Alert-driven Attack Graph Generation using S-PDFA

Azqa Nadeem, Sicco Verwer, Stephen Moskal and Shanchieh Jay Yang

Abstract—Ideal cyber threat intelligence (CTI) includes insights into attacker strategies that are specific to a network under observation. Such CTI currently requires extensive expert input for obtaining, assessing, and correlating system vulnerabilities into a graphical representation, often referred to as an attack graph (AG). Instead of deriving AGs based on system vulnerabilities, this work advocates the direct use of intrusion alerts. We propose SAGE, an explainable sequence learning pipeline that automatically constructs AGs from intrusion alerts without a priori expert knowledge. SAGE exploits the temporal and probabilistic dependence between alerts in a suffix-based probabilistic deterministic finite automaton (S-PDFA) — a model that brings infrequent severe alerts into the spotlight and summarizes paths leading to them. Attack graphs are extracted from the model on a per-victim, per-objective basis. SAGE is thoroughly evaluated on three open-source intrusion alert datasets collected through security testing competitions in order to analyze distributed multi-stage attacks. SAGE compresses over 330k alerts into 93 AGs that show how specific attacks transpired. The AGs are succinct, interpretable, and provide directly relevant insights into strategic differences and fingerprintable paths. They even show that attackers tend to follow shorter paths after they have discovered a longer one in 84.5% of the cases.

Index Terms—Alert-driven attack graphs, Explainable machine learning, Suffix automaton model, Attacker strategy, Intrusion alerts.

1 INTRODUCTION

Alert investigation is one of the main responsibilities of security operations centers (SOC); and it is largely used for reactive defense capabilities. However, alert management can also be used to derive proactive cyber threat intelligence (CTI), e.g., by deducing attacker strategies specific to a network under observation. The biggest hurdle to this aim is the large volume of alerts that SOCs receive on a daily basis: alert fatigue is one of the most prevalent problems faced by analysts working in SOC environments [1]. A survey conducted during the RSA conference in 2018 revealed that security analysts receive more than a million alerts each day, many of which they cannot even address the same day [2]. Alert correlation reduces the volume of alerts by grouping alerts from the same attack stage [3], [4], [5]. However, it does not provide a bigger picture of the attack, and the subsequent analysis to obtain actionable insights into attacker strategies is still manual and labor-intensive.

Attacker strategies are often represented via attack graphs (AG), which are commonly used for visual analytics [6], [7], [8] and forensic analysis [9], [10]. Existing AG generation approaches fall under the Topological Vulnerability Analysis (TVA) [11] that requires extensive amount of expert knowledge and published vulnerability reports [12], [13]. As such, expert-driven AG generation is time-consuming; and it is ineffective to constantly rely on vulnerability scanning — the delayed nature of vulnerability reporting leaves blind-spots in an organization’s security [14]. Additionally, shared threat intelligence reports are often not directly relevant to one’s own network [15]. To the best of our knowledge, it is still an open problem to construct attack graphs that provide directly relevant intelligence regarding attacker strategies without expert input.

In this paper, we formally define our proposed system, SAGE (IntruSion alert-driven Attack Graph Extractor) [16]. SAGE generates AGs directly from intrusion alerts without a priori vulnerability and network topology information. It adopts an explainable sequence learning pipeline to exploit the temporal and probabilistic dependence present between intrusion alerts. SAGE can directly augment existing intrusion detection systems (IDS) for triaging large volumes of alerts to produce only a handful of AGs. These alert-driven AGs unlock a new means to derive intelligence regarding attacker strategies without having to investigate thousands of intrusion alerts. Fig. 1 shows the boxology diagram for SAGE, according to the modular design patterns by van Bekkum et al. [17].

A particular challenge for machine learning-enabled attacker strategy identification is the scarcity of severe alerts — the majority of alerts are associated to network scans, which are not interesting for an analyst due to their widespread use [18]. Therefore, frequency analysis methods like frequent pattern mining and longest common subsequence are inherently unsuitable, since they discard infrequent behavior. Instead, we learn an interpretable suffix-based probabilistic deterministic finite automaton (S-PDFA) using the FlexFringe automaton learning framework [19]. We tune the learning algorithm and transform the alert data such that the resulting model accentuates infrequent severe alerts, without discarding any low-severity alerts. The model summarizes attack paths leading to severe attack stages. It can...
distinguish between alerts with the same signature but different contexts, i.e., scanning at the start and scanning midway through an attack are treated differently, since the former indicates reconnaissance and the latter indicates attack progression. Targeted attack graphs are extracted from the S-P DFA on a per-victim, per-objective basis.

We demonstrate SAGE’s effectiveness on distributed multi-stage attack scenarios, i.e., where multiple attackers collaborate to compromise various targets progressing through numerous attack stages. Discovering attacker strategies in this setting is inherently difficult because host information cannot be used to aggregate alerts from different collaborating attacker(s). Security testing competitions provide an ideal setting to study such attacks in a controlled setting. To this end, we use three open-source datasets collected through penetration testing competitions [20] and blue team exercises [21] that have significantly different statistical properties and target infrastructures.

On one of the datasets, SAGE compresses over 330k alerts into 93 AGs in under a minute. Even with an imperfect IDS, the AGs capture the strategies used by the participating teams. They reveal that 84.5% of the time, attackers follow a shorter path to re-exploit an objective, after they have already discovered a longer path. Moreover, the AGs provide the visual means to compare attacker strategies. We show how to use this comparison to find fingerprintable paths and to rank various attackers based on the severity of their actions. Finally, SAGE is agnostic to the specific inner workings of an IDS, and can process any alert dataset as long as it contains IP addresses, port-numbers, and a description of the observed attack event. Our main contributions are:

1) We propose suffix-based probabilistic deterministic finite automaton (S-P DFA), an interpretable sequence model that focuses on infrequent severe alerts without discarding any low-severity alerts. The model summarizes attack paths in the dataset.

2) We provide formal definitions for SAGE's components, including a thorough explainability analysis of SAGE and the alert-driven AGs it generates.

3) We utilize three security testing competition datasets to extensively evaluate SAGE. We show it is generalizable and the AGs provide actionable intelligence regarding attacker strategies, strategic differences, and fingerprintable paths.

Section 2 describes two practical use-cases for SAGE. We provide a brief overview of the related works in Section 3. The architecture of SAGE, along with its explainability aspect is illustrated in Section 4. Sections 5 and 6 describe the experimental setup and a thorough analysis of alert-driven attack graphs. We discuss the limitations of SAGE in Section 7 and conclude in Section 8.

2 USE-CASES FOR SAGE

SAGE uses intrusion alerts to generate attack graphs (AG) that succinctly display all the paths that reach a given objective, making it an interpretable visual analytics tool. Below, we highlight use-cases for two types of users.

SOC analysts. The primary use-case explored in this paper is about enabling SOC analysts extract threat intelligence about attacker strategies from previously observed malicious activities. As such, SAGE augments existing SIEMs and IDSs by triaging the attack scenarios of interest, e.g., for specific assets in a network. The selected alert-driven AGs can be analyzed and compared to the AGs that were generated from the SAGE system. These differences can be used to identify new vulnerabilities or to refine existing security policies.

