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

- *A. Nadeem and S. Verwer are with the Department of Intelligent Systems, Delft University of Technology, 2628 XE Delft, Netherlands. E-mail: azqa.nadeem@tudelft.nl, s.e.verwer@tudelft.nl.*
- *S. Moskal and S. J. Yang are with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623, United States. E-mail: sfm5015@rit.edu, Jay.Yang@rit.edu.*

Manuscript received December 1, 2020; revised May 1, 2021; accepted September 24, 2021.

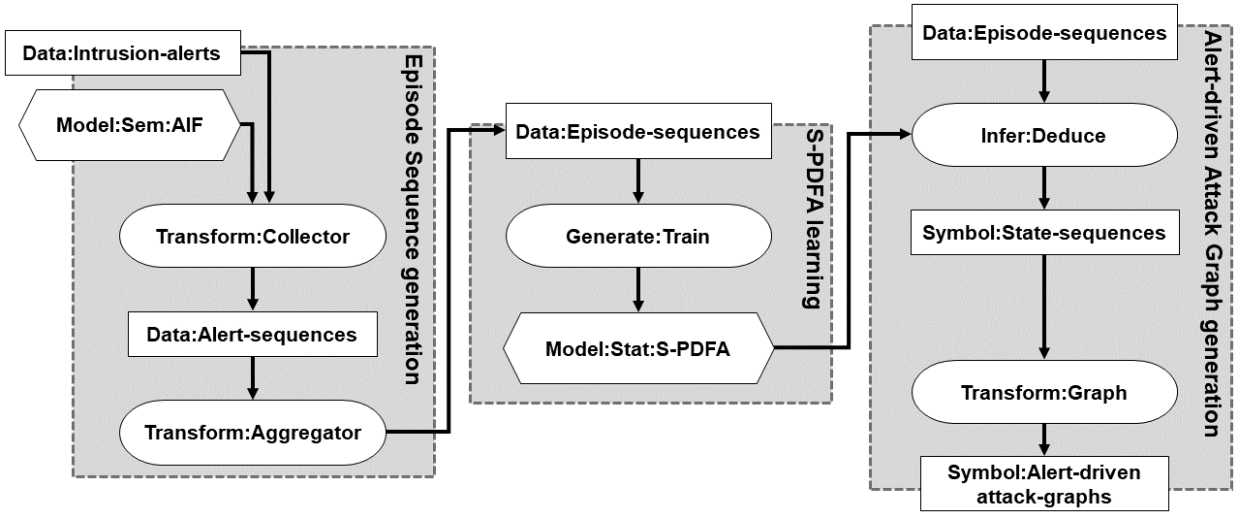


Fig. 1: SAGE takes intrusion alerts as input and generates attack graphs. Intrusion alerts are transformed into episode sequences (Section 4.1). An interpretable S-PDFA model is learned from those sequences (Section 4.2). The sequences are replayed through the S-PDFA and transformed into targeted attack graphs (Section 4.3).

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-PDFA 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-PDFA), 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 attacker strategies can be derived for corroborating specific evidences. Sections 6.1.1 and 6.1.2 discuss concrete examples of interpreting and comparing attacker strategies. The occurrence of certain paths in an AG can serve as fingerprints (see Section 6.1.3). Additionally, attacker hosts can be ranked based on the severity of alerts they raise (see Section 6.1.4).

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

(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. MulVAL [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-source[1]. 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.

	Alerts (raw)	Alerts (filtered)	Episodes	ES/ ESQ	ESS/ Traces	AGs
T1	81,373	26,651	655	103	108	53
T2	42,474	4,922	609	86	92	7
T5	52,550	11,918	622	69	74	51
T7	47,101	8,517	576	63	73	23
T8	55,170	9,037	439	67	79	33
T9	51,602	10,081	1,042	69	110	30

4.1.1 Alert pre-processing

An intrusion alert 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, $sPort$ 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 $t = 1.0\ sec$ 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 $mcat = Map(sign)$ from alert signatures to *attack stages* (see appendix). (iv) Finally, the filtered set $\widetilde{O}$ of intrusion alerts $\widetilde{o}$ is a 5-tuple $\widetilde{o} = \langle sIP, dIP, tServ, ts, mcat \rangle$ 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.

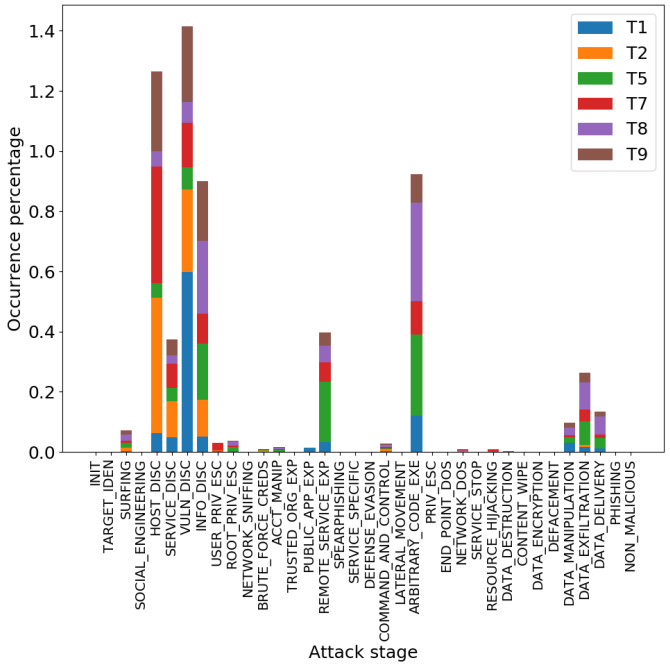


Fig. 2: The distribution of alerts per attack stage for the CPTC-2018 teams. Scanning-alerts are significantly more frequent than exploitation-alerts.

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 ($mcat$), then $AS_{av,c} = \ddot{o}_1^{av,c} \ldots \ddot{o}_n^{av,c}$, where $(a, v) \in A \times V$, $c \in C$. Here, $\ddot{o}_i^{av,c} = \{\hat{o}_1^{av,c} \ldots \hat{o}_\omega^{av,c}\}$ is a multi-set of alerts for $1 \leq i \leq n$. For a window w and given $\widetilde{o}_j = \langle a, v, tServ, ts, c \rangle \in \widetilde{O}$, we define $\hat{o}_j^{av,c} = \langle tServ, ts, c \rangle$ such that $\Pi_{ts}(\hat{o}_1^{av,c}) = i \cdot w$, $\Pi_{ts}(\hat{o}_\omega^{av,c}) - \Pi_{ts}(\hat{o}_1^{av,c}) \leq w$, and $\Pi_{ts}(\hat{o}_j^{av,c}) \leq \Pi_{ts}(\hat{o}_{j+1}^{av,c})$, for $1 \leq j \leq \omega$.

