Investigating the impact of PDFA implementation on alert-driven attack graphs

A comparison between the Suffix-based PDFA and PDFA models

Ioan-Cristian Oprea[1]

Supervisor(s): Sicco Verwer[1]**, Azqa Nadeem**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 28, 2023

Name of the student: Ioan-Cristian Oprea
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Azqa Nadeem, Asterios Katsifodimos

Abstract

SAGE is a deterministic and unsupervised learning pipeline that can generate attack graphs from intrusion alerts without input knowledge from a security analyst. Using a suffix-based probabilistic deterministic finite automaton (S-PDFA), the system compresses over 1 million alerts into less than 500 attack graphs (AGs), which are concise and manageable. Unlike other frequency analysis methods, SAGE does not discard infrequent high-severity alerts, which are crucial for learning the penetration strategies of attackers. This paper compares the baseline algorithm (i.e. S-PDFA) with a modelling assumption generated by swapping the S-PDFA with a PDFA. The aim is to validate the quality of SAGE and propose possible solutions for PDFA usage, allowing the algorithm to generate AGs in real-time. We compare them both quantitatively and qualitatively using size, complexity, completeness and interpretability metrics. Our findings show that AGs generated by the PDFA are more readable and as complete while being slightly larger (i.e. 16% larger) than the baseline S-PDFA. In certain cases, it can also better capture different attack strategies, proving that, if further optimized, it can perform better than the baseline.

Keywords: attack graph, S-PDFA, PDFA, cybersecurity, intrusion alerts

1 Introduction

Security operations centres (SOC) receive thousands of alerts daily. However, most of those threats are either low-severity threats or false positives [1]. Nonetheless, the daily responsibility of a SOC analyst is to inspect and filter out all these alerts, which in most cases, is an unfeasible task. This much cleaning is a pilot to alert fatigue [2], which can lead to burnout, misclassified alerts or even ignored alerts altogether. All this increases the probability of a real cyberattack going undiscovered.

One of the approaches to easing out this process is to analyze attack graphs (AGs), which in essence, are a graphical representation of all the paths that an intruder takes to achieve a potential goal [3]. Systems that automatically generate these graphs from security alerts are already in place. However, they require input knowledge about the network and the topology of the systems [4], making them impractical against newly introduced vulnerabilities (i.e., zero-day attacks.). Therefore, a reliable system that can generate these attack graphs without prior knowledge from a security analyst is in need.

Recently, *Nadeem et al.* has bridged the knowledge gap by developing SAGE [5] (Intrusion alert-driven attack graph extractor), which generates attack graphs from security alerts without the input knowledge of a security analyst. It uses a Suffix-based Probabilistic Deterministic Finite Automaton [6] (S-PDFA) to model the attacker's paths learned from the intrusion alerts into concise and more manageable graphs, compressing over 1 million alerts into less than 500 AGs.

There is still much work to be done in this area, mainly due to the difficulty of measuring the quality of resulting attack graphs. One way is to learn and compare different models to the baseline model based on different metrics. This paper aims to *compare the attack graphs generated by swapping the baseline S-PDFA implementation with a PDFA implementation*. The analysis is done using several quantitative metrics such as *size* or *complexity* and qualitative metrics such as *completeness* or *interpretability.*

Additionally, an improved PDFA implementation would allow us to generate real-time attack graphs, which at the moment is not possible, and extend SAGE's uses beyond only forensics analysis. With these in mind, the research topic was split into multiple subquestions:

- What evaluation criteria can be used to quantifiably compare attack graphs in terms of: **size, complexity, completeness and interpretability**?
- How does a PDFA implementation, as opposed to an S-PDFA, influence the learning algorithm?
- Is the PDFA implementation better at modelling the **real world attack paths** than the S-PDFA?

Before any experimental analysis was conducted, a hypothesis on the changes in attack graphs generation was proposed: *The PDFA implementation will result in a higher number of high severity nodes, which creates bigger and more complex attack graphs.* This hypothesis will be analyzed further in Section 5.

The main contribution of this paper is the analysis of the attack graphs and the alerts generated from them. This is in the form of both automated scripts and manual investigations. We propose metrics for analyzing the interpretability and completeness of the graphs without the existence of prior ground truths to compare. We create several Python scripts to filter out candidates and reduce the search space for the manual examination.

The paper is structured as follows: Section 2 describes related work that helps the reader understand the content better. Section 3 outlines the methodology and the metrics used during the analysis. Section 4 discusses details on recreating the experiments. Section 5 summarizes the results of the analysis and their inherent causes. In contrast, Section 6 presents the conclusion of the analysis, possible limitations and any future work about the paper's topic. Section 7 describes any ethical concerns which might have arisen during research. Finally, Section 8 acknowledges individuals who helped towards this research goal.

2 Related Work

Understanding the nuances of this paper requires the reader to be familiar with certain frameworks and algorithms used throughout the research. These can be systems that are either utilized *directly* (i.e. SAGE) or *indirectly* (i.e. FlexFringe) during or for the analysis. This section will provide a concise overview of these mechanisms, beginning with FlexFringe and concluding with SAGE and its attack graph generation.

2.1 FlexFringe

FlexFringe [6] is a software package that learns a Probabilistic Deterministic Finite State Automaton (PDFA) from a series of traces. By learning automata from trace data, FlexFringe can analyze and model various types of complex software systems. In our case, it is used to learn the attackers' behaviour and model their attack paths.

