SAGE: Intrusion Alert-driven Attack Graph Extractor

Azqa Nadeem
Delft University of Technology
Delft, The Netherlands
azqa.nadeem@tudelft.nl

Sicco Verwer
Delft University of Technology
Delft, The Netherlands
s.e.verwer@tudelft.nl

Stephen Moskal
Rochester Institute of Technology
Rochester, United States
sfm5015@rit.edu

Shanchieh Jay Yang
Rochester Institute of Technology
Rochester, United States
Jay.Yang@rit.edu

ABSTRACT

Attack graphs (AG) are used to assess pathways availed by cyber adversaries to penetrate a network. State-of-the-art approaches for AG generation focus mostly on deriving dependencies between system vulnerabilities based on network scans and expert knowledge. In real-world operations however, it is costly and ineffective to rely on constant vulnerability scanning and expert-crafted AGs. We propose to automatically learn AGs based on actions observed through intrusion alerts, without prior expert knowledge. Specifically, we develop an unsupervised sequence learning system, SAGE, that leverages the temporal and probabilistic dependence between alerts in a suffix-based probabilistic deterministic finite automaton (S-PDFA) – a model that accentuates infrequent severe alerts and summarizes paths leading to them. AGs are then derived from the S-PDFA. Tested with intrusion alerts collected through Collegiate Penetration Testing Competition, SAGE produces AGs that reflect the strategies used by participating teams. The resulting AGs are succinct, interpretable, and enable analysts to derive actionable insights, *e.g.,* attackers tend to follow shorter paths after they have discovered a longer one.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; **Usability in security and privacy**; • **Human-centered computing** → *Visual analytics*; • **Computing methodologies** → *Unsupervised learning*.

KEYWORDS

Alert-driven attack graphs, Deterministic finite automaton

1 INTRODUCTION

Security Operations Centers (SOCs) typically receive thousands of intrusion alerts per day[1]. While alert correlation techniques help reduce alerts from intrusion detection systems (IDS) [1, 21, 22], they do not show the attack progression and attacker strategies, *i.e.,* they show *what* the attackers did, but do not provide insight into *how* the infrastructure was exploited.

Attack graphs (AG) are visual models of attacker strategies that have been used for attack scenario detection and network hardening [11, 12]. Existing approaches to generate AGs are time-consuming due to their expert-driven nature — they utilize a significant amount of domain knowledge [1, 16, 17] and published vulnerability reports [7, 9, 18, 20]. In real-world operations however, it is costly and ineffective to rely on constant vulnerability scanning and expert-crafted AGs. Meanwhile, SOCs often possess intrusion alert datasets from prior security incidents. We show that these alerts can be used as a basis to generate AGs.

In this paper, we propose SAGE — IntruSion alert-driven Attack Graph Extractor. SAGE leverages sequence learning to mine patterns from intrusion alerts, models them using an automaton, and represents them in the form of an attack graph. The three core components of SAGE are shown in Figure 1. A tool such as SAGE can have important implications for evaluation of defensive controls and for cybersecurity education. For example, i) missing paths in the generated attack graphs can indicate faulty or missing IDS rules; ii) attack graphs can help improve penetration testing teams' performance, by showing potential shorter paths to obtain an objective or redundant paths caused by lack of communication.

A major challenge in machine learning-based attacker strategy identification is the scarcity of severe intrusion alerts — most of the alerts are related to low-severity events, *e.g.,* network scans, which are not very valuable for analysts [24]. While most machine learning (ML) solutions discard infrequent patterns, we learn a suffix-based probabilistic deterministic finite automaton (S-PDFA) — a model that accentuates infrequent severe alerts, without discarding low-severity alerts. The S-PDFA summarizes attack paths leading to severe attack stages. It can differentiate between alerts that have identical signatures but different contexts, *e.g.,* scanning at the start indicates reconnaissance, while scanning midway through an attack indicates attack progression. Attack graphs are then extracted from the S-PDFA on a per-objective, per-victim basis.