Here, $\Pi_X(\hat{o}_j^{av,c})$ is the projection of the X attribute of $\hat{o}_j^{av,c}$. Furthermore, we use $f'(i)$ to denote the first derivative of the number of alerts per-window over time, *i.e.,* $f'(i) = \frac{\Delta|\ddot{o}_i^{av,c}|}{\Delta i}$ (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 *up*), we consider it the start of an episode; when the frequency is continuously decreasing and reaches a global minimum (a *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

def CONVERT_TO_ES(as)
  es = []
  for ($mcat_x$, as_x) in SPLIT_ON_MCAT(as) **do**
    $timed\_as$ = LEN(sub) **for all** sub **in** as_x
    $slope = f'(x)$ **for all** x **in** $timed\_as$
    ups = GET_POSITIVE_SLOPES($slope$)
    $downs$ = GET_NEGATIVE_SLOPES($slope$)
    $episodes$ = GET_EPISODES(ups, $downs$)
    es.append(($mcat_x$, ep)) **for all** ep **in** $episodes$
  end for
  es = SORT_BY_EPISODE_START(es)
  return es
def GET_EPISODES(ups, $downs$)
  $episodes$ = []
  for i **in** [0,...,LEN(ups)−1] **do**
    if IS_DOWN_BETWEEN_UPS(i, $i+1$, $downs$) **then**
      $down$ = GET_LAST_DOWN(i, $i+1$, $downs$)
      $episodes$.append(($ups[i]$, $down$))
    end if
  end for
  return $episodes$

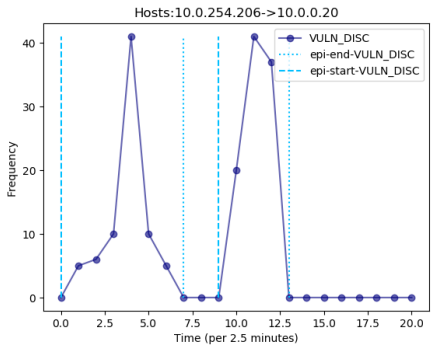


Fig. 3: Bursts of alerts from the same attack stage are aggregated into episodes. Here, an attack sequence related to vulnerability scanning is aggregated into two episodes.

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, $ES_{av} = epi_1^{av} \dots epi_m^{av}$. An episode is a 4-tuple $epi_j^{av} = \langle st^{av}, et^{av}, mcat^{av}, mServ^{av} \rangle$ for $1 \le j \le m$, where $st^{av}, et^{av} \in \mathbb{R}$ denote the start and end time of an episode, $mcat^{av}$ is the attack stage of an episode, and $mServ^{av}$ is the most frequently targeted service in an episode.

In essence, ES's are aggregated sequences of alerts (see Fig. 4). We construct ES_{av} from a windowed alert sequence of attack stage c, *i.e.*, $AS_{av,c} = \ddot{o}_1^{av,c} \dots \ddot{o}_n^{av,c}$. For each $1 \le s \le e \le n$, the start time is $st^{av} = \min(\Pi_{ts}(\ddot{o}_s^{av,c}))$ if $f'(s) = 0$ and $f'(s+1) > 0$; the end time is $et^{av} = \max(\Pi_{ts}(\ddot{o}_e^{av,c}))$ if $f'(e) = 0$ and $f'(e-1) < 0$; the attack stage is $mcat^{av} = c$, and the most frequently targeted service is $mServ^{av} = \arg\max_{mserv} |\{\Pi_{tServ}(\ddot{o}_i^{av,c}) = mserv : s \le i \le e\}|$.

a1→v2: [<0,150,infoD,http>, <900,1200,infoD,http>, <901,1200,CnC,http>, <2250,2550, infoD,http>, <2251,2550,exfil,http>]
a3→v11: [<3497,3797,surf,http>, <3498,3647,vulnD,http>, <3499,3797,infoD,http>, <3500,3647,rPrivEsc,http>, <3501,3647,acctManip,http>, <3502,3647,remoteexp,http>, <3503,3647,CnC,http>, <3504,3647,ACE,http>, <3505,3797,netDOS,http>, <3506,3647, resHJ,http>, <3507,3647,dManip,http>, <3508,3647,exfil,http>, <3509,3647,delivery, http>, <3797,4097,hostD,http>, <3798,4097,serD,ssh>, <3799,4097,vulnD,mysql>, <4097, 4397,infoD,http>, <4098,4397,netDOS,http>, <4099,4397,resHJ,http>, <4697,4847,infoD, http>, <4698,4847,rPrivEsc,http>]
a5→v2: [<27379,27529,infoD,http>, <27380,27529,CnC,http>, <29479,29629,infoD,http>, <29480,29629,CnC,http>]

Fig. 4: Episode sequences from CPTC-2018: Each sequence is a list of tuples $\langle st, et, mcat, mServ \rangle$, 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 $A = \langle Q, \Sigma, \Delta, P, q_0 \rangle$ defining the machine structure: Q is a finite set of states; Σ is a finite alphabet of symbols; Δ is a finite set of transitions; $P : \Delta \to [0, 1]$ 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 $\langle q, q', a \rangle$, where $q, q' \in Q$ are the target and source states, and $a \in \Sigma$ is a symbol. P is a function such that $\sum_{q,a} P(\langle q, q', a \rangle) = 1$. Additionally, Δ is such that for every $q \in Q$ and $a \in \Sigma$, there exists at most one $\langle q, q', a \rangle \in \Delta$, 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 Σ^n for all $1 \le n$. The probability of a sequence $s = a_1 \dots a_n$ is computed along the reverse path $q_0 a_n q_1 a_{n-1} q_2 \dots a_1 q_n$, with $\langle q_i, q_{i+1}, a_{n-i} \rangle \in \Delta$, called the S-PDFA run. The sequence probability is then $P(s) = \prod_{0 \le i < n} P(\langle q_i, q_{i+1}, a_{n-i} \rangle)$, where $\prod$ 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

