104	C:\Users\cassie\Documents\UsnJrnl.bin

Figure 4.4: Example records extracted from $UsnJrnl

change journal is activated by default [30]. USN journal can be extracted from the live system into a binary file by MFTECmd from Eric Zimmerman's tools and then converted to a CSV file by UsnJrnl2Csv developed by Joakim Schicht. Example records from extracted USN journal are shown in Fig 4.4. The records contain various information including the file name, sequence numbers which are the identifier to the files, update timestamp, and update reasons.

Now we are able to know all the file changes done to the system after the malware execution. In the next section, we will further discuss how we combine this with the result generated by the sandbox to identify the exact file.

Registry

The registry is commonly used as forensic evidence itself and has a lot of modifications consistently. To the best of our knowledge, there exist no intended registry forensic artifacts in Windows operating systems.

Windows 10 includes auditing capabilities that can be configured to track registry changes (Event ID 4657: A registry value was modified) [18]. However, such auditing is not enabled by default and requires manually enabled in advance to serve as forensic evidence.

There are two other logs related to the registry. The first one is transaction logs (.LOG). They are used when registry hives cannot be immediately written into due to locking or corruption [49]. The changes will be saved in a transaction log and can be later applied to the incomplete registry hive (a dirty hive). Transaction logs are only generated when certain data can not be written into the registry hive and do not have complete records of all modifications on the registry. The other one is transactional registry logs (.TxR). The registry operations can be written in a transactional file, to perform compound registry operations as a single unit [26]. It can be used by application installers as it simplifies failed installation rollback. This is not commonly used and is also unlikely utilized by malware authors as they do not want to leave such evidence.

Due to the fact that registry keys are usually for specific purposes, for example, adding persistence or turning off a certain service. The key usually has a specific location and name. It is unlikely to have a random name at a random location. Thus, we consider the registry key's name is already a good identifier to check from the live system, even without the verification from forensic evidence. We directly use the reported registry keys from the sandbox result.

4.4.3. Identify Exact Changes

In the previous section, we have identified the forensic evidence we are going to utilize. In this section, we will explain how to combine the sandbox result with the forensic evidence, to identify the exact changes left on the user's system and build the recovery procedures.

Files

Created Files

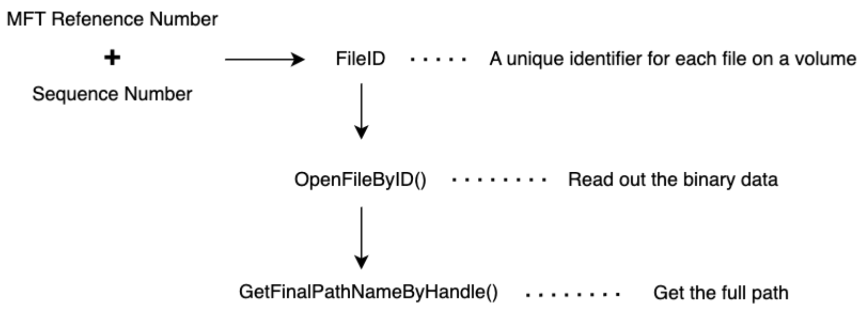


Figure 4.5: Getting the file full path from UsnJrnl Record

From running the sample in the sandbox, we are able to get the malware's dropped files (the binary data). From the Prefetch files and the extracted USN journal from the infected system, we are able to know all the newly created files after the malware's execution. We can query these files on the live system, and compare them with the dropped files from the sandbox.

In the USN journal, it only contains the file name without the full path, thus we can not directly locate the file and read out the data. The USN journal record contains an MFT reference number and a sequence number, from which we are able to get the FileID, a unique identifier for files in one volume. We can then read out the file data by Windows API `OpenFileById`, and get the full path by `GetFinalPathNameByHandle` [4]. The process is shown in Fig 4.5.

The typical way to compare files is using hashes. A hash function maps data of arbitrary size to a fixed-size value. Cryptographic hashes (eg. SHA256) are widely used to identify identical files by computing and comparing the hash, but one single change in the file will result in the hash being completely different. We are not able to use a direct hash comparison, as those dropped files might contain small configuration changes.

A naive way to compare file similarity is to compare it byte by byte. However, it takes up to 1 minute to compare two files with a size of 1MB (tested with Python's library difflib.sequencematcher). In $UsnJrnl record, there exist over 10,000 records within the time range of 5 minutes. Thus, we need something more efficient.

In order to identify similar files (almost-identical files), we use **fuzzy hashing**, invented by Jesse Kornblum in 2006 [20]. Instead of directly calculating the hash of the entire file, we first divide the file into multiple parts. We separately calculate the hash of each part, and then mathematically combine them into one single value, as shown in Fig 4.6. Fuzzy hashes are context-triggered piecewise hashes (CTPH) and are able to demonstrate the similarity level of two files. It is also currently used in antivirus engines such as VirusTotal to identify malware [50].

Modified Files

Malware sometimes modifies existing files. This happens very often in the case of ransomware infection, but some malware also writes into legitimate Microsoft system files and adds its malicious code to these files. In this case, we need to compare the newly added part to confirm the change was made by malware. However, in the forensic evidence, we can only know what file has been changed, instead of the exact changed content. Since it's a system file that gets modified (contrary to the malware-dropped files' randomized name), we directly use the file name as an identifier. For each modified file reported in the sandbox, we find whether a $UsnJrnl record with the update reason "DataOverWrite" and the same file name exists. If so, we add it to the recovery list.

Deleted Files

In the case of deleted file, the file is no longer on the system thus we have no reference to

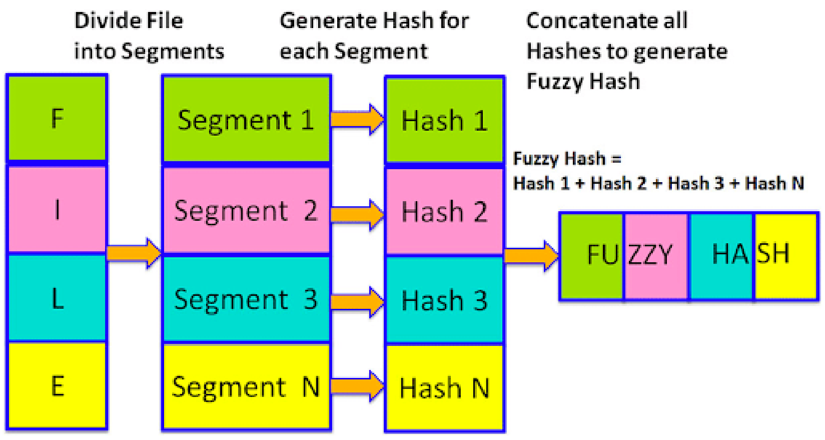

Divide File into Segments
Generate Hash for each Segment
Concatenate all Hashes to generate Fuzzy Hash
F
I
L
E
Segment 1
Segment 2
Segment 3
Segment N
Hash 1
Hash 2
Hash 3
Hash N
Fuzzy Hash = Hash 1 + Hash 2 + Hash 3 + Hash N
FU ZZY HA SH

Figure 4.6: Fuzzy hash generation process [28]

compare with. Since it's a system file that gets deleted, we directly compare the deleted files' names in the record. With each entry of the deleted files from the sandbox report, we query whether a record with the same file name and the update reason "FileDelete" exists in the USN journal. If we find a match, we add it to the recovery lists.

Registry keys

As mentioned in the previous section, there exists no forensic evidence to track the registry changes. However, registry entries need to use a specific name to achieve their purpose. We directly query the affected registry keys shown in the sandbox on the user's system.

If the key's name and value's name are existing (in the case of creation and modification) or non-existing (in the case of deletion) on the system, we assume it's affected by the malware and we add it to the recovery resource list.