We demonstrate the effectiveness of SAGE on distributed multi-stage attack scenarios, *i.e.,* a setting where multi-member teams progress through various attack stages in order to compromise numerous targets. Penetration testing competitions provide an ideal

[1] https://blog.paloaltonetworks.com/2020/09/secops-analyst-burnout/

AI4Cyber'21, August 14-18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

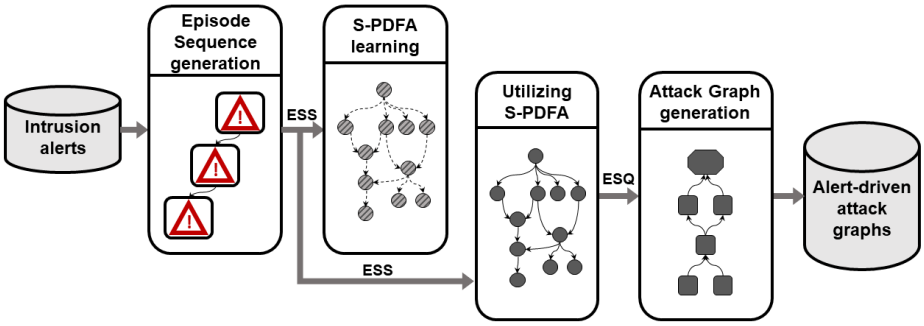

Figure 1: SAGE workflow: Intrusion alerts go in, Attack Graphs come out.

setting to study distributed multi-stage attacks in a controlled setting. To this end, we use intrusion alerts collected through Collegiate Penetration Testing Competition (CPTC)[2]. The AGs reflect the actual pathways taken by the penetration testers, even with an imperfect IDS. The AGs are succinct, and effective in highlighting strategic differences between participating teams. They reveal that attackers often follow shorter paths after they have discovered a longer one. In short, our contributions are:

(1) We develop SAGE, a tool that automatically generates succinct high-severity attack graphs from intrusion alerts, without prior knowledge.
(2) We apply SAGE on alerts from a penetration testing competition. The AGs are effective in attacker strategy comparison.

2 RELATED WORK

Generating succinct and realistic attack graphs (AG) has long been an open problem. Kaynar *et al.* [12] proposed a taxonomy of the existing expert-driven AG generation approaches specifically in the network security domain. These approaches generally utilize a knowledge base [9, 18, 20], making them unsuitable for zero-day vulnerabilities. Only recently has machine learning being used to automate parts of the AG generation pipeline. They can be classified in two main categories: i) *Process mining* has been used to visualize alert datasets [3, 4] without actually extracting attack graphs. Process mining uses alert signatures as identifiers, so it cannot be used for modeling contextually different attack stages having the same alert signature. ii) *Markov-based* methods such as, Markov chains have been used to build alert correlation systems [6, 15], and Hidden Markov Models have been used to build alert forecasting systems [8]. Specifically, Moskal *et al.* [15] use Markov chains to construct sequences of attacker strategies from IDS alerts. In this paper, we build upon [15] and leverage the temporal and probabilistic dependence between alerts to generate alert-driven attack graphs. The probabilistic deterministic finite automaton (S-PDFA) that we use has more expressive power than Markov chains and is easier to interpret. Importantly, the states represent the context of alerts based on their past and future.

[2]CPTC: https://www.nationalcptc.org/

3 ALERT-DRIVEN ATTACK GRAPHS

SAGE (IntruSion alert-driven Attack Graph Extractor)[3] takes raw intrusion alerts as input, and transforms them into aggregated sequences that are used to learn a model summarizing attack paths in the data. AGs are extracted from this model on a per-victim, per-objective basis. Even with an imperfect IDS, this type of alert-driven attack graph shows the actual steps taken by the attackers. The resulting attack graphs are succinct, effective and interpretable.

The first step towards building attack graphs is to arrange intrusion alerts in sequences that characterize an attacker strategy. An IDS alert contains the attacker and victim IP addresses, the targeted service *tServ* derived from destination port[4], and the attack stage *mcat* derived from the existing Attack-Intent framework by Moskal *et al.* [14] (see appendix), based on MITRE ATT&CK [23].