The process begins by utilizing a tree-like model (i.e. *prefix tree*) that directly models the input traces. Afterwards, it combines states by verifying their past and future behaviours. The result is a compact model that reveals the data's hidden system states and transition structure. An example of the output of FlexFringe for the CPTC-2017 dataset can be seen in Figure 1.

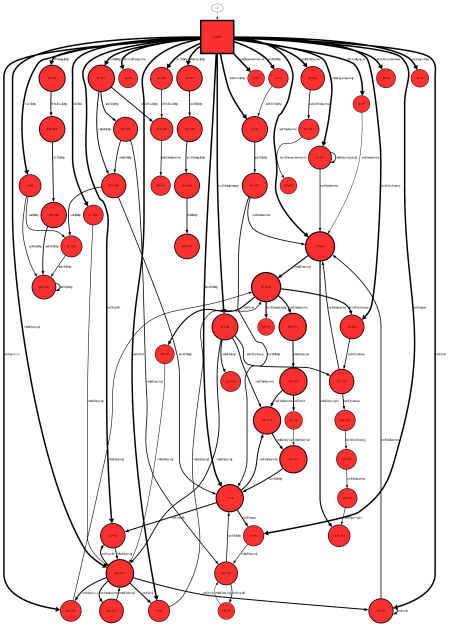


Figure 1: S-PDFA for the CPTC-2017 dataset. Traversing the tree and reading the edges shows the progression of an attack, but in reverse order as the input traces are flipped before the learning process.

2.2 SAGE

SAGE [5] is a deterministic algorithm that can generate attack graphs from security alerts without prior knowledge of the network topology. It uses a suffix-based implementation (i.e. the input traces are reversed) of FlexFringe to learn the attackers' behaviour. The reason behind using FlexFringe instead of other machine learning approaches is that they are notorious for discarding infrequent data [7]. In our case, most high-severity threats are infrequent and discarding them would break the underlying principle of SAGE.

SAGE starts by preprocessing the alerts and extracting certain features from their signature. Then it gathers the alerts into alert sequences (AS) based on source and destination IP addresses. Afterwards, it aggregates these alert sequences into episode sequences based on the timestamp and creates the traces from which FlexFringe learns the suffix-based PDFA.

Lastly, attack graphs (AGs) are created individually for every victim and objective, with the objective representing a high-severity attack stage. An example of an attack graph can be seen in Figure 2.

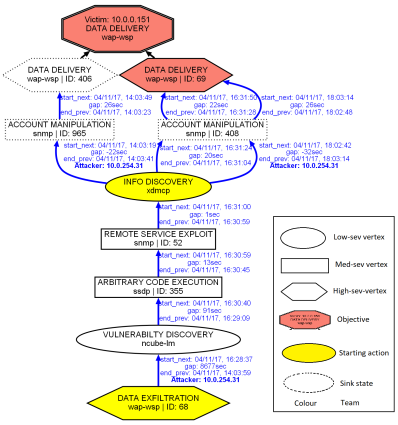


Figure 2: Attack graph legend. To follow a path, you start at a yellow state and follow a path with the label *Attacker*. Then, per node, you follow the edge that has the same end_prev timestamp as the $start\_next$ timestamp of the previously followed edge. All attack paths have only one correct traversal.

The current problem is that the validity of this model is difficult to assess. It is mainly due to the inexistence of ground truth attack graphs to compare. One proposed strategy is to learn different models, in this paper's case, a PDFA, and pick the one that creates the most intuitive graphs. Details on the analysis and the discussion of the results themselves will be presented in the subsequent sections.

3 Methodology

This section presents the methodology of this research project. It first starts by presenting the problem statement in 3.1. Afterwards, it briefly overviews the hypothesis in 3.2 and the experimental workflow in 3.3. Finally, Subsection 3.4 defines all the metrics used throughout this research.

3.1 Problem Statement

A current limitation of SAGE is the inability to generate real-time attack graphs (i.e. while the attack is underway). This is because it uses a Suffix-based PDFA to predict the past from the future [5]. Unfortunately, it makes the pipeline usable only for forensics, not for real-time monitoring of possible intrusions. This paper aims to verify how a PDFA implementation compares with the baseline model and to explore and suggest possible improvements of the modelling assumption for possibly extending the use cases of SAGE.

3.2 Hypothesis

Before conducting any experiments, we hypothesise how the new learning algorithm affects the outputted attack graphs. We believe that because the PDFA implementation does not flip the prefix tree [6], it will result in more high-severity nodes either due to the algorithm not merging them or due to

the way the traces are traversed in the post-processing. These statements will be discussed and explored in Section 5.

3.3 Experimental Workflow

The investigation used quantitative measures such as *size and complexity* and qualitative measures such as *completeness and interpretability*. The qualitative analysis was accomplished by first filtering out candidates based on quantitative analysis (i.e. graphs with more drastic changes in complexity or size were analyzed first) and then manually inspecting the differences within the graphs.

The theories of the differences were backed up by an investigation of the alert sequences related to the victim's IP address and targeted objective. In the absence of ground truth values, these alerts were also used to compute the completeness metric, which will be discussed in the subsequent paragraphs. More details on the setup of experiments can be found in Section 4.

3.4 Metric Definition

Size

The chosen metric for **size** is the *number of nodes*. This metric is easy to compute and understand, and is also used in the original SAGE paper [5]. This way, the comparison between the two reports becomes easier.

