

**Understanding Adversary Behavior via XAI
Leveraging Sequence Clustering To Extract Threat Intelligence**

Nadeem, A.

DOI

[10.4233/uuid:4a1519ba-3542-4d8f-ab91-2342e8f5bb1a](https://doi.org/10.4233/uuid:4a1519ba-3542-4d8f-ab91-2342e8f5bb1a)

Publication date

2024

Document Version

Final published version

Citation (APA)

Nadeem, A. (2024). *Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering To Extract Threat Intelligence*. [Dissertation (TU Delft), Delft University of Technology].
<https://doi.org/10.4233/uuid:4a1519ba-3542-4d8f-ab91-2342e8f5bb1a>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

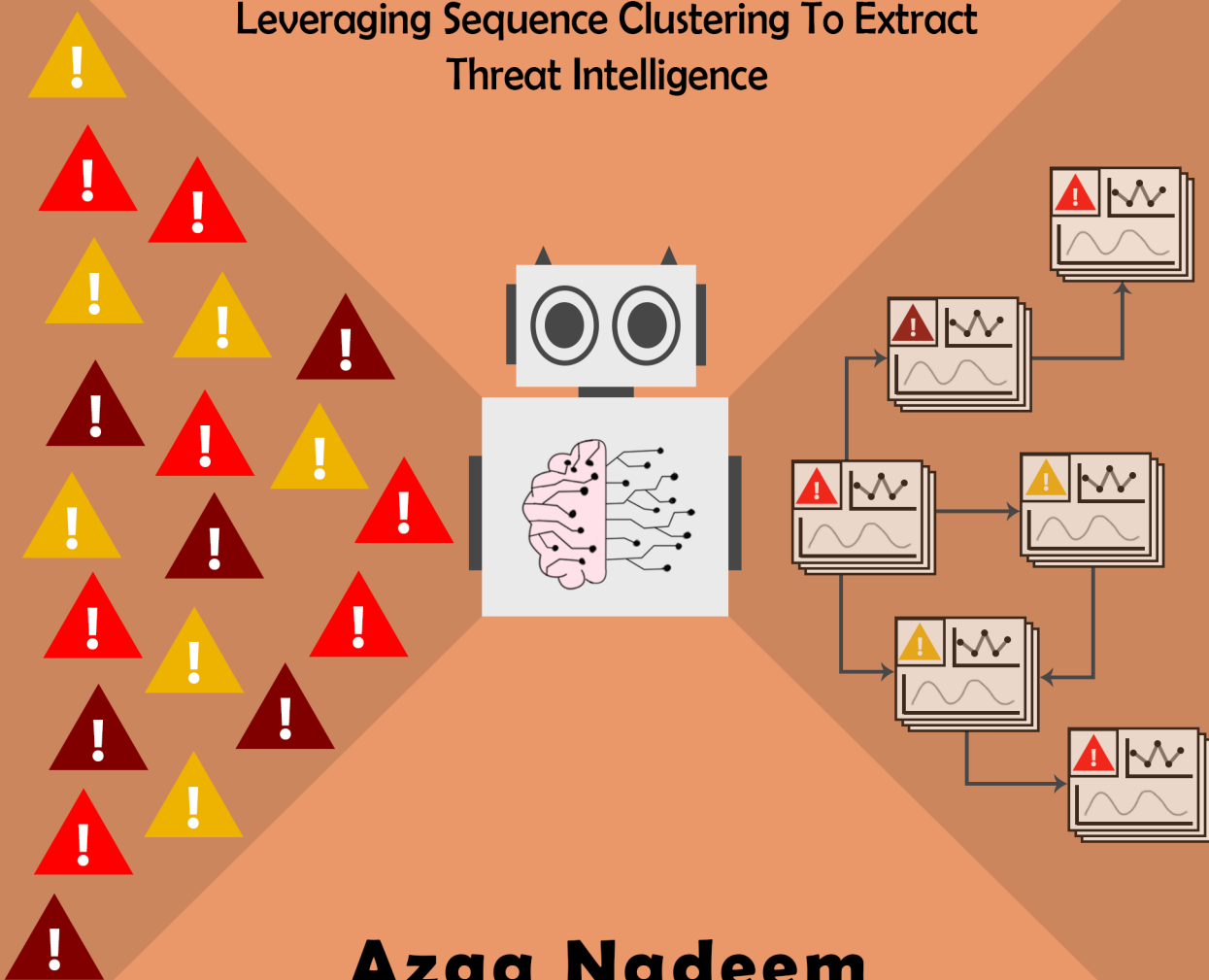
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

Understanding Adversary Behavior via XAI

Leveraging Sequence Clustering To Extract
Threat Intelligence



Azqa Nadeem

UNDERSTANDING ADVERSARY BEHAVIOR VIA XAI

LEVERAGING SEQUENCE CLUSTERING TO EXTRACT
THREAT INTELLIGENCE

UNDERSTANDING ADVERSARY BEHAVIOR VIA XAI

LEVERAGING SEQUENCE CLUSTERING TO EXTRACT
THREAT INTELLIGENCE

Dissertation

for the purpose obtaining the degree of doctor
at Delft University of Technology
by the authority of Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Tuesday 2 April 2024 at 15.00 o'clock

by

Azqa NADEEM

Master of Science in Computer Science,
Delft University of Technology, Delft, The Netherlands,
born in Quetta, Pakistan.