Raw intrusion alerts are often noisy and contain duplicate alerts. Thus, cleaning and aggregating them is necessary. We aggregate alerts into groups, such that they likely belong to the same attacker action. In literature, such an aggregation is called a hyper-alert or an attack episode. We use the method in [15] to aggregate alerts into *attack episodes*, and assume that the episodes closely characterize attacker actions. For an attack stage *mcat*, an episode is defined as $\langle st, et, mcat, mServ \rangle$, where *st* and *et* are start/end times, and *mServ* is the most frequently targeted service during the episode. We create *episode sequences* (ES) for each attacker-victim combination, sorted with respect to time. We partition the ES whenever a low-severity episode follows a high-severity one, signaling the start of a new attack attempt (see *mcat* $\rightarrow$ severity mapping in appendix).

The resulting *episode sub-sequences* (ESS) are used to learn a *suffix-based probabilistic deterministic finite automaton* (S-PDFA). An S-PDFA is the suffix-variant of the probabilistic deterministic finite automaton (see [25] for definitions). The high-severity *mcat*s are at the end of episode sub-sequences, *i.e.,* in the future. This is the main reason for using a suffix-based model – instead of predicting the future, it can be used to predict the past, *i.e.,* which episodes eventually lead to a high-severity attack stage. In order to find similarities in different attacker strategies, the entire alert dataset, *i.e.,* CPTC-2018, is converted into ESS and provided to the Flexfringe automaton learning framework [26]. Flexfringe learns from univariate *symbol* sequences comprised of $\langle mcat, mServ \rangle$. The

[3]We release SAGE: https://github.com/tudelft-cda-lab/SAGE
[4]Derived from open-source Port→Service mapping from IANA.

resulting S-PDFA (see appendix) clusters paths based on their temporal and behavioral similarity. Additionally, it brings infrequent high-severity actions into the spotlight, without discarding low-severity ones. The latter is tricky because while most clustering approaches discard infrequent patterns, it can to some extent be achieved by setting parameters in Flexfringe.

The states in S-PDFA can be considered milestones achieved by attackers, providing contextual meaning to the episodes' attack stages. Prior work by Lin *et al.* [13] has utilized this context to cluster similar car-following behaviors. We follow the same idea and convert episode sequences into state sequences (ESQ): we run every ESS through the S-PDFA and record the state identifiers (sID), resulting in its corresponding ESQ. Finally, the ESQs are converted into a graph (AG) using Graphviz, see Figure 2. These graphs are generated on a per-victim (vic) and per-objective basis (obj). The high-severity attack stages from [14], together with the targeted services form the objectives, since they specify end-goals and usually appear as the last actions in ESS. For an AG with the root vertex $\langle obj, vic \rangle$, only the ESQs concerning the victim vic and containing an episode with obj are shown. If an objective is achieved multiple times in an ESQ, each attempt is shown as an individual path in the graph. All attackers that obtain obj are shown in one graph to aid strategy comparison.

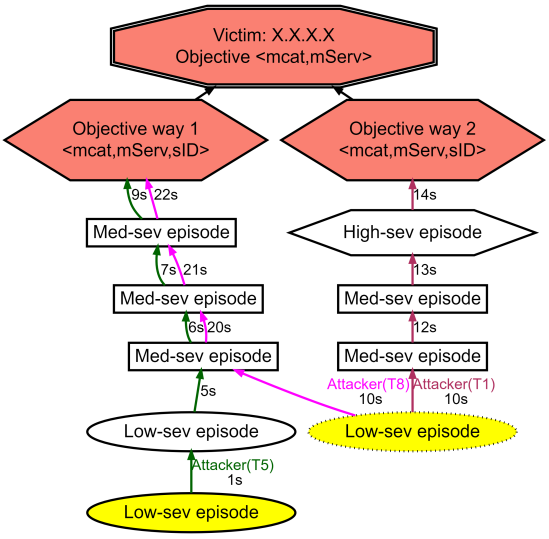

Figure 2: An alert-driven attack graph showing paths to obtain an objective. *Vertex labels* are $\langle mcat, mServ, sID \rangle$, *i.e.,* attack stage, targeted service and state identifier. Low-severity actions are *ovals*, medium-severity are *boxes*, high-severity are *hexagons*. The first state in a path is colored *yellow*, while the objective is *red*. States that occur too infrequently for Flexfringe are *dotted*. Edge label shows time since first alert. Edge color shows team: T1 (*Red*), T5 (*Green*), T8 (*Pink*).