Note that all the affected resources that we can find out are the behaviors that have been shown in the sandbox. If the behaviors are not shown in the sandbox, we do not have a reference to match the real actions. We address this as the fundamental limitation of behavior-based recovery and further discuss it in Chapter 7.

5 Implementation

5.1. Overview

We implement our prototype recovery system based on Windows 10. Although Windows 7 is a more popular and mature choice for malware analysis, now there are very few users still using Windows 7 for their daily usage. Microsoft stopped the support for Windows 7 in January 2020 [46], and by the time of April 2023, only 3.78% percent of desktop Windows users still use Windows 7 [44]. Thus we believe using Windows 10 for the implementation and evaluation is more practical and useful regarding malware recovery for normal users.

The sandbox deployment is done on a Linux Host machine with Ubuntu 22.04.2 LTS, CAPEv2 sandbox, KVM hypervisor, and Windows 10 guest VM. CAPE is a sandbox derived from Cuckoo, with additional automated malware unpacking and config extraction. Cuckoo is an open-source sandbox that has been a very popular choice for malware dynamic analysis, both in the industry and in academia. However, during the time of this thesis, Cuckoo has been publicly archived and is no longer maintained. The newer version, Cuckoo3 (written in Python3, instead of the original Cuckoo written in Python2), is still under the development phase and a lot of crucial functionality we need is not ready yet. Therefore, we decide to choose CAPE as the sandbox platform to conduct our ideas.

The implementation code has been made public on GitHub.

5.2. Environment Retrieve

In our prototype implementation, the environmental variable extraction and configuration are implemented in PowerShell. PowerShell is a command-line shell and object-oriented scripting language and has been an integral part of the Windows operating system since 2006 [24]. The extraction script is run on the user's system, and the retrieved variables are stored in a JSON file. An example result is given in Fig 5.1. The environmental variables are passed to the sandbox's virtual machine and be executed by the configuration script. We reboot the virtual machine to let the configurations take effect, and save the snapshot ready for the sandbox analysis.

5.3. Sandbox Analysis

After setting up the environmental variables to mimic the user's system in the analysis virtual machine, we submit the malware to the sandbox system. In our recovery scenario, we are only interested in the "persistent" changes related to the file system and registry, without the need for

```
{
    "mac-address": "08-00-27-00-00-00",
    "language-list": [
                          "en-GB",
                          "nl-NL"
                     ],
    "serial-number":  "ABC1234XYZ5678",
    "manufacturer":  "innotek GmbH",
    "username":  "cassie",
    "system-model":  "DA243A-ABA 6415cl NA910",
    "timezone":  "W. Europe Standard Time",
    "user-domain":  "DESKTOP-DD12121",
    "ip-address":  "10.0.2.15",
    "uuid":  "617D0B90-C77A-4318-B406-97546EAA0B68",
    "host-name":  "DESKTOP-DD12121"
}
```

Figure 5.1: Environmental variables result example

information in process dump and network recording. Therefore, we disable the process/memory/ASMI dump and the network sniffer. We set up the sandbox in a minimal way, which is intended to build the analysis environment less intrusive and can reflect the behavior on real hosts more accurately.

The networking routing is set up to Internet Routing mode, which means full Internet access is given to the sandbox virtual machine. This reflects the common scenario of users using their computers with a network connection. Considering security risks, we have set up firewall rules on the host machine, which block all the traffic attempting to go into the university's subnet.

After running the sample in the sandbox, we process the generated report to extract relevant information for our recovery system.

For file creation, CAPE saves the binary data of the dropped files, as well as information including name and path. For other file and registry changes, we extract the behavior based on API call traces. Each high-level behavior is indicated by the following APIs:

- File modification: `NtWriteFile`
- File deletion: `DeleteFileA`, `DeleteFileW`, `NtDeleteFile`
- Registry key creation: `RegCreateKeyExA`, `RegCreateKeyExW`, `NtCreateKey`
- Registry key writing: `RegSetValueExA`, `RegSetValueExW`, `NtSetValueKey`
- Registry key deletion: `RegDeleteKeyA`, `RegDeleteKeyW`, `RegDeleteValueA`, `RegDeleteValueW`, `NtDeleteValueKey`

5.4. Recovery Procedures Generation

The main part of the recovery generation program is implemented in Python, together with a few forensic tools. Since the NTFS change journal can not be directly accessed by users, we need specific forensic tools to extract them. Therefore, in our recovery program, we also pack a few tools together with it. They are light-weighted and can be directly used without the installation process.

UsnJrnl extraction

We utilize ExtractUsnjrnl to extract the UsnJrnl records from the live system. The output is a binary file. We then use MFTECmd to parse the binary file into CSV format, which we can easily query and process in the later steps.

Find execution time

We search in the `C:\Windows\Prefetch` folder and find the file starting with the sample's name. We save the creation time of this file, which means the first time this malware was executed.

If such a file does not exist, it means either the sample did not execute successfully, or the malware intentionally deleted the Prefetch file to hide its traces.

Identify file changes

Because we run the sample in the sandbox for 5 minutes, all the changes that are possible to be identified also happen within this time period. Therefore, we also check 5 minutes after the malware execution timestamp in the UsnJrnl records. After checking the UsnJrnl records, we notice it is possible to have over 10,000 entries within 5 minutes, and we need to compare them with all the files from the sandbox report. In order to improve efficiency, when comparing file content, we only compare the ones with the same file extension.

For created files, we find out the records with the update reason "FileCreate", and read out the date through Windows API `OpenFileByID`, where ID is reconstructed from MFT reference number and sequence number. We use the library ssdeep to calculate the fuzzy hashing of the file and compare it to the fuzzy hashes of dropped files in the sandbox. During our testing, we notice that the similarity score of the fuzzy hashes is very distinguishable: they are either 0 or above 90. Therefore we select 90 as the threshold score. If the similarity score between the local file and any dropped file from the sandbox is above 90, we conclude this is a malware-created file. Then we use Windows API `GetFileInformationByHandleEx` to get the full path of the file and include it in the affected resources list.

For file modification, we find the records with the update reason "DataOverwrite". For file deletion, we find the records with the update reason "FileDelete". If the file name matches any record in the sandbox report, we include it in the affected resources list.

Query registry key existence

For created and modified registry keys, we check whether the key/value name exists on the system. If it exists, we assume it was affected by the malware and we include it in the affected resources list.

For deleted registry keys, if the key/value name does not exist on the system, we assume it was deleted by the malware and include it in the affected resources list.

Executable recovery procedures

Within our prototype implementation, we generate a JSON file with all the affected resources, since the coverage of affected resources is more relevant for the evaluation rather than the executable recovery program. An example of the JSON result is shown in Fig 5.2. With the presence of a clean installation of the same system, it is trivial to generate the script to perform the actions. We remove the newly created resources and fetch a clean version of the deleted/modified components back to the computer.

```
{
    "created_files": [
        {
            "name": "C:\\Windows\\MSBLT.EXE",
            "fileid": 1970324837121274,
            "sha256": "08c338479f18f2d1bcd2b08201a3ba43",
            "ssdeep": "12288:yoxejOONAM7GUC1Jr+4o628gx2Jw+tP3Jzm8JOdHXC3X+pd167QhEQO:
    hxY3NtGUmJr+4Obxd+tPZSZZiE6EhE"
        },
        {
            "name": "C:\\Windows\\CSRLT.EXE",
            "fileid": 1970324837121290,
            "sha256": "ac6f0b7204916fee9861efae4b93f8b932456ed69b22b59fcd2c2263e918fdd8
    ",
            "ssdeep": "12288:yoxejOONAM7GUC1Jr+4o628gx2Jw+tP3Jzm8JOAHXC3X+pd167QhEQO:
    hxY3NtGUmJr+4Obxd+tPZSZciE6EhE",
        }
    ],
    "deleted_files": [],
    "modified_files": [],
    "created_keys": [
    "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
        "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce"
    ],
    "wrote_keys": [
        {
            "regkey": "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\CSRLT.
    EXE",
            "content": "C:\\Windows\\system32\\CSRLT.EXE"
        },
        {
            "regkey": "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce\\
    MSBLT.EXE",
            "content": "C:\\Windows\\MSBLT.EXE"
        }
    ],
    "deleted_keys": []
}
```

Figure 5.2: Affected resources list example

6 Evaluation

6.1. Overview

In this section, we evaluate the effectiveness of our recovery system. The evaluation is divided into two parts, corresponding to the before-sandbox phase and after-sandbox phase in our system. The first part evaluates the environment configuration phase of our system. Since environment-sensitive is not a universal characteristic of malware, we perform selected case studies in this part. We discuss several environment-sensitive malware examples, and how our system succeeds or fails to expose their full behavior. The second part evaluates the coverage rate of the generated recovery procedures on a large scale. We first introduce the evaluation methodology and explain the experiment's setup, then we present the results.