This dissertation has been approved by the promoters.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof.dr.ir. R.L. Lagendijk,	Delft University of Technology, promotor
Dr.ir. S.E. Verwer,	Delft University of Technology, promotor

Independent members:

Prof.dr. D. Balzarotti,	Eurecom, France
Prof.dr. S.J. Yang,	Rochester Institute of Technology, U.S.A.
Prof.dr. M.I.A. Stoelinga,	University of Twente & Radboud University, The Netherlands
Prof.dr.ir. R.E. Kooij,	Delft University of Technology & TNO, The Netherlands
Dr. M.A. Migut,	Delft University of Technology, The Netherlands
Prof.dr. M.M. de Weerdt,	Delft University of Technology, <i>reserve member</i>



SIKS Dissertation Series No. 2024-06.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords: Cybersecurity, Explainable machine learning, Behavior modeling

Printed by: IPSKAMP printing

Cover design: Raima Nadeem

Copyright © 2024 by A. Nadeem

ISBN 978-94-6366-828-6

An electronic version of this dissertation is available at
<http://repository.tudelft.nl/>.

A sign of intelligence is an awareness of one's own ignorance.

Niccolo Machiavelli

CONTENTS

Summary	xi
Samenvatting	xiii
1 Introduction	1
1.1 Understanding Adversary Behavior	2
1.1.1 Expert Knowledge Based Manual Approaches	3
1.1.2 Machine Learning Based Data-driven Approaches	4
1.1.3 Examples: Manual vs. Data-driven Approaches	5
1.2 Challenges of Learning Adversary Behavior	7
1.3 Explainable Sequential Machine Learning	9
1.4 Problem Statement	12
1.4.1 Attacker Strategy Discovery	12
1.4.2 Network Attack Characterization.	13
1.5 Contributions of this Thesis.	14
1.5.1 Our Approach To Learning Adversary Behavior	15
1.5.2 Thesis Outline	16
1.5.3 List of Excluded Publications.	18
References	19
2 A Taxonomy of Explainable Machine Learning in Cybersecurity	25
2.1 Introduction	26
2.2 Background and Methodology	28
2.3 Systematization	29
2.3.1 XAI-enabled User Assistance.	32
2.3.2 XAI-enabled Model Verification	38
2.3.3 Explanation Verification & Robustness.	40
2.3.4 Offensive Use of Explanations	42
2.4 Use Case: Debugging a Malicious Network Traffic Detector via XAI	44
2.4.1 XAI for Discovering Spurious Correlations	45
2.4.2 XAI for Finding Causes of Misclassifications	47
2.4.3 Utility of Different XAI Types.	47
2.5 Discussion and Open Problems	48
2.6 Conclusions.	49
2.7 Supplementary Material	50
2.7.1 Selected Literature	50
2.7.2 XAI Pipeline Design	50
2.7.3 LEMNA Implementation.	53
2.7.4 Explanations from the Tutorial.	53