4 EXPERIMENTAL SETUP

Dataset. In this paper, we generate attack graphs for the 2018 Collegiate Penetration Testing Competition (CPTC) dataset. It contains Suricata[5] alerts generated by different student teams tasked with compromising a fictitious network, *i.e.,* an automotive company. Each team has access to fixed-IP machines. Beyond the attackers' IP information, no ground truth is available regarding the attacker strategies and attack progression. Six teams (*i.e.,* T1, T2, T5, T7, T8, T9) produce 330,270 alerts. The competition lasted 9 hours.

Parameter selection. SAGE has five parameters: we set $t = 1.0$ sec to discard repeated alerts [15], and window length $w = 150$ sec to aggregate alerts into episodes. We set three Flexfringe parameters: *symbol_count*, *state_count* and *sink_count*, all set to 5. The experiments are executed on Intel Xeon W-2123 quad-core processor, 32 GB RAM.

S-PDFA model quality. Quantifying S-PDFA model quality is a difficult problem [5, 19]. A common option is to measure its prediction power using *Perplexity* [2, 25]. The S-PDFA learned for SAGE achieves low perplexity showing its effectiveness in finding patterns in traces, see appendix.

Table 1: Workload reduction in the CPTC-2018 dataset.

	# alerts (raw)	# alerts (filtered)	#episodes	#ES/ #ESQ	#ESS	#AGs
T1	81373	26651	655	103	108	53
T2	42474	4922	609	86	92	7
T5	52550	11918	622	69	74	51
T7	47101	8517	576	63	73	23
T8	55170	9037	439	67	79	33
T9	51602	10081	1042	69	110	30

5 RESULTS AND DISCUSSION

SAGE generates 93 attack graphs (AG) from the CPTC-2018 dataset. They contain 70 contextual objectives obtained by targeting 19 victim hosts. Instead of analyzing 330,270 alerts, the analysts only have to analyze 93 AGs, which is a significant workload reduction. Table 1 shows this reduction on a per-team basis. Note that multiple teams can share one AG if they all obtain that objective. Furthermore, the AGs are succinct and effective in highlighting strategic differences between attackers. We evaluate the complexity of the AGs using the model simplicity metric given in [4], *i.e.,* $Simplicity(AG) = \frac{|V|}{|E|}$, where $|V|$ and $|E|$ are the number of vertices and edges, respectively. The AGs have 21.7 vertices on average, which is significantly lower than existing methods [3, 4].

1. AGs show attack pathways: The AGs provide insight into the individual pathways. Figure 3 shows that three teams (T1, T5, T8) use remoteware-cl to exfiltrate data from 10.0.0.20 (absence of other teams means they were unable to obtain this objective). The teams self-reported that they had found a chatting application on this host that contained credentials, which were exfiltrated using a combination of privilege escalation and arbitrary code execution. Team 1 finds two distinct paths to complete this objective, first after 5.8 hours and then again after 7.5 hours since the start of the competition. Team 5 also finds two paths, but significantly earlier in the competition. The S-PDFA model identifies three distinct exfiltration states because it finds different ways to reach the same