Red teams. As an adversarial use-case, SAGE can act as a monitoring intermediary during red team training. After a training session, the teams review alert-driven AGs for gaining intelligence, such as (i) identifying the shortest path to an objective that was discovered by a team member, or

Fig. 1: SAGE takes intrusion alerts as input and generates attack graphs. Intrusion alerts are transformed into episode sequences (Section 4.1). An interpretable S-P DFA model is learned from those sequences (Section 4.2). The sequences are replayed through the S-P DFA and transformed into targeted attack graphs (Section 4.3).
(ii) showing redundant paths, for instance, due by lack of communication between the team members. Enumerating all paths toward an objective can help the teams develop creative strategies (example in Section 6.3). Teams can use such feedback to further improve their performance.

3 Related Work

Cyber Threat Intelligence. CTI refers to evidence-based situational awareness, which typically involves insights into the tactics, techniques, and strategies employed by cyber adversaries [22]. Intrusion detection systems (IDS) generate thousands of alerts on a daily-basis. Alert triaging techniques have been proposed to model attack scenarios, such as alert correlation [3], [4], [5], [23], [24], [25], [26], [27] and alert prioritization [28], [29]. Alert correlation groups alerts from the same attack stage, while alert prioritization highlights and summarizes alerts for speeding up the response time. Although these methods drastically reduce alert volume, they do not provide a bigger picture of the specific strategies employed by the attackers.

Attack graph generation. SOC analysts rely on labor-intensive processes for obtaining intelligence regarding attacker strategies. Attack graphs (AG) provide a concise way of displaying these strategies [8], [14]. Specifically in the network security domain, Kaynar et al. [30] have proposed a taxonomy of the existing AG generation approaches. Many of them fall under the topological vulnerability analysis (TVA) [11], which relies heavily on a priori knowledge about the topology of, and vulnerabilities in a network, making them unsuitable for zero-day attacks. MultiVAL [12] and NetSPA [13] are popular tools in this category. Next to this, there are many approaches to improve pre-existing AGs, e.g., works focusing on AG completeness [31], [32], AG complexity reduction [33], [34], and what-if analyses [6], [7].

Learning from observables. Cyber data from prior security incidents can be utilized to gain insights into attacker behavior, e.g., using log data [35], [36], [37], sensor data [38], and network traffic [39]. Process mining and Markov models are particularly well-suited for sequential learning problems. Process mining (PM) has been used to provide a visual summary of the intrusion alert datasets [40], [41]. While great for modeling concurrent events, PM models are dense and cannot be used to model context: they use alert signatures as identifiers, which makes it impossible to distinguish between alerts with different contexts but identical signatures. Markov models, however, have no such limitation. Moskal et al. [42] use suffix-based Markov chains to represent attacker strategies as sequences of hyper-alerts. They measure attack sequence similarity using Jensen-Shannon divergence. In this paper, we propose SAGE, which is a purely alert-driven approach for generating attack graphs. We borrow initial ideas from Moskal et al. [42]. We leverage the temporal and probabilistic dependence between alerts to generate targeted attack graphs without a priori expert knowledge. The probabilistic deterministic finite automaton (S-PDFA) that we use has more expressive power than Markov chains, while being easier to interpret.

Explainability. SAGE provides an explainable and automated alternative to the manual process of finding attacker strategies. It is important to note that while explainability is widely considered for classification decisions, SAGE is not a classifier, and the explainability lies in the attack graphs instead. Because the explainability aspect of SAGE is an important design consideration, we do not consider inherently black-box models, such as neural networks [43]. While attention mechanisms [44] and linear proxy models [45] help explain the decisions of such black-box models, they offer post-hoc interpretability on a per-input basis. Instead, SAGE relies on the interpretable nature of its entire pipeline. As opposed to black-box models that often make use of randomization and soft decision boundaries to avoid local minima and over-fitting, SAGE relies on statistical tests, making every step in its pipeline discrete and deterministic. In addition to model interpretability, this provides design- and algorithmic transparency. We make conscious design decisions to enhance the interpretability of the S-PDFA, and the way the attack graphs are constructed makes them explainable. These notions are described by Roscher et al. [46], who list the three components of explainable machine learning as: transparency, interpretability, and explainability. In short, interpretability is about the model, while explainability is about the output of a learning pipeline. Model interpretability allows a user to: 1) examine (visualize) a learned model, 2) reason about the discovered patterns, 3) draw inferences, and 4) combine it with subsequent analysis methods. A model is design transparent if design decisions can be motivated from the application domain, and it is algorithmically transparent if it allows a user to reverse the learning pipeline to obtain the input data that led to modeling decisions. We show examples of all of these in Sections 4.4 and 6.1.

4 SAGE: Intrusion Alert-Driven Attack Graph Extractor

SAGE (Intrusion alert-driven Attack Graph Extractor) is a purely alert-driven approach for attack graph generation. SAGE has 3 core components, as shown in Fig. 1. It takes raw intrusion alerts as input, aggregates them into sequences of attacker actions. An automaton model is learned using these sequences, summarizing attacker strategies. Finally, attack graphs are extracted from the model on a per-victim, per-objective basis. SAGE is released as open-source1. It is implemented in Python and released in a docker container for cross-platform support.

In this section, we use the Collegiate Penetration Testing Competition dataset from 2018 [47], i.e., CPTC-2018, as a running example. CPTC-2018 contains intrusion alerts generated by six teams (T1, T2, T5, T7, T8, T9) attempting to compromise the infrastructure of a fictitious automotive company (See Section 5 for details). Table 1 shows how the volume of alerts is reduced by each component of SAGE.

4.1 From intrusion alerts to episode sequences

As a first step, we arrange intrusion alerts in sequences that characterize an attacker strategy. Raw intrusion alerts are noisy and often multiple alerts are triggered by a single attacker action. Thus, the main goal of this step is to clean and aggregate alerts into sequences of attacker actions.

1. https://github.com/tudelft-cda-lab/SAGE
TABLE 1: For each CPTC-2018 team, the number of raw alerts and how they are compressed in each phase of SAGE.

<table>
<thead>
<tr>
<th>Alerts (raw)</th>
<th>Alerts (filtered)</th>
<th>Episodes</th>
<th>ES/ESQ</th>
<th>ESS/Traces</th>
<th>AGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>81,373</td>
<td>26,651</td>
<td>655</td>
<td>103</td>
<td>108</td>
</tr>
<tr>
<td>T2</td>
<td>42,474</td>
<td>4,922</td>
<td>69</td>
<td>86</td>
<td>92</td>
</tr>
<tr>
<td>T5</td>
<td>52,550</td>
<td>11,918</td>
<td>622</td>
<td>69</td>
<td>74</td>
</tr>
<tr>
<td>T7</td>
<td>47,101</td>
<td>8,517</td>
<td>5/6</td>
<td>73</td>
<td>23</td>
</tr>
<tr>
<td>T8</td>
<td>55,170</td>
<td>9,037</td>
<td>4,39</td>
<td>79</td>
<td>51</td>
</tr>
<tr>
<td>T9</td>
<td>51,602</td>
<td>10,081</td>
<td>1,042</td>
<td>69</td>
<td>110</td>
</tr>
</tbody>
</table>

4.1.1 Alert pre-processing

An intrusion alarm is composed of attributes such as, source and destination IP addresses, a timestamp, a descriptive signature, and some protocol specific fields. SAGE utilizes fields that are available for all alerts, regardless of the attack vector. The input to SAGE is a set of observable intrusion alerts \( O \). Let \( o \in O \) be an intrusion alert, with attributes \( o = \langle sIP, dIP, sPort, dPort, ts, sign \rangle \). Here \( sIP, dIP, sPort \) are the attacker’s IP and port number and \( dIP, dPort \) are the victim’s IP and port number. \( ts \) is the time elapsed since the first alert in seconds. \( sign \) is the alert signature attribute.