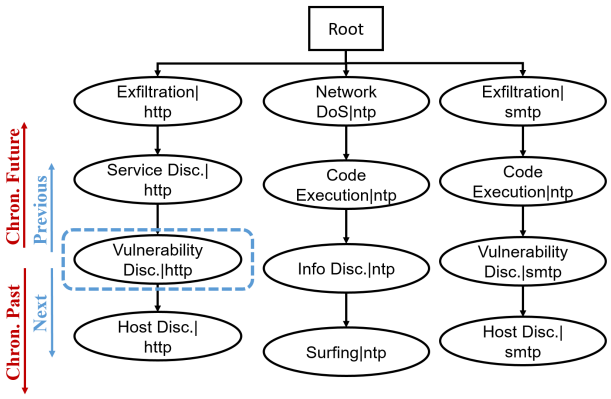


Fig. 5: A suffix tree for three traces. For any vertex, the previous vertex happens chronologically in the future.

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. $Severity(epi)$ 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 $ES_{av} = epi_1^{av} \dots epi_m^{av}$, define a break-point as an index i such that $Severity(epi_{i+1}^{av}) < Severity(epi_i^{av})$. An Episode Subsequence $ESS_{av} = epi_s^{av} \dots epi_{s'}^{av}$ is a contiguous subsequence of ES_{av} without break-points, *i.e.*, $ES_{av} = epi_1^{av} \dots epi_s^{av} \dots epi_{s'}^{av} \dots epi_m^{av}$. Every ES_{av} is broken into its break-point-free subsequences $ES_{av} = ESS_{av,1} \dots ESS_{av,k}$.

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 $\langle mcat, Theme(mServ) \rangle$. $Theme()$ 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.

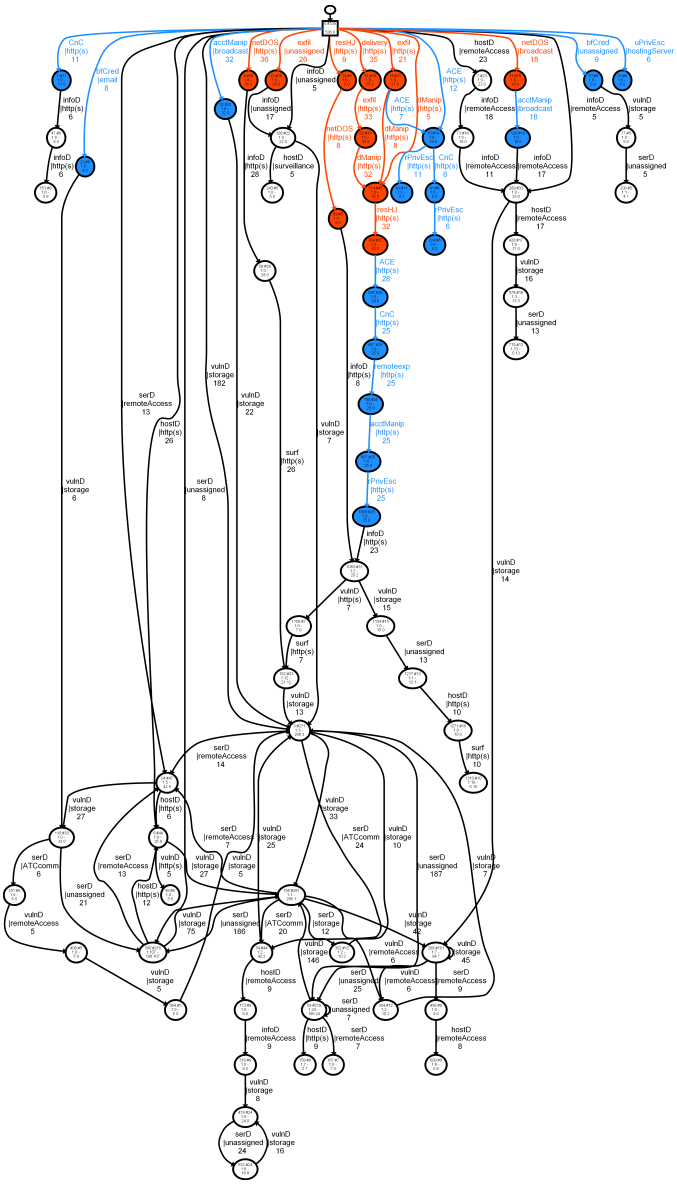


Fig. 6: The S-PDFA model for CPTC-2018. The states are colored according to the severity of the incoming symbol's attack stage: red is high, blue is medium, white is low.

We use three important settings for learning an interpretable S-PDFA: (i) We limit which states are used to compute statistics. The learning algorithm merges two states if it does not find sufficient evidence that the states are different. A lower bound on the data required for this evidence is controlled by the *state_count* and *symbol_count* parameters. Intuitively, it is better to use only frequently-occurring states and transitions in the statistical tests, but the default values of 50 and 25 are much too large for the limited amount of high-severity episodes in the dataset. We set both to 5, implying that a state in the suffix tree that occurs only 5 times in total can provide sufficient evidence to prevent a merge from happening. (ii) We use the *Markovian* property, which dictates that for any given states q_1 and q_2, the previous transition labels have to be identical, *i.e.*, $\langle q_1', q_1, a \rangle$ and $\langle q_2', q_2, a \rangle$. It enforces that the incoming transition label for states is unique, which makes the model

easier to interpret. (iii) We utilize *sink states*. The core algorithm continues merging until all states have either been merged or added to the model. For infrequent states, there is typically insufficient evidence to prevent a merge and they can, therefore, be merged with any of the states added in the previous iterations. The *sink_count* parameter avoids this by disallowing merges that occur *sink_count* times or less, which we set to 5. The states that occur less than *sink_count* times are not displayed in the learned model, which makes it easier to interpret. That said, high-severity sink states are interesting from behavioral perspective since they show the rare exploitative actions. We perform post-processing to include such high-severity sink states in the learned model. This process salvages 13% of the sinks, which otherwise would not have appeared in the attack graphs.

