Real-time attack graph generation using intrusion alerts

Master Thesis

Ion Băbălău

Real-time attack graph generation using intrusion alerts

Master Thesis

by

Ion Babalau

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday November 30, 2023 at 11:00 AM.

Student number:	5626110	
Project duration:	November 14, 2022 – November 30, 2023	
Thesis committee:	Dr. Ir. S. Verwer,	TU Delft, chair
	Prof. Dr. Ir. R. E. Kooij,	TU Delft
Supervisors:	Dr. Ir. S. Verwer,	TU Delft, supervisor
	Ir. A. Nadeem,	TU Delft, daily supervisor
	Ir. J. Keijer,	Northwave Cybersecurity, daily supervisor

This thesis is confidential and cannot be made public until November 30, 2023.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover image created with the assistance of DALL-E

The content of this thesis was enhanced with the assistance of ChatGPT.

TU Delft

Preface

Well, here we are. This thesis is the result of my work for the last year and concludes my master's degree at TU Delft. Looking back, there were many occasions when I didn't know how to proceed, questioned whether what I was working on would actually be useful, and when the end of this thesis seemed so *so* far away. Thankfully, I was lucky enough to have amazing people in my life, who were there to listen and motivate me to push forward.

First and foremost, I'd like to thank all my friends who were with me on this journey: Dan Plămădeală, Ioana Savu, Wessel Thomas, Gerben Timmerman, Bram Verboom, Konrad Ponichtera, Marin Duroyon, Mariana Samardžić, Dan Andreescu, Natalia Struharova, Laura Muntenaar, Andrei Geadău, Radu Rebeja and many others. Our weekly Tuesday drinks, during which we shared our frustrations about the thesis, supervisors, or life, but also shared a lot of great moments (and beers), provided a much-needed break from the routine, which made this entire period much more fun and enjoyable.

Speaking of supervisors, I would like to thank Sicco Verwer and Azqa Nadeem, my university supervisors, for all the guidance, insights, and feedback they provided throughout the duration of this thesis. I would also like to thank my company supervisor from Northwave Cybersecurity, Julik Keijer, for our weekly meetings, helpful suggestions, and for helping me see this thesis through.

Additionally, I want to thank Northwave Cybersecurity for the opportunity to do an internship and for providing the data which helped enhance this research.

Finally, I'd like to express my deepest gratitude to my parents and family for all their love and support, for continuously encouraging me even while being far away, and for helping me become the person I am today.

Ion Băbălău
Rotterdam, November 2023

Abstract

In an era where cyber threats evolve with alarming speed and sophistication, the role of Security Operation Centers (SOCs) has become increasingly pivotal in safeguarding digital infrastructures. SOCs serve as the frontline defence against malicious entities, where they continuously monitor and analyze network traffic, as well as the activity of users and systems for potential threats. The rapid growth of advanced cyber-attacks has amplified the reliance on Intrusion Detection Systems (IDS) to generate alerts for anomalous activities, and on SOC analysts to analyze those alerts. However, these systems often yield an overwhelming number of alerts, many of which are false positives, leading to alert fatigue among analysts. The scarcity of effective visualization tools, coupled with the analysts' dependence on manual investigation and correlation of events aggravates this issue, resulting in extended alert analysis times. Moreover, the number of attack scenarios keeps increasing daily, making it difficult to understand the possible next actions of an attacker and apply preventive measures.

This thesis introduces an innovative approach to aid SOC analysts in managing the large influx of alerts, mitigating alert fatigue, and enhancing the efficiency of threat identification and response. We present an attack prediction tool with alert visualization capabilities that produces real-time attack graphs, summarizing the alerts associated with a specific host. Our method utilizes a Suffix-based Probabilistic Deterministic Finite Automaton (SPDFA) to predict future attacker actions, promoting a proactive defence strategy, and achieving an accuracy of 33.71 %. We validate the practicality and relevance of our contributions through interviews with six security experts, confirming the utility of our methods in a live SOC context. Furthermore, we demonstrate the applicability of our approach by testing it with three datasets collected in the real world. Our work stands apart by simultaneously addressing alert correlation, attack visualization, and predictive modelling of attacker behaviour.

Contents

1 Introduction

1.1. Context

In the rapidly evolving landscape of cybersecurity, Security Operation Centers (SOCs) play a pivotal role in defending organizational networks and systems. Intrusion Detection Systems (IDS) are integral components of SOCs, constantly monitoring network traffic and system activities to identify potential threats. Based on the observed behaviour, these systems generate intrusion alerts, which are then analyzed by SOC analysts in order to identify any potential threats. However, these systems are also notorious for generating a high volume of alerts, including a substantial proportion of false positives [1]. This overwhelming influx of alerts can lead to alert fatigue among SOC analysts [2], impairing their ability to effectively prioritize and respond to genuine threats. The challenge lies not just in the sheer quantity of alerts but also in the difficulty of quickly discerning the context and significance of individual alerts. Moreover, despite advances in automation [3], human analysts remain essential in SOCs due to their irreplaceable skills in complex decision-making and adapting to new threats.

With new software vulnerabilities being discovered daily, attacker scenarios are in a constant state of evolution. Moreover, the large amount of software applications running in a corporate network extends the attack surface even further, as each application could potentially serve as an entry point for attackers. This complexity, combined with the vast number of possible attacks, makes it challenging for even the most seasoned analyst to understand what type of attack is unfolding in a network. This further hinders the ability of an analyst to predict the next step of an attacker, which could be valuable in proactively deploying countermeasures.

The lack of visualization tools is another ongoing challenge for SOC analysts [4]. In most cases, there is too much data to be able to visualize it properly, while also maintaining the resulting visuals simple, precise and informative [5]. This challenge also leads to other problems, such as the increased difficulty of correlating alerts to a particular incident, as the analyst would not have an efficient way to view the data from multiple alerts.

Attack graphs can be of great help in dealing with this challenge[6]. An attack graph is a representation of all the devices in the network together with their vulnerabilities and illustrates the possible paths that an attacker can exploit during an attack. An attack graph also showcases how the vulnerabilities interact with each other and can offer a much higher level view of existing security flaws in the network. However, building such an attack graph requires apriori knowledge of the network, devices, and vulnerabilities.

Recently, the notion of an ”alert-driven attack graph” has been introduced. SAGE [7] uses intrusion alerts to build an attack graph showcasing attacker actions throughout the network. Currently, the attack graphs are built ”offline”, which is useful for cyber threat intelligence and studying attacker strategies observed in the data. This however makes it less useful in a real-world scenario, where the threat landscape might change from one alert to the next, and new attack graphs would need to be generated in real-time.

1.2. Research objectives

This research aims to develop an attack prediction tool with alert visualization capabilities, specifically designed to assist SOC analysts during alert analysis and consequently reduce the effects of alert fatigue. We aim to develop a way of predicting the potential future attacker actions based on currently observed behaviour. This predictive capability is crucial for proactive defence, enabling analysts to anticipate and counteract malicious activities, thereby minimizing the impact of potential future threats and bolstering the overall security posture. By generating attack graphs in real-time, we can provide enriched context for each triggered alert, allowing analysts to quickly grasp the overall situation and understand the relationships between different alerts associated with a single host. This form of alert correlation is vital for a more streamlined and efficient analysis process. The main research question that we attempt to answer is:

How can we generate attack graphs in real-time, providing a comprehensive overview of the triggered intrusion alerts, predicting the subsequent actions of a malicious actor, and assisting a SOC analyst during the analysis of alerts?

We further decompose this question into three parts: predicting the next attacker action, real-time attack graph generation, and real-world applicability. First, we want to examine the methods of predicting what an attacker might do next, given his currently observed actions. SAGE already utilizes a Suffix-based Probabilistic Deterministic Finite Automaton (SPDFA) to learn and summarize attacker scenarios. For this sub-question, we are interested in researching the prediction capabilities of the SPDFA, which has been trained on an intrusion detection alert dataset. In the second part, we want to re-examine the current SAGE pipeline and modify it to be usable in real-time alert streaming scenarios. Here we will look into aspects such as when to re-generate the attack graphs and how to incorporate the next action prediction into this pipeline. Finally, we will evaluate our approach on real-world data, by collaborating with Northwave Cybersecurity[1]. To further help us evaluate our method, we will also conduct interviews with SOC analysts. To summarize, the following research sub-questions have been formulated:

1. How can the SPDFA be used to predict the next attacker action?
2. How can we generate attack graphs in real-time, which will aid a SOC analyst when handling alerts?
3. Can attack graphs be generated using data collected in the real world?

1.3. Contributions

In this work, we propose a new method of attack prediction with alert visualization capabilities designed to enhance the SOC analyst's ability to analyze alerts. We demonstrate its effectiveness on real-world data and confirm its capacity to address prevalent challenges in the field by conducting interviews with security experts. Our main contributions are:

- **Prediction of Future Attacker Actions** - We have developed a method of predicting the potential future attacker actions based on the observed intrusion alerts using an SPDFA. We have implemented three SPDFA traversal strategies and compared them with a PDFA-based approach under different circumstances.
- **Real-time generated attack graphs** - We have created a tool that aids SOC analysts in interpreting and responding to alerts generated by intrusion detection systems. Real-time generated attack graphs provide a visual overview of the triggered alerts for a host, while also displaying the possible next action of the attacker. This not only helps in correlating different alerts but also offers valuable context to each triggered alert, aiding in the more efficient identification of malicious activities. The predictive capability helps in proactively identifying and mitigating threats, potentially minimizing the impact of harmful activities. SOC analysts can now focus on the most critical alerts, enhancing their ability to respond to potential threats swiftly.

[1] https://northwave-cybersecurity.com/

- **Validation through Expert Interviews** - We have conducted interviews with six security experts to validate our approach, ensuring that our method is aligned with the real-world needs of SOC analysts. Additionally, the interviews offer valuable perspectives on the existing difficulties encountered by SOC analysts related to visualization and correlation methods, which could be beneficial for other studies seeking to assist SOC analysts.
- **Application on Real-World Data** - We have demonstrated the applicability and effectiveness of our method in real-world settings, by testing it on three datasets created using real-world industry data, showcasing its practical utility.

1.4. Outline

The rest of this thesis is structured as follows: In chapter 2 we describe background knowledge about intrusion detection systems, alerts, finite state automata, and provide a brief explanation of how SAGE currently creates attack graphs. This chapter also delves into other methods related to data-driven attack visualization tools, real-time attack correlation, and attack prediction. In chapter 3 we introduce an algorithm which uses the SPDFA to predict the next attacker action and evaluate its prediction performance in different scenarios. Following this, in chapter 4 we incorporate the prediction algorithm into the attack graph generation pipeline and adapt it to real-time data streaming. In chapter 5 we generate attack graphs using real-world data. To better evaluate and validate our approach, we organise interviews with security experts and report the findings in chapter 6. Finally, chapter 7 presents a discussion of our main findings and the limitations of our work, while in chapter 8 we conclude and identify potential areas of future research.

2

Background

In this chapter, we provide background to the concepts which will be used in this thesis. We start by describing intrusion detection systems and alerts. We then look into finite-state automatons and their applications. We then provide a summary of SAGE, a tool for generating attack graphs using intrusion alerts. Afterwards, we provide a brief literature survey, which covers the topics of offline as well as real-time (or online) data-driven attack visualization methods, attack correlation and attacker action prediction.

2.1. Intrusion Detection Alerts

Security Operations Centers, or SOCs, are frequently used in corporate environments to monitor the security state of the company network. It is a centralized unit responsible for monitoring, detecting, analyzing, and responding to cybersecurity incidents and threats in real-time. A company can either have its own SOC or hire an external one, depending on its needs. The primary purpose of a SOC is to protect an organization's sensitive data, assets, and information systems from various cyber threats, such as malware, data breaches, unauthorized access, and other cyber attacks. A SOC continuously monitors an organization's network, hosts and other digital assets using various security tools and technologies, such as Intrusion Detection Systems (IDS), firewalls, antivirus software, and Security Information and Event Management (SIEM) solutions. These tools generate logs and alerts for potential security incidents.

When an alert is triggered, the SOC analysts investigate it to determine if it is a real threat or a false positive. They do this by investigating the available log data from the time of the event and verifying public and private threat intelligence platforms. If the analyst believes that the alert has a chance of being a real incident, the proper incident response measures will be used, such as deleting the malware or even isolating the host from the network. SOC analysts also use threat intelligence feeds and databases to stay updated on the latest cyber threats, hacking techniques, and vulnerabilities.

One example of a network IDS is Suricata[1]. Suricata works by defining a set of rules which trigger on known malicious behaviour, making it a signature-based IDS. The rules can specify anything from the destination IP address of the packet, the protocol, port, to some byte sequence found in the payload. If a packet is found that matches any rule, an alert will be generated, which will contain the information about the packet which triggered the alert. Figure 2.1 shows an example of a suspicious user agent Suricata alert. Multiple useful fields can be seen, such as the timestamp, severity, and signature, which will offer some insight about the rule and the malicious behaviour.

Microsoft Sentinel is a cloud-native SIEM as well as a Security Orchestration, Automation, and Response (SOAR) solution offered by Microsoft. It is designed to provide security analytics and threat detection for enterprises and organizations. Its most important feature is its data ingestion capabilities. Microsoft Sentinel can ingest vast amounts of security data from various sources, including Azure services, on-premises infrastructure, and third-party security solutions. It collects data from logs, events, and other sources, enabling organizations to gain a comprehensive view of their security landscape.

[1] https://suricata.io/

```
{"timestamp":"2018-11-03T23:29:59.754160+0000","flow_id":223930225
1270834,"in_iface":"ens4","event_type":"alert","src_ip":"10.0.0.17
6","src_port":60308,"dest_ip":"169.254.169.254","dest_port":80,
"proto":"TCP","tx_id":0,"alert":"action":"allowed","gid":1,"signat
ure_id":2013031,"rev":6,"signature":"ET POLICY Python-urllib/
Suspicious User Agent","category":"Attempted Information
Leak","severity":2}}
```


Figure 2.1: Suricata alert example

It also uses AI and machine learning to analyze the data and identify potential threats. It can correlate events from different sources to provide a more comprehensive understanding of an attack or security incident. Its other features include the ability to automate repetitive tasks through automation, integration with various threat intelligence services, integration with the Microsoft ecosystem and large scalability and flexibility.

Although IDS and SIEMs are tuned and optimized for each network they are deployed in, they still face many challenges. As stated in chapter 1, the most common issue is the high volume of alerts. Usually, a SOC utilizes a combination of signature and anomaly-based detection software, in order to handle both known and unknown attacks. This however can lead to a high rate of false positives [8], because the system often classifies normal behavior as malicious one. This in turn leads to the phenomenon known as alert fatigue, where the large amount of alerts can desensitize the analyst, making him more likely to miss critical alerts while focusing on the false ones too much.

2.2. Finite State Automatons

A Finite State Automaton (FSA), is an abstract mathematical model used to describe various systems that exhibit discrete or sequential behaviour. A FSA consists of the following components:

- States: A finite set of states that the automaton can be in. Each state represents a specific condition or configuration of the system. States are typically represented as circles in diagrams, and they can be labelled for identification. The state machine can only be in one state at a given point in time.
- Transitions: A set of rules that define how the automaton can move from one state to another based on input symbols. These transitions represent the behaviour of the system. Transitions are typically represented as arrows between states, with each arrow labelled by the input symbol that triggers the transition.
- Alphabet: A finite set of input symbols, also known as the alphabet, that the automaton can recognize. These symbols serve as triggers for state transitions.
- Start State: One of the states is designated as the initial state, where the automaton starts its operation. It represents the starting configuration of the system.
- Accepting States (Final States): A subset of the states that are considered accepting states. When the automaton reaches an accepting state after processing a sequence of input symbols, it indicates that the input is accepted by the automaton.

A FSA operates by reading input symbols one by one from the input sequence, and based on the current state as well as the symbol read, moves to another state using the transitions. This process continues until all input symbols are processed, after which it is evaluated whether the automaton is in an accepting state. If yes, the input is accepted, otherwise it is rejected.

There are multiple flavours of an FSA. A deterministic FSA has a unique sequence of states in the model for every possible input sequence, while a non-deterministic one has multiple paths possible for the same input. Figure 2.2 shows an example of a DFSA with two states. S0 is the starting state, and as can be seen, for any possible input, there is a deterministic set of transitions. Figure 2.3 shows an

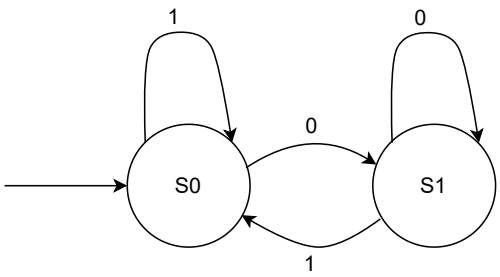

Figure 2.2: Example of DFSA, where every possible input sequence has a unique sequence of states in the model.

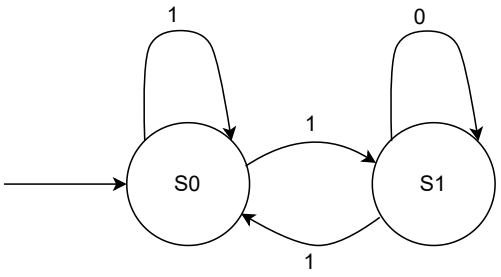

Figure 2.3: Example of NDFSA, where for the input sequence ”1”, there are multiple possible sequences of states in the model, S0->S0 and S0->S1

example of a non-deterministic FSA, and it can be seen that S0 has two transitions for the same input ”1”.

A probabilistic automaton has a probability value associated with every transition, which was calculated using the training data. This means that for a given state and input symbol, there may be multiple possible next states, each associated with a probability. The choice of the next state is made stochastically based on these probabilities. These probabilities represent the likelihood of the automaton taking a particular transition when it reads a specific input symbol in a given state. The probabilities assigned to all possible transitions from a state must sum up to 1. The introduction of probabilistic transitions in a PDFA allows it to exhibit non-deterministic behaviour. This means that when presented with the same input multiple times, a PDFA may produce different outcomes due to the probabilistic nature of its transitions. PDFA models are particularly useful in situations where there is uncertainty or randomness involved in the processing of input, such as in natural language processing tasks or machine learning algorithms that involve probabilistic decision-making.

2.3. Flexfringe: a passive automaton learning package

Flexfringe [9] is a machine learning framework used for learning Finite State Automata from sequential data, making it a valuable tool in automata learning and pattern recognition tasks. FlexFringe combines concepts from automata theory and machine learning to infer an automaton model that best represents the underlying sequential patterns in the input data. The tool uses a state-merging approach to iteratively build a compact automaton from the input sequences.

Flexfringe uses a set of traces as its training data. A trace is a sequence of symbols, which describe some particular behaviour. For example, a trace can be a sequence of Windows event logs. After initializing the model with some random states, Flexfringe begins the state merging algorithm. It iteratively merges states of the initial automaton based on statistical measures and similarity metrics derived from the input data. States that are similar in terms of their observed behaviour are combined into larger, more abstract states, reducing the overall complexity of the automaton. After each iteration of state merging, the resulting automaton is evaluated using standard performance metrics, such as accuracy and generalization to unseen data. The goal is to find the best-fitting automaton that adequately represents the sequential patterns in the training data. The state merging process continues until certain stopping criteria are met, such as performance on validation data, number of iterations, or if no more possible state merges are possible. The result is a model which characterizes the input data. The model will be split between two files, the main model and the sinks. If a sequence was present enough times in the training data, for the model to ”learn” about it, it will be present in the main model, whereas if the sequence was infrequent, it will usually be located in the sinks.

2.4. SAGE: intruSion alert-driven Attack Graph Extractor

SAGE, introduced in the paper *Alert-Driven Attack Graph Generation Using S-PDFA* [7], is an explainable sequence-learning pipeline that generates attack graphs using intrusion alerts, without prior knowledge of the network. First, the intrusion alerts generated using Suricata are grouped and transformed into attack episodes, which represent the actions the attacker took within the network. Then, the episodes are summarized using an SPDFA, which clusters similar attack paths based on temporal and

behavioural similarity. Finally, the learned SPDFA is used for creating attack graphs on a per-victim, per-objective basis. SAGE is explainable due to its transparent pipeline, where every step is discrete and deterministic. Next, we will go into more detail and describe the pipeline.

The first step is the pre-processing of intrusion alerts, which is done by:

1. removing informative alerts, or those with bad data (for example, same source and destination IP)
2. removing duplicate alerts, which are alerts that detect the same behaviour and occur close to one another
3. inferring the targeted service using the destination port
4. using signatures to augment alerts with their attack stage, using the Action-Intent Framework [10]

The AIF is a custom attack modelling framework, which aims to describe *what* the attacker is doing well as *how* he did it. To achieve this, Moskaal et al. define macro action-intent states (AIS), which describe the high-level objectives of an attacker, and micro AIS, which in turn describe what technique the attacker used to achieve that objective. The signatures of intrusion alerts can then be mapped to a micro AIS, which describes the attack stage.

To better illustrate the relationship between an attacker and the victim hosts, the intrusion alerts are clustered into alert sequences, which represent a group of alerts with the same attacker and victim, occurring within a time window. The alert sequences are further used to create the attacker actions, also called *attack episodes*. To create an episode, the frequency of the alerts is examined for a given attack stage, per time window. An increase in frequency denotes the start of an episode, while a decrease means the end of an episode. Thus, an attack episode consists of the start and end time, the attack stage and the most targeted (most frequent) service within that episode. The result of this stage of the pipeline are episode sequences, which are time-sorted episodes for an attacker-victim combination, such as `a1->v1: [<0,150, infoD, http>, <900, 1200, infoD, http>, <901, 1200, CnC, http>, <2250, 2550, infoD, http>, <2251, 2550, exfil, http>]`.

Afterwards, the episode sequences are transformed into traces, which in our case will model the attacker's actions. To achieve this, each ES is split into multiple subsequences, whenever a lower severity episode follows a higher severity one (Medium →Low, High →Medium, etc.). The attack stage and the most frequent service are extracted from each episode, creating a symbol. Thus, each episode sub-sequence represents a trace, which in turn consists of multiple sequences of symbols of increasing severity, in the format of `attack stage | most targeted service`.

The traces are then used to learn a Suffix-based Probabilistic Deterministic Finite Automaton (SPDFA) to summarize and model attacker behaviour. It is a probabilistic model due to the fact that the transition function also defines a probability for each transition in the model, which was calculated based on the training data. More formally, the SPDFA can be defined as "a 5-tuple $A = \langle Q, \sum, \Delta, P, q_0 \rangle$, where:

- Q is a finite set of states
- $\sum$ is a finite alphabet of symbols
- Δ is a finite set of transitions
- $P : \Delta \rightarrow [0, 1]$ is the transition probability function
- $q_0 \in Q$ is the final state (due to suffix model)

A transition $\delta \in \Delta$ in an S-PDFA is a tuple $\langle q, q', a \rangle$ where $q, q' \in Q$ are the target and source states, and $a \in \sum$ is a symbol. P is a function such that $\sum_{q,a} P(\langle q, a', a \rangle) = 1$. Additionally, Δ is such that for every $q \in Q$ and $a \in \sum$, there exists at most one $\langle q, a', a \rangle \in \Delta$, making the model (suffix) deterministic" [7].

As mentioned previously, due to the way the sequences (and consequently traces) are constructed, the high-severity episodes will be located towards the end of the trace. High-severity episodes are also less frequent compared to low and medium ones. If a model was to be learned with these traces, it would contain very few high-severity episodes in the main model, due to the model being unable to "learn" their context. Thus, most of these episodes would be excluded from the main model and put

in the sink states, which model infrequently seen behaviour. To combat this issue, and to bring the less frequent sequences into focus, a suffix-based model is learned instead of a regular one. For this purpose, each trace is reversed, meaning that the higher severity symbols will now be at the beginning of the trace. As a consequence of learning a suffix model, the model has a final state instead of a starting one, and the chronological order of the transitions is inverted.

The SPDFA is learned using the Flexfringe framework. The Alergia algorithm [11] is used for learning the automata, and other parameters are tweaked manually in order to ensure that only states with similar pasts are merged, to conserve the context of different high-severity alerts. Moreover, the sink states are saved, which are states that Flexfringe normally would not include in the final state machine due to the sequences corresponding to them not occurring enough times in the input traces. Since the high alerts are infrequent yet important, the high-severity sink states are included in the learned model.