[5] A well-known IDS: https://suricata-ids.org/

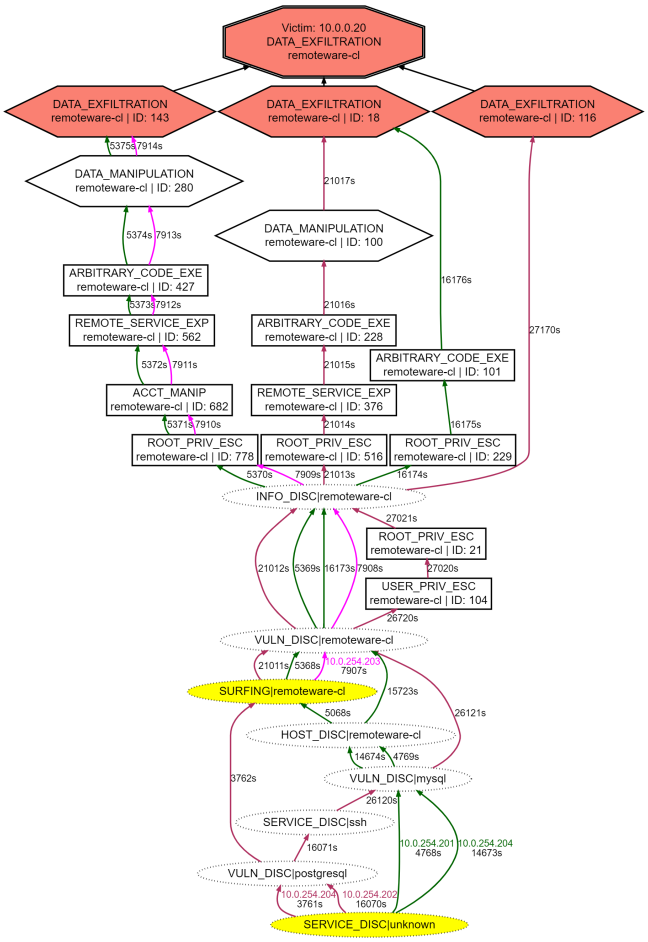

Figure 3: Data exfiltration/remoteware by T1, T5 and T8.

objective, *e.g.,* state ⟨`data_exfiltration, remoteware-cl, 116`⟩ can be reached with much fewer steps compared to the others. This also happens much later in the competition, implicitly capturing attackers' increasing experience.

2. AGs show strategic differences: Interestingly, an AG of data manipulation (Figure 4) over the same victim and service (from Figure 3) are partial subsets of each other, due to overlap in paths that attain both objectives. There are three variants of data manipulation, of which two are also present in the exfiltration AG (*i.e.,* ⟨`data_manipulation, remoteware-cl, 100`⟩ and ⟨`data_manipulation, remoteware-cl, 280`⟩. Team 5 finds an additional path to reach ⟨`data_manipulation, remoteware-cl, 19`⟩ after it has reached the objective ⟨`data_exfiltration, remoteware-cl, 18`⟩ from the previous AG. This actionable intelligence can be used to disrupt the cyber kill-chain [10]. Additionally, the graph shows strategic differences between the attackers, *e.g.,* T5 and T8 perform account manipulation while T1 does not; and that resource hijacking is a step in one of the T5's paths but not in the other. It also shows that T1 has found the shortest path to perform data manipulation on the victim using remoteware-cl.

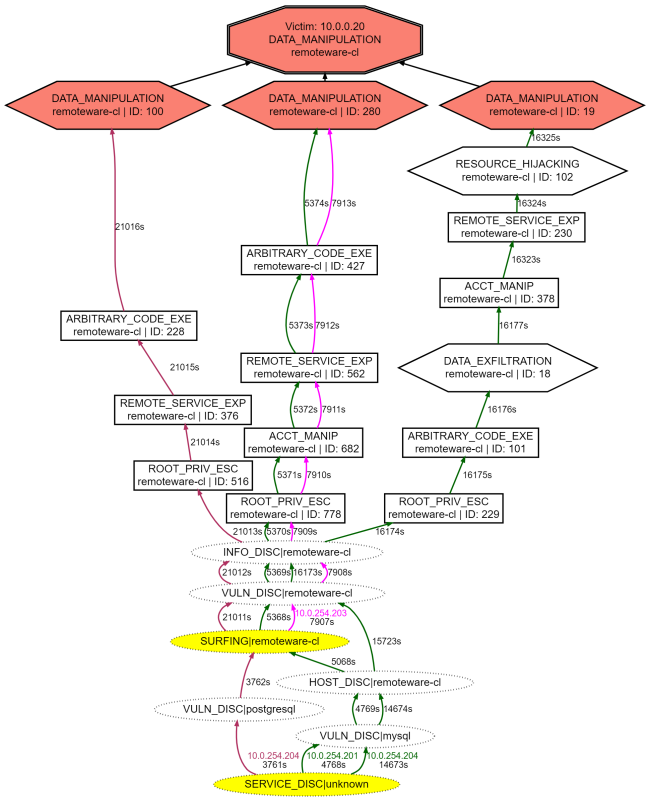

Figure 4: Data manipulation/remoteware is a subset of Fig 3.