Complexity

As for complexity, we use the process introduced by De Alvarenga *et al.* [8]. This method uses minimum and maximum values to determine how the complexity is calculated. If the number of vertices is smaller than the minimum, the graph automatically has a complexity of 0. In comparison, if the number of vertices is larger than the maximum, the graph has complexity 1. However, if the number of vertices falls between these thresholds, we use the inverse of the simplicity measure: $simplcity(G) = \frac{|\mathrm{V}|}{|\mathrm{E}|}$, where |V| and |E| are the number of vertices and edges of the graphs.

In our analysis, the minimum and maximum values selected were 11 and 26 for the CPTC-2017 and 13 and 24 for CPTC-2017. Due to the low number of resulting attack graphs, a knowledge base procedure [8] cannot be effectively used to pick these parameters. Therefore, looking at the distribution of the number of nodes, the lower threshold was picked as the 25^{th} percentile and the upper threshold as the 75^{th} percentile.

$$complexity(G) = \begin{cases} 0 & \text{if } |\mathrm{V}| < \min \\ \frac{|\mathrm{E}|}{|\mathrm{V}|} & \text{if } \min \leq |\mathrm{V}| \leq \max \\ 1 & \text{if } |\mathrm{V}| > \max \end{cases} \quad (1)$$

Afterwards, we perform linear regression on the graphs with vertices between the minimum and maximum threshold in order to decide on a decision boundary t_s and on a classification function:

$$isComplex(G) = \begin{cases} \text{No} & \text{if complexity(G)} < \mathrm{t_s} \\ \text{Yes} & \text{if complexity(G)} > \mathrm{t_s} \end{cases} \quad (2)$$

Completeness

Determining the completeness of an alert-driven attack graph is a complicated task. This is mostly due to the inexistence of ground truth values to compare the results. Consequently, we assume that the alert sequences can be used to derive these ground truth statements.

This observation does not hold in a holistic evaluation because Security Information and Event Management (SIEM) systems can present their own problems. For example, SIEM software generates a large number of false positives [1] for which every complete alert-driven attack graph generation tool should be able to filter out.

However, in our context, the only adjustment over the baseline model is introducing a PDFA instead of the S-PDFA. Given that this stage is not responsible for data cleaning, we believe that a layered evaluation that uses the alert sequence as ground truth is an appropriate method of computing completeness.

Subsequently, for our analysis, we adapt the definitions of schema and population completeness of knowledge graphs [9] in relation to attack paths and the final objectives of the AGs. In essence, it measures the ability of the algorithm to model the real world with regard to additions or subtractions of properties of graphs.

Schema completeness refers to the extent to which the classes and properties within an ontology are adequately portrayed in a linked data dataset. In our context, it represents the degree to which paths are missing in one particular attack graph and can be computed as:

$$schemaComp(G) = \frac{\text{\#paths present in AG}}{\text{\#total individual paths}} \quad (3)$$

Population completeness refers to the extent to which a linked data dataset adequately represents all real-world objects of a specific type. In the context of SAGE, it describes the degree to which all the high-severity alerts have a matching attack graph generated by the algorithm. It can be computed as:

$$popComp(G) = \frac{\text{\#unique objectives present in AG}}{\text{\#total unique objectives}} \quad (4)$$

It is important to look at completeness from both these two perspectives because, in real life, there can be a substantial difference between not detecting an attack for a service that was previously attacked and is already vulnerable or not detecting an attack for a service that has not been previously attacked and is not known as vulnerable yet.

Details on how this data is generated will be discussed in Section 4.

Interpretability

Interpretability is the last of the qualitative approaches that we used throughout the analysis. From the perspective of a security analyst, it is an essential measurement to discuss, as it is redundant for them that the graph is complete unless it is also readable and interpretable. Without this characteristic, one can find themselves drawing the wrong conclusions. At the same time, it is also a challenging metric to quantify.

With these in mind, a protocol was designed to assess the readability of the attack graphs using inspiration from [10]. A set of experts is asked to perform a series of tasks, and the completion of these tasks is timed. The tasks are:

1. Estimate the number of nodes (part of intervals, e.g. 0-10, 10-20)
2. Estimate the number of attack paths
3. Locate a node based on a given label
4. Locate all medium-severity states
5. Locate all high-severity states
6. Follow an attack path from a start state to the victim state

However, performing the protocol for all the attack graphs and both the baseline and the PDFA models is very time-consuming for this project's scope. At the same time, it is also unnecessary because some graphs remain the same between the implementations and do not aid the comparison. Therefore the following selection was made:

- The biggest difference in the node count (top 5 AGs)
- The biggest difference in complexity (top 5 AGs)
- Simple AGs that became complex (at most 5 AGs)
- Complex AGs that became simple (at most 5 AGs)

In addition to this protocol, we also gave some insights which give an idea of the interpretability of the graphs concerning the ability to detect distinct penetrations strategies of attackers. These will be presented in Section 5.

4 Experimental Setup

The attack graphs analyzed in this paper were generated using the 2017 and 2018 datasets of the Collegiate Penetration Testing Competition [11] [12] (CPTC). The algorithm was run twice on each dataset, once with the original implementation, which flips the prefix tree and thus learns an S-PDFA, and once with a regular PDFA model. FlexFringe and SAGE were run on the versions containing only the Pull Requests up to 9^{th} of June. The repository linked in the appendix already contains the correct versions.

To change between the PDFA and S-PDFA implementation, the input traces were reversed before loading them into the FlexFringe algorithm. Afterwards, the graphs between the different implementations were matched head-to-head based on the victim's IP address and the target objective.