6.2. Case Studies

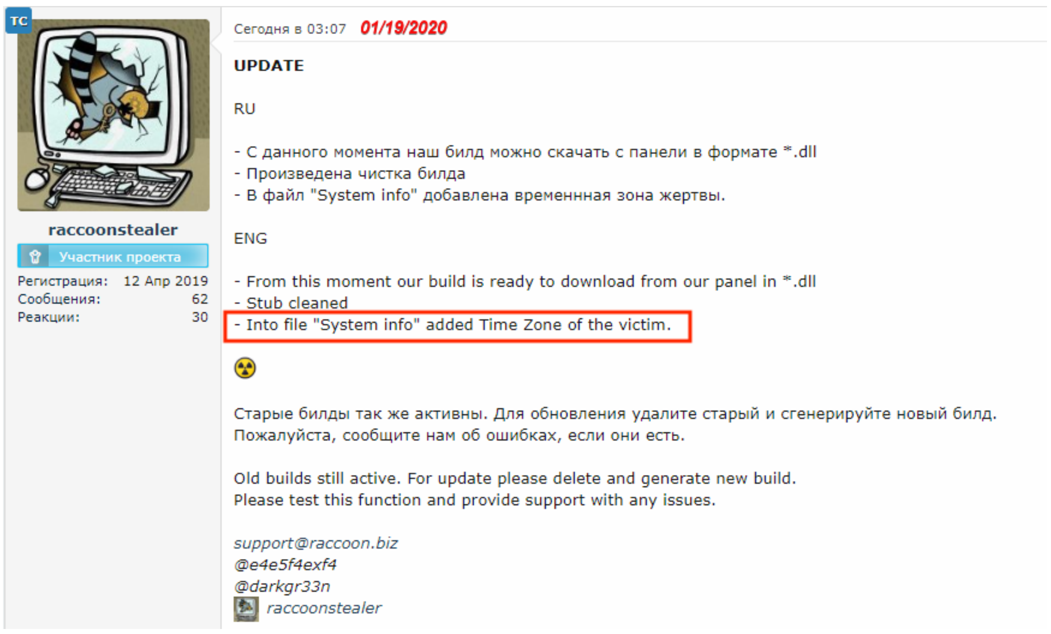
We hereby give three examples of environment-sensitive malware, namely Raccoon, Operation ShadowHammer, and OopsIE. We demonstrate and discuss how our system can expose the full behavior of the first two samples, but fail on the third one.

6.2.1. Raccoon Stealer

Raccoon family is an info stealer type of malware. It first appeared in 2019 and was sold in underground forums available as malware-as-a-service (MaaS) [7], where the customers not only get the malicious software but also an administrative page to track the harvest and technical support from malware authors.

Raccoon can be used to steal information including credit card information, login credentials, and cryptocurrency wallets. Fig 6.1 is a post when a newer version of Raccoon was released, which supports the configuration of the victim's timezone.

We demonstrate with a Raccoon sample (MD5 hash: 08a87b5af76a7d9f47d0bdd7453d77a4) to show how changing the timezone will change the malware's behavior. We first run the sample with the virtual machine setting the timezone to UTC+3 Moscow Standard Time and the language to Russian, and the behavior is shown in Fig 6.2a. Suspicious content is detected in the code but no real action shows. We then change the timezone to UTC-5 American Eastern Time and the language to English and run the sample again, and the behavior is shown in Fig 6.2b. We can notice an encrypted HTTPS connection is established, which is the malware communicating to the command & control server. Therefore, setting the timezone as the infected host can expose the malware's capability.



TC

raccoonstealer
Участник проекта
Регистрация: 12 Апр 2019
Сообщения: 62
Реакции: 30

Сегодня в 03:07 *01/19/2020*

UPDATE

RU

- С данного момента наш билд можно скачать с панели в формате *.dll
- Произведена чистка билда
- В файл "System info" добавлена временная зона жертвы.

ENG

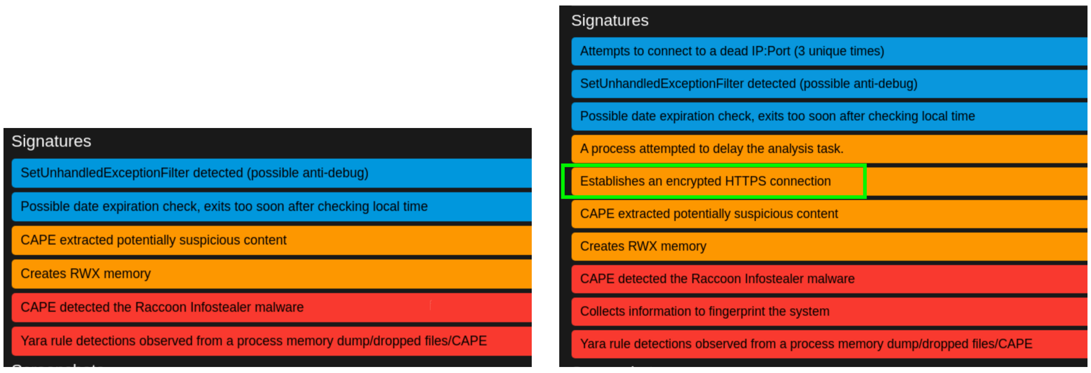
- From this moment our build is ready to download from our panel in *.dll
- Stub cleaned
- Into file "System info" added Time Zone of the victim.

Старые билды так же активны. Для обновления удалите старый и сгенерируйте новый билд.
Пожалуйста, сообщите нам об ошибках, если они есть.

Old builds still active. For update please delete and generate new build.
Please test this function and provide support with any issues.

support@raccoon.biz
@e4e5f4exf4
@darkgr33n
raccoonstealer

Figure 6.1: Raccoon supports configuring the timezone of the victim



Signatures
- SetUnhandledExceptionFilter detected (possible anti-debug)
- Possible date expiration check, exits too soon after checking local time
- CAPE extracted potentially suspicious content
- Creates RWX memory
- CAPE detected the Raccoon Infostealer malware
- Yara rule detections observed from a process memory dump/dropped files/CAPE

(a) Timezone in Russia

Signatures
- Attempts to connect to a dead IP:Port (3 unique times)
- SetUnhandledExceptionFilter detected (possible anti-debug)
- Possible date expiration check, exits too soon after checking local time
- A process attempted to delay the analysis task.
- Establishes an encrypted HTTPS connection
- CAPE extracted potentially suspicious content
- Creates RWX memory
- CAPE detected the Raccoon Infostealer malware
- Collects information to fingerprint the system
- Yara rule detections observed from a process memory dump/dropped files/CAPE

(b) Timezone in US

Figure 6.2: Raccoon sample's different behavior under different timezones

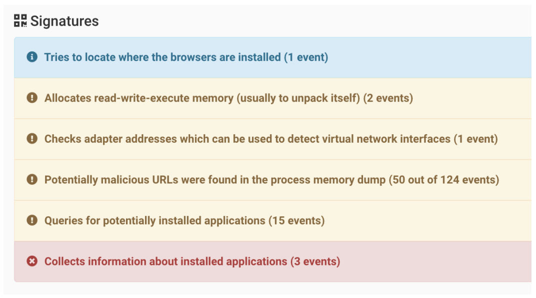
6.2.2. Operation ShadowHammer

Operation ShaowHammer is a high-profile supply chain attack targeting ASUS computers and was discovered in early 2019. It compromised ASUS Live Update Utility and distributed its malware through the official domain of a trusted manufacturer [21].