References	53
I Interpretable Sequential Learning for Attacker Strategy Discovery	65
3 Alert-driven Attack Graph Generation	67
3.1 Introduction	68
3.2 Related Work	70
3.3 Intrusion Alert-driven Attack Graph Extractor – SAGE.	71
3.3.1 Attack Episode Aggregation	71
3.3.2 Suffix-based Probabilistic Deterministic Finite Automaton	74
3.3.3 Alert-driven Attack Graphs.	79
3.4 Explainability Analysis of SAGE	80
3.5 Dataset and Experimental Setup	81
3.6 Results and Discussion	82
3.6.1 S-PDFA Model Quality Evaluation	82
3.6.2 Comparing Infrastructural Differences Across Datasets	83
3.6.3 Alert Triaging via Alert-driven Attack Graphs.	85
3.7 Limitations and Future Work	88
3.8 Conclusions.	89
3.9 Supplementary Material	89
3.9.1 Attack Stage Definition.	89
3.9.2 Targeted Service Mapping	89
3.9.3 S-PDFA Model for CCDC-2018.	89
References	89
4 Enabling Visual Analytics via Alert-driven Attack Graphs	95
4.1 Motivation & Related Work	96
4.2 Methodology	98
4.2.1 Alert-driven Attack Graphs.	98
4.2.2 Critical Path Exploration Dashboard for Alert-driven Attack Graphs .	100
4.3 Dataset and Experimental Setup	103
4.4 Results and Discussion	105
4.4.1 Explaining Attacker Strategies in CPTC-2018.	105
4.4.2 Use Case A: Applying SAGE on an Unexplored Blue Team Exercise. .	111
4.4.3 Use Case B: Attack Investigation via the Dashboard	114
4.5 Practical Implications for Cyber Threat Intelligence.	114
4.6 Limitations and Future Work	115
4.7 Conclusions.	115
References	116
II Explainable Sequential Learning for Network Attack Analysis	119
5 Beyond Labels: Automated Behavior Profiling of Malware	121
5.1 Introduction	122
5.2 Survey of ML-based Malware Defense	125
5.2.1 Challenges in Malware Labeling	125
5.2.2 Research Objectives: Detection vs. Analysis	126

5.2.3	Challenges in Malware Behavior Modeling	126
5.3	Malware Packet Sequence Clustering and Analysis – MalPaCA	128
5.4	Explainability Analysis of MalPaCA	134
5.5	Dataset and Experimental Setup	135
5.6	Results and Discussion	136
5.6.1	Malware Capability Assessment via <i>Behavioral Profiles</i>	136
5.6.2	Comparative Analysis with Existing Methods	139
5.7	Limitations and Future Work	143
5.8	Conclusions.	143
	References	143
6	Real-time Sequence Clustering for Network Attack Summarization	149
6.1	Introduction	150
6.2	Preliminaries	152
6.3	Sequence Clustering in Evolving Data Streams – SECLEDS	152
6.3.1	Stable Cluster Definition via Multiple Medoids.	153
6.3.2	Center of Mass Estimation	154
6.3.3	The SECLEDS Algorithm	154
6.3.4	Real-time Cluster Explanations	156
6.4	Dataset and Experimental Setup	158
6.5	Empirical Results	160
6.6	Use Case A: Intelligent Network Attack Summarization via SECLEDS	163
6.6.1	Clustering Performance	164
6.6.2	Network Bandwidth Support	165
6.7	Use Case B: Malware Behavioral Profiling via SECLEDS.	165
6.7.1	Behavioral Profiling via Random Sampling.	168
6.7.2	Behavioral Profiling via SECLEDS Sampling	168
6.7.3	Discussion	170
6.8	Conclusions.	170
6.9	Supplementary Material	171
6.9.1	Dataset Generation Process	171
6.9.2	Cluster Initialization Quality	172
6.9.3	Network Traffic Sampling with SECLEDS.	172
6.9.4	Clustering Handwritten Character Strokes	173
6.9.5	Cluster Explanations for Multivariate Network Traffic	174
	References	174
7	Conclusions, Outlook, and Societal Relevance	179
7.1	Addressing the Challenges of Cyber Data	181
7.1.1	S-PDFA for Infrequent Data	181
7.1.2	SECLEDS for Evolving Data	182
7.1.3	Societal Relevance	183
7.2	Explainability in Cybersecurity	184
7.2.1	Limitations.	185
7.2.2	Societal Relevance and Future Directions	185

7.3	Attacker Strategy Discovery	186
7.3.1	Alert-driven Attack Graphs	186
7.3.2	Behavioral Analytics by Alert-driven Attack Graphs	187
7.3.3	Follow-up Efforts.	188
7.3.4	Limitations and Future Directions	189
7.3.5	Societal Relevance	190
7.4	Network Attack Analysis	191
7.4.1	Malware Capability Assessment	191
7.4.2	Network Attack Summarization	192
7.4.3	Limitations and Future Directions	193
7.4.4	Societal Relevance	193
7.5	Final Words on Adversary Behavior Analysis	194
	References	195
	Acknowledgements	197
	Curriculum Vitæ	201
	List of Publications	203
	SIKS Dissertation Series	205

SUMMARY

Understanding the behavior of cyber adversaries provides threat intelligence to security practitioners, and improves the cyber readiness of an organization. With the rapidly evolving threat landscape, data-driven solutions are becoming essential for automatically extracting behavioral patterns from data that are otherwise too time-consuming to discover manually. This dissertation advocates the use of machine learning (ML) to obtain insights into adversary behavior for creating *AI-assisted practitioners*. However, developing adversary behavior models is challenging since cyber data is often unlabeled, noisy, infrequent, and contains intricate patterns that evolve over time. We demonstrate that sequential features are effective at addressing these challenges. Yet, they have limited interpretability and algorithmic support.