	CPTC-2018	CPTC-2017
# alerts	330,270	43,611
# teams	6	9
Duration (hrs)	8	11
Victim hosts known?	Yes	No
Competition type	Pen. testing	Pen.testing

Table 1: Summary of experimental datasets [13].

The alert sequences were extracted after the episodes were broken up into sub-sequences and before traces were generated to be fed into the FlexFringe algorithm. Subsequently, candidates for the manual analysis of alert sequences were filtered out using the following procedure:

1. For each attack graph, the starting edge of all attack paths was stored as a tuple containing the following information $< victim\_IP, end\_prev, mcat, protocol >$
2. Each alert was modified to contain the same information as the attack paths
3. If a sequence contained an alert which matched the attack path information, the entire sequence was filtered out as there is enough evidence that it is modelled in the results
4. SAGE takes into account only paths that lead to a high-severity node, therefore sequences that did not contain at least one high-severity alert were removed

The remaining alerts were manually inspected to determine whether the sequences are already accounted into the attack graphs but at different timestamps or whether they are truly missing in the interpretation.

NetworkX was also used to perform operations on the resulting graphs, for example, for extracting the starting states and starting edges or for the computation of the complexity metric. It can read the ".dot" files outputted by SAGE and store them as a graph data structure.

5 Results

This section presents the results of the various analysis methods conducted throughout the research. It first outlines some statistics that confirm the hypothesis presented in the introduction. Then it discusses the results of the size, complexity, completeness and interpretability analysis between the baseline and PDFA models.

5.1 High-Severity Nodes Analysis

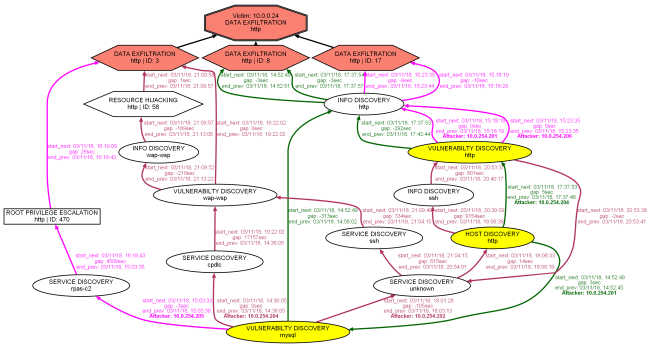
We started this paper by introducing a hypothesis on how the learning algorithm is affected by the PDFA model. We assumed that the attack results would have more high-severity nodes. From Table 2, we can see that for the CPTC-2017, there is an increase in the number of sub-objectives (i.e. high-severity nodes directly connected to the final objective) of 1.48 over the baseline model, while for the 2018 dataset, the ratio is 1.593. When looking at the high-severity nodes, these numbers increase to 1.722 and 1.908, respectively. Therefore, these results confirm the hypothesis.

Dataset	Sub-objectives	Total high-sev. nodes
CPTC-2017	1.722	1.480
CPTC-2018	1.908	1.593

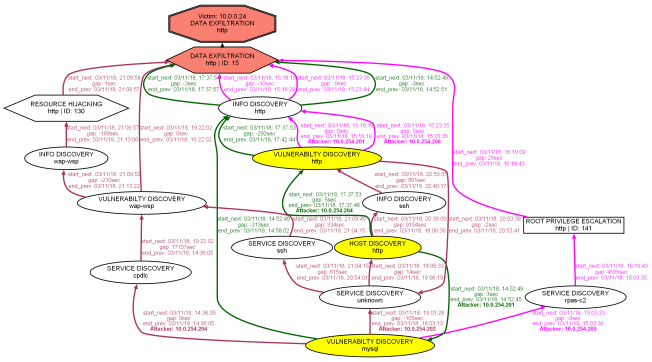
Table 2: Ratio of high-severity nodes in the PDFA implementation over high-severity nodes of the baseline per AG.

Figure 3 shows a clear example of this behaviour. Here we can see that, compared to the baseline model, there are three sub-objectives, even though the two rightmost sub-objectives have similar nodes going into *DATA MANIPULATION*.

At first glance, one might think that this behaviour is caused due to the merging algorithm not being able to merge these correctly. However, by investigating this behaviour more in-depth, we can see that the more probable cause is



(a) PDFA



(b) S-PDFA

Figure 3: Attack graph for *DATA EXFILTRATION http* for victim 10.0.0.24. It shows how the modelling assumption has more sub-objectives than the baseline and how the paths are grouped based on the starting node of the attack.

the traversal of traces in the learned automaton. In the baseline model, the traces are traversed from a high-severity node to a low-severity node. With the PDFA model, it is the opposite. In both models, paths are grouped based on the first alert of the trace. Consequently, this means that in the PDFA, paths are grouped based on low-severity nodes rather than high-severity nodes, which explains why in 3a each path of an individual sub-objective starts in the same node: leftmost three nodes in *VULNERABILITY DISCOVERY mysql*, middle two paths in *HOST DISCOVERY http* and the last two nodes in *VULNERABILITY DISCOVERY http*. Although the individual paths might differ in the baseline model, they will start at the same high-severity node and thus be visually merged in the AG.

5.2 Size Analysis

The behavioural difference between the PDFA and S-PDFA also affects the size of the resulting attack graphs. From Table 3 and 4, we can see an average increase of around three nodes between the implementations, while for some graphs, we even discovered an increase of more than ten nodes.