Figure 2.4 shows the SPDFA which was learned using the intrusion alerts generated by team 1 of the CPTC 2018. The Collegiate Penetration Testing Competition (CPTC) is an annual cybersecurity competition that challenges college students to demonstrate their skills in performing real-world penetration testing and security assessments. The red circles are the states, state 0 being the exception because it is the final one. To explain what each annotation means, let us examine the state with identifier 29 (top right):

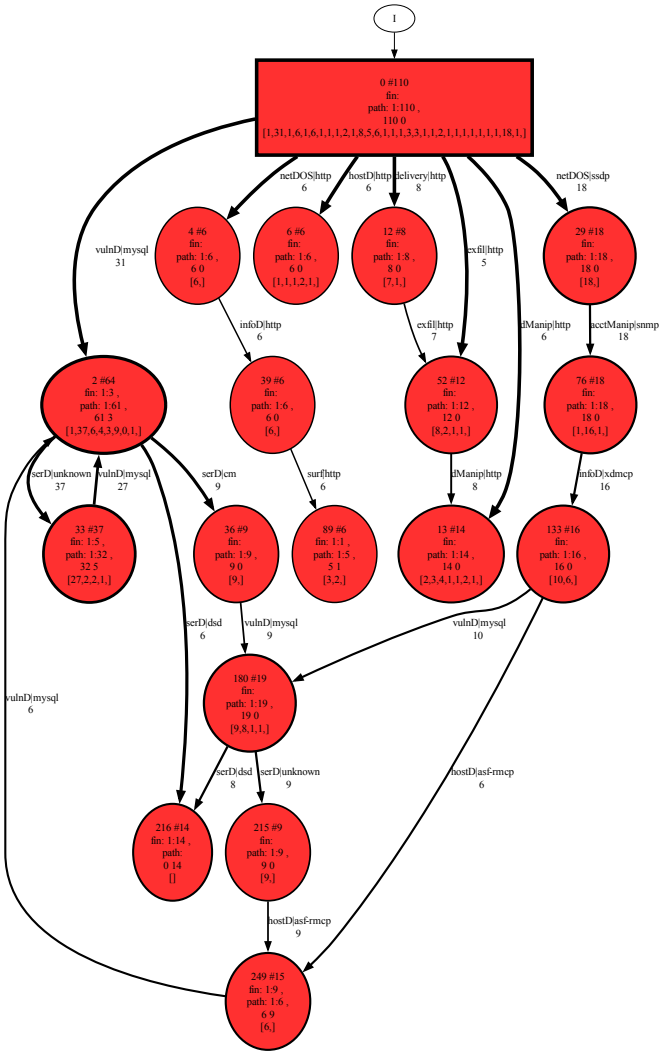

Figure 2.4: SPDFA learned using team 1 intrusion alerts. The transitions represent the attacker's actions, while the states provide context to those actions.

- the first line contains the state identifier, followed by the number of total occurrences for the given state, which is also the sum of all incoming transition counts; this number symbolizes the number of times the incoming symbol appeared in the training traces in the given context
- the second line illustrates how many traces end in this state, in this case, none, meaning that all of the traces will continue; this means that in the training data, there is no trace starting with `netDOS|ssdp` and immediately ending
- the third line tells us how many outgoing transitions this state has, in this case, 18
- the fourth line contains a summary of how many traces continue to other states and how many end in the current state
- the fifth line contains the count of each individual outgoing transition; in this case, it is only one transition, and all of the traces belong to it

It is worth pointing out that because the traces are inverted, the *ending* probabilities of traces can also be interpreted as *starting* probabilities. Transitions are annotated with the symbols of the traces as well as a number, which represents the number of times the sequence occurred in training data. For example, the transition from state 29 to state 76 has a count of 18, meaning that the training data contains 18 traces where a trace begins with `netDOS|ssdp`, and is followed by `acctManip|snmp`. Looking at the transitions, it can be seen that higher severity alerts, which happened later in time, such as `exfil|http` (data exfiltration over HTTP) are closer to the final state, showcasing the reversed order.

Finally, to create attack graphs, the episode sequences are augmented with their corresponding states from the SPDFA, assigning each episode its corresponding state identifier. When constructing an attack graph, the state identifiers are used to cross-match them to the SPDFA states. Each attack graph is then generated on a per-victim, per-objective basis, where an objective corresponds to a high-severity attack stage from the AIF. If the attacker has reached the objective in multiple ways which are significantly different (occurring in different contexts), there will be one high severity node for every attempt made, to showcase the different path the attacker took to reach it.

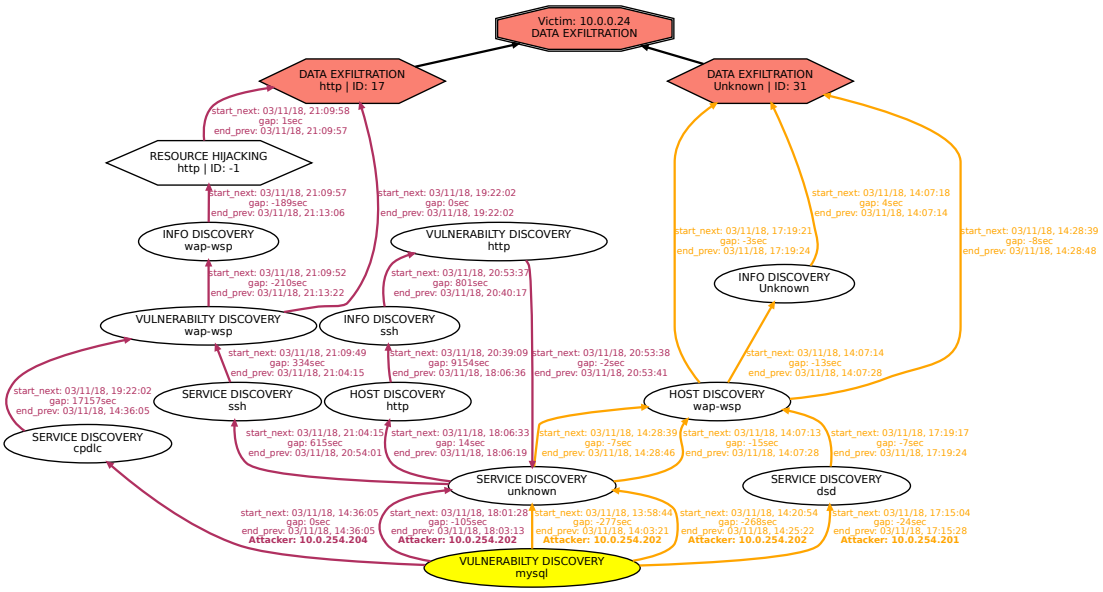

Figure 2.5: Attack graph for Data exfiltration objective using team 1 and team 2 intrusion alerts of CPTC-2018 dataset.

Figure 2.5 showcases an attack graph generated using team 1 and team 2 intrusion alerts from the CPTC 2018 dataset, for the Data exfiltration objective. Vertices represent an aggregation of alerts. Their labels show the attack stage and targeted service, as well as the ID of the states from the automaton, in case of medium and high severity states. Vertex form denotes the episode severity, low-severity

- oval, medium - boxes, high - hexagons. Yellow ones represent the first episode in the sequence, while red ones are the objective. Vertices which correspond to SPDFA sink states are dotted. Edge labels on the other hand have three important annotations: the time of the first alert in the next episode (start_next), the end time of the last alert from the previous episode (end_prev) and the time gap between them. The edge colour showcases the team affiliation of the attacker. The top node is an artificially added node, which connects all of the other objective nodes.

2.5. Related Work

In this section, we provide an overview of existing literature on the topics of data-driven attack visualization, real-time alert correlation and attacker action prediction.

2.5.1. Data driven attack visualization

Yang et al. [12] propose ASSERT, an informational theoretic unsupervised learning approach for the extraction and generation of attack models in near real-time. It uses intrusion alerts as input and does not require any prior expert knowledge neither of attack scenarios nor of the network and device vulnerabilities. The aim is to summarize intrusion alerts in real-time into a visual representation, to help SOC analysts get a better idea of the current situation and deal with alert fatigue. To achieve this, the model processes intrusion alerts in real-time and dynamically creates a set of models representing attack behaviour and scenarios. The authors define a metric called *closeness*, used to decide if when processing more data, a new model will be created or the old one will be updated, all of this done in an unsupervised manner. To test their method, the authors then deploy their model in a real-world SOC, to see how well it can identify attacks. An example of attack models generated by ASSERT can be seen in figure 2.6, where the arrow points to a model that has features of Arbitrary code execution and Persistent attack over Kerberos. A disadvantage of this approach is the lack of information regarding individual hosts which are under attack, as well as their relationship with the attacker.

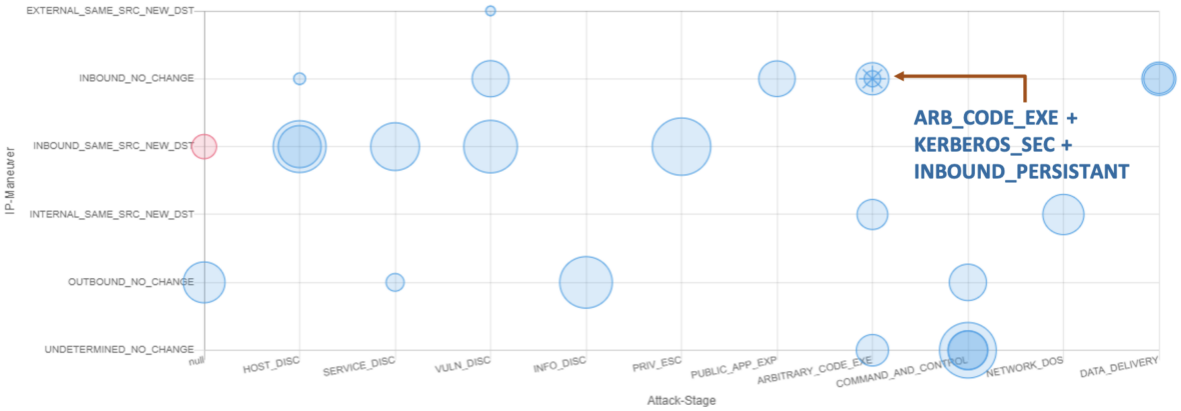

Figure 2.6: Attack models produced by ASSERT [12]

ATLAS [13] uses a combination of causality analysis, natural language processing and machine learning techniques to build a sequence-based model that can help in identifying attack steps in an Advanced Persistence Threat attack. During the training stage, ATLAS processes the audit logs to build a causal dependency graph, which it also uses to create NLP-augmented sequences describing malicious and benign scenarios. The sequences are undersampled and oversampled in order to balance the dataset. The text sequences are then transformed into word embeddings and fed to a deep-learning model consisting of a CNN and an LSTM. During the inference time, a security expert can feed the model an attack indicator, such as a malicious IP or host, using which ATLAS will extract the related entities from the causal graph. Using those entities, it will build the NLP sequences and run them through the trained model, which will predict whether they are part of an APT attack or not. The sequences which ATLAS found during this process will constitute the attack story, an example of which can be found in figure 2.7 (C).

Moskal et al. [14] introduce HeAT, which given an Indicator of Compromise such as a critical alert, generates an ”attack campaign”, which showcases the events leading to the IoC. Their approach con-

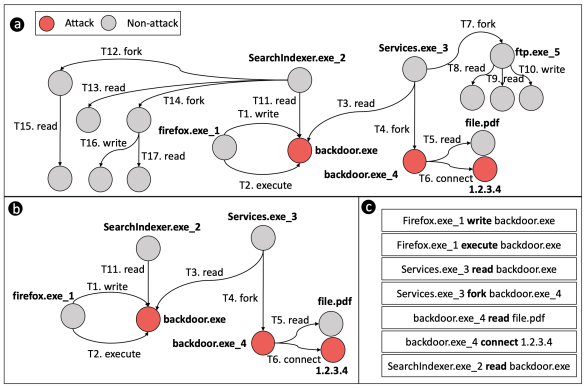

Figure 2.7: Causal graph and attack story generated by ATLAS [13]

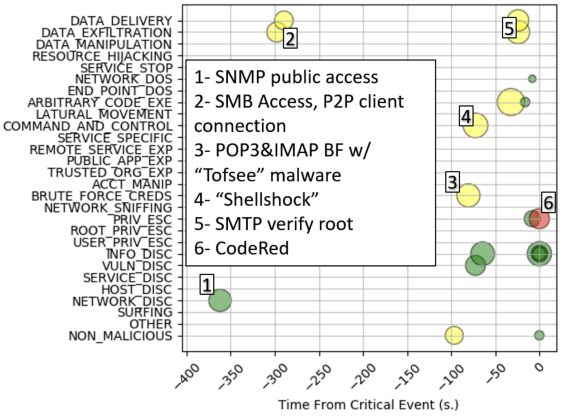

Figure 2.8: Example of HEAT attack campaign [14]

sists of first summarizing alerts by creating attack episodes and then presenting a subset of them to the analyst, who will be asked to rate their degree of relatedness to the IoC, called ”Alert Episode Heat” (AEH). These episodes will then be used to derive network agnostic features, as well as predict the AEH for the other episodes using machine learning. Finally, the ”heated” episodes are used to construct the attack campaign, an example of which can be found in figure 2.8. A disadvantage of this approach is the human factor which is involved. Because the episodes labelled by the analyst will be used to predict the ”heat” for the other episodes, if the analyst makes a mistake, the attack campaigns will not be accurate.

Hu et al. [15] argue that true negative alerts and unsuccessful attack paths are useful when reconstructing the attack scenario. First, the attack graph of the network is generated using MulVAL [16], which showcases the devices and their vulnerabilities. Afterwards, the intrusion alerts are mapped to the attack graph, and attack paths are clustered based on sequence similarity. As a last step, the attack paths are used to identify unreported true negative alerts. An attack scenario reconstructed by this approach can be seen in figure 2.9, where the first path is unsuccessful, while the second one succeeds.

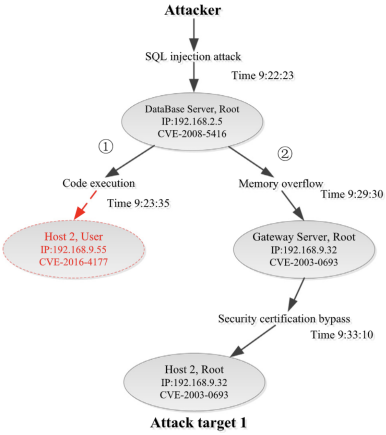

Figure 2.9: Attack scenario generated by Hu et al. [15]

2.5.2. Real-time attack correlation

Hofmann et al. [17] present a probabilistic approach to online attack correlation in distributed intrusion detection systems environments. Their method clusters alerts in real-time using probabilistic models and generates meta alerts which contain all the information from the summarized alerts within a cluster. When streaming data, every new alert is examined, and a decision is made whether a new cluster should be created, or whether the alert should be associated with an existing one. In theory, each

cluster should represent an attack scenario/objective. In practice, however, they were not able to achieve this, because for three reasons: the nature of the attack can vary, the long attack duration and bidirectional communication. However, their method showed an impressive reduction rate of up to 99.96 %, while maintaining a low number of missing meta-alerts.

Werner et al. [18] propose grouping alerts based on their inter-arrival times. Thus, an alert aggregate is defined as a consecutive set of alerts with similar IAT and is constructed using EWMA control charts. Aggregates whose IAT distributions are statistically similar can further be clustered into concepts, based on the two sample KS-test. So, alerts belonging to different concepts have a bigger statistical dissimilarity in their IAT. When streaming alerts in real-time, every alert is first added to the current alert aggregate, and the IAT is calculated. Afterwards, the aggregate is either kept and attributed to a concept or if the IAT is too different, a new aggregate will be started. They then tested their approach on the CPTC 2018 dataset, showing that concepts can indeed identify different temporal behaviours of alerts, even for attackers belonging to different teams.

2.5.3. Attack prediction

Holgado et al. [19] study the performance of Hidden Markov Models (HMMs) for predicting the next attacker action in real-time in the context of DDoS attacks. They represent attacker actions as hidden processes of the HMM, while the observations are represented by the intrusion alerts from the IDS. Each state in the HMM is also augmented with the mean number of alerts and the number of alerts in progress. The alerts are clustered based on keywords in their CVE description and their severity. The models are then trained using both supervised (counting the transitions and emissions of every state) and unsupervised (Baum-Welch Algorithm). Afterwards, the Viterbi Algorithm is used for calculating the best state sequence for a given observation sequence, which is then used to predict the next attack stage as well as the probability of the attacker reaching the objective (the final state), while using intrusion alerts as input. To test their approach, the authors simulate a DDoS attack in a virtual environment and collect the generated intrusion alerts. Although they obtained good prediction performance using the supervised model, a downside is that they only studied DDoS attacks, and each attack type requires training a separate HMM, questioning the performance of their method on other attack types.

Ramaki et al. [20] propose a method to predict the next attacker actions in real-time using a Bayesian Graph. The alerts are first pre-processed by verifying them against a vulnerability and topology database of the current network. They are then aggregated into meta-alerts based on their features, and causal relationships are defined between them, which are then used to create the Bayesian attack graph, where the nodes are meta-alerts. Finally, using a set of observed meta-alerts as input, the probability of the meta-alerts most likely to happen can be calculated in real-time using posterior probabilities. Even though most of the training and creation of the BAG is done offline, a nice part of their approach is that the BAG can be re-trained online, by recalculating the probabilities based on the observed meta-alerts.

Fava et al. [21] develop a way to predict the actions of an attacker inside a network using variable-length Markov models (VLMM). To achieve this, they first introduce the concept of attack tracks, which are sequences of various fields from intrusion alerts. Thus author chose to construct attack tracks using three alert fields: the destination IP from the packets, alert category (scanning, vulnerability exploitation etc.) and alert description. These sequences are then used to train three separate VLMMs, which combine Markov models of different orders. The models are then used to predict the most likely next attack category, description or IP address, given a sequence of observed symbols. The authors also propose a real-time version of their approach, where the models are trained in real-time as more data becomes available, to further improve their performance against new attacks which were not observed before in the training data. Although they achieve good prediction results, a limitation of their approach is that they predict the three alert attributes separately.

Thanthrige et al. [22] introduce a method of alert prediction using Hidden Markov Models (HMMs). They first create a bag of words model using a training dataset of intrusion alerts. This model would contain the important information from each alert, such as source and destination IP addresses, alert category and signature. The vocabulary of the model was then used to create alert clusters using the k-means Clustering algorithm. A cluster contains both IP information as well as an attack description. Finally, a HMM was trained using the alert clusters, which can then be used to predict the next cluster given a sequence of observed clusters.

Sendi et al. [23] propose a method of predicting whether an intrusion is happening in the network

using HMMs. The paper proposes an Alert Severity Modulating approach that increases alert severity exponentially through correlation. The Alert Correlation Matrix (ACM) is used to store correlation strengths between different types of alerts. The enhanced alert severities are then used to train an HMM, whose hidden states denote the state of the network: Normal, Attempt, Progress, and Compromise. For predicting the current status of the network, the model is presented with a sequence of observations in the format of the enhanced severities of currently observed alerts.

3

Prediction of next attacker action

In this chapter, we look into the first research subquestion: *How can the SPDFA be used to predict the next attacker action?* We use attack episodes (described in section 2.4) to denote attacker actions, and we aim to predict the next episode using a sequence of observed episodes. We first discuss the dataset and describe the methodology of our approach. We then describe other prediction methods to which we will compare our method, including a PDFA-based one. Finally, dive into the experiments we have performed in order to evaluate our approach.

3.1. Dataset

The dataset which we will use for this research question consists of traces generated from the CPTC-2018 intrusion alert dataset. The Collegiate Penetration Testing Competition (CPTC) is an annual cybersecurity competition that challenges college students to demonstrate their skills in performing real-world penetration testing and security assessments. An IDS (Suricata) is also deployed in the competition environment, and the generated intrusion alerts are collected and made publicly available for research purposes. The dataset which resulted from the 2018 competition contains a total of 331,554 intrusion alerts generated by 6 teams, over a duration of 10 hours. One downside of this dataset is the fact that the intrusion alerts which have been triggered are not labelled as true or false positives. In a real-world SOC, all of the alerts are investigated by an analyst, who ultimately decides if the alert is a real threat or just benign behaviour which was misclassified as malicious. When creating the dataset, the participants were not asked whether the observed alerts indeed corresponded to their performed actions, meaning that it might contain false positives.

Since we are interested in predicting the next attacker action rather than the next alert, we first need to transform the alerts into episodes. For this purpose, we use the same process described in 2.4 to generate traces. As a reminder, the intrusion alerts are first filtered by removing duplicates that occur within a 1-second interval as well as informative alerts, and grouped on an attacker-victim basis. The alerts are also augmented with the attack stage and the targeted service. They are then aggregated into attack episodes, using their frequency within a time window, for each given attack stage. Finally, the episodes are time-sorted and used to create the trace file, where every line corresponds to an episode sequence from an attacker against a certain victim. A trace consists of multiple symbols, where each symbol is in the format `attack stage|most targeted service`, based on the episode from which it has been created. An example of a trace is `vulnD|http rPrivEsc|http remoteexp|http dManip|http`, illustrating that an attacker started with vulnerability discovery, followed by privilege escalation and remote service exploit, finishing with data manipulation, using services which run over HTTP. Each trace is written in reverse chronological order because we are interested in learning a suffix-based model which will bring high-severity actions into focus.

Table 3.1 shows the result of each stage of the pipeline when transforming alerts to traces, which ultimately results in 555 traces (384 unique traces) and 148 unique symbols. To give some insight into the dataset, we calculate the frequency distribution of the length of the traces, which can be seen in figure 3.1. Traces of length 3 to 14 are considerably more frequent compared to others, the most common lengths being 3 to 6, where 61.9% of the total traces are contained.

Table 3.1: Number of alerts, episodes and traces resulting from the CPTC-2018 dataset

	Raw alerts	Alerts after pre-processing	Episodes	Episode Sub-sequences	Traces
#	331,554	71,126	5022	739	555

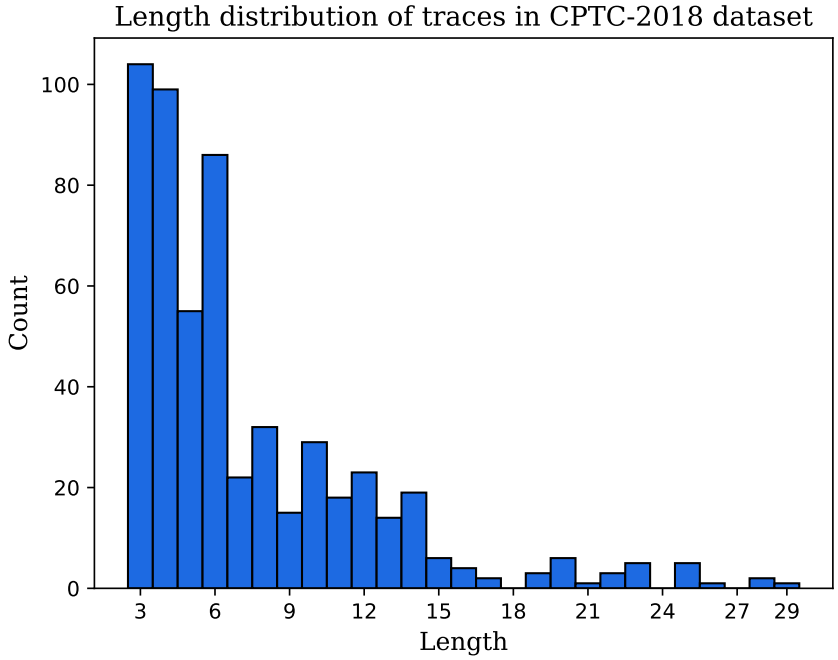

Figure 3.1: Trace length distribution of traces generated using CPTC-2018 dataset. A considerable part of the traces (344, or 61.9%) are contained within lengths 3 and 6.

The frequency distribution of the severity of the first and last actions of these traces can be seen in figures 3.2 and 3.3 respectively. It can be seen that the first action in a sequence is almost always guaranteed to be a low-severity one. On the other hand, in the last action severity distribution, we can see that most of the actions are also low severity. Previously we assumed that attacker actions eventually progress to medium and high severity ones, such as data exfiltration. This distribution can indicate that an attacker can use some hosts as initial devices to establish a foothold in the network, before moving to others where he can perform more critical actions. Thus, the sequences for some hosts will only contain low to medium-severity actions.