This dissertation starts by defining the notion of explainability as it is currently used within cybersecurity by systematizing available literature in Chapter 2. We find that the literature frequently relies on black-box models that use off-the-shelf explanation methods without considering the explanation stakeholders. In contrast, literature on sequence learning models that are interpretable by design is severely limited.

We address these challenges by developing special algorithms that learn *sequential patterns* from *infrequent events*, and *evolving data* in an *unsupervised setting*. We utilize these algorithms to create *interpretable tool-chains* for understanding the behavior of various types of adversaries. We show that it is possible to learn interpretable models (even for complex sequential data in an unsupervised setting) that provide more insights than just prediction probabilities, while achieving competitive performance. In doing so, we encourage the security community to look beyond accuracy scores, and focus on extracting actionable insights from ML models. We make our tool-chains open-source.

The first part of this thesis models the strategies employed by *human threat actors*. Chapters 3 and 4 develop a novel paradigm of attack graphs (AG) that are learned directly from intrusion alerts for capturing attacker strategies. The attacker strategies are learned using our S-PDFA model, which is interpretable, fast, and effective. We learn alert-driven AGs from 3 open-source datasets, and show their ability to compress over 1.4 million alerts in 401 AGs in under 5 minutes. The AGs provide actionable intelligence regarding strategic differences and fingerprintable paths. They also reduce analyst alert fatigue by triaging critical attacks.

The second part of this thesis models the capabilities exhibited by *automated threat actors (malware)*. Chapters 5 and 6 develop an explainable sequence clustering tool-chain to automatically characterize the network behavior of malware samples. We use this tool-chain to create behavioral profiles of 1196 real-world malware samples for explaining their capabilities. We also develop a streaming sequence clustering algorithm for real-time behavior profiling, which is evaluated on 5 datasets and against 4 clustering algorithms. By automatically creating behavioral profiles of bot-infected hosts in real-time, we distinguish between benign and malicious hosts with 100% accuracy.

SAMENVATTING

Het gedrag begrijpen van cyberaanvallers levert dreigingsinformatie op aan veiligheidsprofessionals en verhoogt de cybergereedheid van een organisatie. Door het snel evoluerende dreigingslandschap worden data gebaseerde oplossingen steeds belangrijker om gedragspatronen automatisch te extraheren die anders handmatig een zeer tijdrovend taak zou zijn. Deze proefschrift bepleit voor het gebruik van machine learning (ML) om gedragsinzichten te verkrijgen van een cyberaanvaller, met het oog op het creëren van door AI-ondersteunde beoefenaars. Het ontwikkelen van gedragsmodellen voor een cyberaanvaller is echter een uitdaging, gezien dat cyberdata is meestal ongelabeld, ruizig, en ingewikkelde patronen bevatten die in de loop van de tijd evolueren. We laten zien dat sequentiële kenmerken effectief zijn bij het aanpakken van deze uitdagingen. Toch hebben ze een beperkte interpreteerbaarheid en algoritmische ondersteuning.

Dit proefschrift begint met het definitie van “verklaarbaarheid” zoals het nu wordt gebruikt binnen cybersecurity door beschikbare literatuur systematisch te analyseren in Hoofdstuk 2. We stellen vast dat de literatuur vaak gebaseerd is op black-box modellen die gebruik maken van kant-en-klare verklaringsmethoden zonder rekening te houden met de belanghebbenden van de verklaring. Daarentegen is de literatuur over sequentiële modellen die interpreteerbaar zijn door het ontwerp zeer beperkt.

Deze uitdagingen worden aangepakt door specifieke algoritmen te ontwikkelen die sequentiële patronen leren van zeldzame gebeurtenissen en evoluerende data in een onbeheerde omgeving. We maken gebruik van deze algoritmen om “interpreteerbare tool-chains” te creëren voor het begrijpen van het gedrag van verschillende soorten van cyberaanvallers. We tonen aan dat het mogelijk is om interpreteerbare modellen te leren (zelfs voor complexe sequentiële data in een onbeheerde omgeving) die meer inzicht opleveren dan alleen voorspellingswaarschijnlijkheden, terwijl er competitieve prestaties worden behaald. Hiermee moedigen we de security gemeenschap aan om verder te kijken dan nauwkeurigheidsscores van modellen en zich richten op het verkrijgen van een bruikbare inzichten uit ML-modellen. We maken onze tool-chains open-source.