This allows us to conclude that, in the future, we could possibly use a PDFA implementation without drastically compromising the size of individual attack graphs.

Dataset	Avg size	Min size	Max size
CPTC-2017	17.703	3	48
CPTC-2018	17.173	3	32

Table 3: Size values of the baseline model.

Dataset	Avg size	Min size	Max size
CPTC-2017	20.842	3	70
CPTC-2018	19.946	3	43

Table 4: Size values of the PDFA model. It shows how the differences in size between the modelling assumption and the baseline are minor

5.3 Complexity Analysis

In terms of complexity, we can see from Table 5 that there are not many differences. For CPTC-2017, there is no change in the number of complex graphs between the baseline and PDFA model, while for 2018, the difference is negligible (i.e. 3).

Dataset	Complex S-PDFA	Simple S-PDFA	Complex PDFA	Simple PDFA
CPTC-2017	50	58	50	58
CPTC-2018	36	39	39	36

Table 5: Complexity values between the baseline and PDFA. It shows how, even with the increase in size, the modelling assumption is not drastically affected in terms of complexity.

These can be explained using the hypothesis about the behavioural difference between the S-PDFA and PDFA algorithms. As discussed in *High-Severity Nodes Analysis*, the PDFA model has problems visually merging high-severity nodes. These, however, appear close to the unique objective, and therefore, for each new node, there is a new edge that goes to this unique objective node. The consequences are that, because of how the complexity is defined, the differences are very marginal and, thus, Equation 2 does not affect the output.

This allows us to conclude that, in the future, we could possibly use a PDFA implementation with marginal effects on the complexity of individual attack graphs.

5.4 Completeness Analysis

The completeness analysis is the first qualitative measurement conducted between the two implementations. We aimed to verify whether the models can correctly represent the real world concerning additions or subtractions of properties of graphs. With this regard, we verified if there are any inconsistencies between the episodes that Flexfringe is trained on and the attack sequences of the AGs generated by SAGE.

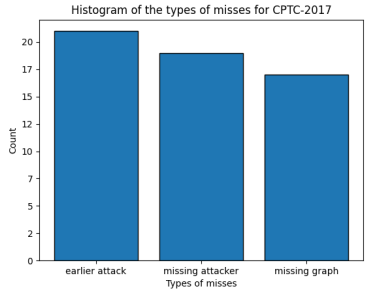
As discussed in the *High-Severity Nodes Analysis* subsection, the proposed model does not affect the underlying paths that SAGE generates, but only the merging approach and the number of nodes/edges. Therefore, we can expect no differences in terms of completeness given the metrics defined in Section 3. Considering the reason for analysing the differences between the two designs is to assess the validity of SAGE, we decided not to make a head-to-head comparison in this section.

From Table 6, it is evident that for CPTC-2017, the algorithm correctly models 87.33% of the total paths into the AGs and 82.44% of the total high-severity alerts into unique AGs. For the 2018 dataset, these numbers become 88.01% and 76.53%, respectively. These values may seem adequate. However, the objective of cybersecurity software is to minimise the production of false negatives and positives [14], which does not apply in our case.

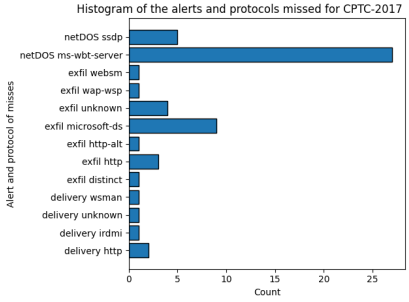
Dataset	schemaComp	populationComp
CPTC-2017	87.33%	82.44%
CPTC-2018	88.01%	76.53%

Table 6: Completeness for baseline and PDFA model (values are the same between the models). It shows how completeness is not affected between implementations, but also how it is fairly low for cyber-security software.

Given Figure 4a, we can observe a somewhat even distribution of false negatives types for the 2017 dataset, with attacks not being detected as early as possible being the most frequent miss. From Figure 4b, one can notice that *NETWORK DOS ms-wbt-server* has the most inconsistencies (i.e. 23) followed by *DATA EXFILTRATION microsoft-ds* with nine inconsistencies, and *NETWORK DOS ssdp* with five inconsistencies.



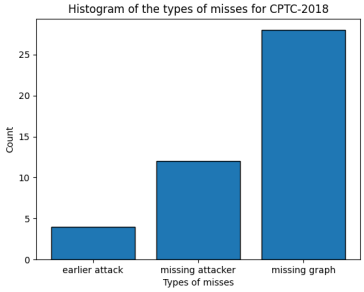
(a) Distribution over the type of false negatives



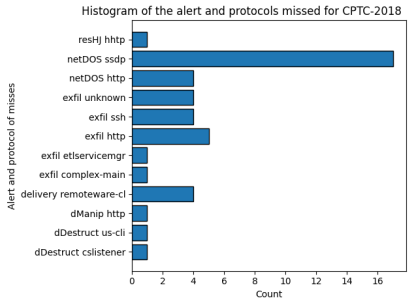
(b) Distribution over the alerts and protocols

Figure 4: Distribution of false negatives on the CPTC-2017 dataset. It gives an idea of what attacks are missed when episodes of length smaller than three are discarded.