3.2. Methodology

Given a sequence of observed attacker actions, we want to use the SPDFA to predict the next action, as the SPDFA contains a summarised version of the past observed attacker behaviour. Having a set of traces, we use Flexfringe to generate the SPDFA, using the same configuration as described in section 2.4. The states resulting from training on the data can be divided into two: states included in the main model and the sink states. If during training Flexfringe finds infrequent sequences from which it could not effectively ”learn” the behaviour, they will be placed in the sinks. However, when predicting the next action, we are interested in examining all the behaviours found in the training data, even those which are infrequent. To this extent, we merge all sink states with the main model and use the resulting SPDFA for prediction. Since all incoming transitions of a state will always have the same symbol (as can be seen in figure 3.4 left), we augment each state with its corresponding symbol.

Due to the fact that we have reversed the traces and learned a suffix-based model, when traversing the SPDFA from the starting (root) state towards other states, the visited states will be in reversed chronological order (top →bottom means future →past). However, since we are interested in predicting future events based on events that occurred in the past, we want to traverse the SPDFA in reversed

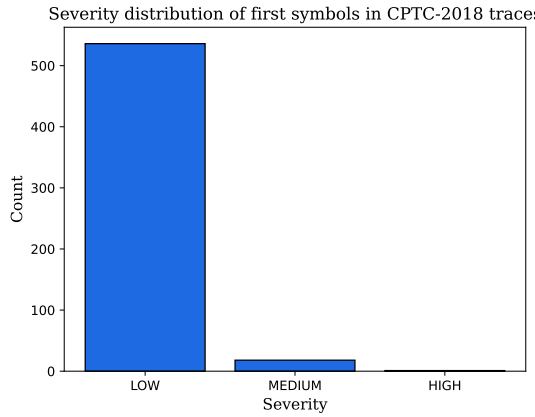

Figure 3.2: Severity distribution of first actions of traces generated using CPTC-2018 dataset. As expected, most are low severity.

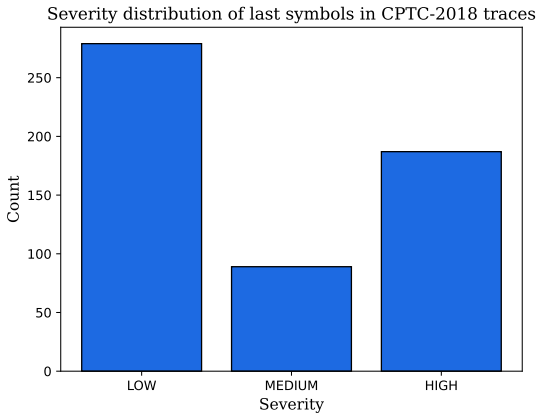

Figure 3.3: Severity distribution of last actions of traces generated using CPTC-2018 dataset. Even though the last action in a chain of symbols is supposed to be of higher severity, we can see that most of the actions are low severity.

order, from leaf or intermediary states towards the root one. To achieve this, we reverse all of the transitions, creating a *Reversed SDPFA (rSPDFA)* (note that a reversed SPDFA does not equal a PDFA), which has the following properties:

1. There is no single root/starting state, instead, the states which previously did not contain any outgoing transitions can be considered the new root states
2. The state machine becomes non-deterministic since a state can now have multiple transitions with the same symbol
3. The transition probabilities are recalculated as per formula 3.1, which will be discussed in more detail below

An example of an SPDFA pre and post-inversion can be seen in figure 3.4. Each state has a state ID and total count of traces (first line), followed by the number of traces that continue and the number of traces that start in this state (second line). The transition labels illustrate the transition symbol, the transition count and the transition probability. After the inversion, the state machine becomes non-deterministic, because state 4 now has two outgoing transitions with the same symbol.

For every transition in the SPDFA, with source state S, destination state D, and transition symbol $symb$ the new probability is calculated as:

$$P_{trans}(D \to S) = \frac{count(S \to D)}{totalcount(D)} \tag{3.1}$$

where $count()$ is the count of the transition from S to D, and $totalcount()$ is the number of total occurrences of a state D. For example in figure 3.4, for the transition from state 2 to state 4 with symbol `vulnD|http`:

$$P_{trans}(4 \to 2) = \frac{count(2 \to 4)}{totalcount(4)} = \frac{30}{42} = 0.71$$

As mentioned in section 2.2, every state also has a starting probability, which symbolizes the probability of a sequence starting from this state. When inversing the SPDFA, it is also recalculated as

$$P_{start}(S) = \frac{startCount(S)}{totalStartCount(symb)} \tag{3.2}$$

where S is a state, $symb$ is the symbol corresponding to that state, $startCount()$ is the number of traces that start in that state, and $totalStartCount()$ for a symbol *symb* is defined as $\sum_{state \in states} startCount(state)$, where $symbol(state) = symb$. For example, in figure 3.4 if we were to calculate it for state 4:

$$P_{start}(4) = \frac{startCount(4)}{totalStartCount(vulnD|http)} = \frac{42}{42} = 1.0$$

The starting probability for this state is 1 because this is the only state whose symbol is `vulnD|http`, respectively all of the traces in the training data which for which this symbol is the first one in the sequence will also start from this state.

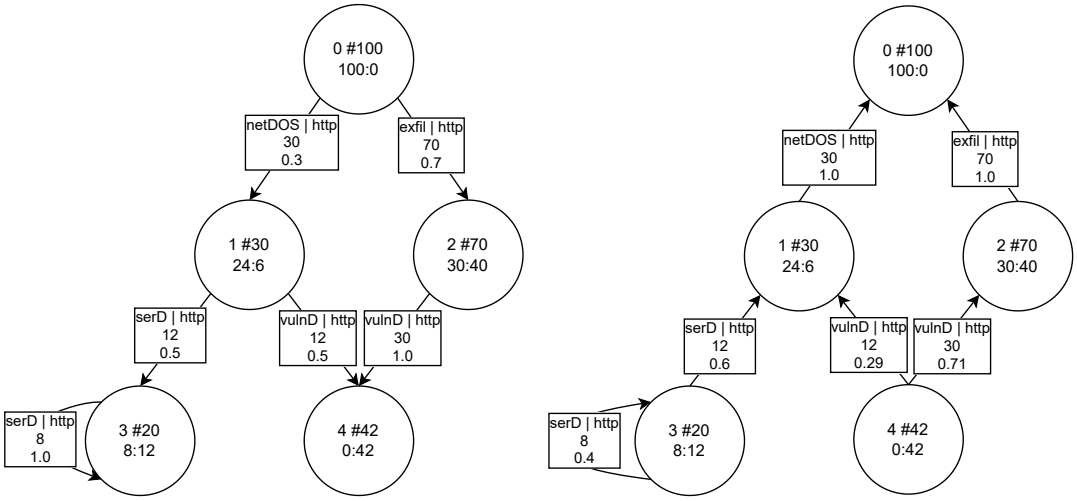

Figure 3.4: An example of a SPDFA before(left) and after(right) inversion. The transition probabilities change after the inversion, re-calculated using the formula 3.1. The state machine also becomes non-deterministic after the inversion.

3.2.1. Path finding algorithm

In order to predict the next action given a sequence of observed actions, we need to search the rSPDFA and find all of the sequences of transitions which have the same symbols in the same order as the observed actions. The same sequence can appear in different contexts, and can result in multiple sequences of visited states, thus it is important to examine all of them to calculate an accurate probability distribution of the next actions. Having found all such sequences of states, we then can look at the symbol of the outgoing transitions corresponding to the last state in the sequence, and calculate the probability distribution of the next actions using all such symbols.

This section describes how the rSPDFA is used to find all such sequences of states, which will be further used for predicting the next action. We use the term *path* to describe a possible sequence of states that can be reached by exploring transitions in the rSPDFA given a starting state. Because of the fact that the SPDFA becomes non-deterministic after the reversal of the transitions, there will be multiple possible paths for a given input trace. The core idea of the algorithm is to explore all possible paths non-deterministically, given a starting state and an input sequence. Each symbol from the input sequence will correspond to one transition in the rSPDFA, thus the algorithm will move from state to state until either there are no more symbols in the input trace or a state with no outgoing transitions is reached.

Since the rSPDFA is similar to an acyclic graph, we adopt a depth-first search approach for finding the paths. As mentioned, we start with a starting state and the first symbol from the input sequence. We add the current state to the path and examine the symbols of the outgoing transitions, to decide whether or not to to visit them based on our chosen visiting strategy. We define the following strategies for visiting the next states based on the transition symbol and the next symbol from the sequence:

1. Strategy 1 - full symbol match - we will visit the next states only if the transition symbol matches the next symbol of the trace
2. Strategy 2 - attack stage match - we will visit the next states only if the attack stage of the transition symbol matches the attack stage of the next symbol of the trace
3. Strategy 3 - any symbol match - we will visit the next states regardless of the next symbol of the trace

As can be seen, each strategy is less restrictive than the previous one and will result in more visited paths. The intuition behind this was the nature of the cyberattacks, which are constantly evolving. For

example, let us assume our training set contained the sequence of actions `service discovery, vulnerability discover, vulnerability exploitation`. If the attacker were to perform these actions in a different order during a cyberattack, or even change the second action to something else, for example `brute force credential gain`, the first and second strategies would result in no prediction, due to not finding any viable paths. On the other hand, strategy 3 can continue exploring the rSPDFA even in case of input symbol mismatch, and eventually output a prediction.

After deciding on the next states, we visit them one by one and create a new path for each one we visit horizontally. Once in the next state, we once again examine the transition symbol and the next symbol from the input trace. We repeat this process until either a state with no outgoing transitions is reached, or when there are no more symbols in the input sequence. Finally, we return a list of all possible paths.

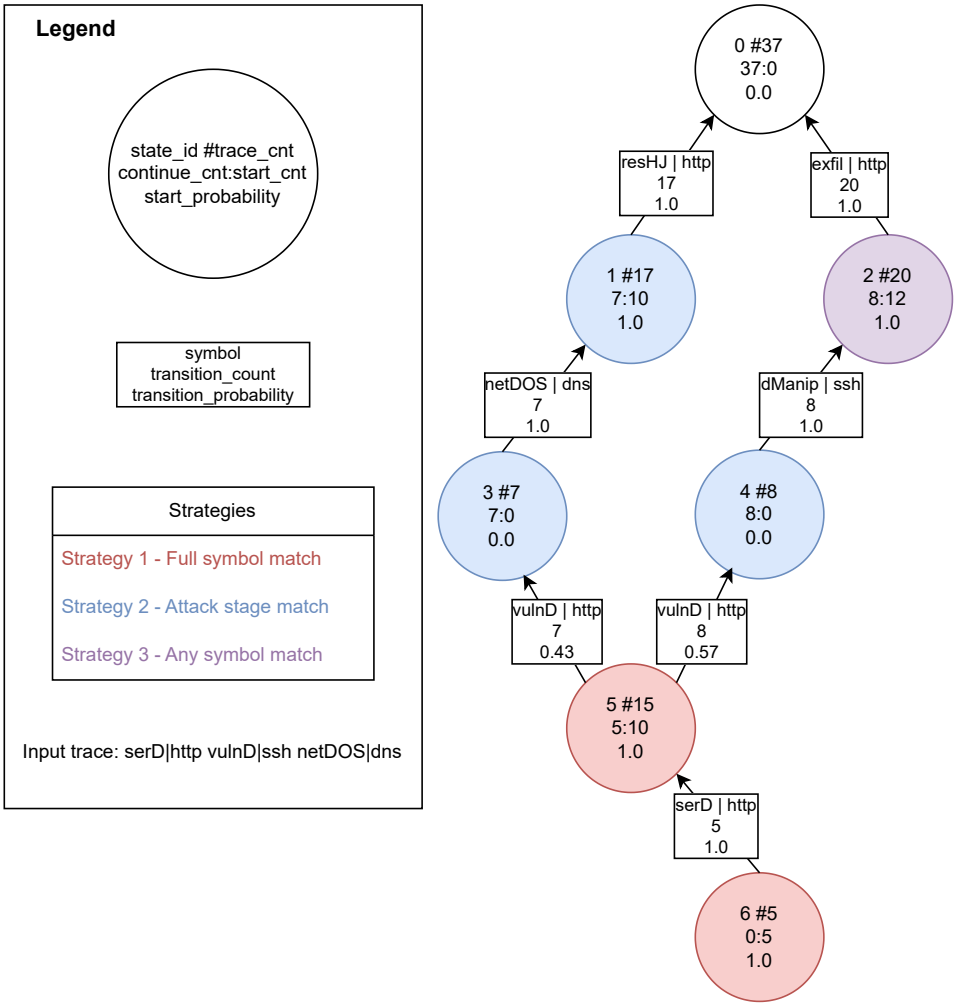

Figure 3.5: An example of a rSPDFA and the resulting paths per strategy for the input trace `serD|http vulnD|ssh netDOS|dns`. The red states indicate the states which were visited while using the full symbol match strategy. When using the attack stage strategy, the red as well as blue states are visited. The any-symbol match strategy has the most relaxed requirements, thus it visits the states from the previous strategy, as well as the purple ones. The full symbol match strategy returns the path `6 -> 5` since it cannot explore past state 5 due to the symbol not matching with the input one. The attack stage match strategy returns the paths `6 -> 5 -> 3 -> 1` and `6 -> 5 -> 4`, while the any symbol one returns `6 -> 5 -> 3 -> 1` as well as `6 -> 5 -> 4 -> 2`.

Figure 3.5 showcases how the algorithm would run for each different chosen strategy, given an rSPDFA and an input trace. The first symbol from the input is `serD|http`, thus we start in state 6 (bottom). Regardless of the strategy used, the first state will always have at least one outgoing transition with the matching symbol, thus we visit state 5 and create a path containing the sequence `6, 5`. We then examine the next transition symbol, which is `vulnD|http`, as well as the next input one,

which is `vulnD|ssh`. We can see that the attack stage matches, while the service does not. When using strategy 1, we cannot visit any more states, thus we return the previously mentioned path. Now let us assume we are using Strategy 2 and matching the attack stage, in which case we will visit both states 3 and 4. We randomly choose 3 as the next state, our path becoming `6, 5, 3`. We once again compare the input symbol with the one from the outgoing transition, and since it matches, we continue to state 1. Since we have no more symbols in the input, we add the current path `6, 5, 3, 1` to the lists of found paths. Next, we return back and visit state 4. The next symbol from the input is `netDOS|dns`, while the one from the transition is `dManip|ssh`, so neither the attack stage nor the service match, so we add the path `6, 5, 4` to the list of results and return. In this case, there are no more states left to explore for the attack stage match strategy. However, if we were using any symbol match strategy, we could also visit state 2, and return the path `6, 5, 4, 2`. These paths can also be seen in the figure, the red states corresponding to Strategy 1, red + blue to Strategy 2, and red + blue + purple to Strategy 3.

To speed up the path-finding process for every subsequent use, we also implement memoization. We use a dictionary where the key is `state identifier, input trace`, and the value is the list of paths corresponding to this combination. Before returning the found paths, we store them in this dictionary. Thus, on subsequent calls, if the current state has already been visited in the past with the same input trace, we can return the resulting paths without having to find all of them again. The complete algorithm for path finding is illustrated in algorithm 1.

Algorithm 1: Find possible paths for a given starting state and a trace

```
findPathsDFS(state, inputTrace, strategy, dp)
Data: paths = [];
key = (state, inputTrace);
nextStates = [];
if inputTrace is empty :
    /* return a list containing only this state                    */
    return [state];
if state has no transitions :
    return [state];
if key in dp :
    return dp[key];
transitionSymbol = state.transitions.symbol;
inputSymbol = inputTrace[0];
if strategy == FULL_MATCH :
    if transitionSymbol == inputSymbol :
        nextStates.append(state.transitions.states)
elif strategy == ATTACK_STAGE_MATCH :
    if transitionSymbol.attackStage == inputSymbol.attackStage :
        nextStates.append(state.transitions.states)
else:
    nextStates.append(state.transitions.states)
for nextState in nextStates do
    for foundPath in findPathsDFS (nextState, inputTrace[1:], strategy, dp) do
        /* append the current state to the path, and add it to the list
           of found paths                                           */
        paths.append([state] + foundPath);
dp[key] = paths;
Result: paths
```

3.2.2. Prediction algorithm

This section describes the algorithm for predicting the next action in a sequence given a list of found paths. Given an input trace for which we want to predict the next action, we first select the first symbol of the trace and find all states where that symbol occurs, which will form the set of starting states. We can start from intermediary as well as leaf states because our model has Markovian properties, meaning that given the current states (present), future states only depend upon them, and not on the past ones.

After using the path-finding algorithm on each starting state and obtaining all possible paths, we can calculate the probability of each path, which is done by multiplying the transition probabilities of all states for a given path. This product is also multiplied by the starting probability of the first state, due to the fact that we are also interested in the probability of a trace starting in a particular state.

For strategies 2 and 3, because we explore paths which contain symbols that are different from the ones in the input trace, either based on the attack stage or disregarding the input symbol completely, we would like to give a significantly higher priority to the transitions which match the entire symbol from the input trace, and a slightly higher priority to those which match the attack stage. The justification behind this is that when calculating the probabilities of each path, the paths which contain symbols closest to the input trace should have a higher probability compared to others. For this purpose, we multiply the transition probability with a chosen *factor* if it matches the attack stage, and with *factor * 2* if it matches the entire symbol. The value of this factor will be further explored in section 3.3.3. The formula 3.3 illustrates how the probability of a path is calculated.

$$P_{path}(path) = P_{path}(S_1..S_N) = P_{start}(S_1) * \prod_{i=1}^{N-1} P_{trans}(S_i \rightarrow S_{i+1}) * getProbFactor(S_i \rightarrow S_{i+1}, strategy) \tag{3.3}$$

Since we multiply some of the probabilities with a probability factor based on the strategy, we need to normalize them before we proceed. To achieve this, we calculate the sum of all path probabilities and divide each one by that sum. As a result, all of the path probabilities sum up to 1.

The traversal results in both valid paths and invalid paths. Invalid paths are those for which the trace could not be explored completely, either because a state with no outgoing transitions was reached, or because of the traversal strategy (when the transitions symbols did not match the input ones). Thus, we filter them out, calculate the sum of the probabilities of these paths, and redistribute the resulting probability equally to the remaining valid paths. This can be seen in algorithm 2. We consider the next action of a path as the symbol of the outgoing transitions of the last state of a given path. We then calculate the probability mass distribution of all possible next actions,

$$P_{nextAction}(symbol) = \sum_{\substack{p \in paths \\ nextAction(p)=symbol}} P_{path}(p)$$

Finally, using this distribution we choose the action with the maximum probability, which represents the prediction result. The entire process can be seen in Algorithm 3.

Algorithm 2: Redistribute probabilities of invalid paths to valid ones

Input: $paths, pathProbs$

1 $validPathsProbs, invalidPathsProbs = splitPaths(pahts, pathProbs)$;
2 $s = sum(invalidPathsProbs)$;
3 $redisAmount = s \,/\, length(validPathsProbs)$;
4 **for** $prob$ *in* $validPathsProbs$ **do**
5 $\quad prob \mathrel{+}= redisAmount$

Result: $validPathsProbs$

Let us exemplify this process using figure 3.5, any symbol match strategy, a probability multiplication factor of 2 and the same input trace `serD|http vulnD|ssh netDOS|dns`. The path finding algorithm returns the two possible paths: `(6, 5, 3, 1)` (path 1) and `(6, 5, 4, 2)` (path 2). We first calculate

Algorithm 3: Find the next attacker action for a given trace

Input: $trace$
Data: $startNodes$ = getStates($trace$[0]) ;
$paths$ = [];
1 **for** *every* $startNode$ *in* $startNodes$ **do**
2 $paths$.append(`findPathsDFS` ($startNode$, $trace$))
3 $pathProbs$ = $getPathProb(path)$ for $path$ in $paths$;
4 $normalizedProbs$ = $normalize(pathProbs)$;
5 $validPathsProbs$ = $redistributeProbabilities(paths, normalizedProbs)$;
6 $nextActionsPD$ = {} ;
7 **for** $path$, $pathProb$ *in* $(validPaths, validPathsProbs)$ **do**
8 na = $getNextAction(path)$;
9 $nextActionsPD[na]$ += $pathProb$;
Result: $nextAction$ = max($nextActionsPD$)

the path probabilities of each path:

$$P_{path}(path1) = P_{start}(6) * P_{trans}(6 \to 5) * 4 * P_{trans}(5 \to 3) * 2 * P_{trans}(3 \to 1) * 4 = 13.76$$

$$P_{path}(path2) = P_{start}(6) * P_{trans}(6 \to 5) * 4 * P_{trans}(5 \to 4) * 2 * P_{trans}(4 \to 2) * 1 = 4.56$$

It can be seen that the probability was multiplied by 4 where the full transition symbol matched the input one, and by 2 when the attack stage of the transition matched the attack stage of the input symbol. Next, we normalize them by calculating their total sum $S = 13.76 + 4.56 = 18.32$ and dividing each one by this sum. The new path probabilities are:

$$P_{path}(path1) = P_{path}(path1)/S = 13.76/18.32 = 0.75$$

$$P_{path}(path2) = P_{path}(path2)/S = 4.56/18.32 = 0.25$$

In this case we have no invalid paths because both of them have the same number of transitions as the number of symbols in the input trace. Finally, we calculate the probability distribution of the next actions using the valid paths. The next action of path 1 is `resHJ|http` (Resource Hijacking over HTTP),

$$P_{nextAction}(resHJ|http) = \sum_{\substack{p \in paths \\ nextAction(p) = resHJ|http}} P_{path}(p) = P_{path}(path1) = 0.75$$

The next action of path 2 is `exfil|http` (Data Exfiltration over HTTP), so the probability is

$$P_{nextAction}(exfil|http) = \sum_{\substack{p \in paths \\ nextAction(p) = exfil|http}} P_{path}(p) = P_{path}(path2) = 0.25$$

The maximum amongst these two is `resHJ|http`, thus we return it as the predicted next action.

3.2.3. Baseline prediction methods

Random guess

For evaluation purposes, we also define two baseline prediction methods. The first one is based on random guessing, and as the name suggests, its accuracy is calculated as if the prediction results were to be random guesses from the space of possible predictions. So the accuracy for this method can be defined as:

$$Accuracy_{randomGuess} = \frac{1}{\text{\# of distinct symbols}}$$

Frequency-based baseline algorithm

The second baseline method works in a purely probabilistic fashion and is shown in algorithm 4. Using the training traces, we examine all `(symbol, nextSymbol)` tuples, and calculate the frequency of every possible *nextSymbol* for a particular *symbol*. When predicting the next attacker action for a particular trace, we select the last symbol in the trace and using the calculated frequencies, we consider the next action to be the symbol with the highest rate of occurrence for the last symbol.

Algorithm 4: Baseline Frequency-based algorithm

Input: $trainingTraces$, $testTrace$, $symbolList$
Data: $frequencies$ = dictionary()
for $symbol$ *in* $symbolList$ **do**
  for $nextSymbol$ *in* $symbolList$ **do**
    $frequencies[symbol][nextSymbol] = 0$
for $trace$ *in* $trainingTraces$ **do**
  for $(symbol, nextSymbol)$ *in* $trace$ **do**
    $frequencies[symbol][nextSymbol]$ += 1
$lastSymb$ = $testTrace$[-1]
$nextAction$ = max($frequencies[lasySymb]$)
Result: $nextAction$

3.2.4. PDFA-based prediction algorithm

We also want to compare the performance of the SPDFA-based prediction algorithm with a PDFA-based one. For this purpose, we train a PDFA using the same training data (this time however the traces are not reversed). The resulting PDFA can then directly be used for our purposes, without needing any further changes. For this purpose, we implement a simple deterministic traversal algorithm for finding the next symbol using an input trace, which can be seen in algorithm 5. We start in the root state of the automaton, and while iterating through the symbols of the input trace, we move state by state using the transitions of each state. The next state is decided based on the current symbol from the input trace. If the symbol cannot be found in the transitions of a state, we choose the transition with the maximum count and the state it corresponds to. We do this until the end of the input trace is reached, and we end up in the final state. We then examine the outgoing transitions of the final state. We pick the symbol corresponding to the transition with the maximum count and return it as the prediction result, along with its probability. The probability is calculated as $P = \frac{\text{transition count}}{\text{sum across all transitions}}$. If the traversal ever reaches a state with no outgoing transitions, we return fail and consider this a failed attempt.