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 $2^{-\frac{1}{N}\sum_{i=1}^{N} log_2 P(x_i)}$, 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 provides the best fit with the training data, as expected. The SAGE S-PDFA is about twice as "perplexed". It is hard to quantify how good this is exactly, but it is better than what the Markov chain and the default S-PDFA achieve. On the test data, SAGE S-PDFA gives the best perplexity value, demonstrating that it accurately captures many patterns present in the data.

TABLE 2: Model quality evaluation (Perplexity) of four suffix variants on the CPTC-2018 traces. Suffix tree and SAGE S-PDFA are the best on training and test data, respectively.

	Suffix tree	Markov chain	Default S-PDFA	SAGE S-PDFA
Training set	`1265.4*`	13659.6	15136.5	2397.8
Holdout test set	13020.7	11617.8	11241.5	`9884.6*`

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 (ES) into state sequences (ESQ). We run each episode subsequence $a_1 \dots a_n$ through the model, which produces $q_n \dots q_0$. A state subsequence is an episode subsequence augmented with state identifiers, *i.e.,* $q_0 a_n q_1 a_{n-1} q_2 \dots a_1 q_n$.

Definition 5. A State Sequence (ESQ) for an episode sequence $ES_{av} = ESS_{av,1} \dots ESS_{av,k}$ is the concatenated sequence $ESQ_{av} = sq_1 sq_2 \dots sq_k$, 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 $\langle mcat, mServ, q \rangle$ 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 $\langle mcat \rangle$, attack stage and targeted service $\langle mcat, mServ \rangle$ or the full tuple $\langle mcat, mServ, q \rangle$. 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 $obj = \langle mcat, mServ, q \rangle$ and $obj' = \langle mcat, mServ, q' \rangle$, by adding a new root node $\langle mcat, mServ \rangle$. 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 $\langle mcat, mServ \rangle$. 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

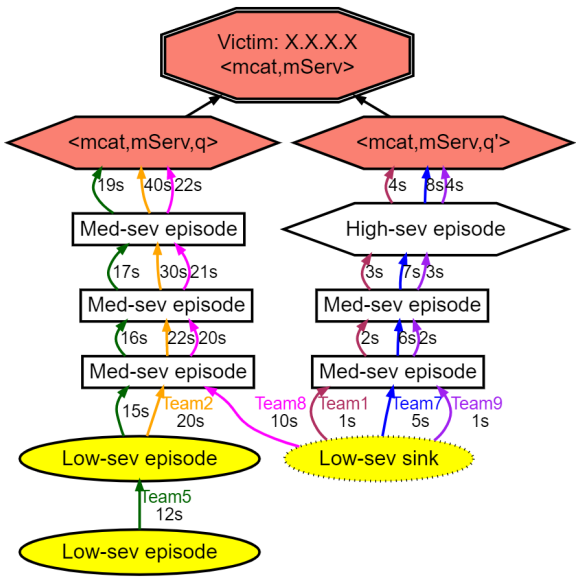


Fig. 7: An alert-driven attack graph: Vertices: Labels show ⟨*attack stage, targeted service, state identifier*⟩. Low-severity episodes are *oval*, medium-severity are *boxes*, high-severity are *hexagons*. The first episode in a *path* is *yellow*, the objective is *black*. Sinks are *dotted*. Edges: Labels show *seconds* since the first alert. Colors show team affiliation: T1 (*Maroon*), T2 (*Orange*), T5 (*Green*), T7 (*black*), T8 (*Magenta*), T9 (*Purple*).

the strategy comparison easier, all teams that achieve an objective are shown in one graph, distinguishable by their edge color. Fig. 7 shows an alert-driven attack graph's anatomy.

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.*, $Simplicity(AG) = \frac{|V|}{|E|}$, 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*.

TABLE 3: Experimental dataset summary(before filtering).

Dataset/Properties	CPTC-2018	CPTC-2017	CCDC-2018
# alerts	330,270	43,611	1,052,281
# teams	6	9	Unknown
# IPs	42	494	2138
# services	160	168	2050
Duration (hrs)	9	11	25
Attacker hosts known?	Yes	No	No
Victim hosts known?	Yes	No	No
Dataset type	Penetration testing	Penetration testing	Blue teaming

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 sever. 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

analysts i) determine how an attack happened, and ii) compare attacker strategies for fingerprintable behaviors.

Experiments. We perform three set 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.

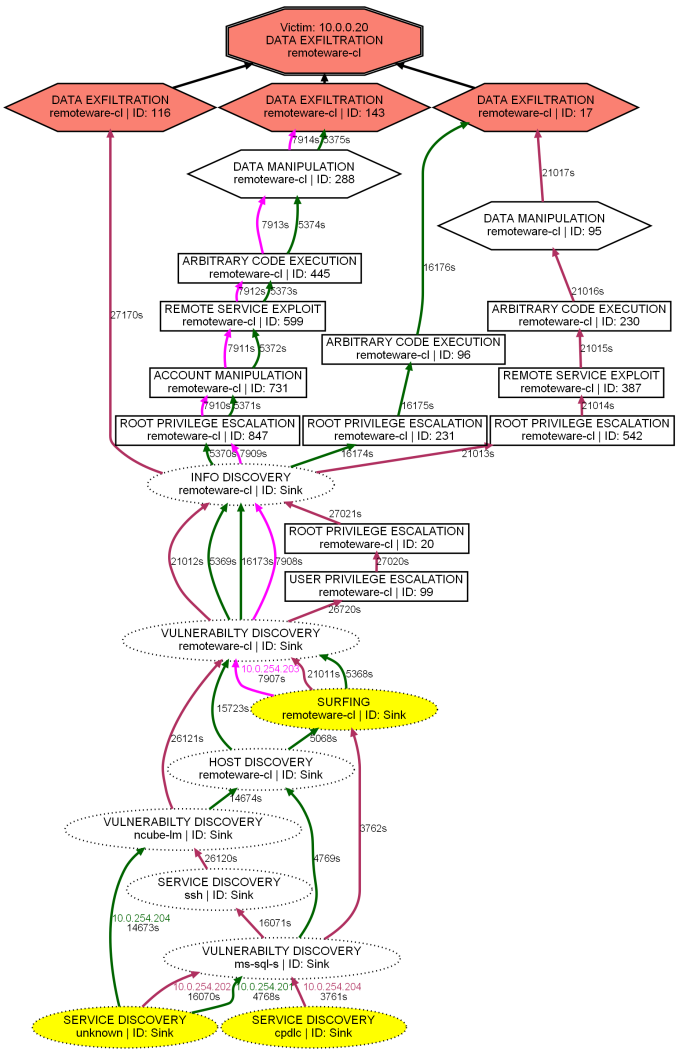


Fig. 8: Attack graph of data exfiltration over remoteware-cl. Three attacker teams successfully exploit it: Teams 1 and 5 exploit it twice, and each subsequent attempt is shorter than the first. The S-PDFA identifies three ways of exploiting the objective based on the actions that lead up to it.