Although the malware was delivered to a large amount of ASUS users, Operation ShadowHammer was designed to specifically target a small number of users, of which they already had prior knowledge and have hardcoded the MAC addresses inside the malware.

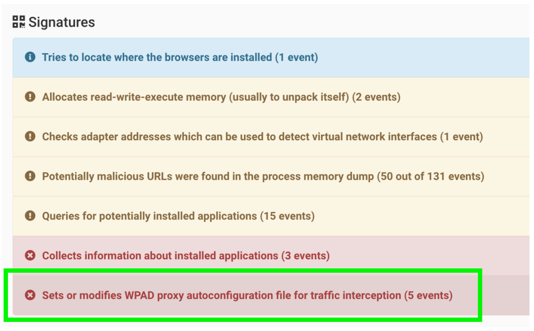
It was estimated that around 600 specific MAC addresses were targeted, and a backdoor was installed on the affected systems, enabling the attackers to have unauthorized access to the computer [13].

When running the sample (MD5 hash: 55a7aa5f0e52ba4d78c145811c830107) in our sandbox with a randomized MAC address, the behavior is shown in Fig 6.3a. The malware only collects information but does not perform any action. After changing the MAC address to one of its targets [13], we ran the sample again and the behavior is shown in Fig 6.3b. We can observe that the network configuration is changed and allows the download of further payload. Therefore, setting the MAC address of the infected host to the sandbox virtual machine can expose this malware's behavior, and thus generate effective recovery procedures.



Signatures
- Tries to locate where the browsers are installed (1 event)
- Allocates read-write-execute memory (usually to unpack itself) (2 events)
- Checks adapter addresses which can be used to detect virtual network interfaces (1 event)
- Potentially malicious URLs were found in the process memory dump (50 out of 124 events)
- Queries for potentially installed applications (15 events)
- Collects information about installed applications (3 events)

(a) With randomized MAC address



Signatures
- Tries to locate where the browsers are installed (1 event)
- Allocates read-write-execute memory (usually to unpack itself) (2 events)
- Checks adapter addresses which can be used to detect virtual network interfaces (1 event)
- Potentially malicious URLs were found in the process memory dump (50 out of 131 events)
- Queries for potentially installed applications (15 events)
- Collects information about installed applications (3 events)
- Sets or modifies WPAD proxy autoconfiguration file for traffic interception (5 events)

(b) With targeted MAC address

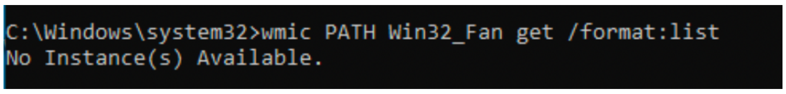
Figure 6.3: Operation ShadowHammer sample's different behavior under different MAC address

6.2.3. OopsIE

With the previous two examples, we show how changing certain environment variables help expose the malware's behavior, which demonstrates the effectiveness of our recovery system's first phase. However, although we can set the context the same as the victim, we are still not fully capable of coping with evasive techniques. We hereby give another example of OopsIE.

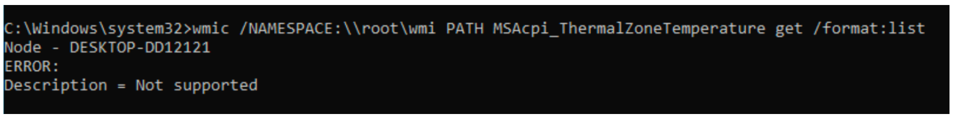
OopsIE is a trojan developed by the OliRig APT group and targets Middle Eastern organizations. It only operates on the computer with Middle Eastern time zones (from UTC+2 to UTC+4) [29]. Meanwhile, it also contains several advanced anti-VM techniques which help further evade detection and analysis, including checking whether the computer is connected to a fan, checking the temperature of the CPU, checking the existence of DDLs from Virtual Box and VMWare, etc [37].

In our system, we modify the timezone to Middle Eastern timezones, however, we still fail at the anti-VM checks, as the checking connected fan shown in Fig 6.4a and checking the CPU temperature shown in Fig 6.4b. In order to successfully analyze this sample, we not only need to set the correct geo-location but also need "anti-anti-analysis" methods to return non-VM values to these virtual machine checks. However, this is beyond the scope of this thesis.



```
C:\Windows\system32>wmic PATH Win32_Fan get /format:list
No Instance(s) Available.
```

(a) Checking connected fan



```
C:\Windows\system32>wmic /NAMESPACE:\\root\wmi PATH MSAcpi_ThermalZoneTemperature get /format:list
Node - DESKTOP-DD12121
ERROR:
Description = Not supported
```

(b) Checking CPU temperature

Figure 6.4: Anti-analysis check in OopsIE sample

6.3. Evaluation Methodology

Besides discussing a few case studies on environment-sensitive malware, we evaluate how many affected resources can be covered in our generated procedures. To get the most accurate result, the most ideal evaluation is to evaluate the effectiveness of our recovery system on real host malware infections. However, it is not realistic to infect bare-metal computers. When there is no upper layer in control (eg. hypervisor), it is difficult to monitor the behavior and also poses serious security risks. Therefore, the only feasible solution is to still use a virtual machine to mimic the user's computer.

The evaluation of a recovery system is not as straightforward as the evaluation of malware detection (is malware or not malware). The ultimate goal of recovery procedures is the machine returning to the previous state, however, comparing the machine state (before malware infection vs after running the recovery procedures) is not realistic: there are consistently legitimate changes done to the computer in the background as well. As already given and discussed in Chapter 4, the persistent changes of malware done to the system can be categorized into files and registry, both including creation, deletion, and modification. Therefore, instead of comparing the machine state, we test how many malware-generated changes are covered by our recovery procedures.

We need a "ground truth" for evaluating the recovery effectiveness. We generate the affected resource list from our recovery system and we want to compare it with what actually happened on the user's system. We do this by performing an "in-the-box" monitor during a malware infection. Using monitoring inside the system is contrary to the sandbox technology's "out-the-box" monitoring, as API hooking and the underlying network communication with the host might affect the malware behavior. Monitoring inside the system also avoids using sandbox report results to match changes monitored by the sandbox: If we also use the sandbox to get the "ground truth", we can expect the infection behavior will most likely be identical to the sandbox report, which we base our recovery procedure on. It then becomes a "self-fulfilling prophecy" and the evaluation result is not sound and persuasive.

The ground truth is collected using Sysmon and Procmon. Sysmon, short for System Monitor, is a tool from the Sysinternal suite developed by Microsoft. It is a service and device driver that can monitor and log various system events after installation. Sysmon is able to capture file creation and deletion, registry creation, modification, and deletion. File changes, which are more frequent and extensive events, are not covered by Sysmon's functionality. We utilize Procmon to track file modifications. Procmon, also part of the Sysinternal suits, is a real-time monitoring tool, that shows file system, registry, and process/thread activity. Since Procmon records a large volume of information in real time, we apply filters only to record write file events and exclude common legitimate processes. More detailed configuration and processing are given in Section 6.4.3.

6.4. Evaluation Set-up

6.4.1. Dataset

The data used in the evaluation is from Virustotal Academic Set 2020 Win32 EXE. We randomly selected 894 samples, consisting of 128 malware families and 55 singleton samples. Wapomi, Upatre, Berbew, and Mydoom are the most appeared families among our samples. During the random selection, we excluded a dominant ransomware family Virlock, which took up half the originally selected samples due to its polymorphism, and ransomware samples' data is not useful for our recovery evaluation.