Algorithm 5: PDFA-based prediction algorithm

Input: $pdfa$, $trace$
Data: $state$ = $pdfa[root]$
for $symbol$ *in* $trace$ **do**
  if $state.transitions$ == *[]* :
    return Fail
  else:
    if $symbol$ *in* $state.transitions$:
      $state$ = $state.transitions[symbol]$
    else:
      $maxCountSymbol$ = maxCount($state.transitions$)
      $state$ = $state.transitions[maxCountSymbol]$
$finalState$ = $state$
Result: $nextAction$ = maxCount($transitions$ of $finalState$)

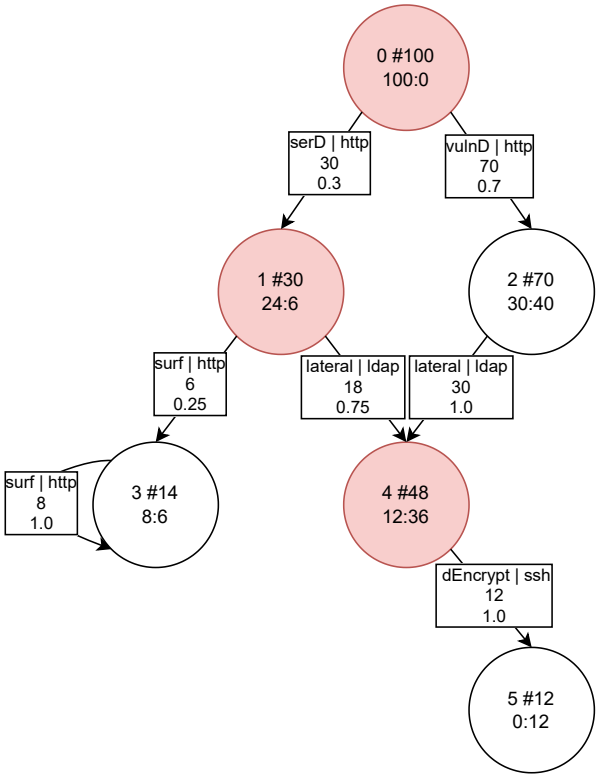

Figure 3.6: An example of a PDFA and the path the algorithm would take (highlighted in red) for the input trace `serD|http privEsc|ssh`, the predicted action being `dEncrypt|ssh`.

Figure 3.6 shows an example of a PDFA and the path the algorithm would take for the input trace `serD|http privEsc|ssh`. We start in the root state and examine the first input symbol `serD|http`. Since we could find it in the transitions, we move to state 1 and examine the next input symbol, `privEsc|ssh`. Since there is no outgoing transition with this symbol, we pick the one with the maximum count, which is `lateral|ldap`, and move to state 4. Now the input trace is finished, so we just pick transition with the maximum count from the last state in the path, which is `dEncrypt|ssh`, and return it as the predicted action, along with the probability of 1.0.

3.3. Experiments and Results

In this section, we describe the experiments we performed in order to evaluate our prediction algorithms, as well as the results we obtained. For each experiment, we first describe its methodology and what we aimed to evaluate by performing it, following with the results we obtained and finishing with a small discussion over those results. The experiments aim to evaluate different aspects of our method, such as accuracy and execution time, and which factors are they influenced by. We also test our method with traces which only contain the attack stage, without the service, to find out whether that will improve accuracy. Finally, we compare our method to a PDFA-based one by testing them in different scenarios and configurations and summarize our findings.

3.3.1. Experimental setup

When performing initial experiments, we observed that the run-time of strategies 2 and 3 increases exponentially by the length of the input trace (this is examined in more detail in section 3.3.5). This made running experiments very time-consuming, as for some inputs, strategy 3 could take tens of minutes to run. Due to this fact, and because we want to compare all our strategies on the same test data, we decided to limit the length of testing traces to 6 symbols. Due to the small number of samples, we also decided to use K-fold cross-validation. This would allow us to get a better overview of the performance of our methods while excluding the input bias.

Thus, when creating the training and testing datasets, we first remove duplicate traces, as well as

traces which have a length longer than 6 symbols. From this subset, using K-fold cross-validation, we select random a subset of traces which will represent the testing dataset. The rest of the traces from the initial trace set will then become the training set. To be able to compare the accuracy of our approach, we would need to know the true next symbol of an input trace, in order to compare it to our predicted symbol. To achieve this, we split each testing trace: the last symbol in the trace will be the true next action, while the rest of the trace will be used as input to the prediction algorithms.

In terms of metrics, we will use the following:

- accuracy = $\frac{\text{correct prediction}}{\text{total predictions}}$, which reflects whether the predicted next symbol matches the true next symbol
- top-n accuracy = $\frac{\text{correct prediction is in top-N candidates}}{\text{total predictions}}$, reflects whether the correct prediction is within top-n predictions; for example, top-1 accuracy is equal to normal accuracy
- attack stage accuracy = $\frac{\text{correct prediction of the attack stage}}{\text{total predictions}}$, that reflects whether the attack stage of the predicted next symbol matches the attack stage of the actual next symbol
- No path found rate = $\frac{\text{no path found count}}{\text{testing set size}}$. This describes for what percentage of the input traces the algorithm could not find a single viable path in the SPDFA, which can happen either due to strategy (e.g. full match strategy and no paths were found in the SPDFA which matched the input symbols entirely) or when all of the found paths were invalid (e.g. all paths were too short and did not match the input sequence length).
- execution time - average execution time in seconds of prediction for an input sample

3.3.2. K-fold cross-validation - K selection

This experiment aims to study the impact of K on the accuracy and find the optimal K for K-fold cross-validation, which we could then use for our future experiments. We run every method with a varying K from 5 to 15 and check the accuracy. As can be seen in figure 3.7, the accuracy of the best-performing strategies is not majorly impacted by a higher K value, and there are minor variations in accuracy, the best accuracy is achieved by K=13. This could be explained by the fact that even a K as low as 5 captures a good overview of the testing set. Therefore a higher K value will not guarantee a better overview of the performance. Still, based on our results, we choose a K=13 and use it for further experiments.

3.3.3. Finding the optimal multiplication factor for strategy 2 and 3

Methodology

As mentioned before, in strategies 2 and 3 we are less strict when selecting viable paths because we also the match attack stage or any symbol, regardless of the input one. We would still like to give higher priority and thus probability to the paths where more symbols match the original input sequence. For this purpose, we introduced a multiplication factor, by which the transition probability is multiplied every time the attack stage matches (*prob * factor*) or the entire symbol matches (*prob * factor * 2*).

This experiment aims to find the optimal multiplication factor which would maximize the accuracy of these strategies. We choose a range of factors from `1` to `95`. We then run our method with the varying factors and calculate the accuracy of each run.

Results

The result can be seen in figure 3.8. We can see that the biggest increase in accuracy happens at the beginning until the factor reaches a value of 10. Afterwards, a minor increase in accuracy is seen, which stops altogether at 45 for strategy 2 and 75 for strategy 3. The increase in accuracy obtained by increasing the factor demonstrates that some paths resulting from these strategies, which contained symbols not matching the ones from the input trace, had a higher probability, and thus had a higher impact on the next action prediction. By artificially increasing the probabilities of the transitions with symbols matching the input trace, we helped prioritize such paths which led to a higher prediction accuracy. A larger increase in accuracy is seen for strategy 3 compared to strategy 2. This could indicate that most of the paths obtained by accepting the attack stage of a transition already had a

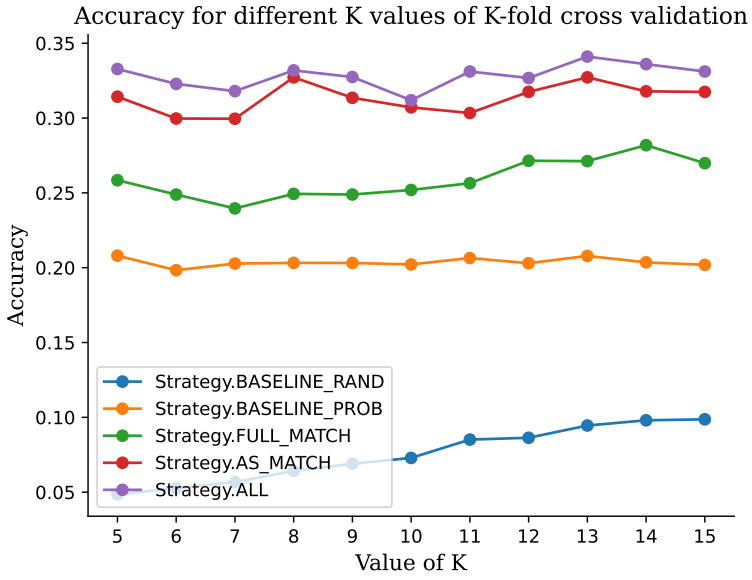

Figure 3.7: Accuracy of each strategy for a different K value used for K-fold cross-validation testing. It can be seen that the only strategy benefiting from higher K values is the baseline random selection. `BASELINE_RAND` corresponds to the baseline random guess method, `BASELINE_PROB` to baseline frequency based, `FULL_MATCH` to full symbol match, `AS_MATCH` to attack stage match and `ALL` to strategy 3, which is any symbol match.

correct result, and increasing the factor helped in fewer cases compared to strategy 3. Nevertheless, we choose *factor=75* for further experiments, as it is the first factor to obtain the best results for both strategies.

3.3.4. Evaluation of the proposed SPDFA-based prediction methods

Methodology

This experiment aims to evaluate the prediction performance of our strategies which use the SPDFA against both the baseline methods. As mentioned in section 3.3.1, we limit the test trace size to 6 symbols, out of which 5 we use as input and the 6th one as the true next action. We use a multiplication factor of 75 for strategies 2 and 3. We use 13-fold cross-validation to run with different variations of the test set while comparing the predicted symbol to the true one. If the given method fails to find a single viable path for an input trace, or if the first symbol cannot be found in the SPDFA, we count that as a failed attempt and increment the ”no path found” count. We also measure the execution time for each run on the test set, excluding the baseline methods, because we consider that in that case, it can be negligible (< 0.001 seconds).

Because our method returns a probability distribution of the next actions, we are interested in studying the number of times when the correct prediction is within the top 3 actions with the highest probability from this distribution. To verify this, we also calculate the top 2 and top 3 accuracy for each strategy.

Results

The results for all the methods can be seen in table 3.2. The best-performing method in terms of accuracy is the any symbol match strategy, with an accuracy of 33.71 % and attack stage accuracy of 42.05 %. This method also has the lowest no-path found rate among the SPDFA-based models, having not found a path for only 0.48 % of the traces. That being said, strategy 3 is also the worst in terms of execution time, with an average of 1.5873 seconds per test set. This is explained by the fact that this strategy explores every path nondeterministically regardless of the transition symbols, thus the execution time grows exponentially with the length of the input.

We can see that the first strategy, which explores a state only if the full symbol matches the input one, has the highest no path found rate, at 27.14 %. This indicates that this strategy is indeed too restrictive, and does not perform well in cases where the test trace was not seen in training data. On the other hand, strategy 2, which only explores a state if the attack stage matches, is a good middle

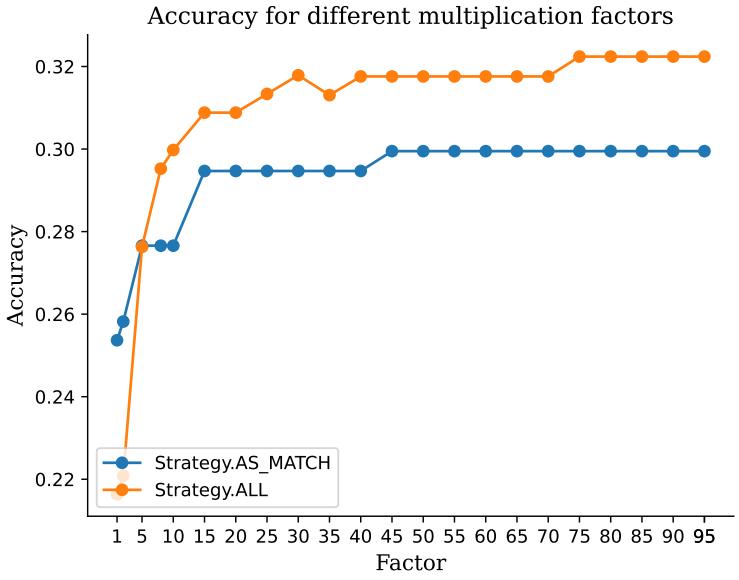

Figure 3.8: Accuracy for different multiplication factors for strategies 2 and 3. Increasing the factor also increases the accuracy, a more drastic increase being seen at the start, which then plateaus at 45 for strategy 2 and at 75 for strategy 3.

Table 3.2: Results for all strategies using 13-fold cross-validation

Strategy	**Metric**			
	Accuracy (%)	**AS Accuracy (%)**	**No path found rate**	**Execution time (seconds)**
Baseline random guess	0.67	4.76	x	x
Baseline probability	20.75	29.52	x	x
Strategy 1 - Full symbol match	28.16	35.52	0.2714	**0.0025**
Strategy 2 - Attack stage match	33.22	41.11	0.1049	0.0432
Strategy 3 - Any symbol	**33.71**	**42.05**	**0.0048**	1.5873

ground between accuracy and execution time, as its accuracy is only 0.49 % lower than that of strategy 3, while its execution time still being under one second.

Table 3.3 shows the top 1, top 2 and top 3 accuracies and attack stage accuracies per strategy. It can be seen that accuracy improves when we take into account the top 2 or top 3 actions, the best one being obtained by strategy 3, with 54.38 % accuracy and 61.73 % attack stage accuracy. This means that for 54.38 % of the testing data, the correct prediction could be found within the top 3 predictions, even though it was not the symbol with the maximum probability in the distribution. This could also mean that the method suffers from predictive uncertainty, which can also be related to the non-deterministic nature of the model, and the fact that there exist multiple paths for the same input sequence. The probability distribution of the next actions is calculated using the probabilities of the found paths, which in turn depends on the frequencies of the sequences from the training data. Thus the more times a sequence is present in the data, the higher probability the resulting path will have, due to the states which are part of it having a bigger state count. Consequently, the next action of this path will also have a higher probability. Therefore, it is possible that our method, by choosing the next action with the maximum probability from the distribution, ignores other paths, which correspond to less frequently seen sequences in the training data, yet still represent a viable attack strategy.

3.3.5. Runtime evaluation based on SPDFA size and input length

In this section, we evaluate the execution time of our strategies which use the SPDFA, by examining its dependence on both the size of the training set and the length of the input trace.

Table 3.3: Top 3 accuracy and attack stage accuracy for all strategies. It can be seen that accuracy improves when taking into account the top 2 actions, and further improves for the top 3 actions, reaching a maximum of 54.38 % for any symbol match strategy.

Strategy	Accuracy (%)			AS Accuracy (%)		
	Top 1	Top 2	Top 3	Top 1	Top 2	Top 3
Strategy 1 - Full symbol match	26.75	36.87	43.32	33.68	42.84	47.00
Strategy 2 - Attack stage match	31.36	45.19	51.15	39.19	51.16	58.05
Strategy 3 - Any symbol	33.17	45.67	54.38	41.45	53.05	61.73

Methodology

The first experiment aims to examine the impact of the size of the training set on the performance of the model. We generate the SPDFA using training sets of increasing length, starting with 300 traces and ending with 500 traces, with a step of 25. We test our methods using a random trace of 5 symbols while measuring the execution time of finding the next action for that input trace. We also write down the size of the SPDFA for every trace count, in terms of the number of states and transitions.

With the second experiment, we evaluate the impact of the length of the input on the execution time. We generate the SPDFA using 500 randomly selected training traces. Subsequently, we execute each strategy using input traces of varying lengths, from 2 symbols to 7 symbols. We have chosen this range as we believe it accurately captures the differences in execution time and allows us to make the necessary conclusions for this experiment.

Results

The results for the first experiment can be seen in figure 3.9, where the execution time increases linearly to the training set size. This can be explained by the fact that the more traces used for learning the SPDFA, the bigger the SPDFA will become, thus increasing traversal times. Table 3.4 confirms this, showing that the more traces are used for training, the more states and transitions the SPDFA will have. This increase however is very dependent on the training data, as more traces can also mean more opportunities for the model to merge states with similar contexts.

The results for the second experiment can be seen in figure 3.10. Strategies 2 and 3 are the most susceptible to the increase in the input size, as they are able to explore more paths compared to strategy 1. We can also see that the runtime is more dependent on the input length than on the size of the training set, as the execution time increase per additional input symbol is more significant.

Table 3.4: Number of states and edges in the SPDFA depending on the length of the training set. The number of states as well as edges in the SPDFA increases linearly with the number of traces used for training.

Training set length	300	325	350	375	400	425	450	475	500
SPDFA States	157	162	166	168	172	178	183	190	194
SPDFA Transitions	526	556	577	601	624	649	677	706	733

3.3.6. Generating traces with attack stage only

Methodology

This experiment aims to examine the prediction performance of an SPDFA which was trained using traces which contain only the attack stages, without the service. The motivation is based on the fact that a security analyst might be interested only in the next action the attacker is going to perform, and not necessarily the targeted service. For this purpose, we create trace files whose symbols contain only the attack stage and use them for training and validation. As a consequence, this reduces the total amount of unique symbols to 21, while also reducing the size of the automaton to an average of 63 states and 242 edges (for 500 traces). We then re-run our SPDFA-based methods using the newly created traces and compare the accuracy, while also removing strategy 2, since now it will be equal to strategy 1 (a full symbol match would be equal to an attack stage match since the symbol only contains the attack stage).

Figure 3.9: Execution time in seconds per different training set size, while testing with an input trace of 5 symbols long, plotted on a logarithmic scale. The execution time increases linearly with the training set length, showcasing the impact of more states and transitions in the automata.

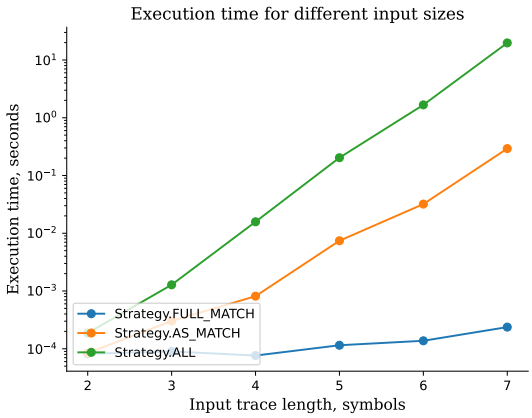

Figure 3.10: Execution time in seconds per different input sizes, while using an SPDFA trained with 500 traces, plotted on a logarithmic scale. Strategies 2 and 3 are the most affected by the increase in input size, as their execution time increases exponentially.

Table 3.5: Results when using traces that contain only the attack stage, thus eliminating the services.

Strategy	Metric		
	Accuracy (%)	**No path found rate**	**Execution time (seconds)**
Baseline random guess	12.26	-	-
Baseline probability	14.45	-	-
Strategy 1	28.02	0.093	**0.002**
Strategy 3	**28.73**	**0.000**	0.160

Results

Table 3.5 shows the results when using traces with only the attack stage. All of the strategies have a decreased accuracy compared to normal traces, however, the best method is still strategy 3. Execution time has also decreased, due to the fact that the SPDFA is now much smaller. The decrease in accuracy might be related to the fact the targeted services offered extra context for the sequences, and the attack scenarios could be differentiated based on them. Without the services and the extra context, the SPDFA could see two different attack scenarios as the same one, resulting in a wrong prediction.

3.3.7. Comparison with PDFA

Methodology

In this section, we look into the performance of a PDFA-based approach for predicting the next attacker action and compare it to our SPDFA-based method. For this purpose, we **do not** reverse the traces, thus we train a prefix-based model using Flexfringe. We evaluate the prediction using the algorithm described in 3.2.4. We then compute the metrics mentioned in section 3.3.1 using 13-fold cross-validation, and compare the results with the best-performing SPDFA-based method, which is strategy 3, any symbol match. Since the PDFA prediction has a very short run-time due to it being deterministic, we repeat the experiment this time removing the input length constraint and write down the results. Moreover, to further compare the performance of both models in different scenarios, we have devised the following experiments:

1. Experiment 1 - Accuracy per severity class - In this experiment, we study the performance of the models when predicting actions of different severity. For this purpose, we evaluate the performance of the models on testing sets where the true next actions have the chosen severity.

2. Experiment 2 - Accuracy on unseen traces - This experiment showcases the ability of the models to generalize and predict the next action for sequences never seen before. We create a testing

Table 3.6: Comparison of best method using SPDFA against the PDFA.

Strategy	Metric			
	Accuracy (%)	**AS accuracy (%)**	**No path found rate**	**Execution time (seconds)**
SPDFA - Strategy 3	**33.71**	**42.05**	**0.004**	1.5873
PDFA	31.33	39.12	**0.004**	**9.25e-05**
PDFA - any input trace length	31.47	36.70	0.018	17.8e-05

Table 3.7: Accuracy of both models for Experiments 1-3. The SPDFA performs better when predicting low and medium severity actions, while the PDFA on high severity ones. Accuracy on unseen traces is lower for both models compared to their accuracy on a mixed set of traces (both seen and unseen). Finally, using a model which does not contain the sinks drastically reduces the accuracy of both models.

	Accuracy (%)				
	Experiment 1			**Experiment 2**	**Experiment 3**
Model	**Low sev.**	**Medium sev.**	**High sev.**	**Unseen Traces**	**Not merging sinks**
SPDFA - Strategy 3	45.6	13.5	27.2	27.0	18.4
PDFA	37.8	5.4	33.7	26.3	17.4

set which contains traces which were not present in the training data, neither as individual traces, nor as part of another longer trace, and test both models with it.

3. Experiment 3 - Accuracy when using models without sink states - As mentioned previously, we merge the sink states with the states from the main model in order for the model to contain all scenarios seen in the training data. We are interested in studying the impact of these merged sink states for prediction accuracy. Thus, in this experiment, we do not merge the sink states, and test the performance on a random set of traces.

Results

As can be seen in table 3.6, the PDFA performs almost as well as the best SPDFA method, achieving an accuracy of 31.33 %, which is only 2.38 % lower than the one achieved by the SPDFA. On the other hand, due to this method being deterministic, the execution time is significantly shorter, with an average of 9.25e-05 seconds per testing set. Furthermore, when testing on traces of any length, we can see a small drop in the attack stage accuracy, while the normal one remains approximately the same, which can be also said about the execution time. The decrease in accuracy might be attributed to the fact that longer sequences are more infrequent (as seen in figure 3.1), and the model has less data to learn from. Thus, unlike the SPDFA non-deterministic methods, this method can be used on traces of any length, while still maintaining a good runtime.

Table 3.7 shows the accuracy per severity for each model. We can see that both models perform best when predicting low-severity actions, which is also the most frequent severity class of the last actions seen in the data. This is followed by high-severity actions and finally medium ones. Based on the fact that the ranking of these results is directly proportional to the severity distribution of the last actions illustrated in figure 3.3, we conclude that it is possible for the training set to impact the performance of both models. Moreover, it can be observed that the PDFA performs better on High severity actions, while the SPDFA achieves higher accuracy on Low and Medium severity ones.

Table 3.7 also shows the accuracy of the models on unseen traces. The accuracy decreases by about 5 % compared to the average one, which means that both models are slightly worse at generalizing and predicting the next action for new traces. When using a model which does not contain the sinks, the accuracy is also seen to decrease. This is explained by the fact that the sinks contain valuable information which is needed for prediction, even if the sequences which they model were infrequent in the training data.

3.4. Discussion

In this chapter, we have implemented an SPDFA-based approach for predicting the next attacker action using an observed sequence of actions as input. Three SPDFA traversal strategies have been

proposed, whose performance as well as execution time has been evaluated in different scenarios. We have also implemented a PDFA-based deterministic method for prediction and compared it to the SPDFA-based one. Given the complexity of the problem at hand, which is a multi-class classification task encompassing 148 distinct classes (symbols), the performance of our method can be regarded as commendable and demonstrates promising results in this challenging context.

Nevertheless, the diminished accuracy of the SPDFA could be rooted in various factors. Firstly, the inherent complexity of our method of finding paths in the rSPDFA, which involves non-deterministically evaluating numerous paths, might lead to the diffusion of probabilistic predictions across potential actions, thereby diluting the confidence in any particular forecast. The exploration of multiple paths can further introduce ambiguities, complicating the resultant probability distribution and possibly misleading the model. Additionally, the automata's state representation might not capture enough information about the past sequences, causing a loss of essential context for accurate predictions. Furthermore, the performance of the SPDFA is highly dependent on the training dataset from which it has learned the sequences.