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 exfiltration 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].

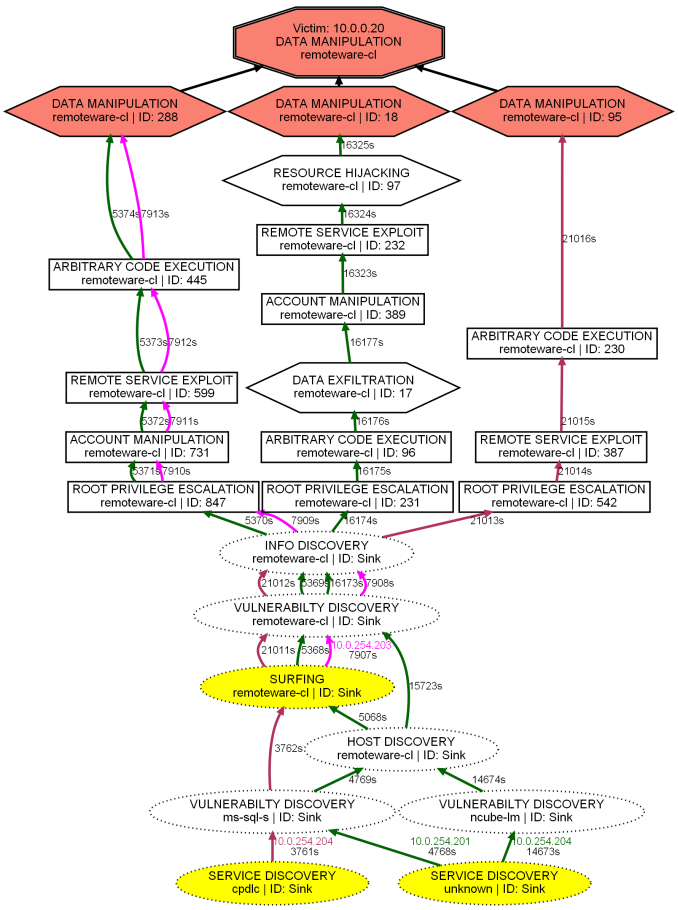
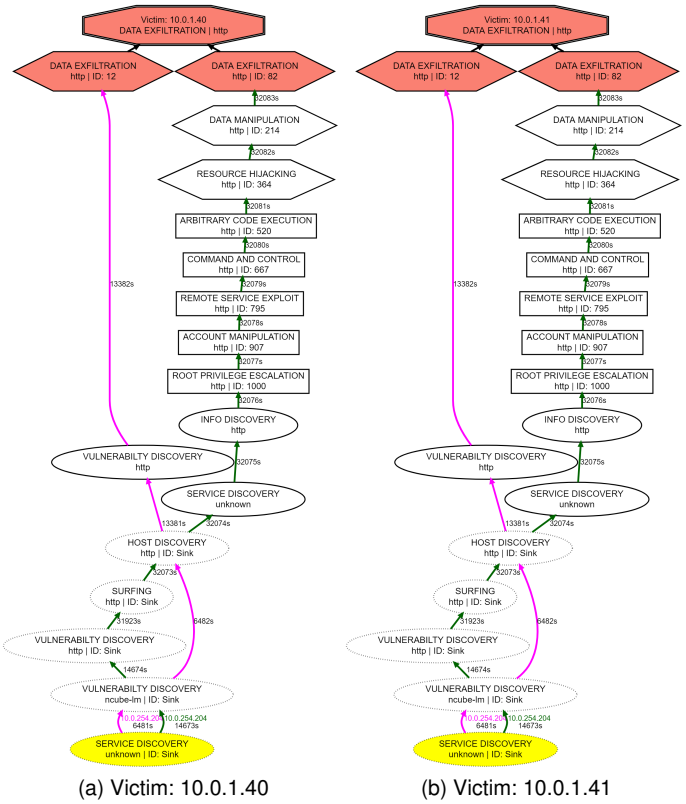


Fig. 9: An attack graph of data manipulation is a partial sub-graph of Fig. 8 because of overlapping attack paths.



(a) Victim: 10.0.1.40 (b) Victim: 10.0.1.41

Fig. 10: Identical and simultaneous attacks targeting multiple victim hosts result in identical attack graphs.

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.

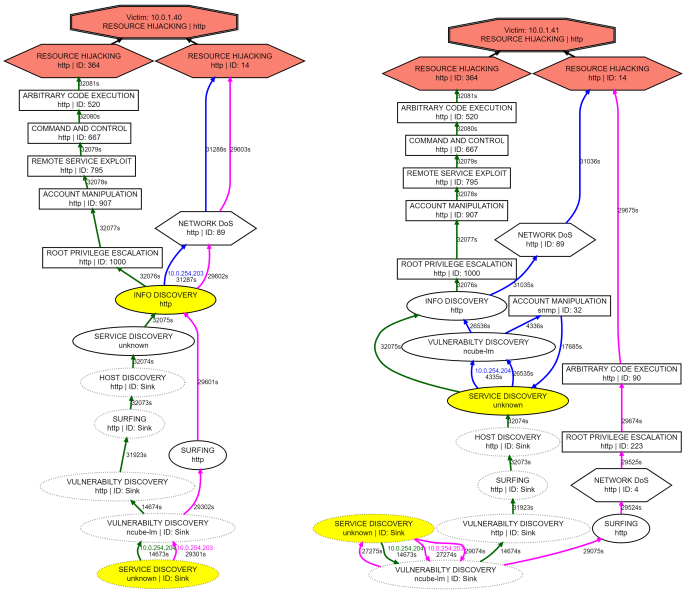
(2) Fig. 11 shows T5, T7, and T8 conducting resource hijacking over two hosts (`.40`, `.41`) using http, resulting in highly similar AGs. T5 has an identical strategy for both hosts. T7 does scans before manipulating accounts and conducting a network DoS over `.41`, while later they only perform a scan and a network DoS over `.40`. Similarly, T8 does a privilege escalation and code execution after network DoS over `.41`, while they later only do a network DoS over `.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



(a) Victim: 10.0.1.40 (b) Victim: 10.0.1.41

Fig. 11: Similar attacks targeting multiple victim hosts result in overlapping attack graphs.