Het eerste deel van dit proefschrift modelleert de strategieën die door menselijke bedreigingsactoren worden gehanteerd. Hoofdstukken 3 en 4 ontwikkelen een nieuw paradigma van “Attack Graphs” (AG), die rechtstreeks worden geleerd van inbraakwaarschuwingen om aanvalsstrategieën vast te leggen. De aanvalsstrategieën worden geleerd met behulp van ons S-PDFA model, dat interpreteerbaar, snel en effectief is. We leren ons alert-driven AGs van drie open-source datasets en tonen hun capaciteit om meer dan 1.4 miljoen waarschuwingen in 401 AGs te comprimeren in minder dan 5 minuten. The AGs bieden bruikbare intelligentie met betrekking tot strategische verschillen en identificeerbare trajecten. Ze verminderen de vermoeidheid van analisten door een triage te maken voor kritieke aanvallen.

Het tweede deel van dit proefschrift modelleert de capaciteiten vertoond door geautomatiseerde bedreigingsactoren (malware). Hoofdstukken 5 en 6 ontwikkelen aan

uitlegbare tool-chain van sequentieclustering die is om het netwerkgedrag van malwarevoorbeelden automatisch te karakteriseren. Met behulp van deze tool-chain creëren we gedragsprofielen van 1196 malwaremonsters uit de echte wereld die hun capaciteiten verklaren. We ontwikkelen ook een streaming sequentie clustering algoritme voor real-time gedragsprofilering, dat wordt geëvalueerd op basis van 5 datasets en tegen 4 clustering algoritmen. Door in real-time automatisch gedragsprofielen te creëren van botgeïnfecteerde hosts, onderscheiden we goedaardige en kwaadaardige hosts met 100% nauwkeurigheid.

1

INTRODUCTION

The perpetual arms-race between cyber adversaries and defenders has fueled the creation of sophisticated cyber attacks over the years. Astra Security estimates that in 2023, a cyber attack occurred every 39 seconds, and 300,000 malware samples were generated on a daily basis. Meanwhile, it took organizations 49 days to detect malware, on average [2]. Moreover, cyber attacks create large volumes of alerts that security analysts investigate manually. For instance, it is estimated that security analysts receive more than a million alerts each day [3]. It takes security teams an average of 130 hours per week to monitor threats, and more than 20 minutes of manual effort to detect, prioritize, and remediate a vulnerability [4].

Data-driven security solutions are becoming increasingly necessary for defending against modern cyber adversaries. These approaches operate on the intuition that attacker actions can be observed in the data generated by a targeted system, *e.g.*, network traffic generated by a victim host, and can thus be approximated using machine learning-enabled knowledge discovery [5]. Patterns related to attacker actions are characterized using either (a) *unsupervised learning* where groups in observable data are discovered based on similar attributes (*i.e.*, features), or (b) *supervised learning* where associations are found between features of observable data and the corresponding class label (*e.g.*, benign, malicious). Ultimately, the aim of the learning process is to find patterns that can be generalized to unseen data instances [5].

In practice, cybersecurity applications often utilize data-driven solutions for attack detection – the aim is to maximize attack detection while minimizing false alarms and detection times. This typically entails complex optimizations of feature combinations, making the resulting machine learning model a ‘black box’ [6]. This lack of transparency is a well-known reason for the slow deployment of machine learning in industry [7], [8]. Furthermore, the black-box nature of the model removes security practitioners from the learning loop, which makes it difficult for them to understand the model and intervene when it makes mistakes [9].

Parts of this chapter have been published as “*Learning About the Adversary*” by **Nadeem, A.**, Yang, S. J. & Verwer, S. in *Autonomous Intelligent Agents for Cyber Defense*, Springer, 2023 (pp. 105-132) [1].

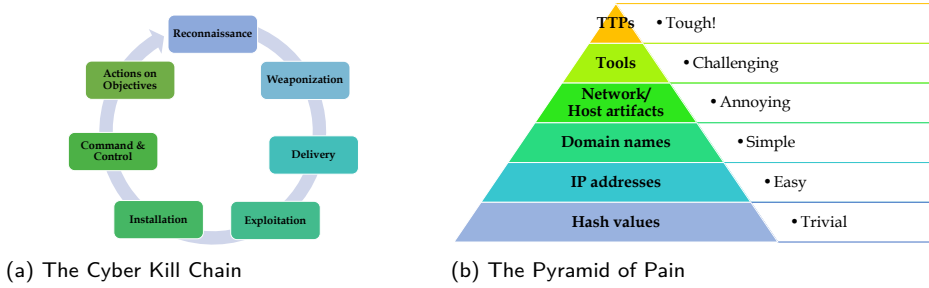


Figure 1.1: (a) The Cyber Kill Chain [11] categorizes a cyber-intrusion in seven phases, starting with scanning for reconnaissance and ending with actions on objectives. (b) The Pyramid of Pain [12] shows the difficulty of obtaining various Indicators of Compromise (IOCs).