Further expanding upon the dataset, the size of the training dataset might also be insufficient to capture the full complexity and variety of attacker behaviours. Moreover, the CPTC-2018 dataset contains all of the alerts which have triggered during the competition, including false positives. False positive alerts introduce noise into the dataset, which means that the traces created from these alerts may contain actions that weren't genuinely indicative of an attack progression but were mistakenly flagged. If the models are trained to predict the next action based on sequences containing false positives, incorrect examples are essentially being learned from. Unfortunately, we cannot remove those alerts from the dataset, because only the members of the competition can confirm which intrusion alerts corresponded to their actions and which were mistakenly triggered. Since the competition participants were students, a wide range of skills is likely to be observed in such a scenario. Different attack paths might be taken, more mistakes might be made, or ineffective strategies might be attempted by novices compared to more experienced participants. This diversity can cause a wide variety of sequences to be introduced, some of which may not represent ”typical” attacker behaviours. Various tactics might be tried by participants in a competition setting just to explore or experiment, even if they don't intend to follow through with them. The unpredictability of sequences can be increased by these exploratory actions. All of these factors contribute to reducing the quality of the dataset. In conclusion, training on noisy data can mislead the models into learning incorrect patterns or giving undue importance to irrelevant actions.

When comparing the SPDFA-based approach to the simpler PDFA-based model, there's no notable accuracy advantage. While the SPDFA evaluates more paths nondeterministically compared to the PDFA, this doesn't necessarily equate to better predictions. As already mentioned, exploring multiple paths can introduce ambiguities, and influence the resulting probability distribution. This suggests that the added complexity of using a non-deterministic model and exploring multiple paths in the rSPDFA might not be justifiable given the almost equivalent performance to the PDFA, especially when considering its longer runtime. Talking about runtime, the SPDFA's long runtime on sequences longer than 6 symbols significantly limits its practicality. If the attacker's action sequences in real-world scenarios are typically longer than this, the SPDFA model would be impractical for timely predictions. Since both models have the same performance on unseen sequences, we can conclude that both models are able to generalize equally. However, the 33 % accuracy suggests there's room for improvement in capturing the underlying patterns of attacker actions.

We can compare our method with three works which also study the prediction of the next attacker action in a sequence, and have been already introduced in section 2.5.3: [20] [21] and [22]. [19] and [23] were also mentioned, however, these works predict the state of the network (normal, compromised, etc). rather than the attacker's action itself. [22] uses a bag-of-words model combined with a hidden Markov model (HMM) and predicts the next intrusion alert based on a sequence of intrusion alerts. The authors use the DARPA 2000 dataset which contains several DDOS scenarios. It achieves a high accuracy (up to 95% for single-step predictions for alert categories). [21] focuses on projecting cyber-attacks by examining the sequential properties of correlated IDS alerts. The authors use a Variable Length Markov Model (VLMM) for predictions and a dataset which was created through scripted multi-stage attacks performed on a VMware network by the authors themselves. Finally, [20] uses Bayesian networks to extract causal knowledge and predict multi-step attack scenarios in real-time, and also uses the DARPA 2000 dataset. All of these methods achieve high accuracy of prediction (>90%).

Firstly, the difference in performance might be attributed to the models used for prediction. The SPDFA-based methods predict the next attacker action based on observed sequences, so we assume that the next action is dependent on the current sequence of actions. The other models work differently. For example, the HMM assumes hidden states that generate observable events, while Bayesian networks model probabilistic relationships between variables. Moreover, models like Bayesian networks offer a lot of flexibility in modelling complex relationships and dependencies between variables. Therefore, it could be the case that the other models are better suited for this particular task.

Secondly, since we used a different dataset for evaluating our method, it is difficult to objectively compare the accuracy with the other works. As mentioned, the CPTC-2018 dataset is noisier and contains false positive alerts, as well as a wider variety of attack scenarios, resulting from different teams. Compared to this, the DARPA dataset only has DDOS-related scenarios, making it very specific. Since it contains only one type of attack, it is easier to learn the underlying sequences which represent the data while also narrowing down the possible actions that need to be predicted. [21] simulate a multi-stage attack in a virtual environment and create the dataset with the resulting alerts, so it is also highly likely that it is less noisy because the researchers were recreating a real attack without performing unnecessary actions. Therefore, even though the other methods achieved a higher accuracy, we cannot affirm whether they are better or worse than our method.

3.5. Conclusions

Summarizing, we can draw the following conclusions:

- Strategy 1 (Full symbol match) should be used in situations where the SPDFA was trained on a large dataset, as in that case, the SPDFA will contain more attack scenarios. That being said, it is generally worse at predicting the next action for sequences never seen before in the training set, compared to the other two strategies.
- Strategy 3 (Any symbol match) has the best performance among the implemented methods, but also the longest execution time due to the nondeterminism of the model. It should be used in situations where execution time is not critical, as it can provide better results compared to the other two, given more time.
- Strategy 2 (Attack stage match) can be used as a middle ground, because it is more ”accepting” when searching for paths in the SPDFA compared to strategy 1(thus exploring more paths), while also having a shorter execution time than strategy 3
- Execution time of prediction is impacted less by the size of the SPDFA and more by the length of the input
- SPDFA performs better when predicting low and medium severity actions, while PDFA performs better for higher severity ones. In practice, we believe that a better accuracy for high-severity actions is desired, as those actions usually have more impact and could result in more dangerous consequences.
- In practical, real-world scenarios the PDFA-based approach is better due to its short runtime and independence from the length of the input.

4

Real-time attack graph generation

In this chapter, we examine the second research subquestion: *How can we generate attack graphs in real-time, which will aid a SOC analyst when handling alerts?* We first describe our implementation for real-time alert streaming and attack graph generation. We use the same procedure as SAGE [7] for transforming intrusion alerts into attack episode sequences. For alert sequences that do not end in a high-severity episode (partial paths), we predict the next action using the PDFA and display the prediction in the resulting attack graph.

4.1. Methodology

In order to be deployed in a real-world scenario such as a SOC, the attack graphs need to be generated in real time. This would allow analysts to get a summarised view of all of the past actions observed for a host, which could help them when analyzing alerts.

Figure 4.1 shows a summarised version of the real-time attack graph generation pipeline. For transforming alerts into episodes, we use the same procedure proposed by SAGE [7], described in more detail in section 2.4. We kept this process the same, due to its short execution time which we observed during initial experiments.

4.1.1. Alert ingestion and episode creation

In real-time scenarios, we believe there is no single ”best” option for the alert ingestion time and re-generation of attack graphs, and it should be configured depending on the alerts generated per minute rate, so the ”noisiness” of the environment. For a low alert rate, an attack graph can be generated for every alert, because of the small execution time of the attack graph generation pipeline (which will be further discussed in section 4.2.3). For a higher alert rate, alerts can be put in a batch, and the generation of attack graphs can be run every minute. Ultimately this depends on the required granularity and the needs of the SOC analysts.

When a new batch of alerts is received, it is pre-processed by removing duplicates and then merged with the current alert pool. The current alert pool can consist of alerts that are triggered the same day, the last couple of days, or even longer periods of time. This depends on the desired amount of context that we want to see in the attack graph for a particular alert. For example, if we suspect that there is an attack campaign happening in the network, it makes sense to increase this window to a longer period, such as multiple weeks, to look for any suspicious behaviour on the hosts which occurred during that time period. Having a larger time window will increase the size of the attack graph, but will also allow us to correlate a larger number of past alerts for a single host. We then aggregate the alerts into episodes and create the episode subsequences.

4.1.2. Trace creation and SPDFA learning

The episode sub-sequences are then transformed into traces, by extracting the attack stage and service from each episode. This will result in a set of *current* traces. Besides them, we also use a set of *past* traces, which contains attack scenarios observed in the past. We merge them with the traces that were generated using the current alert pool and then learn an SPDFA using the resulting trace set.

This design choice is based on the fact that the attacker actions observed in the current alert batch will be merged with the past attacker actions, and the resulting SPDFA will contain both of them. Thus, when mapping the current episodes with the states in the SPDFA, if the episodes can be correlated to a sequence of states located in the main model, it might mean that this is part of an attack which was observed in the past. If we were not to use the past traces, most of the states could be placed in the sinks, which contain the states modelling the data which the model could not learn behaviours from, due to it being too sparse. Thus an analyst would not know whether the currently observed episode sequence was also observed in the past or not.

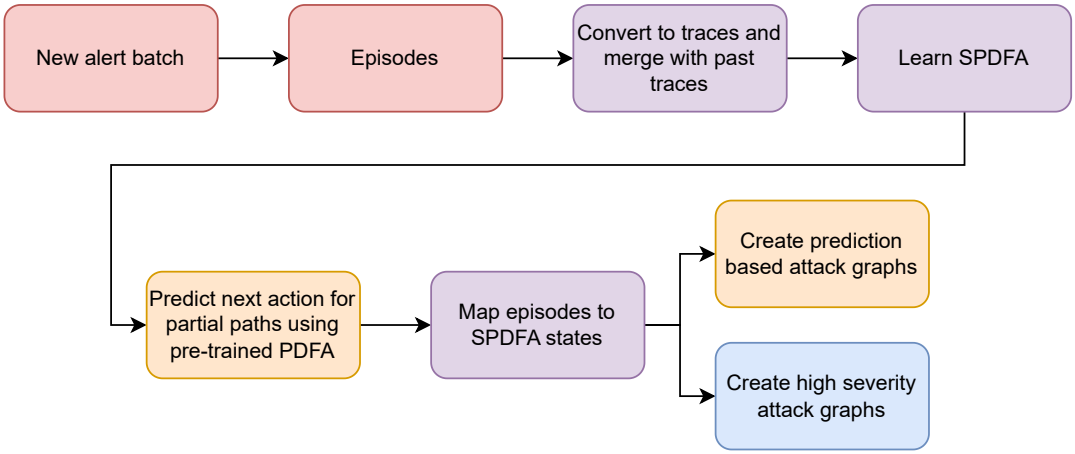

Figure 4.1: Pipeline for real-time attack graph generation and next attacker action prediction.

4.1.3. Prediction of the next actions for partial paths

The set of past traces is also used as a training set for learning a PDFA, which is used by the PDFA-based prediction algorithm. We have chosen the PDFA method due to its short runtime and accuracy comparable with the SPDFA methods. We first identify the partial attack paths, which are episode sequences which do not end in a high-severity episode. Previously these sequences would not be included in any attack graphs, due to the attack graphs being objective-based, and the objective being unknown. Thus, for every created episode sequence, we examine the last episode. If it is a non-high severity episode, we transform the sequence into a trace, by extracting the attack stage and most frequently targeted service from each episode. The result is a sequence of symbols in the format `<attackStage | service>`. We then use the trace as input and save the result if the prediction algorithm succeeds in making a prediction.

4.1.4. Attack graph creation

Due to the previous steps, we now have sequences of episodes that end in a high-severity action, as well as predicted actions. We would like to create attack graphs for both cases. For every sequence where we have predicted the next action, we first examine its severity. Low and medium-severity predictions are generally less interesting for analysts because the impact of those actions is not as critical as compared to high-severity actions. Moreover, creating independent attack graphs for every low-medium severity prediction would result in too many attack graphs, flooding the analyst with information. In order to avoid information loss while still displaying the partial paths, we choose to create attack graphs on a prediction basis, which contain the episodes from all hosts where this prediction was made. For example, if we have three hosts where the predicted action was *vulnerability discovery over HTTP*, we would create only **one** attack graph, which would include the episodes from the three hosts. Predicted nodes are **orange and dashed**, and are connected via a dotted edge with the rest of the nodes. The edge leading to the prediction nodes also displays the prediction probability. Moreover, since each victim can have a different attacker, every attacker will have a different edge colour in order to differentiate them more easily. These types of attack graphs are also great for combining information across multiple hosts, allowing the analyst to get to see the attacker's actions across the entire network.

For high-severity predictions and high-severity actions, we still generate the attack graphs on a per-victim per-objective basis. For example, if the predicted action is *data exfiltration over HTTP* and we already have an attack graph for this action, the partial path and prediction will be included in the same attack graph. If there is no attack graph, a new one will be created. To be able to separate partial paths from complete paths within an attack graph, nodes which are part of a partial path will have dashed edges, and are connected to the orange predicted node via a dotted edge. Examples of both types of attack graphs are shown in section 4.2.2.

4.2. Experiments and Results

This section describes our approach for evaluating the real-time generated attack graphs. For evaluation purposes, we also needed a way to re-identify episodes in between iterations, in order to verify if our prediction of the next episode was correct after the actual next episode had been received. In order to re-identify the newly created episodes with the ones from the previous iteration, we examined the elements of an episode and chose the ones that would stay consistent over two iterations. Thus an episode can be re-identified by the *(host, start time, attack stage)* tuple. The most targeted service was not chosen because it is subject to change when more alerts are received. The same argument is valid for the end time of an episode because as more alerts come in, they might be combined into a previous episode, changing its end time.

4.2.1. Experimental setup

For evaluating our approach, we use the CPTC-2018 dataset, described in more detail in section 3.1. We use alerts of teams 1, 2, 5 and 7 to create the set of past traces, with a total of 354 entries. As mentioned previously, these traces are used to learn a PDFA and are merged with the current set of traces when generating attack graphs. We use the team 8 and team 9 alert datasets for simulating a real-time scenario and generating attack graphs. Figure 4.2 and 4.3 show the number of alerts generated over time by team 8 and correspondingly team 9. It can be seen that the alert generation rate of team 9 is more evenly spread out over the entire period, whereas most of the alerts generated by team 8 are between 15 and 16 o'clock. This could mean that team 9 tried to reach the objectives during the entire duration of the competition, while team 8 achieved them near the beginning, hence the lack of alerts in the period that followed.

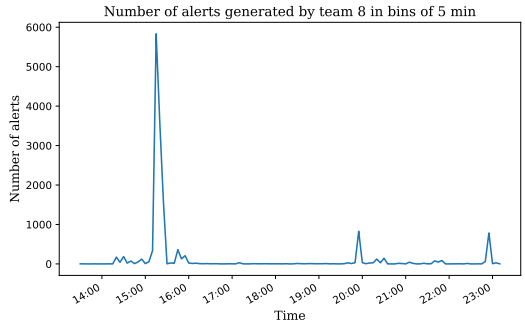

Figure 4.2: Number of alerts generated by team 8 that triggered over the entire period of the competition, grouped in intervals of 5 minutes. It can be seen that most of the alerts were triggered in a short period of time between 15 and 16.

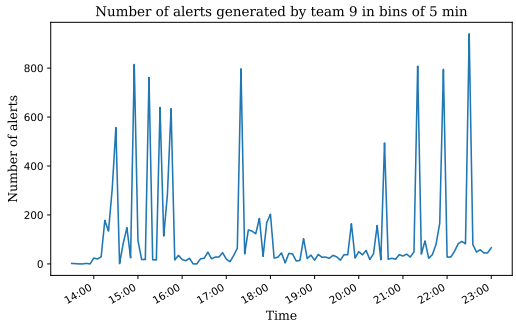

Figure 4.3: Number of alerts generated by team 9 that triggered over the entire period of the competition, grouped in intervals of 5 minutes. Compared to team 8, the alerts for this team are more spread out for the entire period of the competition.

For simulating a real-time scenario, we load all alerts and divide them into batches, each batch containing alerts from a 5-minute interval. The 5-minute interval was chosen because of the high alert generation rate per minute of both teams. Table 4.1 shows an overview of the testing datasets and resulting batches. As can be seen, team 8 generates an average of 69.15 alerts per minute, while team 9 generates 33.52 alerts per minute, thus creating an attack graph for every alert would be unfeasible. Moreover, such a high number of alerts per minute does not occur in the real world, because it would be impossible for the analysts to analyze this many alerts.

We then run the pipeline described in figure 4.1, each time adding more batches of alerts to the

Table 4.1: Overview of team 8 and 9 datasets used for testing

Dataset	Total alerts	Alerts per minute	Number of 5-min batches	Avg. alerts per batch
Team 8	15560	69.15	83	187.4
Team 9	12841	33.52	99	129.7

current pool, and generate the attack graphs only from the new episode sequences. We also save the prediction results for the partial paths, and when the actual next episode is seen, we evaluate whether the prediction was accurate, by comparing the prediction to the actual observed episode. Based on this, we define the following metrics:

- Accuracy = $\frac{\text{correct prediction}}{\text{total predictions}}$
- Attack stage accuracy = $\frac{\text{correct prediction of the attack stage}}{\text{total predictions}}$
- High severity true positive rate = $\frac{\text{High sev. TP}}{\text{High sev. TP + High sev. TN}}$
- High severity false positive rate = $\frac{\text{High sev. FP}}{\text{High sev. FP + High sev. TN}}$

Accuracy is a standard metric used for evaluating the performance of prediction models. We also calculate the accuracy of predicting the correct attack stage, because the service may not be an interest to an analyst, whereas the attack stage can still be a risk indicator. The high-severity true positive and false positive rates are important because high-severity episodes are the most dangerous ones. Thus, an analyst would pay more attention to such predictions, and a high-severity false positive prediction could mean that an analyst prioritized an alert when it should have not been the case. Note that we divide by the total number of predictions for which we have seen the actual next episode, because for some partial paths, no new episode followed, and the prediction remained the last episode in the sequence. We also calculate the execution time of an iteration, by measuring time from the point new alerts are received, until the point where all attack graphs are created. Note that this time will increase as more alerts are added to the current pool of alerts.

4.2.2. Real-time attack graphs

Figure 4.4 shows an example of an attack graph for the low-severity predicted action ”Vulnerability Discovery for mysql”. As mentioned previously, since the action is low severity, it contains the observed episodes for all hosts for which this was the predicted action. Every attacker IP has a different colour, so the actions of a particular attacker can be observed by following the edges of a particular colour.

Figures 4.5 and 4.6 show examples where the prediction for a partial path is a high severity action, resource hijacking and correspondingly data manipulation. The first attack graph contains only the predicted action, while the second one also has an observed high severity one besides the prediction. It can be seen that the partial paths are connected via dashed edges, making it easier to separate them from the complete paths, which are connected via normal edges. Moreover, the predicted nodes are orange and dashed.

4.2.3. Prediction evaluation

Table 4.2 contains the metrics for the prediction of the next actions when streaming alerts in real-time. Even though both teams 8 and 9 have a similar number of alerts, team 9 had a higher execution time per batch. This might be happening because of the fact that the alerts of team 9 were more spread out throughout the duration of the competition, which was also seen in figure 4.3. Because of this, in every iteration new alerts were added and new episodes were created, requiring the regeneration of the attack graphs more frequently, compared to when using the alerts of team 8.

The overall prediction accuracy is also lower compared to the one observed in chapter 3.3.7. This might be explained because of the different evaluation method used in this approach. Previously we only had a set of testing traces, where we removed the last symbol and used the rest of the trace as input. This time, however, we predict the next action for every partial attack path we find, during every iteration. This means that we continuously predict the next action while the sequence increases in size, as more data is processed.

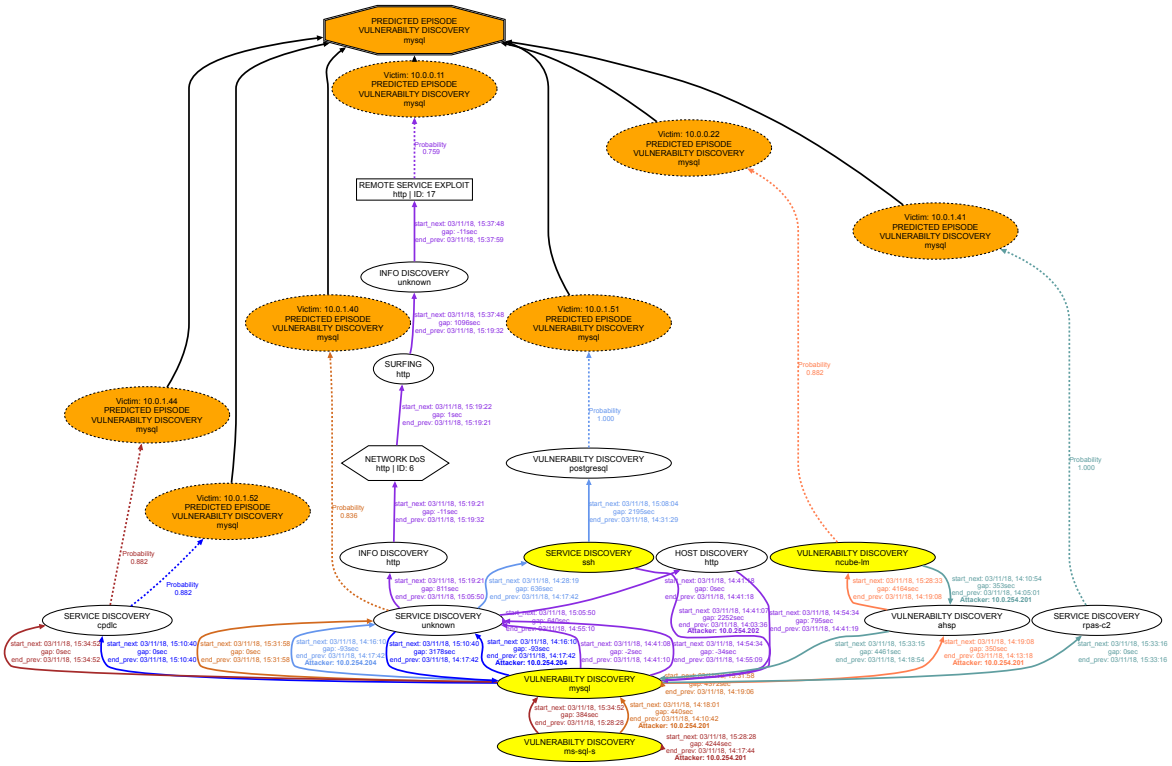

Figure 4.4: Real-time generated attack graph for the predicted action "Vulnerability Discovery for mysql". There is one orange (prediction) node for every host where this was the predicted next action, the rest of the nodes representing the episodes that lead to it. The edge color is based on the attacker IP.

It can also be seen that for both teams the attack stage accuracy is higher than the normal one, showing that the model is better at predicting the attack stage than the attack stage and service. Finally, we can see that the high severity predictions are very infrequent and that our model has trouble predicting them correctly, as the false positive rate is higher than the true positive one.

Table 4.2: Results for the prediction of the next action during real-time generation of attack graphs using alerts of team 8 and team 9.

	Team 8	**Team 9**
Total predictions	84	637
Accuracy (%)	6.6	12.7
Attack stage accuracy (%)	11.1	43.5
Total high sev. predictions	10	12
High sev. TPR	0.1	0.0
High sev. FPR	0.5	0.66
Avg. exec. time per batch (s)	2.3	7.6

4.3. Discussion

In this chapter, we have implemented a method of streaming alerts and generating attack graphs in real-time. The attack graphs contain both the actions of the attacker observed so far, as well as a predicted action. Due to the low execution time of the pipeline which was observed during testing, we believe that attack graphs can be generated for every triggered intrusion alert. This would provide SOC analysts with a means of visualization of the previously observed attacker actions. Moreover, the prediction could inform the analyst of the potential next steps of the attacker, allowing them to implement preventive measures.

The low and medium prediction attack graphs contain the episodes of multiple hosts, which provides

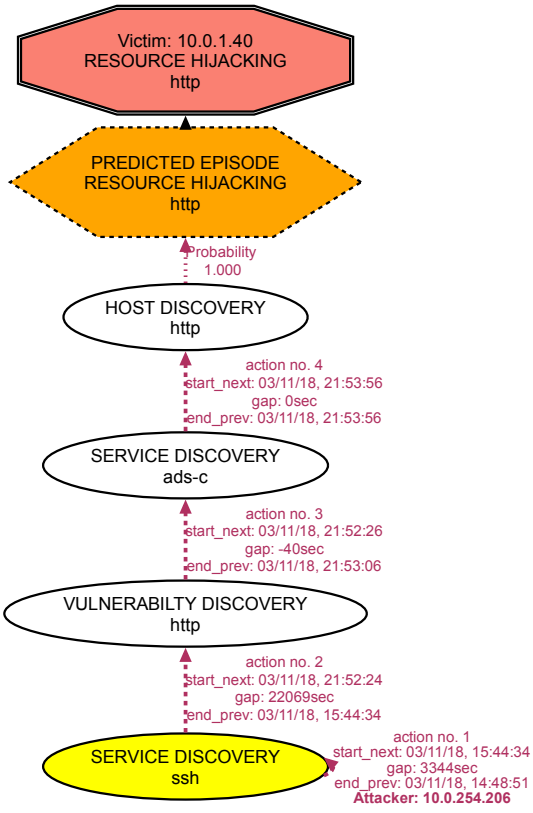

Figure 4.5: Real-time generated attack graph with a high severity predicted action ”Resource hijacking over http”. The nodes with a solid border represent the partial path, while the predicted one is orange. These nodes are connected with dashed edges.