Features are extracted as follows: (i) The destination port number is used to identify the likely targeted service \( tServ = Serv(dPort) \) from open source IANA mapping [48]. (ii) Intrusion alerts typically contain many repeated alerts occurring within a short time interval. Such high-frequency noise creates undesired artifacts in model learning. We filter all alerts with identical attributes that occur within a \( t \)-second interval, keeping only the first occurrence, i.e., we create a set \( O_F \subseteq O \) such that for each observation \( \langle sIP, dIP, sPort, dPort, ts, sign \rangle \in O_F \), there exists no \( \langle sIP, dIP, sPort, dPort, ts', sign \rangle \in O \) with \( ts \neq ts' \), and \( ts - t \leq ts' < ts \). In this paper, we use \( t = 1.0 \text{ sec} \) following [5], [42]. (iii) Instead of using the default alert signature attribute, we augment alerts with attack stages proposed by the Action-Intent Framework (AIF) of Moskal et al. [49] for categorizing them into their respective attack phases. The AIF provides a better representation of the attack stages. Based on the MITRE ATT&CK framework [50], it was proposed specifically to map action-types to dynamic observables, such as intrusion alerts. The AIF provides a mapping \( \text{map} = \text{Map}(\text{sign}) \) from alert signatures to attack stages (see appendix). (iv) Finally, the filtered set \( \hat{O} \) of intrusion alerts \( \hat{o} \) is a 5-tuple \( \hat{o} = \langle sIP, dIP, tServ, ts, map \rangle \) for each \( o \in O_F \). Fig. 2 shows the distribution of the attack stages across all six teams in the filtered CPTC-2018 dataset.

4.1.2 Gathering alerts into Alert Sequences (AS)

There are three main methods for converting discrete observables into sequences: aggregation based on (i) source IP: showing the attacker’s perspective, (ii) destination IP: showing the victim’s perspective, and (iii) source IP, destination IP) pair: showing individual interactions between unique attackers and victims. We select (iii) because the sequences clearly show the interaction an attacker has with a victim, without other attackers polluting the sequence, which helps to preserve the temporal dependence between alerts. Thus an alert sequence is a windowed list of alerts between a unique (attacker, victim) pair.

Definition 1. An Alert Sequence (AS) is a windowed list of alerts occurring within a time window \( w \). Let \( A \) be the set of unique attacker hosts, \( V \) be the set of unique victim hosts, and \( C \) be the set of unique attack stages (\( mc\text{at} \)), then \( AS_{a,v,c} = a_{a,v,c} \ldots a_{a,v,c} \), where \( (a, v) \in A \times V \), \( c \in C \). Here, \( a_{a,v,c} = \{ a_{a,v,c} \ldots a_{a,v,c} \} \) is a multi-set of alerts for \( 1 \leq i \leq n \). For a window \( w \) and given \( \tilde{a}_j = \langle a, v, tserv, ts, c \rangle \in \tilde{O} \), we define the projection of the X attribute of \( \tilde{a}_{a,v,c} \) is \( \Pi_x(a_{a,v,c}) = \langle i \cdot w, \Pi_x(a_{a,v,c}) - \Pi_x(a_{a,v,c}) \rangle \leq w, \) and \( \Pi_x(a_{a,v,c}) \leq \Pi_x(a_{a,v,c}) \) for \( 1 \leq j \leq \omega \).

Here, \( \Pi_x(a_{a,v,c}) \) is the projection of the X attribute of \( a_{a,v,c} \). Furthermore, we use \( f(i) \) to denote the first derivative of the number of alerts per-window over time, i.e., \( f(i) = \frac{d(a_{a,v,c})}{dt} \) (will be used to define slope in Algorithm 1). In contrast to other works that use \( sIP \) and \( dIP \) as explicit features [24], [25], [26], we only use them to construct sequences. This allows identification of related alerts originating from different sources.

4.1.3 Aggregating AS into Episode Sequences (ES)

Intrusion alerts are aggregated into a group, such that they likely belong to the same attacker action. In the literature, such an aggregation is called an attack episode [42]. We assume that these episodes closely characterize attacker actions. Generally, low-severity alerts are so frequent that they subsume high-severity alerts. To overcome this, we treat each attack stage separately. Intuitively, we test the frequency of all alerts in a windowed sequence: when the frequency starts to increase (an \( \text{up} \)), we consider it the start of an episode; when the frequency is continuously decreasing and reaches a global minimum (a \( \text{down} \)), we consider it the end of that episode (see example in Fig. 3). Episodes are the building block of SAGE. All extracted episodes...
**Algorithm 1** Convert alert sequence into episode sequence

**Input:** Alert sequence: `as`

**Output:** Episode sequence: `es`

1. `def CONVERT_TO_ES(as):
2.     `es = []`
3.     `for (mcat_x, as_x) in SPLIT_ON_MCAT(as) do`
4.         `temped_as = LEN(sub) for all sub in as_x`
5.         `slope = f'(x) for all x in temped_as`
6.         `ups = GET_POSITIVE_SLOPES(slope)`
7.         `downs = GET_NEGATIVE_SLOPES(slope)`
8.         `episodes = GET_EPISODES(ups, downs)`
9.         `es.append((mcat_x, ep)) for all ep in episodes`
10. `end for`
11. `es = SORT_BY_EPISODE_START(es)`
12. `return es`
13. `def GET_EPISODES(ups, downs):
14.     `episodes = []`
15.     `for i in [0,...,LEN(ups)–1] do`
16.         `if IS_DOWN_BETWEEN_UPS(i, i + 1, downs) then`
17.             `down = GET_LAST_DOWN(i, i + 1, downs)`
18.             `episodes.append((ups[i], down))`
19.         `end if`
20. `end for`
21. `return episodes`

**Fig. 3:** Bursts of alerts from the same attack stage are aggregated into episodes. Here, an attack sequence related to vulnerability scanning is aggregated into two episodes.

are collected and time-sorted in an episode sequence (see Algorithm 1).

**Definition 2.** An Episode Sequence (ES) for an attacker `a` and victim `v` is a list of episodes, `E_{av} = e_1^{a,v} \ldots e_m^{a,v}`. An episode is a 4-tuple `e_i^{a,v} = (st^{a,v}, et^{a,v}, mcat^{a,v}, mserv^{a,v})` for `1 \leq j \leq m`, where `st^{a,v}, et^{a,v} \in \mathbb{R}` denote the start and end time of an episode, `mcat^{a,v}` is the attack stage of an episode, and `mserv^{a,v}` is the most frequently targeted service in an episode.

In essence, ES's are aggregates of sequences of alerts (see Fig. 4). We construct `E_{av}` from a windowed alert sequence of attack stage `c`, i.e., `A_{av,c} = \langle a_1^{c,v}, \ldots a_n^{c,v} \rangle`. For each `1 \leq s \leq e \leq n`, the start time is `st^{a,v} = \min(\Pi_{t_{\text{a,c}}})` if `f'(s) = 0` and `f'(s + 1) > 0`; the end time is `et^{a,v} = \max(\Pi_{t_{\text{a,c}}})` if `f'(e) = 0` and `f'(e - 1) < 0`; the attack stage is `mcat^{a,v} = c`, and the most frequently targeted service is `mserv^{a,v} = \arg\max_{mserv} |\Pi_{t_{\text{a,c}}}| = mserv : s \leq i \leq e |`

**Fig. 4:** Episode sequences from CPTC-2018: Each sequence is a list of tuples `(st, et, mcat, mserv)`, ordered in time.