Stepping away from the realm of attack detection, the analysis of cyber attacks offers valuable threat intelligence for the development of better detection tools, *e.g.*, investigating attack campaigns provides intelligence for better detection signatures. This, in turn, enhances the cyber readiness of organizations, enabling them to gracefully handle future cyber attacks. In practice, however, attack analysis and threat intelligence generation remain expert-driven manual tasks [10].

In this dissertation, we utilize machine learning to obtain more insights than just predication probabilities and accuracy scores – machine learning is used to extract intricate patterns regarding adversary behavior from observable data. This way, data-driven approaches create a potential human-in-the-loop setting for *AI-assisted practitioners*. The philosophy is to utilize unsupervised machine learning models as virtual autonomous agents that assist in analyzing large datasets, discovering patterns related to adversary behavior, and delivering actionable threat intelligence to security practitioners. They can then use this threat intelligence for downstream tasks, such as risk assessment, signature creation, and defense playbook creation.

In this chapter, we define the notion of data-driven attacker behavior analysis, discuss the challenges of developing unsupervised data-driven solutions for modeling adversary behavior, and broadly explain the proposed approach utilized in this dissertation. We also present an outline of this dissertation and specify our contributions.

1.1. UNDERSTANDING ADVERSARY BEHAVIOR

“Know thy enemy” is one of the first principles of warfare preached by *The Art of War* [13]. In practice, adversaries are normally quantified and classified using *threat assessment models*. These models theoretically define an adversary from different facets, which help security practitioners provide security guarantees for specific abuse cases. For instance, the Capability, Opportunity, Intent (COI) model characterizes an adversary based on its (a) “Capability” – an adversary’s capacity to undertake the task at hand; (b) “Opportunity” – the presence of an operating environment, and (c) “Intent” – the brain processes that make the adversary act upon the task [14]–[16]. Understanding the capabilities of an adversary improves situational awareness and cyber readiness – it provides threat in-

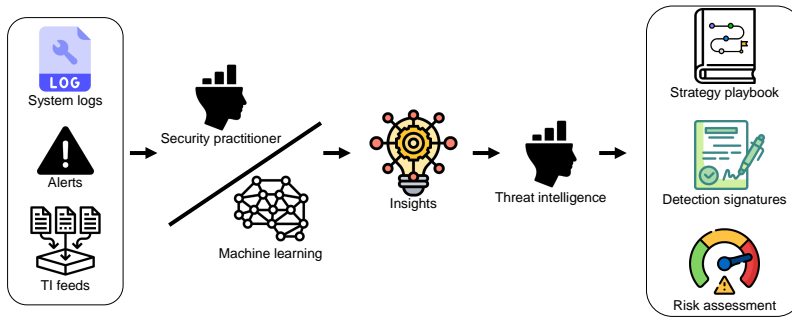


Figure 1.2: Contributions of this thesis: We replace the manual and monotonous labor of analyzing large volumes of data with machine learning for the generation of threat intelligence. The threat intelligence can be used to analyze attacker behavior, create/update training playbooks, and detection signatures.

telligence required for targeted countermeasures. For instance, the intelligence can be used for creating signatures for classification/detection, for assessing the quality of current defenses (risk assessment), and for constructing playbooks of adversary strategies that can be utilized for training incident response staff.

Adversary behavior is often described in terms of Tactics, Techniques, and Procedures (TTP). TTPs are usually an expression of an adversary's training/education, and are thus extremely difficult to alter, once detected. TTPs relate to an adversary's *capability* of employing a strategy to obtain their objectives. A vulnerability in the target system presents an *opportunity* for the adversary, while the adversary's *intent* is often implicitly inferred through their actions. Lockheed Martin's Cyber Kill Chain [11] and MITRE's ATT&CK [17] are two of the most popular frameworks to study the structure of a cyber attack in terms of tactics and techniques. The Cyber Kill Chain, shown in Figure 1.1a, models the attack process as a sequential chain of seven steps that an adversary must complete in order to obtain their objective, while ATT&CK is a comprehensive knowledge-base enumerating the TTPs of cyber adversaries.

Threat assessment models are utilized for understanding adversary behavior in two ways: manual approaches based on expert knowledge, and automated approaches that are data-driven. Expert knowledge based approaches rely on practitioner expertise to manually characterize adversary behavior. In contrast, data-driven approaches aim to reduce practitioner workload by automating the process of extracting insights about adversary behavior from observable data sources. Figure 1.2 graphically shows the standard process of gathering threat intelligence, and how this thesis contributes to it.

1.1.1.1. EXPERT KNOWLEDGE BASED MANUAL APPROACHES

Security practitioners build adversary behavior models based on their knowledge of past attacks, the analysis of threat intelligence feeds, and other domain knowledge. They are also further enriched with insights obtained from investigating historical incident logs.