3. AGs allow attacker performance evaluation: Each vertex in the attack graphs signifies a new milestone achieved by the teams. We argue that the fraction of unique milestones, *i.e.,* $\langle mcat, mServ, sID \rangle$ discovered by a team provides a metric for its performance. A medium-severity attack stage forms a stepping-stone towards a high-severity attack stage. Hence, high-severity vertices are twice as important as medium-severity vertices, *i.e.,* $\frac{(2*sev)+(1*med)}{3}$, where *sev* and *med* are the number of high- and medium-severity milestones discovered by a team, respectively. Table 2 shows the evaluation of all six teams based on the 93 AGs, ranked according to their score. It shows the number of unique high- and medium-severity vertices discovered by the teams during the competition. Team 5 scores the highest points, while Team 2 scores the lowest points. Team 1 comes in second, solely because they discover the highest number of medium-severity vertices compared to any other team. Overall, this table provides an alternate way to automatically rank teams.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose SAGE, an unsupervised sequence learning tool that generates succinct attack graphs (AG) directly from raw intrusion alerts, without a priori knowledge. SAGE models the temporal and probabilistic dependence between alerts in a suffix-based probabilistic deterministic finite automaton (S-PDFA). The S-PDFA brings infrequent severe alerts into spotlight and summarizes paths leading to them. AGs are then extracted from the S-PDFA.

Table 2: Attacker evaluation using the fraction of unique vertices discovered during CPTC-2018 (*Rank = Score*).

Team	Severe vertices (out of 70)	Medium vertices (out of 148)	Weighted average percentage
T5	28 (40%)	40 (27%)	35.67
T1	18 (26%)	62 (42%)	31.33
T9	23 (33%)	36 (24%)	30.0
T7	22 (31%)	26 (18%)	26.67
T8	15 (21%)	32 (22%)	21.33
T2	3 (4%)	8 (5%)	4.33

SAGE is used to generate AGs from 330k alerts collected through Collegiate Penetration Testing Competition with six attacker teams. The resulting AGs provide a clear picture of the attack progression, and show strategic differences across various attackers. Specifically, they show that attackers often follow shorter paths after they have discovered a longer one.

Future work will focus on evaluating alert-driven attack graphs with real analysts, and leveraging readily-available domain knowledge to map the vertices to high-level attacker actions.

REFERENCES

[1] Faeiz M Alserhani. 2016. Alert correlation and aggregation techniques for reduction of security alerts and detection of multistage attack. *International Journal of Advanced Studies in Computers, Science and Engineering* 5, 2 (2016), 1.

[2] Borja Balle, Rémi Eyraud, Franco M Luque, Ariadna Quattoni, and Sicco Verwer. 2017. Results of the sequence prediction challenge (spice): a competition on learning the next symbol in a sequence. In *International Conference on Grammatical Inference*. 132–136.

[3] Yuzhong Chen, Zhenyu Liu, Yulin Liu, and Chen Dong. 2020. Distributed Attack Modeling Approach Based on Process Mining and Graph Segmentation. *Entropy* 22, 9 (2020), 1026.

[4] Sean Carlisto De Alvarenga, Sylvio Barbon Jr, Rodrigo Sanches Miani, Michel Cukier, and Bruno Bogaz Zarpelão. 2018. Process mining and hierarchical clustering to help intrusion alert visualization. *Computers & Security* 73 (2018), 474–491.

[5] Colin De la Higuera. 2010. *Grammatical inference: learning automata and grammars*. Cambridge University Press.

[6] Ouissem Ben Fredj. 2015. A realistic graph-based alert correlation system. *Security and Communication Networks* 8, 15 (2015), 2477–2493.

[7] Ni Gao, Yiyue He, and Beilei Ling. 2018. Exploring attack graphs for security risk assessment: a probabilistic approach. *Wuhan University Journal of Natural Sciences* 23, 2 (2018), 171–177.

[8] Ibrahim Ghafir, Konstantinos G Kyriakopoulos, Sangarapillai Lambotharan, Francisco J Aparicio-Navarro, Basil AsSadhan, Hamad BinSalleeh, and Diab M Diab. 2019. Hidden Markov models and alert correlations for the prediction of advanced persistent threats. *IEEE Access* 7 (2019), 99508–99520.

[9] Hao Hu, Jing Liu, Yuchen Zhang, Yuling Liu, Xiaoyu Xu, and Jinglei Huang. 2020. Attack scenario reconstruction approach using attack graph and alert data mining. *Journal of Information Security and Applications* 54 (2020), 102522.