### 4.2 Suffix-based Probabilistic Deterministic Finite Automaton (S-PDFA)

The insight provided by episode sequences is limited because they fail to capture the temporal dependence between episodes. We use a suffix-based probabilistic deterministic finite automaton (S-PDFA) with Markovian properties to summarize attacker strategies. It clusters similar attack paths based on temporal and behavioral similarity. It also brings infrequent severe episodes into the spotlight. This last requirement is problematic because most clustering approaches ignore infrequent patterns.

In contrast to regular Markov chains, an automaton model is able to distinguish between episodes of the same `mcat` with different contexts, e.g., a scanning event happening at the start, and that happening mid-way through an attack, when attackers have already gained some knowledge, are treated differently. This makes them popular for learning the behavior of software systems, such as communication protocols and even malware, see e.g., [51], [52], [53], [54].

**Definition 3.** A Suffix-based Probabilistic Deterministic Finite Automaton (S-PDFA) is a 5-tuple `A = (Q, \Sigma, \Delta, P, q_0)` defining the machine structure: `Q` is a finite set of states; `\Sigma` is a finite alphabet of symbols; `\Delta` is a finite set of transitions; `P : \Delta \rightarrow [0, 1]` is the transition probability function, and `q_0 \in Q` is the final state (due to suffix model). A transition `\delta \in \Delta` in an S-PDFA is a tuple `(q, q', a)`, where `q, q' \in Q` are the target and source states, and `a \in \Sigma` is a symbol. `P` is a function such that `\Sigma_{q_0} P((q, q', a)) = 1`. Additionally, `\Delta` is such that for every `q \in Q` and `a \in \Sigma`, there exists at most one `(q, q', a) \in \Delta`, making the model (suffix) deterministic.

A suffix automaton contains a single final state and does not model starting states. Instead of generating a sequence from the start, it generates sequences from the end. It still represents a probability distribution over `\Sigma^*` for all `1 \leq n`. The probability of a sequence `s = a_1 \ldots a_n` is computed along the reverse path `\pi_{1 \leq i \leq n} P((q_i, q_{i+1}, a_i))`, where `\pi` denotes a product. For any trace, there exists a unique run due to suffix determinism. The Flexfringe automaton learning framework [19] can be used to learn suffix models. Flexfringe implements several automaton learning heuristics within the well-known state merging algorithms, such as state merging [55] and DFASAT [56] (see [57] for details).

**4.2.1 Input trace construction**

Whereas an episode sequence may contain multiple attempts to exploit a victim host, an S-PDFA models each
attempt separately to find partial overlap in attacker strategies. To this end, an ES is partitioned into episode subsequences (ESS) when a low-severity episode follows a high-severity one. Severity(\(\text{epi}\)) is a user-defined function, determined by the acceptable risk of a SOC. By default, scanning has low severity, exploitation has high severity and the rest of the enabler-actions have medium severity (see appendix).

Definition 4. Given an \(\text{ESS}_{av} = \text{epi}_{1}^{av} \ldots \text{epi}_{m}^{av}\), define a break-point as an index \(i\) such that \(\text{Severity}(\text{epi}_{i}^{av}) < \text{Severity}(\text{epi}_{i+1}^{av})\). An Episode Subsequence \(\text{ESS}_{av} = \text{epi}_{1}^{av} \ldots \text{epi}_{m}^{av}\) is a contiguous subsequence of \(\text{ESS}_{av}\) without break-points, i.e., \(\text{ESS}_{av} = \text{epi}_{1}^{av} \ldots \text{epi}_{i-1}^{av} \ldots \text{epi}_{i}^{av} \ldots \text{epi}_{m}^{av}\). Every \(\text{ESS}_{av}\) is broken into its break-point-free subsequences \(\text{ESS}_{av} = \text{ESS}_{av,1} \ldots \text{ESS}_{av,k}\).

The S-PDFA learns on sequences of univariate symbols, called traces. One trace is constructed per ESS. The symbols signify the most apparent intent of episodes, defined by \(\langle \text{mcat}, \text{Theme(mServ)} \rangle\). \text{Theme()} groups services based on their functionality (see appendix). This gives 536 traces, which is small but sufficient to learn insightful S-PDFAs.

4.2.2 S-PDFA for SAGE

We opt for a suffix model because we are interested in predicting which episodes eventually lead to high-severity attack stages. These attack stages are infrequent, and always lie at the end of our input traces. Therefore, a suffix-automaton model is used to predict the past, instead of predicting the future. Each state in an S-PDFA model can be thought of as a milestone achieved by an attacker.

Although Flexfringe uses prefix-based models, we obtain a suffix-based one by simply reversing the input traces. We choose the Flexfringe implementation of the Alergia algorithm [58] because of limited data. For reversed traces, the algorithm constructs a suffix tree (see Fig. 5 for an example). The algorithm starts at the root of the suffix tree and iteratively tries to merge states based on the chosen merge criteria. The parameter selection for model learning is guided by the properties of input traces and some trial-and-error of visualizing the model until satisfied. Fortunately, the algorithm learns these models in less than 0.5 seconds. Fig. 6 shows the S-PDFA for CPTC-2018, learned from all 536 traces to enable behavior comparison.

![Fig. 5: A suffix tree for three traces. For any vertex, the previous vertex happens chronologically in the future.](image)

![Fig. 6: The S-PDFA model for CPTC-2018. The states are colored according to the severity of the incoming symbol’s attack stage: red is high, blue is medium, white is low.](image)
The chosen state merging algorithm ensures that only the states with similar pasts are merged. The Markovian property, in addition, forces that the immediate-future is identical. Thus, the occurrence of identical episodes leading to different states highlights semantic differences, e.g., data exfiltration http may either be reached by service discovery → code execution, or by vulnerability discovery → privilege escalation. Separate states will be learned for these two types of data exfiltration, capturing their context.

### 4.2.3 S-PDFA model quality evaluation

Evaluating model quality is a hard problem in grammatical inference [57], [59]. Typically, it is measured using a trade-off between model size and fit. We are mainly interested in the insight provided by the S-PDFA. The initial suffix tree shows the data as is, which provides insight but does not show similarities between the different traces. The S-PDFA shows such similarities by performing merges. Every such merge generalizes from the training data, and assigns probability mass to unseen test data. We use Perplexity to quantify model quality. It measures the prediction power of a model, and has been used in grammatical inference competitions [60], [61]. It is defined as $2^{-\sum_{i} \log_{2} P(x_{i})}$, where $N$ is the number of traces, and $P(x_{i})$ returns the probability of the $x_{i}$ trace. The lower the value, the better the model fits with the data. We compute perplexity for both, training and test data, using an 80-20 split, where the former shows how well the model fits the training data, and the latter shows how well it captures patterns in the overall data.