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.*, $\frac{(2*high)+(1*medium)}{3}$.

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

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 x*).

Service ↓	Unique paths discovered						Finger-print?
	T1	T2	T5	T7	T8	T9	
Data Delivery							
http	8		3	2	5	5	
commplex-main				1*			✓
cslistener	1					1	
wap-wsp	1		1				
remoteware-cl	1*						✓
us-cli			1			1	
unassigned			1*				✓
Data Exfiltration							
http	13		8	3	12	6	
commplex-main				3*			✓
cslistener						1*	✓
wap-wsp			1*				✓
remoteware-cl	2		2		1		
us-cli			1		2	1	
etlservicemsgr			2*				✓
unassigned	2	7	1	6	1	7	
Data Manipulation							
http	14		7	3	8	6	
commplex-main				1*			✓
cslistener						1*	✓
wap-wsp	1		1				
remoteware-cl	1		2		1		
us-cli			1		1	1	
etlservicemgr			2*				✓
unassigned			1*				✓
Resource Hijacking							
http	5		6	9	10	5	
commplex-main				1*			✓
cslistener						1*	✓
wap-wsp	1		1				
remoteware-cl			2		1		
us-cli			1			1	
etlservicemgr			2*				✓
unassigned			1*				✓
Network DoS							
http	6	7		7	8	14	
ssdp	8*						✓
Data Destruction							
us-cli				1*			✓

TABLE 5: CPTC-2018 team ranking based on the fraction of unique severe vertices discovered. It also shows the number of attacker hosts responsible for discovering those vertices.

Teams	# Active hosts	# Vertices		Weighted average percentage
		High-sev (out of 70)	Medium-sev (out of 148)	
T5	2/5	28 (40%)	40 (27%)	35.67
T1	5/6	18 (26%)	62 (42%)	31.33
T9	5/5	23 (33%)	36 (24%)	30.0
T7	6/6	22 (31%)	26 (18%)	26.67
T8	6/7	15 (21%)	32 (22%)	21.33
T2	3/6	3 (4%)	8 (5%)	4.33

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 6: S-PDFA differences between CPTC-2017 and CPTC-2018: the number of unique ⟨high-severity attack stage, targeted service⟩ modeled.

Service ↓	S-PDFA version	Unique objectives	Network_DoS	Resource Hijacking	Data Manipulation	Data Exfiltration	Data Delivery	Data Destruction
http(s)	**CPTC-2017**	10	✓			✓	✓	
	CPTC-2018	18	✓	✓	✓	✓	✓	
wireless	**CPTC-2017**	3				✓	✓	
	CPTC-2018	6		✓	✓	✓	✓	
remoteAccess	**CPTC-2017**	8	✓			✓	✓	
	CPTC-2018	9		✓	✓	✓	✓	✓
surveillance	**CPTC-2017**	7				✓	✓	
	CPTC-2018	9		✓	✓	✓	✓	
broadcast	**CPTC-2017**	8	✓					
	CPTC-2018	6	✓	✓	✓	✓	✓	
hostingServer	**CPTC-2017**	0	n/a					
	CPTC-2018	9		✓	✓	✓	✓	
email	**CPTC-2017**	1				✓		
	CPTC-2018	0	n/a					
authentication	**CPTC-2017**	2	✓			✓		
	CPTC-2018	0	n/a					
dataSharing	**CPTC-2017**	2				✓		
	CPTC-2018	0	n/a					
nameserver	**CPTC-2017**	1			✓			
	CPTC-2018	0	n/a					
browser	**CPTC-2017**	2				✓	✓	
	CPTC-2018	0	n/a					
clocksync	**CPTC-2017**	2	✓					
	CPTC-2018	0	n/a					
storage	**CPTC-2017**	3				✓	✓	
	CPTC-2018	0	n/a					
unassigned	**CPTC-2017**	8				✓	✓	
	CPTC-2018	7		✓	✓	✓	✓	

to complete certain objectives. This is important because the 2018 teams exploit each service for completing more objectives, on average. However, the same does not hold for the 2017 model as it has additional pathways for attackers to evade controls.

Table 6 shows that the teams in the election scenario (2017) exfiltrate data using a specific type of browser, while this service is never even scanned in the automotive scenario (2018). They also conduct DoS attacks using the network time protocol (clocksync), and use services associated to authentication and storage that are never used in the automotive scenario. On the other hand, teams conduct privilege escalation on a web hosting service in the automotive scenario, but never in the election scenario. Furthermore, while both team-sets scan and elevate privileges related to email, only the teams in the election scenario manage to exploit it for exfiltrating data. The unassigned service category is particularly intriguing because it refers to high port numbers being targeted. SOC analysts for both the networks should analyze whether these open ports indicate a misconfiguration in their networks.

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 1,052,281 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.

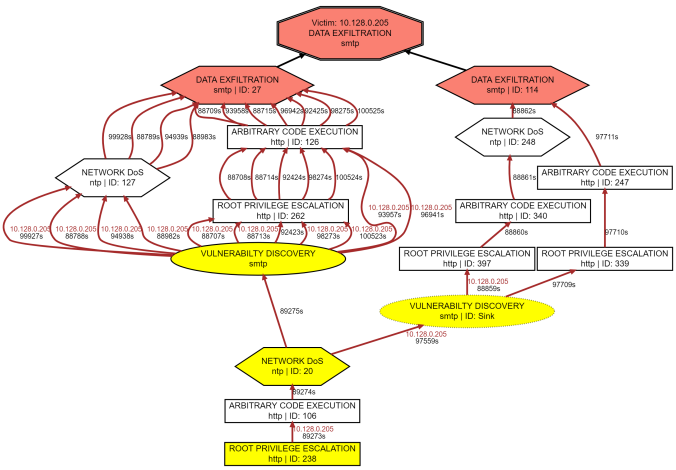


Fig. 12: Attack graph of data exfiltration over smtp for CCDC-2018. The same attacker host makes 13 attempts. Paths starting from severe attack stages are possible because the attack graphs show part of a full attack campaign.