[10] Eric M Hutchins, Michael J Cloppert, Rohan M Amin, et al. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research* 1, 1 (2011), 80.

[11] Somesh Jha, Oleg Sheyner, and Jeannette Wing. 2002. Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. IEEE, 49–63.

[12] Kerem Kaynar. 2016. A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications* 29 (2016), 27–56.

[13] Qin Lin, Yihuan Zhang, Sicco Verwer, and Jun Wang. 2018. MOHA: A multi-mode hybrid automaton model for learning car-following behaviors. *IEEE Transactions on Intelligent Transportation Systems* 20, 2 (2018), 790–796.

[14] Stephen Moskal and Shanchieh Jay Yang. 2020. Framework to Describe Intentions of a Cyber Attack Action. *arXiv preprint arXiv:2002.07838* (2020).

[15] Stephen Moskal, Shanchieh Jay Yang, and Michael E Kuhl. 2018. Extracting and Evaluating Similar and Unique Cyber Attack Strategies from Intrusion Alerts. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 49–54.

[16] Peng Ning, Yun Cui, and Douglas S Reeves. 2002. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 245–254.

[17] Peng Ning, Dingbang Xu, Christopher G Healey, and Robert St Amant. 2004. Building Attack Scenarios through Integration of Complementary Alert Correlation Method.. In *NDSS*, Vol. 4. 97–111.

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

[19] Rajesh Parekh and Vasant Honavar. 2001. Learning DFA from simple examples. *Machine Learning* 44, 1-2 (2001), 9–35.

[20] Sebastian Roschke, Feng Cheng, and Christoph Meinel. 2011. A new alert correlation algorithm based on attack graph. In *Computational intelligence in security for information systems*. Springer, 58–67.

[21] Reza Sadoddin and Ali Ghorbani. 2006. Alert correlation survey: framework and techniques. In *Proceedings of the 2006 international conference on privacy, security and trust: bridge the gap between PST technologies and business services*. 1–10.

[22] Saeed Salah, Gabriel Maciá-Fernández, and JesúS E DíAz-Verdejo. 2013. A model-based survey of alert correlation techniques. *Computer Networks* 57, 5 (2013), 1289–1317.

[23] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. 2018. Mitre att&ck: Design and philosophy. *Technical report* (2018).

[24] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A Kemmerer. 2004. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on dependable and secure computing* 1, 3 (2004), 146–169.

[25] Sicco Verwer, Rémi Eyraud, and Colin De La Higuera. 2014. PAutomaC: a probabilistic automata and hidden Markov models learning competition. *Machine learning* 96, 1-2 (2014), 129–154.

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

A APPENDIX

A.1 Attack stages

Table 3: Attack stages and their severity.

Acronym	Micro attack stage	Severity
SURFING	Surfing	Low
HOST_DISC	Host Discovery	Low
SERVICE_DISC	Service Discovery	Low
VULN_DISC	Vulnerability Discovery	Low
INFO_DISC	Information Discovery	Low
USER_PRIV_ESC	User Privilege Escalation	Med
ROOT_PRIV_ESC	Root Privilege escalation	Med
BRUTE_FORCE_CREDS	Brute force Credentials	Med
ACCT_MANIP	Account Manipulation	Med
PUBLIC_APP_EXP	Public Application Exploitation	Med
REMOTE_SERVICE_EXP	Remote Service Exploitation	Med
COMMAND_AND_CONTROL	Command and Control	Med
LATERAL_MOVEMENT	Lateral movement	Med
ARBITRARY_CODE_EXE	Arbitrary code execution	Med
PRIV_ESC	Privilege escalation	Med
NETWORK_DOS	Network Denial of Service	High
RESOURCE_HIJACKING	Resource hijacking	High
DATA_MANIPULATION	Data manipulation	High
DATA_EXFILTRATION	Data exfiltration	High
DATA_DELIVERY	Data delivery	High
DATA_DESTRUCTION	Data destruction	High

A.2 Example Episode Sequences

a1→v2: `[<0,150,INFO_DISC,http>, <900,1200,INFO_DISC, http>, <901,1200,C&C,http>, <2250,2550, INFO_DISC,http>, <2251,2550,DATA_EXFILTRATION,http>]`

a3→v11: [<3497,3797,SURFING,http>, <3498,3647,VULN_DISC, http>, <3499,3797,INFO_DISC,http>, <3500,3647,ROOT_PRIV_ESC, http>, <3501,3647,ACCT_MANIP,http>, <3502,3647,REMOTE_SER-