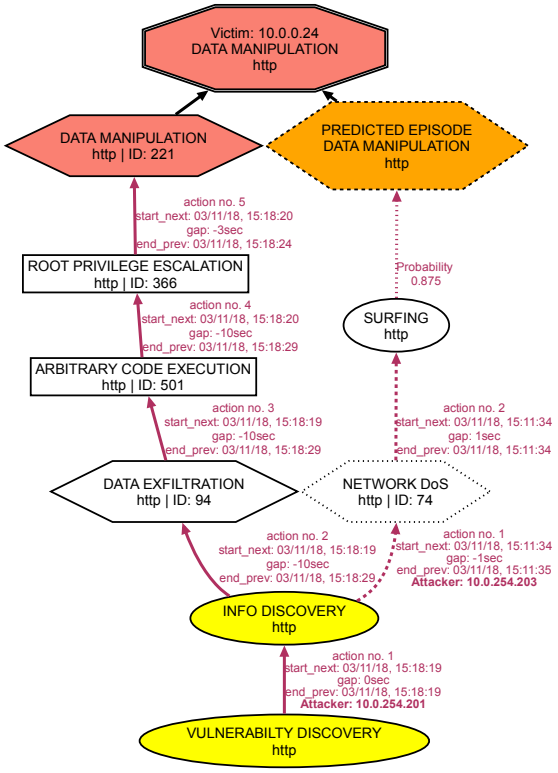

Figure 4.6: Real-time generated attack graph with both an observed and predicted high severity action ”Data manipulation over http”. The partial paths have dashed edges, while the complete ones have normal ones.

a more summarised view of the generated alerts. For example, if all of the alerts displayed in figure 4.4 were to be displayed independently, in text form, it would have been very hard for an analyst to manually correlate them. Such an attack graph could also inform the analyst of the fact that these hosts might be vulnerable to the predicted next action. That being said, the approach of grouping alerts between multiple hosts can also prove to be a disadvantage, as the graph can become very dense and cluttered if it contains many episodes/hosts, making it hard to follow.

The high severity and high severity prediction attack graphs are good for highlighting the alerts leading to the current alert for a single host. Due to them being smaller in size, an analyst would also take less time to understand them and react faster. For example, an analyst could see in the attack graph from figure 4.5 that the host might be vulnerable to data exfiltration. Knowing this, he could perform additional measures to try and minimize the risk such an action could have, such as isolating the host from the network, preventing the attacker from causing additional damage and stopping the attack chain.

When comparing our approach with existing works, we noticed that there are no other methods which perform both real-time attack visualization and attack prediction. [12] displays attacker behaviours observed in the network based on the intrusion alerts in real-time. That being said, it does not highlight them for each individual host. [17] and [18] are both methods of performing alert correlation in real-time. Attack correlation is a method of grouping alerts together, which could aid an analyst in identifying malicious behaviour spread out between multiple hosts and/or alerts.

Our method combines elements of attack visualization, alert correlation, as well as attacker action prediction. Compared to [12], the visualization is done both on a per-host basis (high severity actions and predictions), as well as for multiple hosts when predicting low and medium severity actions. Their approach is good for getting an overview of intrusion alerts for a particular host, but not so much for

getting a summarized view of what is happening in the monitored network. [17] and [18] can correlate alerts across multiple hosts, but they do not provide a means to visualize them. Finally, even though the prediction accuracy achieved by our method is lower than those of other works (which was discussed in section 3.4), we believe it could still be used by an analyst during the alert analysis process. For example, if the prediction is about vulnerability exploitation, the analyst could check whether the host has any vulnerabilities, and consequently patch them. In this case, even if the prediction does not end up happening, the security of the host would still be improved.

In order to further evaluate our method, and its usefulness if deployed in a real-world SOC, we have organized interviews with SOC analysts of Northwave Cybersecurity. During these interviews, we have asked them to identify challenges in their daily operations, as well as evaluate different aspects of our method. Chapter 6 describes the interview process, results and findings.

4.4. Conclusions

In summary, our main findings from this sub-research question are:

- The attack graph generation is fast and takes a small amount of time to run, even with such a high influx of alerts as in the performed experiments. In real-world scenarios, we believe the alert generation rate to be much lower than in our testing scenarios, making the generation even faster.
- Real-time generated attack graphs are a flexible solution for visualization, because they can be deployed in both noisy and normal environments with the correct parameters
- Attack graphs for high-severity actions provide an overview of the events observed on a single host, allowing the analyst to focus on the high-severity events
- Attack graphs for low and medium severity predictions combine the alerts observed on all hosts with the same prediction, providing the analyst with more information
- Compared to existing solutions, our method is the only one which combines alert visualization, correlation and attacker action prediction

5

Real-world data evaluation

In this chapter, we examine the third research subquestion: *Can attack graphs be generated using data collected in the real world?* We have collaborated with Northwave Cyber-security [1], a cybersecurity company located in the Netherlands. Three datasets have been created using intrusion alerts generated by their SOC, which uses Microsoft Sentinel as their SIEM, described in section 2.1. We have decided to generate offline attack graphs, rather than real-time ones, due to the sparsity of real-world data and the similarity of the two approaches.

We first discuss the methodology of creating the datasets. We then dive into the challenges we encountered when adapting the attack graph generation pipeline to the new format of the input data, and how we solved them. We then compare real-world data with open-source one and highlight several identified differences. Finally, we evaluate the generated attack graphs and their potential use in the daily operation of a SOC analyst.

5.1. Dataset description and analysis

In this section, we describe the criteria based on which we have created the datasets, as well as provide a short analysis of the underlying alerts. For creating the dataset, three network environments have been chosen, each corresponding to a separate scenario: an environment where a penetration test has been performed, an environment which had a true positive high severity alert and a more ”noisy” environment with a larger number of alerts compared to the others. These scenarios would allow us to create three datasets with different characteristics and goals. In the pentest dataset, we expect to see attack graphs where the nodes are the actions of the pentester. In the true positive scenario, we expect an attack graph which illustrates all of the actions leading to the true positive high-severity alert. Finally, in the big environment dataset, we want to evaluate the alert summarization capabilities of our approach, as well as examine the attack graphs for any potential past attacks.

In each of the chosen environments, the intrusion alerts were collected over a set period of time: 3 weeks for the pentest environment, and 3 months for the others. The 3-week duration for the pentest dataset was chosen due to the fact that according to literature, the maximum duration of a pentest is usually 3 weeks [24], so we wanted to make sure we caught all alerts related to it. On the other hand, the 3-month period was chosen because that is the maximum amount of time that alerts are stored in Sentinel.

A Sentinel intrusion alert has multiple fields related to the detection. Table 5.1 provides a brief description of those attributes which we find the most useful and will use when generating attack graphs. Note that it is possible for some of these attributes to be missing in an alert, depending on its type.

Figure 5.1 shows the distribution of the ’Severity’ field across the three datasets. One thing common among all of them, is Medium alerts being the most numerous, followed by Low and finally High alerts. The same distribution pattern can be seen in the CPTC-2018 dataset in figure 5.2. An exception to this is the Pentest dataset, which has more High alerts than Low ones. This can be tied to the nature of a penetration test which usually involves more high-impact actions than those that occur during the normal operation, which would trigger a High severity alert.

[1] https://northwave-cybersecurity.com/

Table 5.1: Sentinel alert attributes which will be used when generating attack graphs

Attribute name	Attribute description
Timestamp	Time of generation of alert
Signature	Description of what was detected
Severity	Severity of the detected behaviour: Low, Medium, High
MITRE Tactic	What the attacker is trying to achieve (e.g. Initial Access)
MITRE Technique	How the attacker is trying to achieve his goal (e.g. Phishing)
Entity	What the alert refers to: hostname, IP address, account name

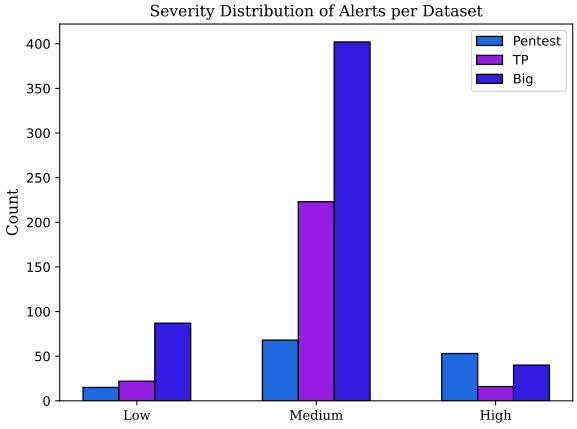

Figure 5.1: Severity distribution of intrusion alerts from the three Northwave datasets. Medium alerts are the most frequent across all three datasets. The Pentest dataset has more High-severity alerts compared to Low-severity ones.

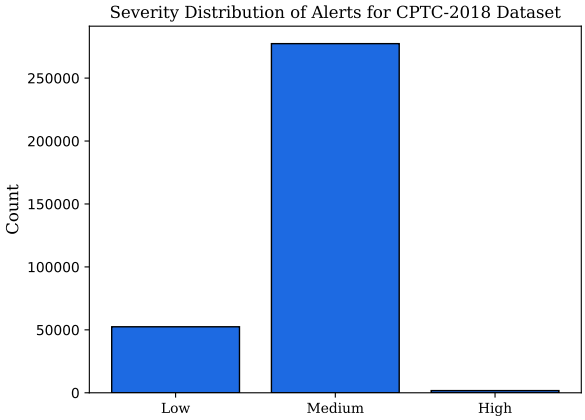

Figure 5.2: Severity distribution of alerts for CPTC-2018 dataset. Similar to the Northwave data, Medium-severity alerts are the most common.

Due to the fact that the three datasets contain a mix of alerts generated by Sentinel, NIDS and EDR (Endpoint Detection and Response) systems, as well as the large variety of behaviours that they detect, there are two major differences in terms of the features present in them compared to CPTC's Suricata dataset. The first one is the lack of a source IP address in some alerts, which is usually the case for EDR alerts, which only contain the victim host. For example, an alert about malware detected on a host cannot have the source IP address of an attacker related to it. The second one is the lack of the destination port in non-NIDS alerts. We will describe how these changes have influenced the attack graph generation pipeline in section 5.2.

5.1.1. Dataset pre-processing

After obtaining the alerts from the specified time periods, we pre-process them using the procedure described in table 5.2. We start with cleaning out alerts that are uninteresting for us because they cannot be used when generating attack graphs. These include alerts that do not contain any entities, where the entity is not a hostname or an IP address, and those that have neither an MTIRE technique nor a tactic. We then remove duplicate alerts, which are two alerts that have the same hostname, signature and have a difference in timestamps of less than 1 second. We then split alerts that contain multiple entities into separate alerts that contain only one entity. For example, if the alert contains a list of three hostnames, it would be separated into three alerts, each having the same details as the original one, but a single host as the entity.

Following this, we extract the attack stage from each alert, using both the MITRE Tactic and Technique. If the alert has both of these attributes, its attack stage becomes <*Tactic.Technique*>. Some alerts also have multiple tactics, or multiple techniques listed for the same tactic, in which case we concatenate all of them, their attack stage becoming <*Tactic1.Technique1, Tactic2.Technique1*>. Otherwise, if the technique is unknown, the attack stage becomes <*Tactic. Unknown Technique*>. This approach was chosen in order to use all the available information from the alerts and prevent information loss. Finally, we anonymize the hostnames, since we are dealing with sensitive data.

Table 5.2: Summary of pre-processing steps performed on the extracted datasets

Pre-processing step	Description
Cleaning	Remove alerts that: do not contain any entities do not contain a hostname or IP address do not have a MITRE technique or a tactic
Duplicate removal	Remove alerts that have the same signature, host, and a difference of timestamps of less than 1 sec.
Entity separation	Split alerts that contain multiple entities into separate alerts with a single entitiy
Alert augmentation	Concatinate the Mitre Tactic and Techniques to create the attack stage of each alert
Host anonymization	Anonymize the hostnames

5.2. Methodology

In this section, we describe the modified attack graph generation pipeline using real-world datasets. Table 5.3 presents a summarised version of the pipeline and its differences from the one previously used by SAGE.

We start with a dataset of pre-processed intrusion alerts. First, we need to convert these alerts into attack episodes and group them by the host to which they belong. The episodes are grouped into sequences per victim host (instead of attacker-victim combination) because not all alerts have a source IP address. Due to the scarcity of the alerts, we opted to convert each alert directly into an episode, rather than clustering them based on alert frequency. The attack stage of the alert then becomes the attack stage of the episode and the start and end times are the same and correspond to the alert timestamp. The result of this step is a list of episode sequences for every host.

The episode sequences are then partitioned into episode sub-sequences. This is done by finding two episodes in a sequence where the severity decreases, dividing it into two subsequences. The first episode will then become the end of the first subsequence, and the second episode the start of the next subsequence. This process is repeated until the end of the sequence is reached.

Next, each subsequence is converted into a trace. Since we have no target service in the alerts, when creating the traces, only the attack stage can be used as a symbol. However, we observed that in some cases the attack stage of two alerts can be the same, whereas their severity could be different. An example of this would be an unauthorised login alert, whose attack stage is Initial Access, and the alert severity would depend on whether the login was during or outside office hours. This would have resulted in an SPDFA which inaccurately represents the data because the same attack stage potentially refers to actions of different severity depending on the context. To counter this issue, when converting an episode into a symbol, we extract its attack stage and severity. An example of such a symbol is `Privilege Escalation. Process Injection | Medium`. This way we can illustrate the different severities of the attacker's actions and attribute different contexts to them.

With the traces, we use Flexfringe to learn an SPDFA, which will represent a summarised version of the data. We then map each episode to a state in the SPDFA using the corresponding state identifier. Finally, we generate the attack graphs using the episodes, one attack graph per *(victim, objective)* combination, where the objective is a high-severity episode. This time the node labels indicate the attack stage as well as the state identifier of the episode, while the edge label illustrates the number of the action in the sequence, the gap between the next episode and the previous one, as well as the timestamp of the next episode. The episode number in the sequence was added to make the order of the attacker's actions easier to understand for an analyst.

5.3. Results

5.3.1. Datasets and SPDFA

Table 5.4 illustrates the difference between the datasets in terms of alert number, number of traces generated, as well as the average length of an episode sequence. The pre-processing of the CPTC-2018 dataset was described in section 2.4. As expected, the big environment dataset has the most

Table 5.3: Description of the pipeline for attack graph generation from real-world data. The second column highlights the changes between the SAGE pipeline and this one.

Pipeline Step Description	**Difference from SAGE**
Transform alerts into episodes and group them per host	Each episode will contain only one alert. Episodes are grouped per host only instead of (attacker, host).
Create episode subsequences, by splitting each sequence into multiple ones, in places where the severity of the episodes decreases	No differences
Use ESS to create traces, by extracting attack stage and severity of each episode	Each symbol will contain attack stage and severity, instead of service.
Use traces to learn SPDFA	No differences
Map each episode to their corresponding state in the SPDFA	No differences
Create an attack graph for every victim, objective combination	Edges will also display the number of action in the sequence. No attacker IPs are displayed.

alerts among the three. The number of alerts before and after pre-processing is not majorly different because even though some alerts got split into multiple ones, others were removed due to referring to a non-host entity.

Table 5.4: Comparison between the three datasets and CPTC-2018, in terms of alert and trace number

	Alert #	Alert # after pre-process	Number of traces	Avg. sequence length
Pentest	136	116	24	3.50
TP	261	261	35	6.14
Big Environment	529	508	83	4.40
CPTC-2018	331,554	71,126	664	5.78

Multiple differences can be seen between the real-world datasets and the CPTC-2018 one. First, there is the raw alert number, which is significantly lower in the real-world datasets. Even if we were to compare the average number of alerts generated per team, which is 55,259, the alert number would still be higher than that of the biggest dataset obtained in the real world. A cause for this might be that the intrusion detection system deployed during CPTC has not been tuned, and as a result, a large amount of the alerts generated were false positives. It can also be noted that the number of alerts is directly proportional to the number of traces generated, while the TP dataset has the highest average sequence length of 6.14. This could mean that this environment has hosts which trigger false positive alerts periodically, creating bigger sequences of episodes.

Figure 5.3 shows the SDPFA generated using the traces of the big environment dataset. Even though the trace file contained 83 traces, we can conclude that most of them were unique and infrequent, because they are not part of the main model. Even though the traces have been reversed, most of the actions are related to either Credential or Initial access. A common observation among all the SPDFAs learned using these datasets is that the automaton is less successful in learning the patterns from the training data. Thus the main model is smaller and contains fewer behaviours, while the sinks (states corresponding to infrequent sequences from the training data) contain more of them. This is mostly due to the small size of the training dataset, and because a lot of the sequences are infrequent, the learning algorithm does not have enough data to learn something useful from them.

5.3.2. Attack graphs

Figures 5.4 through 5.6 show examples of attack graphs for each dataset. The pentest dataset attack graph (figure 5.4) is the biggest one of the three and showcases the actions of the red teamers against a host. It can be observed that the objective ”Defense Evasion.Unknown Technique” was reached in two different contexts, once close to the start, and a second time after a longer sequence of actions.

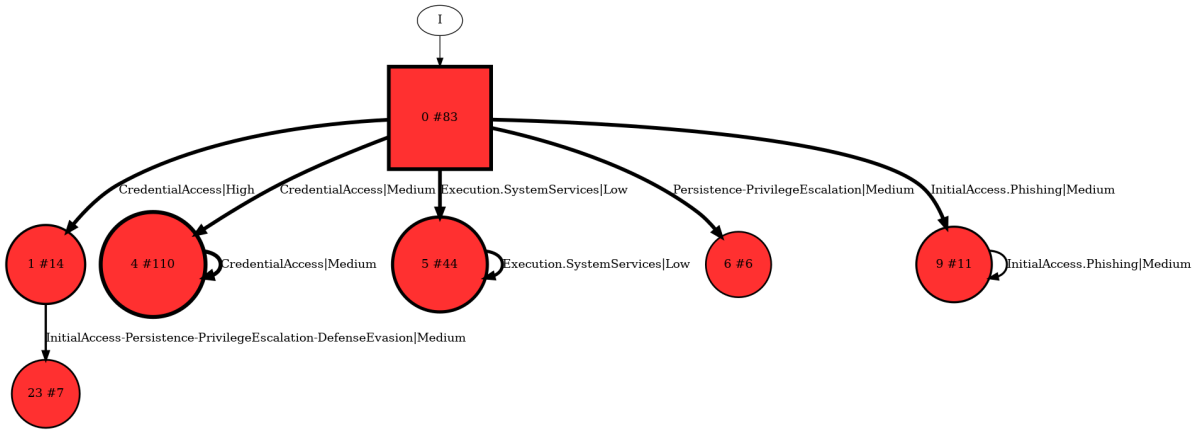

Figure 5.3: SDPFA learned from 83 traces generated using the Big Environment Dataset. We can see that only a small part of the traces were included in the main model, due to their infrequency.

A lot of the nodes are dotted because their corresponding states are in the sinks due to them being infrequent, which also makes sense because this is the first time such actions have been used against a host. Such an attack graph can be very useful for an analyst, as it links multiple alerts together providing a summarised view for this host.

The attack graph generated from the Big environment dataset from figure 5.5 showcases actions for ”Defense Evasion.Impair Defenses” objective, which usually means turning off the antivirus. The first action in the sequence was ”Execution. User Execution”, as seen by the yellow Node. It can also be seen that the first 5 actions are low and medium severity (left part of the graph), and the high severity alerts start with action 6, ”Credential Acccess. OS Credential Dumping”. As was later identified, a penetration test was in process when the data for this environment had been collected, and this host has been part of it.

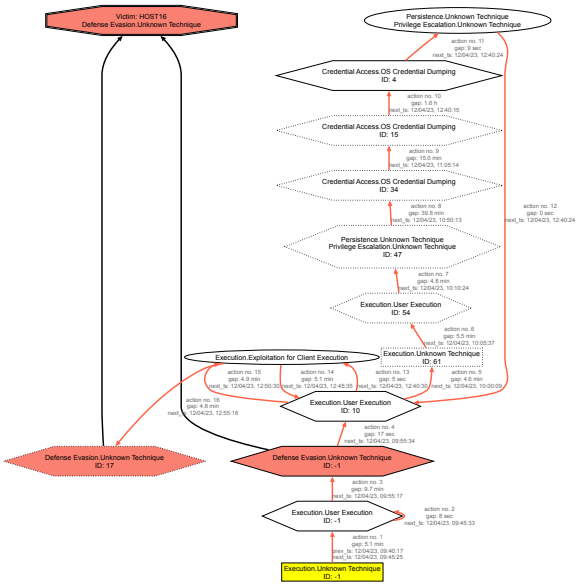

Figure 5.4: Attack graph for a host from the Pentest dataset showcasing the Defense Evasion objective.

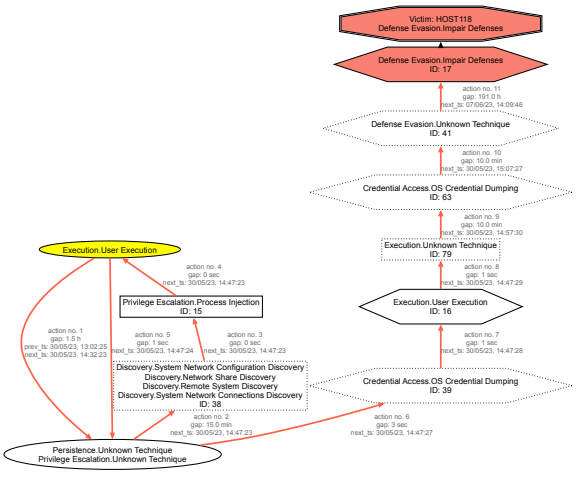

Figure 5.5: Attack graph for a host from the Big Environment Dataset, showcasing the Defense Evasion. Impair Defense objective.

Unfortunately, our hypothesis for the TP dataset did not prove itself, because the attack graph for the host which had the TP high severity alert did not contain any other ones (figure 5.6). This showcases a disadvantage of the approach because the attack graph is highly reliant on the generated alerts. Moreover, it is constructed for each individual victim. Thus, if an attacker performs lateral movement

between hosts, or if the intrusion system does not detect some actions, they will also not be visible in the attack graph.

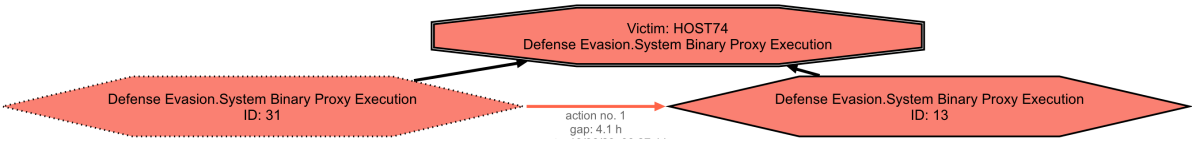

Figure 5.6: Attack graph for the host which had the True Positive high severity alert from the TP dataset. Unfortunately, the attack graph contains only the high-severity actions, due to there being no other alerts for this host.

We have also observed that attack graphs related to false positives will often contain the same action repeated multiple times. An example of such an attack graph can be found in figure 5.7. It showcases only three actions related to ”Initial Acesss”, for the same host. The time gap between the actions is also large, being 142.8 hours between actions 1 and 2, and 409.6 hours between actions 2 and 3. All of the alerts related to these actions were later identified to be false positives and triggered due to a network administrator. We believe that such attack graphs could prove useful to analysts when deciding whether an alert is a false positive or not, as they illustrate actions that re-occur periodically on the same host.

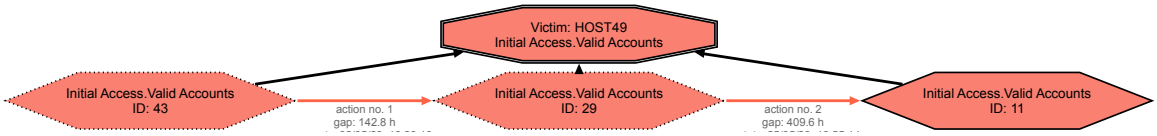

Figure 5.7: Attack graph showcasing only ”Initial Access.Valid accounts” actions. It can be observed that the time gap between these actions is very large. The alerts related to these actions were later confirmed to be false positives and caused by an administrator.