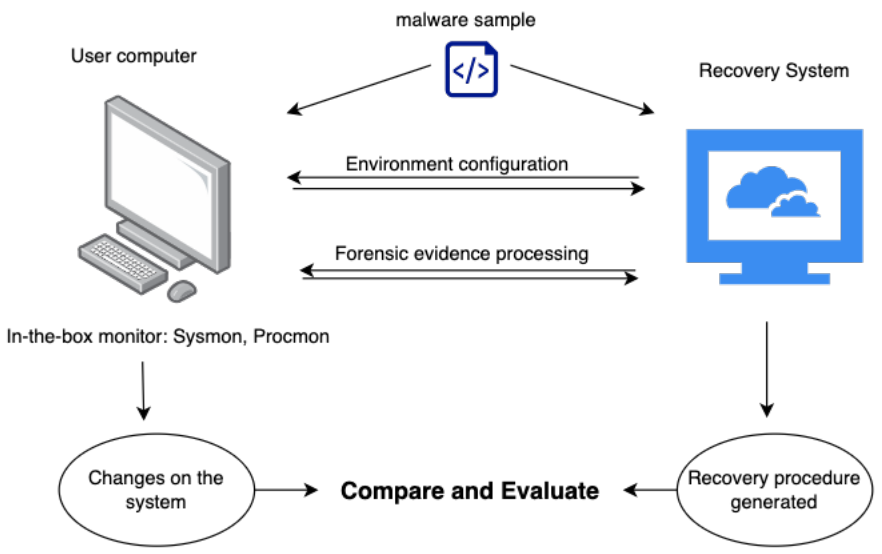

malware sample
User computer
Recovery System
Environment configuration
Forensic evidence processing
In-the-box monitor: Sysmon, Procmon
Changes on the system
Compare and Evaluate
Recovery procedure generated

Figure 6.5: Evaluation Overview

6.4.2. User Machine Preparation

Although we are limited to using virtual machines, we arrange the virtual machine as real hosts as possible. Commonly used software is installed. We have used the virtual machine for normal daily activities for one month. Therefore, the computer consists of real usage logs, cache data, browser history, and documents. All the accounts logged in are testing accounts and do not contain any personal information. All personal information-related documents have been erased before we start to infect the computer with real malware.

6.4.3. Ground Truth Collection

As mentioned in the methodology, we utilize Sysmon and Procmon for collecting the ground truth, which are the changes that malware makes to the system. Both Sysmon and Procmon are highly configurable. For Sysmon, we use the Sysmon config template from SwiftOnSecurity, which excludes most of the legitimate processes and noises. We made our own changes to this configuration file, excluding the events that are irrelevant to our recovery goal and enabling the ones that are relevant. In Procmon, we only monitor the write-file event, and have excluded the common system processes. The specific configuration files for Sysmon and Procmon are also available in the implementation code.

The logs generated by Sysmon are stored in Windows event logs (Applications and Services Logs/Microsoft/Windows/Sysmon/Operational) and we are able to use the Get-WinEvent cmdlet in Powershell to filter out the events with the sample's image name or process IDs. Procmon logs are saved in its PML format, which can be converted to CSV format, and easily filtered by process IDs. After letting the malware sample run for 5 minutes, we run the ground truth collection script and save the results into a JSON file.

We separate the changes made by malware and other processes by process ids. We first identify the starting process in Sysmon logs by looking for the Process Creation Event (Event ID: 1) with the image name of the sample's path and name. Then we get the child processes IDs from the Process Creation Event with the parent process ID equal to the starting process. We iterate this process until there's no new child process.

When malware uses process injection (injecting code into another process' memory space),

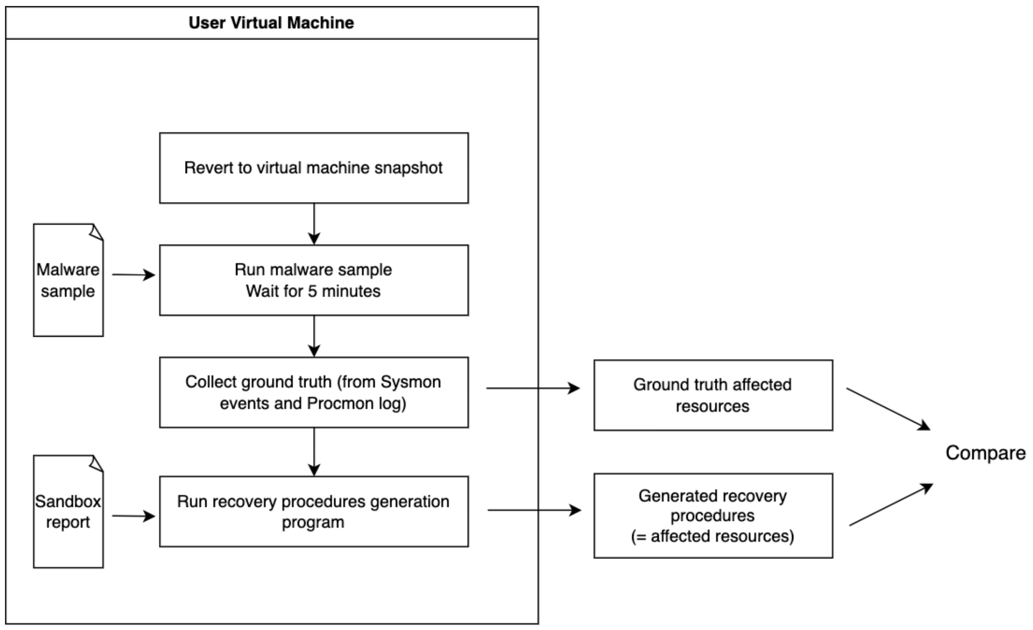

User Virtual Machine
Revert to virtual machine snapshot
Malware sample
Run malware sample
Wait for 5 minutes
Collect ground truth (from Sysmon events and Procmon log)
Sandbox report
Run recovery procedures generation program
Ground truth affected resources
Generated recovery procedures
(= affected resources)
Compare

Figure 6.6: Evaluation pipeline

we are not able to track the changes made from the injected code in our ground truth. Sysmon has the ability to identify the action of process injection (Event ID 8: CreateRemoteThread), but after we know which is the injected process, we still can not distinguish which change is made by the injected code and which is made by the legitimate part of the process, as there is no thread-level monitoring available.

6.4.4. Testing Pipeline

In order to parallel the large scale, we first run the samples in batches in the sandbox and extract the useful data from the sandbox. Then, we begin the testing on the user virtual machine. The pipeline is given in Fig 6.6. For each sample, we first revert the virtual machine to the clean snapshot and we copy the malware sample and the sandbox report into the virtual machine. We run the malware sample and wait for 5 minutes. Then we collect the ground truth from processing Sysmon events and Procmon logs. Afterward, execute the recovery script, and generate the list of infected resources. Finally, we copy out the experiment results for this sample. We compare the ground truth list and the generated list to evaluate the effectiveness of our recovery system: how many affected resources can be precisely pointed out.

6.5. Evaluation Result

Among the 894 samples, we collected 741 sets of results, while the other 153 samples did not successfully execute (the program crashed). And from 46 samples' results, there exists no corresponding Prefetch file, which renders our system unsuccessful directly. In the following part, we analyze the six persistent change situations separately and then give the overall recovery rate result.

Created files

Among the 741 active samples, 483 samples created files on the system. The median number of files created is 5 and the most common number is 1. There are 27 samples that created over 1000 files.

Among the 483 samples which created files, we are able to fully identify the created files from 137 samples, and we can not identify any of the created files from 191 samples. Overall, we observe that the files created in the user VM are more than the reported dropped files from the sandbox.