In addition to being time-consuming and expensive, there are two main drawbacks of purely relying on expert knowledge for modeling adversary behavior: First, because these models are created manually, they must be updated periodically to accurately re-

flect the evolving threat landscape. This means that the practitioners experience fast-paced and monotonous workloads to keep the defenses updated, causing them to burn out [18]. Second, because of these extreme workloads, the task performance goes down and the quality of these models can vary [19]. As such, manually generated adversary behavior models are often incomplete, since analyzing all attacks with full situational awareness is infeasible.

1.1.2. MACHINE LEARNING BASED DATA-DRIVEN APPROACHES

Data-driven approaches rely on patterns in cyber data to create adversary behavior models. These approaches have the benefit of continually learning from new data in order to keep the behavior models updated. They can automatically discover intricate patterns that are too difficult to identify manually. Thus, data-driven approaches reduce practitioner workload by automating the generation of threat intelligence regarding adversary behavior. Note that this paradigm necessitates a mapping between concepts seen in standard threat assessment models (*e.g.*, MITRE ATT&CK) and observable data. For instance, the Action-Intent Framework (AIF) [20] is a wrapper around the ATT&CK framework that maps attacker intent with intrusion alert signatures (see Chapter 3).

Extracting patterns related to adversary behavior from observable data is extremely challenging. The Pyramid of Pain [12], presented in Figure 1.1b, shows different types of Indicators of Compromise (IOCs) that can be extracted from observable data. As one moves up the pyramid, the IOCs get harder to extract. The IOCs regarding adversary behavior (*i.e.*, TTPs) lie at the very top of the pyramid, demonstrating the difficulty of the task. Section 1.2 explains these challenges in detail.

DEFINITIONS: DATA-DRIVEN AGENTS

We define a data-driven approach for modeling adversary behavior as an autonomous, *data-driven machine learning agent* that learns *contextually meaningful cyber adversary behaviors* from *observable data*. We define these terms as follows:

- **Cyber adversary:** A cyber adversary is a single or a group of human actors or automated agents (malware) that intend to perform malicious actions that harm other cyber resources. The actions can also have physical aspects, *e.g.*, as in the case of social engineering attacks. The risk associated with a cyber adversary is related to their perceived capabilities and intent.
- **Adversary behavior:** A behavior is a group of actions, and is a learned abstraction (model or pattern) from observable data that can be interpreted and transferred to other systems.
- **Adversary intent:** Intent is defined as the relationship between an adversary and their action, which lends insights into the motivations that lead to an attack. Intent is inferred through observed actions. A framework such as ATT&CK or Action-Intent Framework (AIF) aims to connect the intended attack stage with the corresponding tactics, techniques, and procedures (TTPs).
- **Observables:** Observables can be extracted from data sources to learn adversary behavior. These include, but are not limited to, software logs (network traffic, in-

```

(timestamp:2018-11-03T14:36:00.340802+0000, flow_id:201450559731735,
 in_iface:ens4, event_type:alert, src_ip:10.0.0.20, src_port:33868,
 dest_ip:10.0.254.204, dest_port:3000, proto:TCP, tx_id:0, alert:{signature:ET SCAN
 Nikto Web App Scan in Progress, category:Web Application Attack, severity:2},
 _serial: 21248, _si: [index01, ids], _subsecond: 540802, _time: 2018-11-03
 14:36:00.340 UTC, index: ids, linecount: 1, source: /var/log/suricata/alert-
 json.log, sourcetype: suricata:alert, splunk_server: index01}

(timestamp:2018-11-03T23:29:58.831562+0000, flow_id:936249975795249,
 in_iface:ens4, event_type:alert, src_ip:10.0.254.204, src_port:35702,
 dest_ip:10.0.0.20, dest_port:3000, proto:TCP, tx_id:0, alert:{signature:GPL
 EXPLOIT CodeRed v2 root.exe access, category:Access to a Potentially Vulnerable
 Web Application, severity:2}, _serial: 0, _si: [index01, ids], _subsecond:
 .831562, _time: 2018-11-03 23:29:58.831 UTC, index: ids, linecount: 1, source:
 /var/log/suricata/alert-json.log, sourcetype: suricata:alert, splunk_server:
 index01}

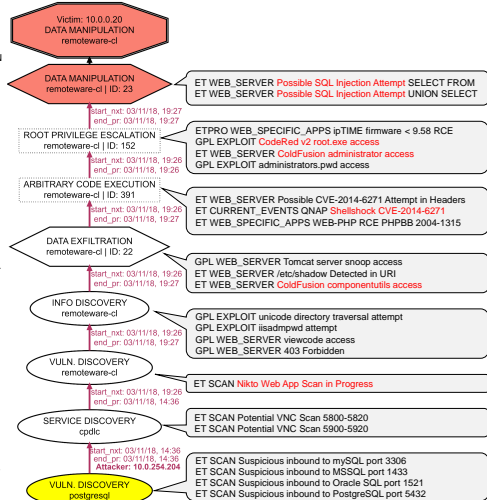
(timestamp:2018-11-03T19:26:40.831562+0000, flow_id:936249975795250,
 in_iface:ens4, event_type:alert, src_ip:10.0.254.204, src_port:35702,
 dest_ip:10.0.0.20, dest_port:3000, proto:TCP, tx_id:0, alert:{signature:ET
 WEB_SERVER ColdFusion administrator access, category:Web Application Attack,
 severity:2}, _serial: 0, _si: [index01, ids], _subsecond: .831562, _time: 2018-11-
 03 19:26:40.831 UTC, index: ids, linecount: 1, source: /var/log/suricata/alert-
 json.log, sourcetype: suricata:alert, splunk_server: index01}

(timestamp:2018-11-03T19:26:40.831562+0000, flow_id:936249975795251,
 in_iface:ens4, event_type:alert, src_ip:10.0.254.204, src_port:35702,
 dest_ip:10.0.0.20, dest_port:3000, proto:TCP, tx_id:0, alert:{signature:GPL
 EXPLOIT CodeRed v2 root.exe access, category:Access to a Potentially Vulnerable
 Web Application, severity:2}, _serial: 0, _si: [index01, ids], _subsecond:
 .831562, _time: 2018-11-03 19:26:40.831 UTC, index: ids, linecount: 1, source:
 /var/log/suricata/alert-json.log, sourcetype: suricata:alert, splunk_server:
 index01}

(timestamp:2018-11-03T13:26:40.831562+0000, flow_id:936249975795239,
 in_iface:ens4, event_type:alert, src_ip:10.0.254.204, src_port:35702,
 dest_ip:10.0.0.20, dest_port:3000, proto:TCP, tx_id:0, alert:{signature:GPL
 WEB_SERVER Oracle Java Process Manager access, category:Access to a Potentially
 Vulnerable Web Application, severity:2}, _serial: 0, _si: [index01, ids],
 _subsecond: .831562, _time: 2018-11-03 13:26:40.831 UTC, index: ids, linecount: 1,
 source: /var/log/suricata/alert-json.log, sourcetype: suricata:alert,
 splunk_server: index01}