Perplexity is computed for four suffix model-variants: (i) suffix tree: plain representation of traces in a tree format, (ii) Markov chain: standard statistical model, (iii) default S-PDFA: an S-PDFA with default settings, (iv) SAGE S-PDFA: an S-PDFA learned using the settings in this paper. Table 2 shows the perplexity for each variant. It shows that a suffix tree contains the data as is, which provides insight but does not show similarities between episodes. The S-PDFA shows such similarities by performing merges. Separate states will be learned for these two types of data exfiltration, capturing their context.

### 4.3 Alert-driven Attack Graphs

The S-PDFA assigns the same context to episodes that are temporally and probabilistically similar, where context is denoted by state identifiers. We first augment episode sequences with their context, and then transform them into attack graphs (AG) on a per-victim, per-objective basis.

#### 4.3.1 Adding context to Episode Sequences

The states of an S-PDFA provide contextual meaning to the episodes’ attack stages. Existing work by Lin et al. [62] have utilized this context to encode traces into state sequences for clustering similar car-following behaviors. We follow the same principle, and convert the episode sequences (ESQ) into state sequences (SSEQ). We run each episode subsequence $a_{1} \ldots a_{n}$ through the model, which produces $q_{a_{1}} \ldots q_{a_{n}}$. A state subsequence is an episode subsequence augmented with state identifiers, i.e., $q_{a_{1}}q_{a_{2}}\ldots q_{a_{n}}$.

**Definition 5.** A State Sequence (ESQ) for an episode subsequence $ES_{av} = ESS_{av,1} \ldots ESS_{av,k}$ is the concatenated sequence $ESQ_{av} = sq_{1}sq_{2}\ldots sq_{k}$, where $sq_{i}$ is the state subsequence for $ESS_{av,i}$, for all $1 \leq i \leq k$.

#### 4.3.2 Attack graph construction

The state sequences are transformed into alert-driven attack graphs based on the specified objective and the victim host. An objective $obj \in Obj$ is a 3 tuple $(mcat, mServ, q)$ associated to a high-severity attack stage, represented by the last six categories of the Action-Intent mapping (see appendix). They are considered as end-goals since (a) they are typically the last actions to appear in ESS, and (b) it is unlikely that medium-severity actions, e.g., privilege escalation, are done to no end. To support episode prioritization, an analyst can choose the granularity of objectives, i.e., only attack stage $(mcat)$, attack stage and targeted service $(mcat, mServ)$ or the full tuple $(mcat, mServ, q)$. By default, SAGE generates AGs on a per-victim, per-objective basis, i.e., for an objective $obj \in Obj$ and a victim $v \in V$, only the state sequences that contain $obj$ are considered, i.e., $(path \in ESQ_{av} | obj \in path)$. In theory, this produces $|V| \cdot |Obj|$ attack graphs, many of which contain shared paths. We aggregate AGs of a victim $v$ and objectives $obj \in Obj$ attack graphs, many of which contain shared paths. We aggregate AGs of a victim $v$ and objectives $obj \in Obj$. This is because paths leading to $obj$ and $obj'$ tend to have shared vertices. On the CPTC-2018 dataset, for 19 victims and 70 objectives, this step results in 93 AGs instead of 1,330 (a reduction of 93%). Each AG compresses over 500 alerts in less than 25 vertices, on average.

In summary, the root of an attack graph is $(mcat, mServ)$. Other vertices are the unique items in path. Edges are obtained by running a sliding window of length 2 over path. The edge label shows the start-time attribute of each episode, showing attack progression. In a state sequence, if an objective is achieved multiple times, each attempt is shown as an individual path in the graph. Also, to make

<table>
<thead>
<tr>
<th>Training set</th>
<th>Suffix tree</th>
<th>Markov chain</th>
<th>Default S-PDFA</th>
<th>SAGE S-PDFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holdout test set</td>
<td>13659.6</td>
<td>13659.6</td>
<td>15136.5</td>
<td>2397.8</td>
</tr>
</tbody>
</table>

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Fig. 7: An alert-driven attack graph: Vertices: Labels show (attack stage, targeted service, state identifier). Low-severity episodes are oval, medium-severity are boxes, high-severity are hexagons. The first episode in a path is yellow, the objective is black. Sinks are dotted. Edges: Labels show seconds since the first alert. Colors show team affiliation: T1 (Maroon), T2 (Orange), T5 (Green), T7 (black), T8 (Magenta), T9 (Purple).

4.3.3 Attack graph complexity analysis

We evaluate the complexity of the AGs using the model simplicity metric proposed by De Alvarenga et al. [40] for process mining, i.e., Simplicity(AG) = |V|/|E|, where |V| and |E| are the number of vertices and edges, respectively. The average simplicity of the CPTC-2018 AGs is 0.81, with 21.7 vertices on average (min: 3, max: 49) and 38.7 edges on average (min: 2, max: 174). Although the average number of vertices is higher than [40], these AGs show the paths for all teams, making strategy comparison much easier. Moreover, Nadeem et al. [63] show that the S-PDFA AGs are more succinct than suffix tree- and Markov chain-based approaches.

4.4 Explainability analysis of SAGE

We make conscious design decisions to make the entire SAGE pipeline explainable. This is so that security analysts can review the attack graphs (AG), reason about attacker strategies, and discover new knowledge [64].

Fig. 7 shows the composition of an alert-driven AG. An AG for a given (objective, victim) is a compressed representation of its relevant intrusion alerts. A vertex represents an aggregation of alerts, i.e., an episode (defined by the severity of its attack stage, its context as determined by the S-PDFA, and the most frequently-targeted service within the alerts). Some episodes may have the same shape, attack stage, and targeted service, but different contexts, i.e., state identifiers. This happens when these episodes are observed in sequences with different futures and pasts. An AG may also have multiple red vertices if the S-PDFA identifies different ways of obtaining the same objective, which happens when the paths leading up to it are significantly different. A path in an AG represents a sequence of episodes that leads to an objective. Two paths overlap iff the S-PDFA has sufficient evidence that they are similar, i.e., the episodes have identical futures or similar pasts. In addition, we remove the influence of (a) other actions in a path by constructing a sequence with only the alerts between a specific (attacker, victim), and (b) other attack attempts by modeling each one as a separate path. A path can be traced starting from a yellow vertex, and following the time progression of the edge labels, ending in one of the red vertices. This makes each AG design- and algorithmically transparent, interpretable, and scientifically explainable.

The S-PDFA is an intermediate step responsible for modeling context. We specifically learn a suffix model to highlight the infrequent severe episodes. The Markovian property, together with sinks, makes the model components interpretable. The deterministic nature of the model makes it algorithmically transparent. The parameter settings are guided by the input data, making the model design transparent.

5 DATASET AND EXPERIMENTAL SETUP

Dataset. Security testing competitions provide an ideal setting for distributed multi-stage attacks in a controlled environment. In this paper, we use three open-source intrusion alert datasets: two datasets from the Collegiate Penetration Testing Competition (CPTC) [65] for showing SAGE’s efficacy, and one dataset from the Collegiate Cyber Defense Competition (CCDC) [66] for showing SAGE’s generalizability. A summary of the datasets is given in Table 3.