Referring to Figure 5a, we can recognise that the most frequent type of false negative is the absence of a unique AG given a high-severity alert, with a count of 27. Analysing Figure 5b, *NETWORK DOS ssdp* shows the most significant number of inconsistencies, specifically 18. Following that, *DATA EXFILTRATION http* has five inconsistencies, and *NETWORK DOS http, DATA EXFILTRATION unknown, DATA EXFILTRATION ssh, DATA DELIVERY remotware-cl* all have five inconsistencies.



(a) Distribution over the types of false negatives



(b) Distribution over the alerts and protocols

Figure 5: Distribution of false negatives on the CPTC-2018 dataset. Compared to CPTC-2017, we can see that for the 2018 dataset, SAGE is more prominent in missing entire graphs.

The reason behind these discrepancies is that SAGE discards all episodes with less than three alerts. This is to reduce the number of paths and attack graphs generated. However, it results in the algorithm missing valuable information if one of these episodes contains a high-severity threat. The next subsection will discuss one possible modification to solve this problem and the result analysis.

5.5 Completeness Improvement Analysis

One possible solution to the problem showcased in the previous sub-section is keeping all episodes, regardless of length. Applying this technique decreases the number of missed paths from 56 to 8 for CPTC-2017 and 43 to 11 for the 2018 dataset. This is a considerable decrease in the number of false negatives. However, it comes at the cost of generating substantially more attack paths, from 393 to 538 and 323 to 386, respectively.

The reason for this is that it considers many false positives, i.e., subpaths of paths that already exist in attack graphs. This is because attackers tend to follow shorter paths to re-exploit the same objective [5]. In most cases, capturing these insights

is useful, though, in our case, the number of such paths is so large that it makes the AGs less interpretable, thus, less qualitative.

Another interesting aspect discovered when analysing this experiment's results is that some episodes with less than three alerts translated into attack paths of more than three alerts when adding them to the learning step of SAGE.

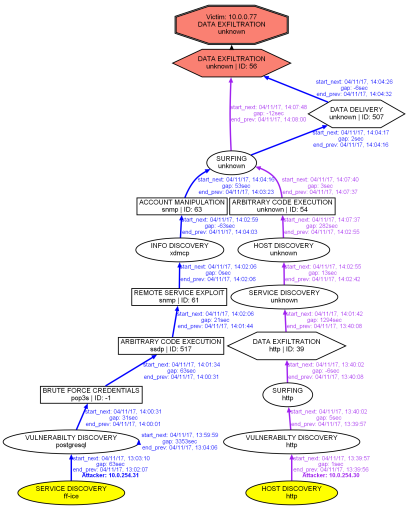


Figure 6: Attack graph for victim 10.0.0.77 for *DATA EXFILTRATION unknown* when SAGE uses all episodes to learn from. It shows how SAGE, by discarding episodes with length smaller than three, removes longer attacks.

For example, for victim *10.0.0.77* and attacker *10.0.254.30*, we have the following episode that SAGE discards because of its length:

- (['2017-11-04 14:07:40.394839+00:00', '2017-11-04 14:08:00.600665+00:00'], 'surf', 'unknown')
- (['2017-11-04 14:07:48.259365+00:00', '2017-11-04 14:08:00.690355+00:00'], 'exfil', 'unknown')

When SAGE does not discard it, it results in the purple attack path from Figure 6. In this case, we can see that the attack does not start at a *SURFING unknown* node as it is showcased in the episode, but from a *HOST DISCOVERY http* node. The cause is that the information for this particular attack is not stored in only one episode. However, it is split into multiple ones, which might result in SAGE incorrectly discarding alerts.

With all these in mind, SAGE cannot learn from all the episodes without drastically affecting the size and interpretability of attack graphs. However, it also cannot discard them without noticeably decreasing the completeness. One possible solution that can be explored in future works will be discussed in Section 6.

5.6 Interpretability Analysis

We define *interpretability* as the process of quickly and correctly reading attack graphs and the ability to draw the correct conclusions from them, given design decisions of SAGE and practices in the cyber security domain. Moreover, the person conducting this process was assumed to be a security analyst who knows how to read a SAGE-generated attack graph. Consequently, this sub-section is split into the readability protocol and the correct retrieval of attack strategies.

Readability Protocol

For a security analyst to draw the right conclusions from an attack graph, they must first be able to read it correctly. Only the results of 6 attack graphs were documented here to save space. However, the readability protocol was, in total, conducted for the 20 most interesting (i.e. with significant changes between baseline and PDFA model) attack graphs.

As seen from Table 7, the PDFA is overall more readable, with the difference being more drastic for the more complex AGs. This time was saved when the expert had to traverse a path from a start state to an end state. The reason behind this outcome is that, because the PDFA does not merge high-severity episodes as much as the baseline model, the paths are more spread out, and the out- and in-degree of these high-severity nodes are lower than their S-PDFA counterparts. Thus, it makes it easier for an expert to identify the edge leading to the next attack node.