For the samples that created a large number of files, mainly from the malware family Sfone, we have a very low rate of identifying the files, as they mainly consist of randomly downloaded files.

Additionally, we observe a prevalent phenomenon in which malware tends to use randomized names and slightly modify the file content. Among the total 1443 found created files, 635 have a different name than the file in the sandbox and 626 have a similar fuzzy hash but not an identical sha256 hash. These files could not be found if we directly use the sandbox result and without combining the forensic evidence, which demonstrates the effectiveness and necessity of our method. The amount of found files is doubled compared to directly comparing hashes or file names.

Modified files

We observed 15 samples modified existing system files, including `autorun.inf`, `system.ini`, `win.ini`. However, none of these were shown in the sandbox.

Deleted files

We observed a large amount of file deletion activities from the sample. However, when examining closely, we notice it's mainly the malware deleting the file created by itself, or cleaning caches and logs related to its execution. We also observed malware samples emptied the recycle bin. There are 4 samples that deleted system files: they deleted `C:\Program Files (x86)\Internet Explorer\iexplore.exe` and `C:\Windows\SysWOW64\notepad.exe`, and replaced them with its own dropped files (with the same name). In our recovery procedures, we did not identify them, as they are not reported in the sandbox analysis.

Create Registry keys

Most created registry keys from our results are `CurrentVersion\Run` and `CurrentVersion\-RunOnce`. Run and RunOnce are commonly used by malware to achieve persistence through a reboot, as programs or scripts under these registry keys are executed during system startup (RunOnce) or user logon (Run). Both keys are enabled by default in Windows 10, and also in our system. They appeared in the created keys because malware samples commonly use `NtCreateKey` or `RegCreateKey`, and if the key exists, the API will open the existing key. Thus, we excluded the Run/RunOnce from the created keys when evaluating recovery.

Besides this, we observe 15 samples created registry keys. We are able to fully identify the changes done by 9 samples. The remaining ones are used for registering DLL or COM (component object model) objects, which contain a unique local ID, and we failed to identify them.

Write Registry keys

166 out of the 741 samples wrote data into registry keys. The majority of the written registry key values are under `CurrentVersion\Run` and `RunOnce`, in order to achieve persistence through a reboot. The others contain modifying firewall policies and security policies, turning off system service, and a few samples added binary data (payload) into the registry.

We are able to fully identify 122 samples' changes and are unable to identify any written entries from 10 samples.

We noticed 5 samples used randomized names under Run/RunOnce, which we can not successfully identify. The other non-covered registry keys consist of CLSID, e.g., `HKCR\CLSID\{79FEACFF-FFCE-815E-A900-316290B5B738}\InProcServer32\(Default)`. CLSID is a globally unique identifier (GUID) that is used to identify a specific COM class or object in the Windows

-	Identify Changes	Fully recovered	0% recovered
Created file	28.3%	53.3%	25.8%
Modified file	0%	98.0%	2.0%
Deleted file	0%	99.5%	0.5%
Created registry key	60%	99.2%	0.8%
Wrote registry key	73.5%	94.1%	1.3%
Deleted registry key	0%	99.5%	0.5%
Total	39.0%	51.3%	0.5%

Table 6.1: Evaluation result: Recovery rate

registry. Since we directly compare the registry key, a different CLSID, which is almost always the case, can not be recognized.

To tackle this problem, instead of direct comparison, an improvement can be first enumerating and then reconstructing possible values. We address this in Chapter 7.

Delete Registry keys

Malware usually modifies the value data instead of deleting the key, which falls under the previous category. Among the 741 active samples, we observe only four samples delete registry keys. Three of which are disabling AlternateShell in safe mode, and one is removing a browser DLL and replaced with its own DLL. None of them showed in the sandbox, thus we are not able to recover any of them.

Recovery Rate

We define the recovery procedures are successful and complete (fully recovered) if there are no malware-made changes left on the system. Thus, if the malware did not drop any file, we still consider it a successful recovery, as we did not leave malicious files on the system.

In Table 6.1, we summarize the percentage of changes that can be identified, the fully recovered rate, and 0% percent recovered rate. As file creation is the most variant behavior, the total recovery rate is mainly subject to the file creation recovery rate. Also, note that our recovery system did not delete any legitimate resources and thus has zero false positives.

7 Discussion

In this chapter, we first give a comparison with other malware recovery approaches and present the trade-offs. Afterward, we discuss the limitation of our approach and point out possible directions for future work.

7.1. Comparison

Since our work is based on the behavior-based recovery idea from Paleari et al. [31], a natural comparison is to compare with their approach. However, in their paper, they only gave a vague conclusion of 98% of the malicious resources being successfully remediated, without detailedly explaining the experiment process and presenting the results. From our experiment, we observe a significant portion of malware demonstrates more behavior in the user VM than in the sandbox, which makes a 98% recovery rate theoretically impossible. Therefore, we think the evaluation methodology might be very different and it's unfair to directly compare with their result.

We also summarize the results from other recovery methods, including log-based methods and snapshot-based methods, and compare them with our result in Table 7.1. Details about the mentioned systems are explained in Section 3.1.1 and Section 3.1.2. However, our approach is fundamentally different from these approaches. Since we do not require any monitoring set up in place beforehand, we have no space required and no performance overhead. Although the recovery rate is significantly lower than the other approaches, only our approach is suitable for the remediation process entirely after the infection. With this table, we present the trade-offs between different approaches.

7.2. Limitations

Our system is based on behavior analysis. Thus, it also has some inherited limitations from sandbox analysis:

- **Unable to recover from interactive malware**

  Some malware installs backdoors or communication channels to the command and control (C&C) server upon infection, which allows the attacker to remotely control and manage infected devices or systems for a longer time period. These further command and malware behaviors are not shown in the sandbox analysis result immediately and thus are not covered by our recovery procedures.

System	Set up in advance	Space required	Runtime overhead	Recovery rate
Back to the future [16]	Yes	MBs per program execution	100%	theoretically 100%
Taser [15]	Yes	GBs per day	7%	theoretically 100%
ExecRecorder [48]	Yes	5.4 GB/hour	4%	theoretically 100%
TimeVM [11]	Yes	thousands network packets	not measured	theoretically 100%
CRIU-MR [52]	Yes	GBs	1%	theoretically 100%
Our system	No	-	-	51.3%

Table 7.1: Comparison with other recovery systems

- **The different behavior from the sandbox and real hosts**

  Although we already try to address this limitation in this research, the behavior difference between the sandbox and real hosts is still significant. In our system, by doing environment configuration, we eliminate the difference from environment-sensitive / targeting malware. But the malware still demonstrates different behavior due to the presence of monitoring and instrumentation. This is also one of the most fundamental limitations of behavior analysis.

Besides the inherited limitation from sandbox analysis, there also exist limitations from our own design and implementation:

- **Effective time of the forensic evidence**

  We rely on forensic evidence to generate specific recovery procedures. The required evidence has a limited timespan, as newer records and logs are constantly written in and older data are replaced. In the case of 4 hours of system usage per day, the USN change journal can store the past 2 weeks' activities [30] [8]. If the infection happened a long time ago, this forensic evidence become absent and hinders the accuracy of our recovery procedures.
- **Oversimplified search strategy**

  It was only after the large-scale evaluation we gained insights into the data and our implementation. We realized improvements can be done in the implementation aspect, but due to the time limitation, we did not have the time to incorporate the improvement and re-run the experiment. Our current search strategy regarding modified files, deleted files, and all registry keys is directly comparing the name. However, through our evaluation result, we observed that sometimes there is a local specific value in the name, for example, an object ID in the registry key. The oversimplified search strategy makes us miss some behaviors which are already shown in the sandbox. An improvement idea is given in the next section.

7.3. Future Works

We hereby point out several possible directions to further continue and extend this work.