5.4. Discussion

In this chapter, we have examined the usability of SAGE on real-world generated data. We have decided to generate attack graphs offline, using all of the alerts in the dataset, rather than in real-time. Such an approach was chosen for multiple reasons. First, our primary goal with this research question was to study real-world data and identify key differences between it and open-source data. Moreover, previously SAGE was used for generating attack graphs using intrusion alerts from an open-source dataset, thus we were interested in testing it with real-world data. Secondly, by using offline generation, we can control the amount of data, meaning we can compensate for the sparsity of the real-world dataset. Finally, the real-time generated pipeline is a modified version of the offline one, so the results obtained with offline attack graphs are also valid for real-time generated attack graphs.

We have identified significant distinctions between open-source datasets and those derived from actual operations. Notably, the quantity of alerts generated in real-world settings is considerably lower, even when aggregated over an extended period, such as three months. It is important to note that our analysis is confined to the alerts from a single Security Operations Center (SOC), and as such, it would be premature to assert that this trend is universally applicable across all real-world environments. Moreover, the elevated number of alerts in the CPTC-2018 dataset may be attributed to the possibility that the Intrusion Detection System (IDS) used during the competition was not optimally configured, resulting in a substantial volume of false positives.

Despite these challenges, our findings show how important it is to have high-quality datasets from the real world for research. When researchers have access to good data that reflects what’s actually happening in operational settings, their results are more likely to be useful and applicable in real-world situations.

We have also evaluated the usefulness of an SPDFA in providing an accurate summarization of the data. For all of the datasets, the SPDFA only contained a small number of behaviours observed in the data. The cause for this is of course the small number of sequences which was used for training this.

Thus, we believe that the SPDFA performs best when it has more data to learn from. Future research could explore adjustments to the learning algorithm's parameters to optimize performance.

We have adapted the SAGE pipeline to generate attack graphs based on intrusion alerts from real-world scenarios. Typically, attack graphs derived from penetration testing activities, which involve simulated attack scenarios, were of higher quality, as they contained more observed attacker actions. This leads us to believe that attack graphs could be a valuable tool in the event of an actual attack, provided that the intrusion detection system successfully identifies malicious activities. Generating attack graphs in real time would allow the analysts to get an overview of the intrusion alerts observed for a single host, and assist them with alert analysis. Additionally, this approach can facilitate the identification of false positives, as attack graphs generated from these often exhibit repetitive actions.

Ultimately, the usefulness of such a method depends on the intrusion detection system itself, as it only displays the actions which were already detected. That being said, this characteristic could also be used as an advantage. For example, if used offline, our approach could be used as an evaluation method for the IDS. For instance, a red team could conduct an attack on a monitored environment, and subsequent analysis of the attack graphs could reveal which actions were detected, which went unnoticed, and which areas require further enhancement.

5.5. Conclusions

Summarizing, we can draw the following conclusions:

- Real-world data is much more sparse and different than artificially generated intrusion alerts dataset, both because the IDS is more tuned (less triggered alerts), and because attacks happen rarely in the real world
- SPDFA is not as useful in this scenario, since it cannot provide an accurate representation of the behaviours observed in the data, due to its sparsity
- Attack graphs are especially useful during real attacks or penetration tests because in these scenarios more intrusion alerts are generated than usual, so the resulting attack graphs will provide a big-picture view of the situation
- AGs related to false positives usually contain only several high severity nodes, without any other actions leading to them (but exceptions can also happen, such as the one shown in figure 5.6).
- Real-time attack graphs generated in the real world can be used as a tool to summarize and visualise all alerts for one host as well as help in identifying false positives
- Offline attack graphs generated in the real world can be used as an evaluation method for the IDS, to see which behaviours were detected by the IDS and which were missed.

6

User study: Interviews with Northwave SOC analysts

In order to further evaluate the applicability of our method in the real world, and get feedback from security experts, we organised 1-on-1 structured interviews with 6 SOC analysts from Northwave. This chapter describes the interview process, a summarised version of the answers of the participants, and finally the implications of those answers for our research.

6.1. Objective and set up

The first objective is to identify how useful would real-time generated attack graphs be in the daily tasks of a SOC analyst. Particularly, here we are interested in the current challenges the SOC analysts face regarding alert handling, if our method addresses any of those challenges, and if the analysts would use it as an alert visualization and summarization tool. The second objective is to evaluate the usefulness of predicting the next attacker's action. We would like to verify if the analysts would trust the result of the prediction, and in what way they would use it when analyzing alerts.

SOC analysts can be classified into three tiers, based on their experience level: Tier 1, 2 and 3. Tier 1 analysts are responsible for analyzing most of the alerts, and deciding whether they are justified or false positives. In the case that the tier 1 analyst requires help with their tasks, they involve a tier 2 analyst, who has more experience and can take a more in-depth look. Finally, Tier 3 analysts are responsible for continuously monitoring and evaluating current processes and routines within the SOC.

The interviews were conducted with five Tier 2 and one Tier 3 analyst, as we wanted the opinion of analysts with more experience. Each interview was done in a structured manner, meaning that we prepared a script with a list of questions, which we asked each of the participants in the same order. We started the interview with background questions related to the experience of the participant while working in the SOC, to get an idea of the current challenges and workflow. Afterwards, we provided the participants with the knowledge required to understand how attack graphs are generated and how can they be read. We then asked the participants to use the attack graphs in four hypothetical scenarios. These scenarios were developed to check whether the participants can correctly extract information from an attack graph, as well as to see how they would integrate them into their current workflow. Finally, we asked them to evaluate the usefulness of the different aspects of our method, as well as provide feedback and point out potential areas of improvement. Each 30-minute interview was conducted on Microsoft Teams and was recorded with the consent of the participants. The recordings were then transcribed, and the transcriptions were analyzed, by extracting the answer of each participant for every question.

The next section presents a summarised version of the answers, grouped per question. To streamline the report, we have grouped answers that align with common themes or sentiments, without attributing the responses to specific participants. Thus, when participants share similar ideas or express a collective opinion on a topic, we aggregate their responses for clarity and conciseness. Conversely, if the answers of participants express different ideas, or if their opinion diverges on a particular topic, we will address their answers independently. We have adopted such a mixed approach to ensure the

report provides maximum value to the reader, focusing on both shared insights and differing viewpoints where necessary.

6.2. Interview questions and results

6.2.1. Background questions

We started the interview with background questions regarding the experience of the participants while working in the SOC.

Q: How long have you been working as a SOC analyst?
The work experience of the participants ranged from 1 year and 3 months to 5 years, with an average of 2.54 years across all participants. We do not disclose the exact numbers for each participant, in order to avoid re-identification.

Q: What would you say is the biggest challenge you experience when handling alerts?
Participants highlighted a range of different challenges associated with handling alerts. First, multiple problems that come from data were identified. Two participants highlighted challenges in retrieving information from all available log sources, especially when an analyst needs to get an overview of what is happening for a particular host. Finding a correlation between multiple events and/or alerts was another identified challenge.

Some challenges that arise from the alerts themselves were also identified. One of the main ones is not having an overview of the alerts for a host, stated Participant 4. They further provided an example of a scenario where an alert seems to be of medium severity when analyzed on its own but is actually high severity if linked with other alerts which have been triggered in the past. Other challenges include the large number of alerts that are generated daily, as well as dealing with false positives and distinguishing them from true positives. Another identified challenge identified was the deep dive investigations, which can take multiple days. Finally, the lack of automation and the large amount of manual work which the analysts have to perform daily was another big challenge.

Q: In your opinion, do you or your colleagues suffer from alert fatigue?
The presence of alert fatigue emerged as a notable concern among participants, with four out of six acknowledging personal experiences or those of their colleagues. Participant 1 pointed out that the high frequency of repeated alerts contributes to analysts overlooking nuances in their analysis. Similarly, Participant 4 highlighted the potential for a decline in work quality attributed to the overwhelming volume of alerts. Conversely, Participant 6 expressed a more cautious stance, noting that while they don't perceive alert fatigue as an immediate issue, they believe it is a topic that demands ongoing awareness from everyone involved.

Q: What do you do when you want to look into all alerts related to a host or a particular incident, in order to get more context for an alert you are analysing?
Four participants stated that in order to get more information about a particular host or alert, they would perform an initial search in the alert ticketing system using the hostname, followed by a more in-depth investigation utilizing specific log sources, such as an Endpoint Detection and Response (EDR) solution. In contrast, the remaining two participants opted for a more direct strategy, bypassing the alert ticketing system and heading straight to the EDR to conduct their search for additional host-related events.

Q: Do you use any visualization tools, and if yes, which ones and for what?
Two participants mentioned not using any visualization tools. On the other hand, three participants shared that they occasionally use correlation graphs linked to alerts from the EDR, which show the entities related to the alert. Additionally, some participants noted using time charts selectively to examine events for a specific host, but only for certain types of alerts.

Q: Do you use any attack prediction tools?
None of the participants use attack prediction tools. Participant 5 noted that sometimes the playbook (which contains information about how to handle a type of alert) also contains information about the possible next actions of the attacker.

6.2.2. Attack graphs scenarios

Next, we provided the participants with the background knowledge required to understand the attack graphs generated by our method, by telling them how are they constructed, how can they be read, and what the different colours, shapes and symbols mean. We then showed the participants four different scenarios and asked them to either describe the actions in the attack graphs or use the attack graph when analyzing an alert.

Scenario 1: Could you describe what you see in the following attack graph? (fig. 6.1)
All of the participants were able to identify the different events and read the attack graph. The participants hovered over nodes in order to read the signatures, and using them and the timestamps from the edges, they could reconstruct the story behind the attack graph. That being said, only one participant noticed that the second node repeats once, and is composed of two separate alerts. One participant confused the prediction node with an objective node at first. Another participant made the observation that the attack stages (the title of the nodes) do not offer a lot of details about what actually happened, and without the signatures, it is difficult to get an idea of all of the events in the timeline.

Scenario 2: After more alerts come in, we regenerate the attack graph. Can you describe what has changed? What do you think is going on in the network? (fig. 6.2)
The majority of participants observed a continuity between the current attack graph and the previous one, noting that the first three nodes remained unchanged. One participant noticed that it is difficult to tell what exactly happened without having more information from the alert (for example, the name of the hack tool which was executed). Two participants inferred host compromise based on the attack graph. For instance, P5 pointed out that an attack graph with only the credential dumping node might suggest a false positive, but the presence of the entire attack chain leading to credential dumping would lead to the assumption of a compromised host.

Nevertheless, some participants faced challenges interpreting certain elements of the attack graph. One participant believed that our method would generate new alerts based on previously observed ones, while another confused the objective node with a prediction node. Lastly, a participant misunderstood the dotted nodes (sink nodes), thinking they corresponded to alerts in the attack chain that had not been triggered.

Scenario 3: You receive a low-severity alert from the NIDS for a host, which concerns suspicious activities of service discovery (such as an NMAP scan). You also see the attack graph corresponding to the alert, which showcases a sequence of past episodes for this host. For now, ignore the prediction node. (fig. 6.3)
Q1: What information from the attack graph would you use during your alert analysis?
Every participant stated that they would utilize the signatures of the nodes to check the specific types of scans and the targeted services. Additionally, some participants mentioned incorporating timestamps into their analysis to improve their understanding of the timeframe in which these events are happening. Additionally, one participant emphasized assigning a higher priority to the alert, due to the presence of multiple discovery actions.

Q2: What are the usual recommended actions for system administrators to perform for this type of alert?
Participants suggested various actions based on whether the source IP is internal or external to the environment. In either scenario, the following recommendations were commonly advised: checking that software running on the target host is up to date, reviewing open ports for any unnecessary ones, and conducting a thorough scan. If the source host is internal, participants recommended additional investigation to identify the cause of the scans. This involves scanning the internal source using anti-malware tools and questioning the owner of the host about any peculiar behaviour observed. On the other hand, if the source IP is external to the organization, a common recommendation is to block it using the firewall.

Q3: How would you change the recommendation given the current attack graph?
The majority of participants indicated that they wouldn't alter the recommendations, citing reasons such as all actions being of low severity or involving only discovery actions. Participant 2 highlighted

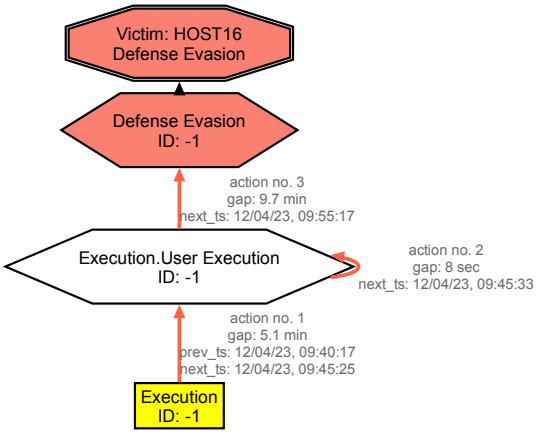

Figure 6.1: Attack graph which was shown in scenario one, generated using Northwave intrusion alerts. The ”Execution” node contains a suspicious command line signature, and the ”Execution. User Execution” contains two signatures: ”Malware detected” and ”hacking tool detected”. Finally, the ”Defense Evasion” node contains a signature of ”multiple alerts on a single host”.

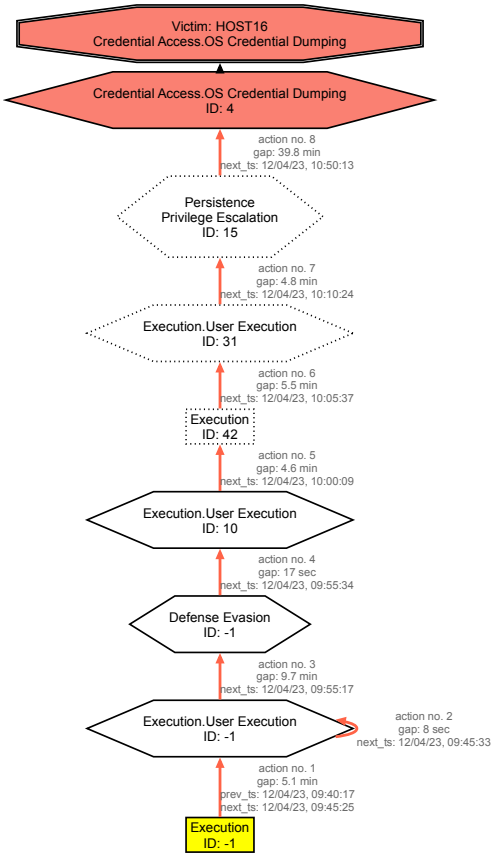

Figure 6.2: Attack graph used in the second scenario, also generated using Northwave data. It was created for the same host as the attack graph in figure 6.1, after more alerts have been triggered, consequently the first actions are the same. It can also be seen that more high-severity actions have occurred, and the attacker is further down the attack chain.

the usefulness of the attack graph in correlating multiple scanning events for the same host.

Among those willing to make changes to the recommendation, one participant suggested amending it to specify the services being scanned, which they obtained by examining the signatures of the nodes. Another participant proposed advising the limitation of traffic originating from the identified source IP as a preventive measure.

Scenario 4: Now also take into account the prediction of the next action from the attack graph, which is a high-severity action related to data manipulation (for example, modification of the privilege rights of a non-admin user)

Q1: Given the prediction, would you perform any additional actions on the host?

The majority of participants expressed a willingness to take additional actions. Among these, some participants advocated for isolating the host from the network, particularly if the prediction had a high probability. Another participant highlighted the importance of conducting further analysis on the host and considering the prediction while analyzing logs. In a slightly different approach, one participant mentioned checking the destination host for other observed behaviours from the past.

Contrarily, a participant took a more cautious stance regarding the prediction, opting not to undertake any additional actions. They justified this decision by suggesting that the current behaviour might align with the normal operations of a network administrator.

Q2: Given the prediction, how would you change the recommendation?

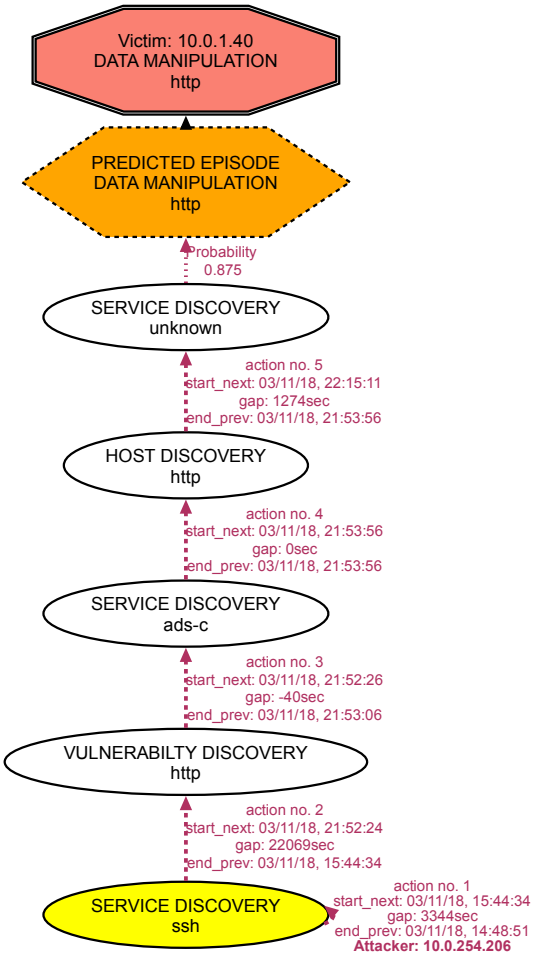

Figure 6.3: Attack graph used in the third scenario, generated using CPTC-2018 alerts. It showcases a sequence of discovery-related actions for a single host, as well as a prediction for a potential ”data manipulation” action.

The majority of participants affirmed that they would maintain the existing recommendations without modifications. However, a participant suggested recommending a check for sensitive data on the host. In contrast, another participant proposed a more proactive approach by advising the blocking of access rights for the user on the destination host and temporarily blocking the source IP pending the completion of the investigation.

6.2.3. Attack graph evaluation and feedback

Finally, after the interviewees had used the attack graph in different scenarios, we asked them to evaluate them based on different aspects.

Q: What situations do you see yourself using intrusion alert-based attack graphs in?

All participants unanimously agreed on utilizing attack graphs for summarization and correlation purposes. For instance, Participant 1 highlighted the value of intrusion-driven attack graphs in cases where multiple weak indicators which are scattered across different alerts, can be combined to form a strong indication of an attack. Several participants expressed their intention to use attack graphs while analyzing alerts to gain an overview of past alerts and the observed behaviour on a specific host. Participant 5 specifically emphasized the utility of attack graphs for high-severity alerts, where the attacker has progressed further down the attack chain, posing a more serious threat. A common sentiment among participants was that the overview offered by attack graphs plays a crucial role in enhancing analysts' understanding of the situation and the possible threats.

Q: What features of the attack graphs did you find particularly useful?
All of the participants highlighted that one of the most useful features of the attack graphs is the fact that they provide the order of the observed events as well as a timeline corresponding to these actions. Another frequently mentioned feature was the ability of attack graphs to effectively summarize information and offer a comprehensive overview of the events from one host. Participant 3 specifically noted that examining the gaps between actions could be insightful in discerning whether an alert is a false positive. Additionally, participant 4 highlighted the utility of predictions, even when appearing lower in the attack chain.

Q: What elements of the attack graphs were confusing, if any?
Participants identified multiple issues, with a shared sentiment that understanding all the elements of attack graphs would be challenging without explanations from the researcher. Assuming explanations are provided, two participants stated that there are no elements of the attack graphs which they find confusing.

The majority of confusing aspects were related to node properties. Participant 1 expressed confusion regarding using node shapes as an indicator of severity and suggested using colours or another textual indicator for more clarity. Participant 6 added that the similarity of node shapes could lead to confusion, and the varying node sizes might mislead analysts into assigning different priorities to nodes of the same severity. Participant 3 further pointed out that nodes sharing the same name (attack stage) could be confusing without having any other additional information available. Lastly, Participant 5 noted the difficulty in understanding negative gaps present in some edges.

Q: What features would you like to be added to them?
Four participants stated that if they could add a feature, they would add the ability to open the alert details in the alert ticketing system by clicking the corresponding node in the attack graph. Participant 2 proposed incorporating additional details for each alert, such as the executed command line or the specific hack tool used, though acknowledging the need to find a balance between the amount of information and the readability of the attack graph.

Moreover, Participant 2 suggested the use of colour to distinguish between different severities. Participant 3 envisioned the addition of recommendations based on the observed actions, offering guidance on how to mitigate the risks of these malicious actions. Participants 1 and 6 stated they would add more timeline-related features. The former stated that they would arrange the nodes horizontally, making the attack graph more similar to a timeline. The latter stated that they would add a separate summarized timeline near the attack graph, which showcases the timespan of the events.

Q: Assuming that the predictions are based on past alerts and are not always accurate, would you still trust them and use them during your analysis of an alert? What information can you infer from them?
Participants provided various insights into the utility of the prediction feature and its potential applications. Participant 5 characterized the feature as a valuable pointer to consider during alert analysis. Many participants emphasized the use of predictions for generating additional recommendations aimed at preventing the predicted actions. Participant 6 added that even if the prediction is incorrect, a well-formulated recommendation could still be valuable. However, concerns about the accuracy of such predictions were expressed, with participants indicating a need for testing. In a different perspective, Participant 1 suggested using predictions during alert triage as a method of increasing the priority of some alerts. For example, if the predicted action for an alert is a high-severity one, its priority would be increased, and it would be analyzed earlier compared to other alerts.

Q: How do the attack graphs compare to other visualization tools you've used in the past? How about prediction tools?
The participants who previously utilized correlation graphs from the EDR portal noted a significant advantage in the attack graphs generated by our method. They highlighted the comprehensive nature of these graphs, incorporating events from multiple alerts along with their respective timeframes, as opposed to only the entities associated with a single alert. Participant 3 further observed that our method offered a notable improvement by providing a cohesive overview, eliminating the need for manual searches to obtain the start and end times of events.

Q: On a scale of 1 to 10, how would you rate the overall usefulness of the attack graphs in assisting with your daily tasks?
The attack graphs received an average rating of 7.45 out of 10. Participant 1 expressed a willingness to elevate their grade from 5 to 8 if our method could correlate alerts from multiple hosts into a single attack graph. Participant 5 evaluated their usefulness based on the severity of the alerts, assigning a rating of 8.5 for attack graphs used for the analysis of high-severity alerts and 7 for low and medium-severity ones. Additionally, Participant 2 provided a rating of 8 for attack graphs related to NIDS (Network Intrusion Detection System) alerts, emphasizing their efficiency in summarizing numerous events associated with such alerts.

Q: On a scale of 1 to 10, how would you rate the usefulness of the prediction feature?
The prediction feature received an average rating of 7 out of 10. Participant 1, while not providing a rating for the current implementation, indicated a potential rating of 8 if the feature could be applied to alert triage. Participant 2 expressed the view that the prediction feature could be valuable for mitigating potential risks. Several participants emphasized that their decision to use the feature ultimately hinges on its accuracy, underscoring the importance of its reliability in practical applications.

Q: On a scale of 1 to 10, how easy to follow/intuitive were the attack graphs?
The intuitiveness of the attack graphs received an average rating of 8.2 out of 10. Participants noted that the graphs were initially somewhat confusing, but after receiving explanations, they found them to be quite easy to follow. Participant 3 also stated that the fact that the order of the actions is from bottom to top is beneficial when reading the attack graphs.

Q: Do you have any other comments/suggestions which weren't covered by the questions?
Participant 1 proposed an area of future improvement for the method, suggesting the creation of attack graphs that connect events from multiple hosts or the entire network. Even in the current iteration, they acknowledged that the method provides value by offering an overview of activities on a single host, aiding the analyst during alert analysis. Additionally, Participant 1 pointed out that the generic nature of the attack stages printed on the nodes limits insight into the actions. They recommended adding more details, such as the executed command line, to enrich the narrative of the ongoing attack. Participant 3 suggested another improvement by incorporating recommendations on mitigating the risks at each node. This feature, according to the participant, would aid analysts during alert analysis and enable the creation of a unified recommendation for multiple alerts.

6.2.4. Summary

In this section, we provide a summary of all of the answers and observed themes in the interview. The interviews conducted with SOC analysts have provided insights into the challenges they encounter when analyzing alerts. These challenges encompass data-related issues, such as dealing with large volumes of data and the need for effective data summarization. Additionally, alert-related challenges include finding correlations between multiple alerts and obtaining a comprehensive overview of events occurring on a single host. One participant stated: "sometimes it (an alert) comes in as medium, but if you look at the events, so many things are happening that you can make it a high (severity)". These challenges are compounded by the lack of automation, compelling analysts to manually search and correlate events. A participant highlighted the untapped potential of the available data, saying "I feel like the data is there, but we're not doing enough with it." Alert fatigue emerged as a shared concern, particularly when combined with the difficulty of distinguishing false positives from true alerts. Moreover, analysts currently lack tools that provide an effective overview of alerts associated with a single host.