Attack Graph	Baseline	PDFA
10.0.0.176 DATAMANIPULATIONhttp	66s	44s
10.0.0.100 DATADELIVERYwsman	186s	116s
10.0.1.46 DATAEXFILTRATIONuscli	147s	104s
10.0.0.100 NETWORKDOSntp	353s	254s
10.0.1.46 RESOURCEHIJACKINGhttp	73s	61s
10.0.0.20 DATAMANIPULATIONremotewarecl	27s	20s

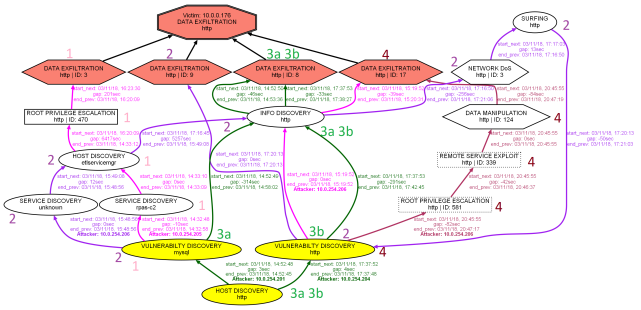
Table 7: Results of the readability protocol(i.e. completion time per model and attack graph). It shows that overall, the PDFA implementation is more readable than the baseline model, with the difference being more drastic with a more complex graph.

A clear example is Figure 7 where the *DATA EXFILTRATION http* node of the S-PDFA has an in-degree of 6, while the *DATA EXFILTRATION http* nodes of the PDFA have in-degrees of 1, 1, 2 and 2 respectively. This leads to the paths being simpler to traverse and the graph being easier to read.

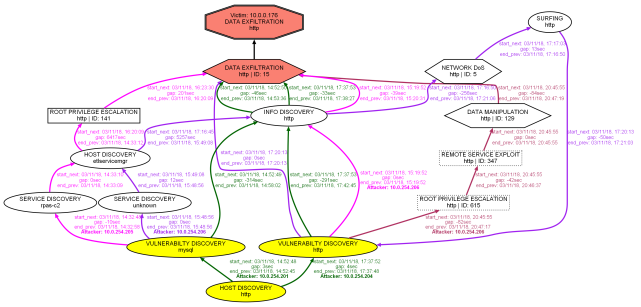
Correct Retrieval of attack strategies

Besides being readable and easily traversable, an interpretable graph should allow a security analyst to draw the right conclusions from the attack strategies executed on a specific service. SAGE was designed such that the number of sub-objectives in the attack graphs signifies different strategies for targetting the same service. Thus the AGs should correctly model such behaviour. With the PDFA implementation not merging sub-objectives, it is possible that the PDFA incorrectly splits paths that are not sufficiently distinct.

Considering this, Figures 3 and 7 clearly illustrate how optimising our modelling assumption further can make it more interpretable than the baseline. In the S-PDFA model of



(a) PDFA



(b) S-PDFA

Figure 7: Attack graph for *DATA EXFILTRATION http* for victim 10.0.0.176. It shows how the PDFA implementation can sometimes better capture the individual penetration strategies of attackers. In the figure, these strategies are encoded with numbers.

DATA EXFILTRATION http 10.0.0.176, there is only one sub-objective, from which we could conclude that the service was exploited similarly by all the attackers. However, analysing the paths more closely, we notice four distinct methods of breaching the victim. This is accurately represented in the PDFA attack graph, with these paths being encoded with numbers from 1 to 4 for ease of recognition but incorrectly depicted in the baseline.

On the other hand, Figure 3 proves the need for optimisation, as the PDFA model incorrectly models the two green paths as being distinct from the two rightmost pink paths, which is not the case, as both of these alternatives have common nodes such as *VULNERABILITY DISCOVERY http* and *INFO DISCOVERY http*.

6 Conclusions and Future Work

The goal of this paper was to compare the baseline model (i.e. S-PDFA) with the proposed modelling assumption (i.e. PDFA) in terms of **size**, **complexity**, **completeness** and **interpretability** and to discuss any other insights discovered during analysis. The key findings were:

- attack graphs generated by the PDFA implementation tend to present around 1.5 more high-severity nodes than the S-PDFA one. This can be attributed to the organization of traces used during the SAGE learning phase, which is structured in a low-to-high severity alert sequence, unlike the baseline model.
- attack graphs generated by the PDFA implementation contain 16.5% more nodes than the S-PDFA one. This can be attributed to the increase in high-severity nodes discussed in the previous point
- attack graphs generated by the PDFA implementation present a negligible difference in the number of simple and complex attack graphs compared to the S-PDFA one.
- attack graphs generated by the PDFA implementation present no difference in completeness compared to the S-PDFA implementation. This phenomenon is because both models possess an equal number of attack graphs and paths, making them equally complete.
- by discarding episodes containing less than three alerts, SAGE misses important information about attack graphs and paths regardless of the model. This is because an attack stage, from start to finish, is not stored in the same episode, and SAGE can incorrectly discard (part of) an attack stage that is essentially longer than three alerts.
- attack graphs generated by the PDFA implementation tend to be more readable than the S-PDFA one. This can be attributed to the increase in high-severity nodes, which makes the paths appear more spread out and easier to traverse.
- attack graphs generated by the PDFA implementation are more sensitive to differences in attack strategies than the S-PDFA one and will, overall, capture more true positive and false positive distinct paths.

Even though we have tried to reduce it as much as possible by using clearly defined protocols and defining explicit research objectives [15], due to the nature of the manual investigation, some analysis bias and human error might have been unknowingly introduced during the research. Moreover, in the absence of ground truth values to compare the AGs in terms of completeness, we assumed the filtered alerts to be these ground truth values. It works for a layered evaluation of the learning automaton. However, in the context of a holistic evaluation, this assumption does not hold. This is due to Security Information and Event Management Systems (SIEM) generating large amounts of false positives [1].