1. **Run malware sample in the sandbox multiple times to generalize behavior**

Previous works [31] [2] have demonstrated that generalized behavior from multiple traces is more effective than behavior from one single trace. Such a strategy can also be incorporated into our system: running the sample in the configured environment multiple times and obtaining a sum of possible behaviors and also generalized behavior (possibly in the form of regular expressions), then using the generalized behavior to match the specific forensic evidence. We anticipate this could improve the behavior's coverage range.

2. **Use multiple sources of forensic evidence**

   From our experiment result, we observe that 6% of the malware samples deleted the Prefetch files, which directly affected our recovery system's function. In our current implementation, we solely rely on the Prefetch file to get the malware execution's timestamp. As discussed in Chapter 4, there exist multiple forensic sources regarding executions, such as ShimCache, jump lists, and AmCache. Using multiple sources of forensic evidence can make our system more robust when malware intentionally deletes its own traces.

3. **Build smarter search strategy**

   As mentioned in the previous section, we observe that there are unique IDs included in the registry keys, instead of directly comparing, we can first enumerate the existing keys and build more possible searches. This also applies to files containing local factors. This can also be combined with the first future work point, to obtain a generalized expression and then search locally.

4. **Use dynamic environment configuration**

   In our system, we research and summarize the influential environmental variables that can affect malware behavior. However, this may not cover the full range of potential environment queries when further techniques are developed, and can also fail to conquer evasive techniques (such as querying whether a fan is connected). There exist systems [51] and academic works [41] on how to dynamically configure an environment, not by changing the actual system, but by returning values to environment query API calls and assembly instructions. The anticipated return values can be found from static analysis of the code or prepared lists. Further research can be how to combine these methods with an existing victim environment, to let the malware believe it is in the exact target environment.

5. **Integrate threat intelligence**

   Our current design sorely relies on the information obtained from sandbox execution, which is quick and formatted but also has its own limitations. Threat intelligence involves information that is collected, analyzed, and interpreted from multiple sources, and often involves human analysts' efforts as well. These data can be obtained from real hosts, such as the data collected by antivirus solutions, and provides a bigger and more accurate picture of malware's behavior. We believe researching how to integrate threat intelligence can help our recovery system remit the limitation of sandbox analysis.

8 Conclusion

We have answered the following research questions:

1. **What are the current malware recovery techniques, and how mature and effective are they?**

   They are three categories of current malware recovery techniques: log-based, snapshot-based, and behavior-based. Log-based and snapshot-based approaches demonstrate high recovery rates and high reliability but need to be set up in advance, and also have significant storage and performance overhead. The behavior-based approach can be used purely after the infection. However, the accuracy and reliability are lower than the other two kinds of approaches. The main challenge is that behaviors in the sandbox do not accurately reflect the behaviors on the real host.
2. **What are the non-deterministic malware behaviors?**

   There are two types of malware non-deterministic behaviors. The first type is subjective environment-sensitive behaviors. Some malware families have a target audience, for example, a specific country. They will first query the environment and only continue the malicious behavior if the environment falls under the target. The correct environment needs to be set for malware to expose its behavior.

   The second type is intrinsic non-deterministic behavior since malware is doing illegal and stealthy actions. It might need to visit ten different folders to see which one it has access to write into and always keep multiple redundancies to increase success and persistent rate. This behavior is impossible to reconstruct but needs to investigate forensic evidence to know about the exact behaviors.
3. **What are the types of forensic evidence left by malware infections, how can they be useful for malware recovery, and how can we automatically collect and process them?**

   In order to recover the persistent changes left by the malware, we are interested in the forensic artifacts regarding execution evidence, file system, and registry.

   We can obtain the execution time from Prefetch files. We can know the changes made on the file system from the USN change journal, and narrow down to the possible changes made by malware with the execution time.

   To the best of our knowledge, there exists no specific forensic evidence about registry modifications.

4. **How can we automatically generate recovery procedures, based on behavior analysis and forensic evidence?**

   From the USN journal and the Prefetch file, we obtain the list of all changes made to the file system within the short time period after the malware execution. We match them with the files reported by the sandbox. We utilize fuzzy hash to identify created files. For all created resources, we remove them from the system. For all modified and deleted system resources, we fetch them from a fresh, clean installation of the same operating system.

5. **How effective is our framework? What are the advantages and limitations?**

   Our framework is able to recover 51.3% of all the infected systems. The recovery procedures are generated based on sandbox reports, our system maximizes the usage of the sandbox-delivered information but is not able to cover any behavior that is not shown in the sandbox. The limitations come from the inherent limitation of sandbox analysis, usage of only a single execution evidence source, and oversimplified search strategy.

Finally, we answer the main research question of this thesis project:

How can we automatically generate user-specific recovery procedures after malware infections on Windows, which is able to revert all the malicious changes on system resources with no extra system monitoring in place?

We run the malware sample in a sandbox environment to observe its behavior and generate a revert operation for each action that modifies the system state.

In order to eliminate the environmental influence on malware's behavior, we extract environmental variables from the infected system and configure the same context to the sandbox virtual machine. After getting the behavior from the sandbox, we combine forensic evidence to understand the exact actions that happened on the system and generate the user-specific recovery procedures.

References

[1] Acronis. *Acronis' mid-year cyberthreats report finds ransomware is the number-one threat to organizations, projects damages to exceed $30 billion by 2023*. Aug. 2022. URL: https://www.acronis.com/en-eu/pr/2022/08/25-09-45.html.

[2] Erin Avllazagaj et al. "When Malware Changed Its Mind - An Empirical Study of Variable Program Behaviors in the Real World". In: Aug. 2021. URL: https://www.usenix.org/system/files/sec21-avllazagaj.pdf.

[3] Tamas Boczan and Brandon McCrillis. *Evade me if you can: Unmasking context-aware malware*. July 2021. URL: https://www.vmray.com/resource/sans-webcast-unmasking-context-aware-malware-2/.

[4] Celestial. *How to get the full path for USN journal query*. 2018. URL: https://stackoverflow.com/questions/31763195/how-to-get-the-full-path-for-usn-journal-query.

[5] A Chailytko and S Skuratovich. *Defeating sandbox evasion: how to increase the successful emulation rate in your virtual environment*. 2016.

[6] Sang-Hoon Choi and Ki-Woong Park. "iContainer: Consecutive checkpointing with rapid resilience for immortal container-based services". In: *Journal of Network and Computer Applications* 208 (2022), p. 103494.

[7] Ben Cohen. *Raccoon: The Story of a Typical Infostealer*. 2020. URL: https://www.cyberark.com/resources/threat-research-blog/raccoon-the-story-of-a-typical-infostealer.

[8] Mike Cohen. 2020. URL: https://velociraptor.velocidex.com/the-windows-usn-journal-f0c55c9010e.

[9] George W Dunlap et al. "ReVirt: Enabling intrusion analysis through virtual-machine logging and replay". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 211–224.

[10] A. Đuranec et al. "Investigating file use and knowledge with Windows 10 artifacts". In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* 00 (2019), pp. 1213–1218. DOI: 10.23919/mipro.2019.8756877.

[11] Khalid Elbadawi and Ehab Al-Shaer. "TimeVM: a framework for online intrusion mitigation and fast recovery using multi-time-lag traffic replay". In: *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security - ASIACCS '09* (2009), pp. 135–145. DOI: 10.1145/1533057.1533077.

[12] Josh Fruhlinger. *Stuxnet explained: The first known cyberweapon*. 2022. URL: https://www.csoonline.com/article/3218104/stuxnet-explained-the-first-known-cyberweapon.html.

[13] Sergiu Gatlan. *MAC Addresses Targeted by the ASUS Supply Chain Attack Now Available*. 2019. URL: https://www.bleepingcomputer.com/news/security/mac-addresses-targeted-by-the-asus-supply-chain-attack-now-available/.