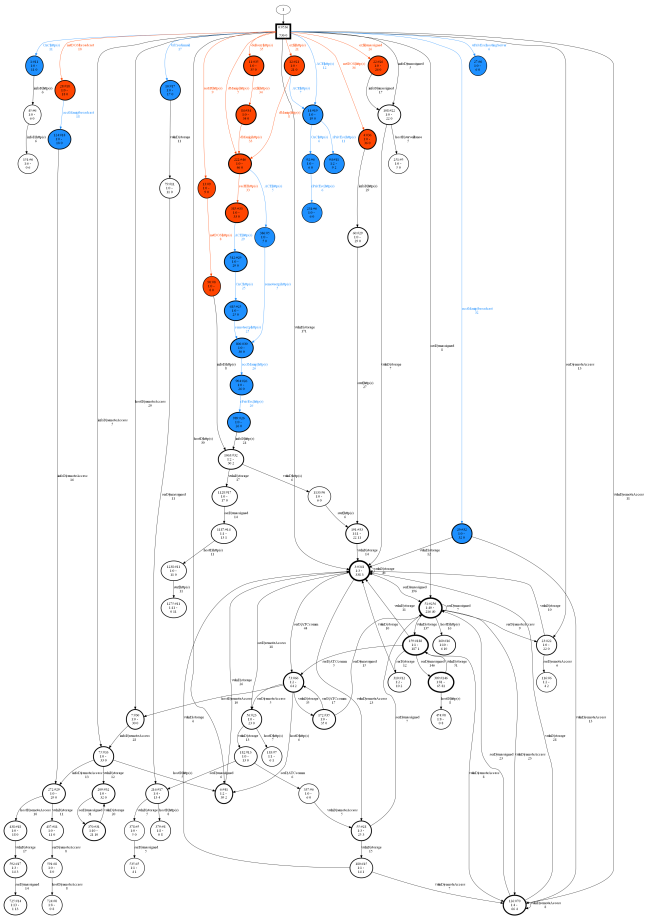

Figure 5: CPTC-2018 S-PDFA model for 6 teams. Color denotes severity: red= high, blue= medium, white= low.

VICE_EXP,http>, <3503,3647,C&C,http>, <3504,3647,ACE,http>, <3505,3797,NETWORK_DOS,http>, <3506,3647, RESOURCE_HIJACKING,http>, <3507,3647,DATA_MANIPULATION,http>, <3508,3647, DATA_EXFILTRATION,http>, <3509,3647,DATA_DELIVERY, http>, <3797,4097,HOST_DISC,http>, <3798,4097,SERVICE_DISC,ssh>, <3799,4097,VULN_DISC,mysql>, <4097, 4397,INFO_DISC,http>, <4098,4397,NETWORK_DOS,http>, <4099,4397, RESOURCE_HIJACKING, http>, <4697,4847, INFO_DISC, http>, <4698,4847,ROOT_PRIV_ESC, http>]
a5→v2: [<27379,27529,INFO_DISC,http>, <27380,27529, C&C, http>, <29479,29629, INFO_DISC, http>, <29480,29629 , C&C,http>]

A.3 S-PDFA for CPTC-2018

Figure 5 shows the S-PDFA learned for the entire CPTC-2018 dataset.

A.4 S-PDFA model quality

Typically, model quality is quantified using a model selection criterion, measuring a trade-off between model size and fit. Perplexity 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 the training data and an unseen test set using an 80-20 split and compare the perplexity values against two models: (a) Suffix tree and (b) Markov chains. Table 4 shows the perplexity for each variant. It shows, as expected, that a suffix tree provides the best fit with the training data. The S-PDFA is about twice as *perplexed*. On the test data, the S-PDFA gives the best perplexity value, demonstrating that the model accurately captures many of the patterns present in the data that are missed by the other models.

Table 4: Perplexity of suffix models. (Bold = Best value)

	Suffix tree	**Markov chains**	**S-PDFA**
Training data	***1265.4***	13659.6	2397.8
Holdout test set	13020.7	11617.8	***9884.6***