Additionally, we have found five bugs throughout the project that affected the results and required rerunning the experiments. These were fixed, however, due to time constraints, the analysis in this paper contains the fixes of only the first 3. Thankfully, the last two bugs did not drastically change SAGE's outcome, thus, the results should still be representable. With all these considered, reproducibility is still not affected, as long as the experiments are run on the versions that contain only the Pull Requests up to 9^{th} of June.

All these conclusions give insight into why our modelling assumption can yield better results than the baseline if optimized further. More precisely, exploring an approach for decreasing the number of false positive paths can generate attack graphs which are more interpretable, slightly bigger, and yet maintain an equivalent level of completeness.

Lastly, completeness could be improved by adding all the episodes of length smaller than 3 to episodes with similar timestamps. This process would be done per attacker per victim and would, technically, add previously discarded episodes to their full attack paths. However, this process is more complex, and it should be studied further in possible future works.

7 Responsible Research

SAGE and Flexfringe with both the baseline and the PDFA implementation are deterministic and would yield the same results as in this paper as long as they are run on the versions that contain only the Pull Requests up to 9^{th} of June. For easier reproducibility, the repository linked in the appendix contains the correct versions. Detailed instructions on how the experiments and the analysis were conducted are found in Section 4. Thus, this helps the reader reproduce any experiment they may desire.

The evaluation was conducted critically, and the reasoning behind any design choices was properly argued. Explanations of the results were given and backed up by evidence in the SAGE pipeline. The analysis was conducted such that bias in the results was as little as possible by defining explicit research objectives and using clearly defined protocols [15]. However, due to the nature of the manual analysis, some result bias may have been unknowingly added.

As for ethical concerns, five bugs were discovered and fixed in the SAGE code throughout the research project. However, due to time limitations and the nature of the manual analysis, the presented results contain only the first three fixes. Thankfully, the last two bugs did not drastically change the outcome of SAGE. Therefore the results should still be comparable to the potential results generated with all the fixes. With all these in mind, the code base could still possess undetected bugs, which, if discovered, should be reported to the maintainers.

8 Acknowledgements

Given the nature of the research questions and the desire of the teaching team to have similar metrics and comparable results, the research methodology was developed in collaboration with *Alexandru Dumitriu* and *Jegor Zelenjak*. Moreover, the bugs mentioned in Section 7 were discovered and fixed with the help of *Senne Van den Broeck*, *Vlad Constantinescu*, *Jegor Zelenjak* and *Alexandru Dumitriu*. Lastly, we want to acknowledge the responsible professor, *Sicco Verwer*, and supervisor, *Azqa Ndeem*, for the help and guidance throughout the research project.

References

[1] Sandeep Bhatt, Pratyusa K. Manadhata, and Loai Zomlot. The operational role of security information and event management systems. *IEEE Security Privacy*, 12(5):35–41, 2014.

[2] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. *Network and Distributed Systems Security Symposium*.

[3] Leevar Williams, Richard Lippmann, and Kyle Ingols. Garnet: A graphical attack graph and reachability network evaluation tool. In John R. Goodall, Gregory Conti, and Kwan-Liu Ma, editors, *Visualization for Computer Security*, pages 44–59, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[4] Steven Noel, Matthew Elder, Sushil Jajodia, Pramod Kalapa, Scott O'Hare, and Kenneth Prole. Advances in topological vulnerability analysis. In *2009 Cybersecurity Applications Technology Conference for Homeland Security*, pages 124–129, 2009.

[5] Azqa Nadeem, Sicco Verwer, Stephen Moskal, and Shanchieh Jay Yang. Alert-driven attack graph generation using s-pdfa. *IEEE Transactions on Dependable and Secure Computing*, 19(2):731–746, 2022.

[6] Sicco Verwer and Christian Hammerschmidt. Flexfringe: Modeling software behavior by learning probabilistic automata, 2022.

[7] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007.

[8] Sean Carlisto de Alvarenga, Sylvio Barbon, Rodrigo Sanches Miani, Michel Cukier, and Bruno Bogaz Zarpelão. Process mining and hierarchical clustering to help intrusion alert visualization. *Computers Security*, 73:474–491, 2018.

[9] Subhi Issa, Onaopepo Adekunle, Fayçal Hamdi, Samira Si-Said Cherfi, Michel Dumontier, and Amrapali Zaveri. Knowledge graph completeness: A systematic literature review. *IEEE Access*, 9:31322–31339, 2021.

[10] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *IEEE Symposium on Information Visualization*, pages 17–24, 2004.

[11] J Pelletier. Collegiate penetration testing competition, 2018.

[12] Nuthan Munaiah, Akond Rahman, Justin Pelletier, Laurie Williams, and Andrew Meneely. Characterizing attacker behavior in a cybersecurity penetration testing competition. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2019.

[13] Azqa Nadeem, Sicco Verwer, Stephen Moskal, and Shanchieh Jay Yang. Enabling visual analytics via alert-driven attack graphs. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2420–2422, New York, NY, USA, 2021. Association for Computing Machinery.

[14] Cheng-Yuan Ho, Yuan-Cheng Lai, I-Wei Chen, Fu-Yu Wang, and Wei-Hsuan Tai. Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems. *IEEE Communications Magazine*, 50(3):146–154, 2012.

[15] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164, April 2009.

A Implementation of experiments and SAGE

The code used for generating the data presented in this paper can be found at the following GitHub Repository: https://github.com/OpreaCristian2002/research-project.

The code for SAGE can be found at the following GitHub Repository: https://github.com/tudelft-cda-lab/SAGE