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

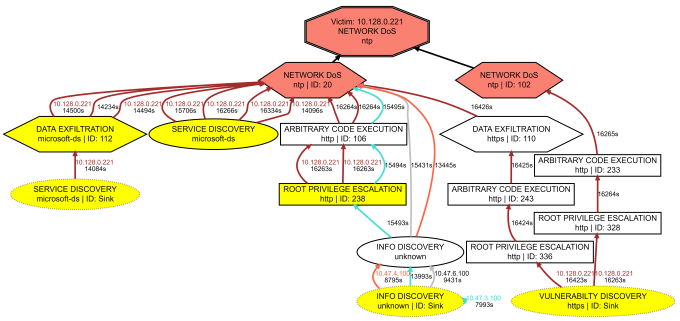


Fig. 13: Attack graph of network DoS over ntp for CCDC-2018. There are six possible starting actions and two possible ways to reach the objective.

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

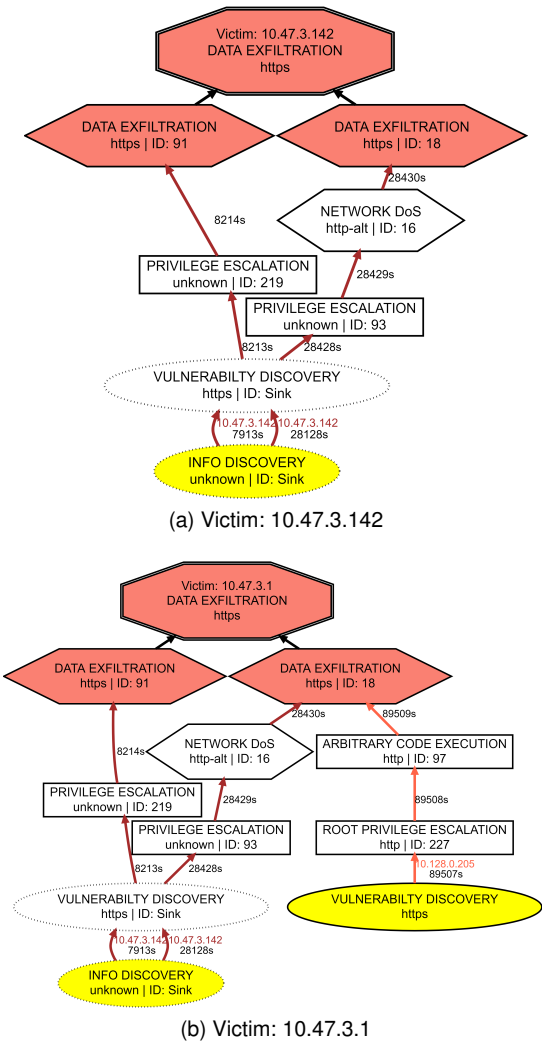


Fig. 14: Highly similar attack graphs of two victims from CCDC-2018. The graphs are identical, except for an additional attack attempt by a new attacker in the second graph.

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

[1] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *NDSS*, 2019.

[2] T. Casey, "Survey: 27 percent of it professionals receive more than 1 million security alerts daily," 2018, accessed: 08-Jul-2021. [Online]. Available: https://www.imperva.com/blog/27-percent-of-it-professionals-receive-more-than-1-million-security-alerts-daily

[3] R. Sadoddin and A. Ghorbani, "Alert correlation survey: framework and techniques," in *PST*, 2006.

[4] S. Salah, G. Maciá-Fernández, and J. E. DíAz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, 2013.

[5] F. M. Alserhani, "Alert correlation and aggregation techniques for reduction of security alerts and detection of multistage attack," *IJASCSE*, 2016.

[6] L. Williams, R. Lippmann, and K. Ingols, "Garnet: A graphical attack graph and reachability network evaluation tool," in *VizSec*. Springer, 2008.

[7] M. Chu, K. Ingols, R. Lippmann, S. Webster, and S. Boyer, "Visualizing attack graphs, reachability, and trust relationships with navigator," in *VizSec*, 2010.

[8] M. Angelini, N. Prigent, and G. Santucci, "Percival: proactive and reactive attack and response assessment for cyber incidents using visual analytics," in *VizSec*. IEEE, 2015.

[9] S. Roschke, F. Cheng, and C. Meinel, "A new alert correlation algorithm based on attack graph," in *CISIS*. Springer, 2011.

[10] C. Liu, A. Singhal, and D. Wijesekera, "Using attack graphs in forensic examinations," in *ARES*. IEEE, 2012.

[11] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, and K. Prole, "Advances in topological vulnerability analysis," in *CATCH*. IEEE, 2009.

[12] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer." in *USENIX Security Symposium*. Baltimore, MD, 2005.

[13] M. L. Artz, "Netspa: A network security planning architecture," Ph.D. dissertation, Massachusetts Institute of Technology, 2002.

[14] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *CSFW*. IEEE, 2002.

[15] C. Sillaber, C. Sauerwein, A. Mussmann, and R. Breu, "Data quality challenges and future research directions in threat intelligence sharing practice," in *WISCS*, 2016.

[16] A. Nadeem, S. Verwer, and S. J. Yang, "Sage: Intrusion alert-driven attack graph extractor," in *VizSec*. IEEE, 2021.

[17] M. van Bekkum, M. de Boer, F. van Harmelen, A. Meyer-Vitali, and A. t. Teije, "Modular design patterns for hybrid learning and reasoning systems: a taxonomy, patterns and use cases," *arXiv preprint arXiv:2102.11965*, 2021.

[18] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *TDSC*, 2004.

[19] S. Verwer and C. A. Hammerschmidt, "Flexfringe: a passive automaton learning package," in *ICSME*. IEEE, 2017.

[20] RIT, 2021, accessed: 08-Jul-2021. [Online]. Available: https://globalcptc.org/

[21] WRCCDC, 2021, accessed: 08-Jul-2021. [Online]. Available: http://www.nationalccdc.org/

[22] D. Shackleford, "Who's using cyberthreat intelligence and how?" *SANS Institute. Retrieved January*, 2015.

[23] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *CCS*, 2002.

[24] X. Qin and W. Lee, "Discovering novel attack strategies from infosec alerts," in *ESORICS*. Springer, 2004.