[14] Jon Giffin. "The Next Malware Battleground: Recovery After Unknown Infection". In: *IEEE Security Privacy* 8.3 (2010), pp. 74–76. ISSN: 1540-7993. DOI: 10.1109/msp.2010.107.

[15] Ashvin Goel et al. “The taser intrusion recovery system”. In: *ACM SIGOPS Operating Systems Review* 39.5 (2005), pp. 163–176. ISSN: 0163-5980. DOI: 10.1145/1095810.1095826.

[16] Francis Hsu et al. “Back to the Future: A Framework for Automatic Malware Removal and System Repair”. In: *2006 22nd Annual Computer Security Applications Conference (ACSAC’06)* (2006), pp. 1–10. ISSN: 1063-9527. DOI: 10.1109/acsac.2006.16.

[17] IBM. *Cost of a data breach 2022*. Dec. 2022. URL: https://www.ibm.com/reports/data-breach.

[18] Ultimate IT. *Windows Security Log Encyclopedia*. 2023. URL: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx.

[19] Ronny Ko et al. “Ultraverse: Efficient Retroactive Operation for Attack Recovery in Database Systems and Web Frameworks”. In: *arXiv preprint arXiv:2211.05327* (2022).

[20] Jesse Kornblum. “Identifying almost identical files using context triggered piecewise hashing”. In: *Digital investigation* 3 (2006), pp. 91–97.

[21] Kaspersky Lab. *Operation ShadowHammer: new supply chain attack threatens hundreds of thousands of users worldwide*. 2019. URL: https://www.kaspersky.com/about/press-releases/2019_operation-shadowhammer-new-supply-chain-attack.

[22] Songsong Liu et al. “Enhancing malware analysis sandboxes with emulated user behavior”. In: *Computers Security* 115 (2022), p. 102613. ISSN: 0167-4048. DOI: 10.1016/j.cose.2022.102613.

[23] Kevin Mandia. *Increase in ransomware attacks ’absolutely aligns’ with rise of Crypto, FireEye CEO says*. June 2021. URL: https://www.cnbc.com/2021/06/28/fireeye-ceo-spike-in-ransomware-attacks-absolutely-aligns-with-crypto-rise-.html.

[24] Microsoft. 2022. URL: https://learn.microsoft.com/en-us/powershell/scripting/overview.

[25] Microsoft. *NTFS overview*. 2023. URL: https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview.

[26] Microsoft. *Working With Transactions*. 2022. URL: https://learn.microsoft.com/en-us/windows/win32/ktm/programming-model.

[27] Najmeh Miramirkhani et al. “Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts”. In: *2017 IEEE Symposium on Security and Privacy (SP)* (2017), pp. 1009–1024. DOI: 10.1109/sp.2017.42.

[28] 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.

[29] Patrick Howell O’Neill. *A well-known hacking group is getting better at evading detection*. 2018. URL: https://cyberscoop.com/oopsie-oilrig-iran-evading-detection/.

[30] Junghoon Oh. 2013. URL: http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf.

[31] Roberto Paleari et al. “Automatic Generation of Remediation Procedures for Malware Infections”. In: (2010), pp. 419–434.

[32] Emanuele Passerini. “Malware Mitigation and Remediation Strategies”. PhD thesis. 2009.

[33] Emanuele Passerini, Roberto Paleari, and Lorenzo Martignoni. “How Good Are Malware Detectors at Remediating Infected Systems?” In: *Lecture Notes in Computer Science* (2009), pp. 21–37. ISSN: 0302-9743. DOI: 10.1007/978-3-642-02918-9_2.

[34] David Patterson et al. *Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies*. Tech. rep. Citeseer, 2002.

[35] Athina Provataki and Vasilios Katos. “Differential malware forensics”. In: *Digital Investigation* 10.4 (2013). hiiiiii goedeavond, pp. 311–322. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.08.006.

[36] PC Risk. URL: https://www.pcrisk.com/.

[37] Riley Porter Robert Falcone Bryan Lee. *OilRig targets a Middle Eastern Government and Adds Evasion Techniques to OopsIE*. 2018. URL: https://unit42.paloaltonetworks.com/unit42-oilrig-targets-middle-eastern-government-adds-evasion-techniques-oopsie/.

[38] Florian Roth. *Sigma: Generic Signature Format for SIEM Systems*. URL: https://github.com/SigmaHQ/sigma.

[39] Dija S et al. “Cyber Forensics: Discovering Traces of Malware on Windows Systems”. In: *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)* 00 (2020), pp. 141–146. DOI: 10.1109/raics51191.2020.9332496.

[40] Arpan Man Sainju and Travis Atkison. “An Experimental Analysis of Windows Log Events Triggered by Malware”. In: *Proceedings of the SouthEast Conference* (2017), pp. 195–198. DOI: 10.1145/3077286.3077295.

[41] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. “GOLDENEYE: Efficiently and Effectively Unveiling Malware’s Targeted Environment”. In: *2012 Proceedings IEEE INFOCOM* (2012), pp. 2846–2850. DOI: 10.1007/978-3-319-11379-1_2.

[42] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. “A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis”. In: *International Journal on Advanced Science, Engineering and Information Technology* 8.4–2 (2018), pp. 1662–1671. ISSN: 2088-5334. DOI: 10.18517/ijaseit.8.4-2.6827.

[43] SonicWall. *Mid-year update to the 2022 sonicwall cyber threat report: Threat intelligence*. June 2022. URL: https://www.sonicwall.com/2022-cyber-threat-report/.

[44] StatCounter Global Stats. 2023. URL: https://gs.statcounter.com/windows-version-market-share/desktop/worldwide/.

[45] Orathai Sukwong, Hyong S Kim, and James C Hoe. “Commercial Antivirus Software Effectiveness: An Empirical Study”. In: *Computer* 44.3 (2011), pp. 63–70. ISSN: 0018-9162. DOI: 10.1109/mc.2010.187.

[46] Microsoft Support. 2023. URL: https://support.microsoft.com/en-us/windows/windows-7-support-ended-on-january-14-2020-b75d4580-2cc7-895a-2c9c-1466d9a53962.

[47] AV-TEST. *Malware statistics amp; trends report: AV-TEST*. URL: https://www.av-test.org/en/statistics/malware/.

[48] Josep Torrellas et al. “ExecRecorder: VM-based full-system replay for attack analysis and system recovery”. In: *Proceedings of the 1st workshop on Architectural and system support for improving software dependability - ASID ’06* (2006), pp. 66–71. DOI: 10.1145/1181309.1181320.

[49] David Via. *Digging Up the Past: Windows Registry Forensics Revisited*. 2019. URL: https://www.mandiant.com/resources/blog/digging-up-the-past-windows-registry-forensics-revisited.

[50] VirusTotal. *VirusTotal: ssdeep, CTPH hash of the file content*. 2020. URL: https://developers.virustotal.com/reference/ssdeep.

[51] VMray. *Nowhere to Hide: Analyzing Environment-Sensitive Malware with Rewind*. 2016. URL: https://www.vmray.com/cyber-security-blog/analyzing-environment-sensitive-malware/.

[52] Ashton Webster, Ryan Eckenrod, and James Purtilo. “Fast and Service-preserving Recovery from Malware Infections Using CRIU”. In: Aug. 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/webster.

[53] Yara-Rules. *Yara-Rules/Rules: Repository of Yara Rules*. URL: https://github.com/Yara-Rules/rules.

[54] Yara-Rules. *Yara:The pattern matching swiss knife for malware researchers*. URL: https://virustotal.github.io/yara/.

[55] Bo Yu et al. “A survey of malware behavior description and analysis”. In: *Frontiers of Information Technology Electronic Engineering* 19.5 (2018), pp. 583–603. ISSN: 2095-9184. DOI: 10.1631/fitee.1601745.

[56] Ningning Zhu and Tzi-cker Chiueh. “Design, Implementation, and Evaluation of Repairable File Service.” In: *DSN*. Citeseer. 2003, pp. 217–226.