Participants identified several uses for intrusion alert-based real-time generated attack graphs. All of the participants agreed that their main strength is their ability to summarize alerts and give an overview for the analyst, by providing a timeline of the observed attacker actions and their timestamps. As noted by one of the participants, "It's amazing that you have everything in one place and you have everything linked together and also have a timeline and when it happened and what kind of action was it".

Participants also identified some uses for the prediction feature. Most of them centered around providing additional recommendations of actions for the system administrator to perform, in order to prevent the predicted action from happening or to minimize its impact. One participant noted, "I think

Table 6.1: Results of quantitative questions from the interview

Question	Average rating (1-10)
General usefulness of attack graphs	7.45
The usefulness of predicting the next attacker action	7.00
Readability of attack graphs	8.20

it's a nice pointer to just take another minute or two to look into whatever the prediction says, just to be sure that you didn't miss anything". Although participants were eager to give the feature a try, they stated that its use in the real world ultimately depends on its accuracy. One participant also suggested using the prediction during alert triage, for modifying the priority of the alerts based on the severity of the predicted action.

Participants also put forth several suggestions for improving attack graphs to enhance their integration into SOC environments. These recommendations included providing more detailed information about observed actions to address issues associated with the generic nature of the attack stage. To further integrate the attack graphs into their existing workflow, they proposed adding the ability to open the alert details in the alert ticketing system by clicking on the corresponding node in the attack graph. Another requested feature was the creation of attack graphs which connect events across multiple hosts, to further improve their correlation ability. Table 6.1 presents the average ratings from the quantitative questions.

6.3. Discussion

As can be seen based on the answers of the participants, the attack graphs address some of their previously identified challenges, particularly for summarizing multiple events observed on a single host and getting an overview of those events. Additionally, attack graphs enhance the process of attack correlation, making it easier for analysts to connect the dots and discern patterns. Moreover, the automatic generation of attack graphs eliminates the requirement for manual event searches, thus contributing to an increase in automation by using currently available data. This automation streamlines the workflow within the SOC, allowing analysts to allocate their time and expertise more efficiently.

Several participants identified the presence of alert fatigue as well as some of the consequences it brings. This shows the need for real-time generated attack graphs, which can help in identifying false positives and reducing the time spent on analyzing alerts. We further observed that most of the participants were able to use the attack graph when analyzing alerts, thus we conclude that we were successful in generating attack graphs that provide additional insights for the analyst.

The feedback from participants points to a clear need for continued research into better tools for correlating and visualizing security alerts, tools that would directly aid SOC analysts in their work. Additionally, it's important that this research is carried out in close cooperation with security professionals. Working alongside these experts ensures that the new methods are not only effective but also fit well within the practical context of daily SOC operations.

The prediction feature was seen as something which could aid the analysts in handling alerts, by providing an idea of potential future attacker actions. This highlights the need for more research on the topic of attack prediction tools, as they hold significant potential in assisting SOC analysts with the early identification and circumvention of emerging threats.

Based on the observed interaction of the participants and the attack graphs, we also believe that the attack graphs could be further simplified. For example, the state ID from each node could be removed, and we could opt for making all nodes the same appearance, regardless if their corresponding state is in the main model or the sink. While these details are important for research purposes and for gaining a deeper understanding of the SPDFA's behaviour, in real-world scenarios the primary objective of attack graphs is to provide a summarised representation of observed alerts. Consequently, SOC analysts prefer a more straightforward and less cluttered representation that is easily digestible and facilitates quick comprehension. This simplification would not only enhance the user-friendliness of attack graphs but also align them more closely with the practical needs of SOC analysts.

Our interviews have provided valuable insights, but we must also acknowledge certain limitations associated with the employed methodology. Firstly, the small sample size is a potential constraint. While the insights gained are significant, they may not comprehensively represent the diversity of per-

spectives, experiences, or demographics present in the broader population, particularly in the context of other SOCs. Additionally, the limitations introduced by a small sample size extend to issues related to variability and generalization. It is important to recognize that the findings may be most applicable to the specific SOC under study.

Secondly, we recognize the potential for confirmation bias in our interviews, which is a known issue for qualitative interviews [25]. This bias can arise when the questions are formulated in a way which favours a particular outcome, thus influencing participants' responses in a certain direction. To mitigate this issue to the greatest extent possible, our approach has been to create questions that are open-ended and neutral in nature. By doing so, we aimed to provide participants with the freedom to express their perspectives without being influenced. Nonetheless, the potential for bias in responses is a consideration that we remain conscious of in our study.

6.4. Conclusions

In summary, the interviews proved that intrusion alert-based real-time generated attack graphs can be used in the real world. We can draw the following conclusions:

- SOC analysts face challenges such as difficulty in correlating events, lack of automation, and alert fatigue
- Intrusion alert-based real-time generated attack graphs are a useful visual tool for alert summarization and correlation, which could aid the analyst when handling alerts by providing more context
- The ability to provide a timeline of observed events and their order, as well as summarize information from multiple alerts proved to be the strongest point of attack graphs
- Predicting the next attacker action is a useful feature for the analysts, however, our method needs further improvements and a higher accuracy before it can be deployed in the real world
- Attack graphs can be further improved to facilitate their deployment in the real world, by integrating them with existing systems that SOC analysts use and providing more details about the observed attacker actions

7 Discussion

In this chapter, we discuss our main findings and take a closer look at the limitations and challenges associated with the method proposed in this thesis. As the results of each sub-question have already been discussed in greater detail in their corresponding chapter, this chapter aims to provide a more high-level and centralized form of discussion. Our approach brings forward innovative techniques for visualizing attacks in real-time and predicting potential future actions. However, it is crucial to acknowledge and understand the constraints that might affect its effectiveness and applicability.

7.1. Real-time generated attack graphs

Our primary research question, as delineated in the introduction, states: **How can we generate attack graphs in real-time, providing a comprehensive overview of the triggered intrusion alerts, predicting the subsequent actions of a malicious actor, and assisting a SOC analyst during the analysis of alerts?**

In this thesis, we have addressed this question by developing a novel methodology for attack prediction and real-time generation of attack graphs, which operates independently of network topology knowledge and solely utilizes intrusion alerts as input data. Furthermore, we have tested our approach using a dataset derived from a real-world SOC environment and have undertaken a thorough evaluation of its practical utility through interviews with security experts.

7.1.1. How can the SPDFA be used to predict the next attacker action?

First, we have developed a method to predict the next attacker action given a sequence of observed actions, using a pre-trained SPDFA. For the best SPDFA-based strategy, we have achieved an accuracy of 33.71 % when predicting the entire symbol, and 42.05 % when predicting only the attack stage. Given the complexity of the problem at hand, which is a multi-class classification task with 148 distinct classes, the performance of our method demonstrates promising results in this challenging context. As stated previously, this performance might be rooted in the non-deterministic nature of the reversed SPDFA. The exploration of multiple paths during prediction might introduce ambiguities, which in the end might result in a worse probability distribution of the next actions. Another possible reason might lie in the way we calculate probability distributions of the next actions, which ultimately depends on the actions observed in the training data. Due to the nature of cyberattacks, attacker strategies tend to constantly evolve, in order to avoid detection. Thus, it is possible that an attacker will perform a different action than the one observed in the training data, resulting in a wrong prediction.

Another potential cause might have been the dataset, which consists of intrusion alerts generated during a penetration testing competition where the participants were students. Due to this, the dataset contained a mix of different strategies with varying frequencies, which might have introduced additional ambiguities in the SPDFA. This underscores the need for high-quality datasets in the domain of predictive modelling. Therefore, an artificially created dataset, designed in collaboration with security experts and based on real-world attack patterns, may offer a more robust foundation for refining predictive algorithms in this realm.

On the other hand, the PDFA-based algorithm has scored an accuracy of 31.47 %, while maintaining a low execution time of 9.25e-05 seconds. Our experiments aimed at identifying the reasons behind the comparable performance of the SPDFA and PDFA algorithms did not yield definitive conclusions. This highlights that more complex algorithms do not invariably result in superior outcomes, and in real-time scenarios, the determinism and efficiency of the PDFA algorithm make it a more pragmatic choice.

7.1.2. How can we generate attack graphs in real-time, which will aid a SOC analyst when handling alerts?

Subsequently, we integrated the predictive functionality into the SAGE attack graph generation pipeline, culminating in a real-time variant. One of the biggest challenges here was finding a method to evaluate our approach. Unlike the previous research question, we could not use metrics such as accuracy, as the results for this question are of a visual nature.

The interviews with SOC analysts of Northwave Cybersecurity proved to be a great way of evaluating our approach. Real-time generated attack graphs were found to be useful for summarizing multiple events observed on a single host and getting an overview of those events. Additionally, attack graphs enhance the process of attack correlation, making it easier for analysts to connect the dots and discern patterns. Moreover, the automatic generation of attack graphs eliminates the requirement for manual event searches, thus contributing to an increase in automation by using currently available data. This automation streamlines the workflow within the SOC, allowing analysts to allocate their time and expertise more efficiently. Ultimately, this shows the ability of our method to help in dealing with alert fatigue, as it can help in identifying false positives, and reducing the time spent on analyzing alerts.

The prediction of the next attacker action was also seen as something which could aid an analyst during alert analysis, obtaining a score of 7 out of 10 on the usability scale (table 6.1). That being said, several participants have stated that its real-world usability highly depends on its accuracy. This highlights the need for more research on the topic of attack prediction tools, as they hold significant potential in assisting SOC analysts with the early identification and circumvention of emerging threats. Furthermore, the research of predictive methods should extend beyond achieving high accuracy. It is imperative that future research delves into the effective integration of these tools into SOC analysts' workflows, enhancing the ease of deployment. Concurrently, it is crucial to ensure that these predictive tools are designed with a strong emphasis on transparency and explainability — attributes that are essential for their successful implementation in real-world settings.

Through interviews with industry professionals, we have empirically validated our method and showed the demand for such tools in practice. Our findings show the presence of alert fatigue and the absence of adequate visualization tools as tangible challenges in real-world contexts, thereby emphasizing the requirement for research to address these specific issues. Future research should aim to work closely with security experts to ensure that their findings are both relevant and can be effectively applied in real-world security settings.

7.1.3. Can attack graphs be generated using data collected in the real world?

Finally, to further assess the applicability of our approach in real-world contexts, we have created and analyzed three datasets corresponding to distinct scenarios, using intrusion alerts from Northwave Cybersecurity's SOC. The datasets were created based on different characteristics of the environments which the SOC was monitoring: an environment where a penetration test had been performed (136 alerts), an environment which had a true positive high severity alert (261 alerts) and a more ”noisy” environment with a larger number of alerts compared to the others (529 alerts). A notable observation was the significantly lower volume of alerts in the real-world datasets, highlighting the discrepancies between research environments and operational realities. This observation underlines the critical importance of utilizing data that closely mirrors real-world scenarios in research aimed at addressing challenges within practical operational environments, such as SOCs. Despite these differences, and the requisite adaptations to accommodate diverse alert formats, the resultant attack graphs remained representative of the underlying data, demonstrating the feasibility of generating attack graphs from real-world data.

7.1.4. Summary

Summarizing, this thesis highlights several critical insights for the field. Firstly, the development of an attack prediction and alert visualization tool presents a significant advancement in aiding SOC analysts to manage alert fatigue and prioritize alerts efficiently. The ability of our method to both provide contextualized overviews of alerts for individual hosts and also to correlate these alerts marks a substantial step towards more coherent security incident analysis. Moreover, the innovative aspect of predicting potential future attacker actions introduces a proactive dimension to the defensive capabilities of SOCs, which are currently mostly reactive. However, this work also underscores the necessity for predictive tools to not only be accurate but to be seamlessly integrated into the analysts' workflow, emphasizing transparency and explainability as indispensable features for real-world application. Additionally, the stark contrast between the volume of alerts in real-world datasets versus research ones shows the need for data that closely reflects operational realities. These findings advocate for a research approach that privileges practical applicability and operational fidelity in developing tools that support SOC analysts in the dynamic and complex landscape of cybersecurity threats.

7.2. Limitations

While the method presented in this work offers innovative solutions for real-time attack visualization and prediction, it is not without limitations.

The reliance on intrusion alerts for graph generation may lead to incomplete representations if the underlying intrusion detection system is not comprehensively configured, potentially resulting in false negatives or positives. The predictive capability, although a novel addition, is dependent on the accuracy and reliability of the underlying model, and current results indicate a need for improvement before real-world deployment. Moreover, we have only tested the prediction algorithm on one dataset, making it difficult to objectively compare it to related works, and also to make conclusions about its performance with other training data.

Additional limitations stem from the singular focus on real-world data procured from a specific Security Operations Center (SOC). This constraint raises questions regarding the generalizability of the findings, as the characteristics of the datasets may be uniquely tailored to the SOC in question.

Lastly, the evaluation methodology employed for real-time attack graphs, specifically the utilisation of interviews, introduces potential biases. The limited participant pool, coupled with their shared workplace environment, may result in a homogeneity of experiences and feedback, potentially skewing the findings and limiting their applicability across diverse SOC contexts.

7.3. Reflections

In reflecting on the design of this thesis, there are several areas where strategic modifications could potentially enhance the quality and applicability of the research outcomes.

Firstly, given additional time and resources, we would prioritize the evaluation of our prediction method across diverse datasets. This expansion would not only allow for a more comprehensive comparison with existing methodologies but also provide a deeper understanding of the model's performance under various operational scenarios. Moreover, designing a broader spectrum of experiments could shed light on the robustness and limitations of the proposed method, offering a clearer assessment of its efficacy.

Furthermore, an exploration of alternative predictive models, such as Hidden Markov Models, could prove of value. Such comparative analysis could illuminate the relative strengths of each approach, guiding the selection of the most suitable method for the task at hand.

Secondly, the involvement of SOC analysts in the developmental cycle of real-time generated attack graphs from the earlier stages of this research would likely yield significant benefits. Establishing a continuous feedback loop with end-users would not only ensure that the resulting tool aligns more closely with operational requirements but also facilitate smoother adoption into existing workflows. This collaborative approach stands to bridge the gap between theoretical research and practical application, ultimately enhancing the real-world impact of our contributions.

8

Conclusion

Security Operation Centers are crucial for protecting companies, with intrusion detection systems playing a key role by setting off intrusion alerts. However, it's up to the SOC analysts to look at these alerts and spot any harmful behaviour.

In this thesis, we've developed a way to help SOC analysts review alerts by giving them a clear overview of the alerts linked to a specific host in the form of an attack graph. The attack graphs are generated in real-time using only intrusion alerts as input data and contain a prediction of the next potential attacker action based on the observed alerts.

The prediction can guide the analyst to notice and stop potential harmful actions in the future. We achieved this using a pre-trained SPDFA which contains known attacker behaviours. We have developed three separate SPDFA-traversal strategies, which could be used in different scenarios, depending on the required accuracy and execution time. Moreover, we have compared this method to a PDFA-based one and evaluated their performance under different circumstances.

We have achieved an accuracy of 33.71 % when predicting the entire symbol, and 42.05 % when predicting only the attack stage using the best SPDFA-based strategy. That being said, this method is limited by the large execution time of the algorithm, which further increases with the size of the input. On the other hand, the PDFA-based algorithm has scored an accuracy of 31.47 %, and has proved to have a low execution time regardless of the size of the input, factors which make it the preferred choice in real-time operational scenarios. Despite the challenges presented by a multi-class classification problem involving 148 distinct classes, coupled with a baseline accuracy of only 0.67 %, the results attained by our approach are noteworthy and indicative of its effectiveness in this complex scenario. That being said, a higher accuracy is required before this method can be used in a real-world operational setting.

We have developed two types of real-time generated attack graphs, based on the predicted next action of a sequence of alerts. Thus, if the action is low or medium severity, the attack graph combines the episodes of all the hosts where that particular prediction was made. On the other hand, if the episode sequence ends in a high-severity action, or if the predicted one is high severity, the attack graph is constructed for only one host. Due to the short runtime of the real-time pipeline, attack graphs can be generated for every alert that triggers, which allows the analyst to have a better understanding of the context of an alert during analysis, reducing the amount of manual work required.

We have demonstrated that our approach can be used on open source as well as real-world data and that in both cases the resulting attack graphs provide a meaningful representation of the data. Moreover, we have evaluated our approach by organizing interviews with security experts. This way, we have grounded our research in practical reality, ensuring that our contributions are relevant and applicable in a real SOC environment.

8.1. Contributions

In summary, we have successfully developed and validated a tool designed to enhance the SOC analyst's ability to analyze alerts, demonstrated its effectiveness on real-world data, and confirmed its capacity to address prevalent challenges in the field. Furthermore, our work distinguishes itself from

existing literature by concurrently offering alert correlation, attack visualization, and attacker action prediction, underscoring the innovative nature of our approach. Our main contributions are:

- **Prediction of Future Attacker Actions** - We have developed a method of predicting the potential future attacker actions based on the observed intrusion alerts using an SPDFA. We have implemented three SPDFA traversal strategies and compared them with a PDFA-based approach under different circumstances.
- **Real-time generated attack graphs** - We have created a tool that aids SOC analysts in interpreting and responding to alerts generated by intrusion detection systems. Real-time generated attack graphs provide a visual overview of the triggered alerts for a host, while also displaying the possible next action of the attacker. This not only helps in correlating different alerts but also offers valuable context to each triggered alert, aiding in the more efficient identification of malicious activities. The predictive capability helps in proactively identifying and mitigating threats, potentially minimizing the impact of harmful activities. SOC analysts can now focus on the most critical alerts, enhancing their ability to respond to potential threats swiftly.
- **Validation through Expert Interviews** - We have conducted interviews with six security experts to validate our approach, ensuring that our method is aligned with the real-world needs of SOC analysts. Additionally, the interviews offer valuable perspectives on the existing difficulties encountered by SOC analysts related to visualization and correlation methods, which could be beneficial for other studies seeking to assist SOC analysts.
- **Application on Real-World Data** - We have demonstrated the applicability and effectiveness of our method in real-world settings, by testing it on three datasets created using real-world industry data, showcasing its practical utility.

8.2. Future work

We now outline potential avenues for future research, aiming to build upon the foundations laid in this work and address its limitations. The enhancement of the prediction method stands as a primary objective. The PDFA-based prediction algorithm, for instance, could be refined through the exploration of multiple paths within the PDFA, rather than following a single one. Alternatively, the SPDFA-based algorithm might see improvements in efficiency and runtime through the implementation of early path-pruning techniques. Adopting a hybrid strategy may also prove beneficial; this would entail utilizing the SPDFA for short-sequence predictions and the PDFA for longer sequences. Furthermore, additional empirical testing with diverse datasets is imperative. Our hypothesis suggests that a training set comprised exclusively of intrusion alerts from specific real-world attack scenarios would yield more accurate predictions, decrease ambiguities, and result in a more compact and efficient SPDFA.

Improvements are not confined to the prediction methods; the attack graphs themselves present substantial opportunities for refinement. Insights gleaned from interviews indicate that analysts frequently deal with events distributed across multiple hosts. Addressing this, future research efforts could be directed towards developing methodologies to generate coherent and comprehensible attack graphs that seamlessly integrate alerts from multiple hosts. Enhancing the alert correlation capabilities is another potential focus area, extending beyond hostname-based correlations to encompass additional indicators. Lastly, by incorporating more detailed information from intrusion alerts, the attack graphs could be rendered more comprehensive, ultimately enhancing their practical utility in real-world scenarios.

Bibliography

[1] Tadeusz Pietraszek. “Using adaptive alert classification to reduce false positives in intrusion detection”. In: *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings 7*. Springer. 2004, pp. 102–124.

[2] Wajih Ul Hassan et al. “Nodoze: Combatting threat alert fatigue with automated provenance triage”. In: *network and distributed systems security symposium*. 2019.

[3] Chuck Brooks. *Advancing The Security Operations Center (SOC): New Technologies and Processes Can Help Mitigate Cyber Threats*. Accessed: 2023-11-04. 2023. URL: https://www.forbes.com/sites/chuckbrooks/2023/04/26/advancing-the-security-operations-center-soc-new-technologies-and-processes-can-help-mitigate-cyber-threats/?sh=5ce4880836c9.

[4] Manfred Vielberth et al. “Security operations center: A systematic study and open challenges”. In: *IEEE Access* 8 (2020), pp. 227756–227779.

[5] Alexandre Kabil et al. “3D cybercop: A collaborative platform for cybersecurity data analysis and training”. In: *Cooperative Design, Visualization, and Engineering: 15th International Conference, CDVE 2018, Hangzhou, China, October 21–24, 2018, Proceedings 15*. Springer. 2018, pp. 176–183.

[6] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. “Practical attack graph generation for network defense”. In: *2006 22nd Annual Computer Security Applications Conference (ACSAC’06)*. IEEE. 2006, pp. 121–130.

[7] Azqa Nadeem et al. “Alert-driven attack graph generation using s-pdfa”. In: *IEEE Transactions on Dependable and Secure Computing* 19.2 (2021), pp. 731–746.

[8] Awalin Sopan et al. “Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC”. In: *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE. 2018, pp. 1–8.

[9] Sicco Verwer and Christian A Hammerschmidt. “Flexfringe: a passive automaton learning package”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2017, pp. 638–642.

[10] Stephen Moskal and Shanchieh Jay Yang. “Cyberattack action-intent-framework for mapping intrusion observables”. In: *arXiv preprint arXiv:2002.07838* (2020).

[11] Rafael C Carrasco and Jose Oncina. “Learning stochastic regular grammars by means of a state merging method”. In: *Grammatical Inference and Applications: Second International Colloquium, ICGI-94 Alicante, Spain, September 21–23, 1994 Proceedings 2*. Springer. 1994, pp. 139–152.

[12] Shanchieh Jay Yang et al. “Near Real-time Learning and Extraction of Attack Models from Intrusion Alerts”. In: *arXiv preprint arXiv:2103.13902* (2021).

[13] Abdulellah Alsaheel et al. “ATLAS: A Sequence-based Learning Approach for Attack Investigation.” In: *USENIX Security Symposium*. 2021, pp. 3005–3022.

[14] Stephen Moskal and Shanchieh Jay Yang. “Heated Alert Triage (HeAT): Network-Agnostic Extraction of Cyber Attack Campaigns”. In: (2021).

[15] Hao Hu et al. “Attack scenario reconstruction approach using attack graph and alert data mining”. In: *Journal of Information Security and Applications* 54 (2020), p. 102522.

[16] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. “MulVAL: A Logic-based Network Security Analyzer.” In: *USENIX security symposium*. Vol. 8. Baltimore, MD. 2005, pp. 113–128.

[17] Alexander Hofmann and Bernhard Sick. “Online intrusion alert aggregation with generative data stream modeling”. In: *IEEE transactions on dependable and secure computing* 8.2 (2009), pp. 282–294.

[18] Gordon Werner, Shanchieh Jay Yang, and Katie McConky. “Near real-time intrusion alert aggregation using concept-based learning”. In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 2021, pp. 152–160.

[19] Pilar Holgado, Víctor A Villagrá, and Luis Vazquez. “Real-time multistep attack prediction based on hidden markov models”. In: *IEEE Transactions on Dependable and Secure Computing* 17.1 (2017), pp. 134–147.

[20] Ali Ahmadian Ramaki, Masoud Khosravi-Farmad, and Abbas Ghaemi Bafghi. “Real time alert correlation and prediction using Bayesian networks”. In: *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*. IEEE. 2015, pp. 98–103.

[21] Daniel S Fava, Stephen R Byers, and Shanchieh Jay Yang. “Projecting cyberattacks through variable-length markov models”. In: *IEEE Transactions on Information Forensics and Security* 3.3 (2008), pp. 359–369.

[22] Udaya Sampath K Thanthrige, Jagath Samarabandu, and Xianbin Wang. “Intrusion alert prediction using a hidden Markov model”. In: *arXiv preprint arXiv:1610.07276* (2016).

[23] Alireza Shameli Sendi et al. “Real time intrusion prediction based on optimized alerts with hidden Markov model”. In: *Journal of networks* 7.2 (2012), p. 311.

[24] N-able. *Penetration Testing Methods*. URL: `https://www.n-able.com/blog/penetration-testing-methods` (visited on 07/25/2023).

[25] Ronald J Chenail. “Interviewing the investigator: Strategies for addressing instrumentation and researcher bias concerns in qualitative research.” In: *Qualitative report* 16.1 (2011), pp. 255–262.