The alert datasets are generated by different student teams who are tasked to compromise a common fictitious network. The CPTC-2017 dataset contains alerts by nine teams (T2 to T10) targeting an electronic election infrastructure, while the CPTC-2018 dataset contains alerts by six teams (T1, T2, T5, T7, T8, T9) targeting an automotive company. Naturally, some vulnerabilities are unique to the network, while the others are typical of any misconfigured web server. Each team has access to fixed-IP machines that they can use, either in collaboration, or in isolation to achieve their objectives. The infrastructure is monitored by a Suricata IDS [67], which records alerts on a per-team basis. Beyond the attackers’ IP information, no other ground truth is available regarding the attack progression and attacker strategies. This imitates the real-world scenario where SOC

<table>
<thead>
<tr>
<th>Dataset/Properties</th>
<th>CPTC-2018</th>
<th>CPTC-2017</th>
<th>CCDC-2018</th>
</tr>
</thead>
<tbody>
<tr>
<td># alerts</td>
<td>350,220</td>
<td>43,611</td>
<td>1,052,281</td>
</tr>
<tr>
<td># teams</td>
<td>6</td>
<td>9</td>
<td>Unknown</td>
</tr>
<tr>
<td># IPs</td>
<td>42</td>
<td>494</td>
<td>2138</td>
</tr>
<tr>
<td># services</td>
<td>160</td>
<td>168</td>
<td>2050</td>
</tr>
<tr>
<td>Duration (hrs)</td>
<td>9</td>
<td>11</td>
<td>25</td>
</tr>
<tr>
<td>Attacker hosts known?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Victim hosts known?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dataset type</td>
<td>Penetration testing</td>
<td>Penetration testing</td>
<td>Blue teaming</td>
</tr>
</tbody>
</table>

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author’s version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2021.3117348
analysts i) determine how an attack happened, and ii) compare attacker strategies for fingerprintable behaviors.

**Experiments.** We perform three sets of experiments:

1) **Strategy explanation.** We analyze attack graphs generated from one infrastructure, i.e., CPTC-2018, and demonstrate SAGE’s explainability aspect.

2) **Model comparison.** We perform a comparison between the CPTC-2017 and CPTC-2018 S-PDFA models to highlight infrastructure-related differences captured by the learning algorithm.

3) **Replication case study.** We analyze attack graphs generated from the CCDC-2018 dataset — it contains alerts from a blue team exercise, where the organizers serve as the red team. Other than a network topology diagram (which seems like a web shop), no other ground truth is available.

**Parameters.** In this paper, we set $t = 1.0$ seconds to filter repeated alerts [5], [42]. For window length $w$, we experiment with $w = \{60, 150, 300, 600\}$ seconds, and choose $w = 150$ as a reasonable value. Smaller window sizes produce longer alert sequences, which may cut the same behavior across multiple episodes. As such, $w$ should be tuned according to the trade-off between analysis resolution and the number of alerts available per sequence. For model learning, $state\_count$, $symbol\_count$, and $sink\_count$ are set to 5. All experiments are run in a Jupyter notebook executed on Intel Xeon W-2123 quad-core processor and 32 GB RAM.

## 6 Results and Discussion

Alert-driven attack graphs (AG) are aggregated representations of intrusion alerts, reflecting the actual pathways taken by the attacker teams. The AGs are succinct, interpretable, and generalizable.

### 6.1 Explaining attacker strategies in CPTC-2018

In this experiment, we analyze the AGs generated from CPTC-2018. The S-PDFA finds a total of 70 contextual objectives that are achieved by targeting 19 victim hosts. 330,270 alerts are represented by 93 AGs, where each AG shows how the attack actually transpired. The end-to-end execution time is 1.65 minutes, where 50% of this time is spent loading the intrusion alerts. Below, we demonstrate how SAGE enables visual analytics for attack path interpretation, and highlights strategic differences for intelligence collection.

#### 6.1.1 Comparing individual attack paths

1) Alert-driven attack graphs provide insights into the paths explored by attackers. Fig. 8 shows the strategies of three teams (the absence of other teams indicates that they were unable to achieve this objective). This graph compresses 300 alerts into 25 vertices, enabling a SOC analyst to follow the attack progression.

2) Fig. 8 shows that T1, T5, and T8 exfiltrate data from 10.0.0.20 using a remote access service. The teams self-reported that they had found a chatting application on this host that contained credentials, which they exfiltrate using a combination of privilege escalation and arbitrary code execution. The AG concretely shows how this was done.

T5 finds two distinct paths to complete this objective: first at around the 1.4-hour mark of the competition, and then later at around the 4.5-hour mark. T1 also finds two paths, but significantly later in the competition. The S-PDFA identifies three distinct exploitation states because of significant differences in the paths that reach these states. Clearly, the states $(data\_exfiltration, remoteware-cl, 17)$ and $(..., 116)$ are reached later in the competition with fewer steps, implicitly capturing attackers’ increasing experience.

3) Interestingly, an AG of data manipulation (Fig. 9) results in a partial sub-graph of the AG from Fig. 8, due to overlap in paths that attain both objectives. It shows three variants of data manipulation, of which two are also present in the exfiltration graph, i.e., $(data\_manipulation, remoteware-cl, 95)$ and $(..., 288)$. T5 finds one additional path to reach $(..., 18)$ right after it has reached $(data\_exfiltration, remoteware-cl, 17)$ from the previous AG. These type of insights provide actionable intelligence to disrupt the cyber kill-chain [68].

---

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
6.1.2 Explaining strategic differences across AGs

(1) In addition to comparing attack paths, SOC analysts can also compare entire AGs for a broader view of the network, e.g., the AGs of victims 10.0.1.40 and 10.0.1.41 for data exfiltration over http are identical, both in terms of the teams that exploit it and the timestamps of their actions (see Fig. 10). According to the network topology, these two hosts handle authentication in the production network. The identical AGs indicate that both, T5 and T8, conduct a scripted attack on these hosts.

(2) Fig. 11 shows T5, T7, and T8 conducting resource hijacking over two hosts (10.0.1.40, 10.0.1.41) using http, resulting in highly similar AGs. T5 has an identical strategy for both hosts. T7 does scans before manipulating accounts and conducting a network DoS over 10.0.1.41, while later they only perform a scan and a network DoS over 10.0.1.40. Similarly, T8 does a privilege escalation and code execution after network DoS over 10.0.1.41, while they later only do a network DoS over 10.0.1.40 to achieve their objective. These differences show that attackers tend to follow shorter paths after having successfully exploited a longer path. Out of all the attack paths discovered in CPTC-2018, 84.5% subsequent paths are shorter than an earlier attempt, for a given objective.

6.1.3 Discovering fingerprintable paths

After analyzing the AGs, we observe that different teams often reach different objectives, and when they do reach the same objective, their paths are very different. Moreover, when a team reaches an objective multiple times, their paths are highly similar. Thus, the uniqueness of the paths can be used by SOC analysts as fingerprints to single-out attacker teams. A fingerprint is a uniquely identifiable sequence of episodes, i.e., path, that leads to a certain objective. It is entirely possible that other paths (or sub-paths) leading to common objectives are also unique, but we take a conservative approach and say that an objective is fingerprintable if only a single team reaches it. Also, an objective can have more than one fingerprint if a team finds multiple unique ways to reach it. Table 4 shows the number of unique paths each team discovers during CPTC-2018. 17 objectives are fingerprintable, with a total of 29 unique fingerprints. We found 9 fingerprints for two objectives reached by T1; 10 fingerprints for four objectives reached by T5; 7 fingerprints for five objectives reached by T7; and 3 fingerprints for three objectives reached by T9. We found no dedicated fingerprintable objectives for T2 and T8. Also, since a fingerprint is a sequence of episodes, longer fingerprints provide more evidence for identifying an attacker. The fingerprints we discover are composed of 15.8 episodes, on average, which provides solid evidence to uniquely identify a team.