[25] B. Zhu and A. A. Ghorbani, "Alert correlation for extracting attack strategies," *IJ Network Security*, 2006.

[26] C.-H. Wang and Y.-C. Chiou, "Alert correlation system with automatic extraction of attack strategies by using dynamic feature weights," *IJCCE*, 2016.

[27] S. Haas and M. Fischer, "Gac: graph-based alert correlation for the detection of distributed multi-step attacks," in *SAC*, 2018.

[28] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and M. Rajarajan, "Intrusion alert prioritisation and attack detection using post-correlation analysis," *Computers & Security*, 2015.

[29] S. McElwee, J. Heaton, J. Fraley, and J. Cannady, "Deep learning for prioritizing and responding to intrusion detection alerts," in *MILCOM*. IEEE, 2017.

[30] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *JISA*, 2016.

[31] P. Ning, D. Xu, C. G. Healey, and R. S. Amant, "Building attack scenarios through integration of complementary alert correlation method." in *NDSS*, 2004.

[32] H. Hu, J. Liu, Y. Zhang, Y. Liu, X. Xu, and J. Huang, "Attack scenario reconstruction approach using attack graph and alert data mining," *JISA*, 2020.
[33] J. Homer, A. Varikuti, X. Ou, and M. A. McQueen, "Improving attack graph visualization through data reduction and attack grouping," in *VizSec*. Springer, 2008.
[34] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs," in *ACSAC*. IEEE, 2009.
[35] J. Navarro, V. Legrand, S. Lagraa, J. François, A. Lahmadi, G. De Santis, O. Festor, N. Lammari, F. Hamdi, A. Deruyver *et al.*, "Huma: A multi-layer framework for threat analysis in a heterogeneous log environment," in *FPS*. Springer, 2017.
[36] J. Navarro, V. Legrand, A. Deruyver, and P. Parrend, "Omma: open architecture for operator-guided monitoring of multi-step attacks," *EURASIP Journal on Information Security*, 2018.
[37] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "A framework for cyber threat intelligence extraction from raw log data," in *Big Data*. IEEE, 2019.
[38] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, "Tabor: A graphical model-based approach for anomaly detection in industrial control systems," in *Asia-CCS*, 2018.
[39] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, "Beyond labeling: Using clustering to build network behavioral profiles of malware families," *Malware Analysis Using Artificial Intelligence and Deep Learning*, 2021.
[40] S. C. De Alvarenga, S. Barbon Jr, R. S. Miani, M. Cukier, and B. B. Zarpelão, "Process mining and hierarchical clustering to help intrusion alert visualization," *Computers & Security*, 2018.
[41] Y. Chen, Z. Liu, Y. Liu, and C. Dong, "Distributed attack modeling approach based on process mining and graph segmentation," *Entropy*, 2020.
[42] S. Moskal, S. J. Yang, and M. E. Kuhl, "Extracting and evaluating similar and unique cyber attack strategies from intrusion alerts," in *ISI*. IEEE, 2018.
[43] J. Liu, B. Liu, R. Zhang, and C. Wang, "Multi-step attack scenarios mining based on neural network and bayesian network attack graph," in *ICAIS*. Springer, 2019.
[44] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *TKDD*, 2019.
[45] M. T. Ribeiro, S. Singh, and C. Guestrin, """ why should i trust you?" explaining the predictions of any classifier," in *SIGKDD*, 2016.
[46] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, "Explainable machine learning for scientific insights and discoveries," *IEEE Access*, 2020.
[47] N. Munaiah, A. Rahman, J. Pelletier, L. Williams, and A. Meneely, "Characterizing attacker behavior in a cybersecurity penetration testing competition," in *ESEM*. IEEE, 2019.
[48] IANA, 2021, accessed: 08-Jul-2021. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml
[49] S. Moskal and S. J. Yang, "Framework to describe intentions of a cyber attack action," *arXiv preprint arXiv:2002.07838*, 2020.
[50] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," *Technical report*, 2018.
[51] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces." in *USENIX Security Symposium*, 2007.
[52] J. De Ruiter and E. Poll, "Protocol state fuzzing of {TLS} implementations," in *USENIX Security Symposium*, 2015.
[53] C. Y. Cho, D. Babi ć, E. C. R. Shin, and D. Song, "Inference and analysis of formal models of botnet command and control protocols," in *CCS*, 2010.
[54] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *Symposium on Security and Privacy*. IEEE, 2009.
[55] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm," in *ICGI*. Springer, 1998.
[56] M. J. Heule and S. Verwer, "Software model synthesis using satisfiability solvers," *Empirical Software Engineering*, 2013.
[57] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
[58] R. C. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," in *ICGI*. Springer, 1994.
[59] R. Parekh and V. Honavar, "Learning dfa from simple examples," *Machine Learning*, 2001.
[60] S. Verwer, R. Eyraud, and C. De La Higuera, "Pautomac: a probabilistic automata and hidden markov models learning competition," *Machine learning*, 2014.
[61] B. Balle, R. Eyraud, F. M. Luque, A. Quattoni, and S. Verwer, "Results of the sequence prediction challenge (spice): a competition on learning the next symbol in a sequence," in *ICGI*, 2017.
[62] Q. Lin, Y. Zhang, S. Verwer, and J. Wang, "Moha: A multi-mode hybrid automaton model for learning car-following behaviors," *T-ITS*, 2018.
[63] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, "Enabling visual analytics via alert-driven attack graphs," in *CCS*, 2021.
[64] A. Adadi and M. Berrada, "Peeking inside the black-box: a survey on explainable artificial intelligence (xai)," *IEEE access*, 2018.
[65] RIT, "cptc dataset," 2018, accessed: 08-Jul-2021. [Online]. Available: https://mirror.rit.edu/cptc/
[66] F. Hassanabad, "ccdc dataset," 2019, accessed: 08-Jul-2021. [Online]. Available: https://github.com/FrankHassanabad/suricata-sample-data
[67] OSIF, 2019, accessed: 08-Jul-2021. [Online]. Available: https://suricata.readthedocs.io/en/suricata-6.0.3/
[68] E. M. Hutchins, M. J. Cloppert, R. M. Amin *et al.*, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, 2011.

Azqa Nadeem received her BSc degree from National University of Science and Technology, Pakistan in 2015. She received JvEffen Excellence 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 (AIF) 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 anticipatory 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.