6.1.4 Ranking attacker performance

Each vertex in an alert-driven AG signifies a new milestone or objective achieved by an attacker. We argue that the fraction of unique milestones discovered by an attacker provides a metric for their performance, which can be used...
by SOC analysts and red teams to rank interesting attacker hosts. A medium-severity episode serves as a stepping-stone towards a high-severity episode. Hence, we propose that high-severity vertices hold twice the weight of medium-severity vertices, i.e., \(2^{\text{high}} + 1^{\text{medium}}\).

Table 5 shows the evaluation of CPTC-2018 teams based on all 93 AGs, ranked according to their performance. It shows, for each team, the number of active attacker hosts, and the unique milestones they discover. T5 is the most high-profile team, even though only two team members were responsible for discovering all the high-severity vertices. T1 comes in second, solely because they discover the highest number of medium-severity vertices. Finally, T2 discovers the least number of severe vertices. These results are also corroborated by Table 4, which shows T2 being unsuccessful in discovering many of the objectives.

### 6.2 CPTC-2017 vs. CPTC-2018 S-PDFA comparison

In this experiment, we analyze the extent to which an S-PDFA model summarizes attacker strategies, including infrastructure-related nuances, present in an alert dataset. We learn two S-PDFA models, one for CPTC-2018 (Fig. 6) and the other for CPTC-2017 (see appendix) using the same method and parameter settings. Both models summarize the various paths taken by the teams to reach high-severity states. Several thousands of alerts are modeled by less than 75 states. The 2017 model is larger than the 2018 model, with significantly more transitions. This is because the 2017 dataset has more traces, and there is more variability per-trace, i.e., the 2017 teams exhibit more diverse sub-behaviors than the 2018 teams.

Table 6 shows an exhaustive comparison between the two models in terms of the services used to carry out the objectives. It shows the number of unique objectives exploited by the teams via a particular service. This includes the different ways of reaching the same objective, as identified by the S-PDFA model. The most striking difference between the models is that there are, on average, more paths leading to severe states in the 2017 model than in the 2018 one. This means that a control could be more easily placed in the 2018 network, making it impossible for attackers

### Table 4: Number of unique paths discovered by the CPTC-2018 teams, per objective. Fingerprintable objectives are highlighted (and the number of fingerprints is shown as \(\times^*\)).

<table>
<thead>
<tr>
<th>Service</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T5</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
<th>Finger-prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>complex-main</td>
<td>1*</td>
<td>1*</td>
<td></td>
<td></td>
<td>1*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cslistener</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wap-wsp</td>
<td>1*</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remoteware-cl</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>us-cl</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>etservicemgr</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>unassigned</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Delivery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>14</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>complex-main</td>
<td>1*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cslistener</td>
<td>1*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wap-wsp</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remoteware-cl</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>us-cl</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>etservicemgr</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>unassigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
<tr>
<td>Data Exfiltration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>complex-main</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cslistener</td>
<td>1*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wap-wsp</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remoteware-cl</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>us-cl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
<tr>
<td>etservicemgr</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>unassigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
<tr>
<td>Data Manipulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>complex-main</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cslistener</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wap-wsp</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remoteware-cl</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>us-cl</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>etservicemgr</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>2*</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>unassigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
<tr>
<td>Network DoS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>(\times^*)</td>
</tr>
<tr>
<td>ssdp</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
<tr>
<td>Data Destruction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>us-cl</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(\times^*)</td>
</tr>
</tbody>
</table>

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Table 6: S-PDFA differences between CPTC-2017 and CPTC-2018: the number of unique (high-severity attack stage, targeted service) modeled.

<table>
<thead>
<tr>
<th>Service</th>
<th>S-PDFA version</th>
<th>Unique objectives</th>
<th>Network DoS</th>
<th>Resource Hijacking</th>
<th>Data Manipulation</th>
<th>Data Exfiltration</th>
<th>Data Delivery</th>
<th>Data Destruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>http(s)</td>
<td>CPTC-2017</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>wireless</td>
<td>CPTC-2017</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>remoteAccess</td>
<td>CPTC-2017</td>
<td>8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>surveillance</td>
<td>CPTC-2017</td>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>broadcast</td>
<td>CPTC-2017</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>hostingServer</td>
<td>CPTC-2017</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>email</td>
<td>CPTC-2017</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>authentication</td>
<td>CPTC-2017</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dataSharing</td>
<td>CPTC-2017</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nameserver</td>
<td>CPTC-2017</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>browser</td>
<td>CPTC-2017</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>clocksync</td>
<td>CPTC-2017</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>storage</td>
<td>CPTC-2017</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>unassigned</td>
<td>CPTC-2017</td>
<td>8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>CPTC-2018</td>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Fig. 12: Attack graph of data exfiltration over smtp for CCDC-2018. The same attacker host makes 13 attempts. Paths starting from severe attack stages are possible because the attack graphs show part of a full attack campaign.

6.3 Case study: Applying SAGE to CCDC-2018

The Collegiate Cyber Defense Competition (CCDC) dataset is given as input to SAGE to verify whether it provides the same interpretability and succinctness on a dataset that is not related to penetration testing (see appendix for the resulting S-PDFA). From 1,082,281 alerts, SAGE produces 139 AGs. The fact that we do not have any information about the attacker/victim hosts and the underlying infrastructure reinforces that SAGE is generalizable, and is agnostic to host, dataset, and infrastructure properties. The cases discussed in this section verify that the alert-driven AGs require no expert knowledge to be insightful.

**Case 1 - Path enumeration.** The AG in Fig. 12 shows two possible variants of data exfiltration over SMTP (email service), which can be achieved using the following paths:

1) RPE, ACE, NetDoS, VulnDisc, RPE, ACE, Exfil
2) NetDoS, VulnDisc, RPE, ACE, Exfil
3) VulnDisc, RPE, ACE, NetDoS, Exfil
4) VulnDisc, NetDoS, Exfil
5) VulnDisc, ACE, Exfil
6) VulnDisc, RPE, ACE, Exfil

where RPE is root privilege escalation; ACE is arbitrary code execution; VulnDisc is vulnerability discovery; Exfil is data exfiltration, and NetDoS is network DoS. Explicitly enumerating attack paths in this way can help red teams come up with creative strategies. The first two paths are especially interesting because they start with a severe attack stage. Since these alert-driven AGs show a segment of an on-going campaign, starting from a severe attack stage indicates that the attackers already had intelligence from elsewhere before targeting this machine. Such paths are not intuitive when constructing expert-driven AGs.

**Case 2 - Shortest path.** Fig. 13 shows the AG for performing Network DoS using NTP. It shows two possible variants, starting from six different vertices. Various services are targeted along the way, including http and microsoft-ds (data sharing protocol). The different attacker hosts are highlighted by different edge colors. This AG shows that it is possible to obtain this objective with just two actions, i.e., data exfiltration and network DoS. This happens at the 4-hour mark. About 30 minutes later, root privilege escalation is achieved.
is done leading to arbitrary code execution and Network DoS. This is a counter-example where a subsequent path is longer than the first, even though only a single IP is involved. SOC analysts can further investigate whether these two attempts are indeed made by the same attacker, or some behavioral artifact is at play.

**Case 3 - An extra attempt.** Fig. 14 shows various ways to conduct data exfiltration over https for victims 10.47.3.142 and 10.47.3.1. Both AGs are nearly identical, with one additional exfiltration attempt in the second AG towards the end of the competition, made by a new attacker. SOC analysts can investigate why only one of the two machines were targeted by this new attacker.

### 6.4 Practical implications for CTI: A discussion

CTI platforms convert cyber data into actionable intelligence. Intrusion alerts play a critical role in this process, and automated attacker strategy derivation is a major challenge. Existing tools that display attacker strategies via attack graphs (AG) require network scans and vulnerability information, which are often time-consuming and outdated.

SAGE generates purely alert-driven attack graphs that provide quick insight into attacker strategies, without expert input. SAGE has an explainable architecture (Fig. 1), and can directly augment existing intrusion detection systems. It is released in a docker container for cross-platform support. SAGE facilities attacker strategy analysis via advanced visualizations. The attack graphs are a compressed representation of numerous alerts. Even though SAGE does not discard any alert, the targeted nature of the attack graphs allow analysts to review large quantities of alerts without being overwhelmed.

The analysis presented in this paper merely scratches the surface of the intelligence that can be acquired from these alert-driven AGs. They show clear attack progression and allow strategy comparison. Fingerprintable paths can be recorded for attacker re-identification. They also show that attackers will often follow shorter paths to re-exploit an objective, after they have already discovered a longer one.

We have rigorously evaluated SAGE with diverse datasets and against alternative modeling approaches. We show that the AGs indeed model the teams’ self-reported claims. As demonstrated in Section 6.3, SAGE is agnostic to network, host, and alert properties: with no ground truth about any aspect of the dataset, SAGE produces succinct and interpretable attack graphs, capable of actionable insights.

As a potential use-case, the attack graphs can also be used to evaluate IDS rules. The quality of alert-driven AGs is directly dependent on the quality of the IDS rules. Thus, if an attacker exploits the system, and that path is missing from the AGs, it is an indication of missing or faulty rules.

### 7 Limitations and Future work

Learning from infrequent sequences is a hard problem. A side-effect of including high-severity sinks in the state sequences is that the corresponding AG might show distinct objective-types for similar sequences. Although this happens rarely, handling this problem is left as future work. Secondly, only the state sequences that reach an objective are part of its corresponding AG. It is possible that the attackers divide their tasks such that the full attack path is visible across multiple sequences. The AS construction resolution needs to be changed in order to handle this scenario. Thirdly, the S-PDFA is sensitive to small perturbations in the sequences at test-time. To build resilience, perturbed
traces can be added to the training dataset at learning time. Note that oversampling will alter the true data distribution, which is why we do not opt for this solution. Lastly, we do not yet have a metric to measure model interpretability. Metrics like AIC, BIC, and Perplexity produce arbitrary values for models learned on different parameters, making the comparison meaningless.

Future work will focus on: (a) evaluating the adversarial robustness of SAGE; (b) deploying SAGE during a security competition to measure its effectiveness; and (c) building alert-driven AGs on the fly to monitor evolving threats.

8 Conclusion

Intrusion alerts play a critical role in extracting intelligence about attacker strategies, which is mostly a labor-intensive and expert knowledge-driven process. To the best of our knowledge, SAGE is the first tool that generates purely alert-driven attack graphs (AG), without a priori expert knowledge. We elaborate upon SAGE’s sequence learning pipeline, which is fully transparent, interpretable and explainable. As a core building block, SAGE utilizes a suffix-based probabilistic deterministic finite automaton (S-PDFA) — a model that leverages the temporal and probabilistic dependence between alerts. The S-PDFA brings infrequent severe alerts into the spotlight without discarding any low-severity alerts. Targeted attack graphs are then extracted on a per-victim, per-objective basis. Using several use-cases, we demonstrate the practical utility of SAGE’s AGs.

Our extensive experiments show that the AGs provide a clear picture of the attack progression, and capture the strategies of the participating teams. Specifically for CPTC-2018, SAGE compresses over 330k alerts in 93 AGs in under a minute. These AGs can be used for both, forensic analysis of the attacks, and intelligence collection: (i) They show exactly how specific attacks transpired and reveal that attackers follow shorter paths to re-exploit objectives 84.5% of the time; (ii) They discover 29 uniquely identifiable attack paths, composed of 15.8 episodes on average; (iii) They rank attackers based on the severity of their actions, showing that Team 5 visits the highest, while Team 2 visits the lowest number of severe vertices. SAGE is agnostic to host and network properties: SAGE is capable of producing insightful attack graphs even when no ground truth about attackers and the target network is available. SAGE is released in a docker container for cross-platform support.

9 Acknowledgments

We thank Profs. Bill Stackpole and Daryl Johnson for their guidance, and the reviewers for their constructive feedback that has tremendously improved this manuscript. This effort is partially supported by United States NSF Award 1742789 and RIT Global Cybersecurity Institute.

References


Azqa Nadeem received her BSc degree from National University of Science and Technology, Pakistan in 2015. She received the JEFF/Ellelence Scholarship to pursue her MSc from Delft University of Technology, the Netherlands. She graduated Cum laude in 2018 and received the Best Graduate award in 2019 from the faculty of EEMCS. She is currently a PhD candidate in the Cyber Analytics Lab at Delft University of Technology. Her research involves development of explainable sequential machine learning systems for network security applications.

Sicco Verwer is currently an Associate professor at Delft University of Technology in machine learning for cybersecurity. He is the head of the TU Delft Cyber Analytics Lab where he works on understandable AI for intrusion detection and software understanding. His team won several AI challenges including ones on learning software models, automated reverse engineering, and adversarial machine learning. He received many grants and awards for his research including prestigious VENI and VIDI grants from NWO, and a test-of-time award from ECMLPKDD for his pioneering work on discrimination-free classification.

Stephen Moskal is currently a Ph.D. of Engineering student at Rochester Institute of Technology (RIT). He has received his B.S. and M.S. degrees in Computer Engineering at RIT in 2016. His current research focuses on the simulation and modeling of cyber-attack scenarios and behaviors along with the application of deep machine learning techniques to cyber security. He has developed concepts such as the Attacker Behavior Model (ABM) for cyber threat simulations and the Attack-Action Framework (AIP) to describe the intentions of a cyber-adversary over an attack scenario. Most recently his effort has been applying transfer learning techniques to leverage information cyber security texts to aid in the interpretation and classification of IDS alert descriptions to the AIF. He is expected to graduate with a Doctor of Engineering degree in 2021.

Shanchieh (Jay) Yang (SM'15) received his MS and Ph.D. degrees in Electrical and Computer Engineering from the University of Texas at Austin in 1998 and 2001, respectively. He is currently a Professor in the Department of Computer Engineering and Director of Global Outreach for Global Cybersecurity Institute at Rochester Institute of Technology. His research focuses on cyber attack modeling, machine learning, and simulation to enhance cyber situational awareness and anticipate cyber defense. He was a NSF Trusted CI Fellow in 2019 and a NSF Trusted CI TTP Fellow in 2020. He was recognized in 2019 with IEEE Region 1 Outstanding Teaching in an IEEE Area of Interest Award – for outstanding leadership and contributions to cybersecurity and computer engineering education.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.