```

(a) An excerpt from alert logs



(b) An attack path derived by a data-driven agent

Figure 1.3: Example 1: Deriving attack campaigns from intrusion alerts is time-intensive. A data-driven approach automatically aggregates correlated alerts, discovers dependencies between alerts, and displays the attack campaign as a path in an attack graph.

trusion alerts, system logs), software code (malware binaries decompiled or otherwise), threat intelligence (feeds shared among organizations, collected through open-source threat intelligence (OSINT)). “Features” are attributes derived from observables that characterize adversary actions. Note that obtaining real-world and usable observables is one of the biggest challenges in constructing data-driven adversary models, as described later in the chapter.

- **Autonomous data-driven agent:** A machine learning model that learns adversary behavior from observable data with either limited reliance on ground truth (*i.e.*, in a semi-supervised setting) or no reliance on ground truth (*i.e.*, in an unsupervised setting) [5]. It does not require frequent human intervention for (re)learning, and is not necessarily used for detection tasks. The model enables an interpretable (potentially) human-in-the-loop setting where a security analyst can understand the adversary behaviors discovered by the model, and validate the model, if needed.
- **Contextually meaningful:** A model that produces contextually meaningful output by correlating several temporally-linked observables from different modalities, instead of viewing a single observable in vacuum.

1.1.3. EXAMPLES: MANUAL VS. DATA-DRIVEN APPROACHES

In the following, we provide examples from two different domains that highlight the benefit of data-driven approaches over manual ones for understanding adversary behavior:

EXAMPLE 1: ATTACKER STRATEGY DISCOVERY

Once an attack is detected, security analysts want to discover the strategies utilized by the attackers to penetrate the network. An attack graph is a well-known example of an adversary behavior model that shows the pathways followed by an attacker. Traditional attack graphs are based on Topological Vulnerability Assessment (TVA), which rely on expert input and system vulnerabilities to model adversary behavior [21]. Due to the reliance on vulnerability scanning, these attack graphs are incomplete since not all vulnerabilities are known in advance [22]. Moreover, these attack graphs are typically huge since they show all the hypothetical pathways an attacker *may follow* based on the limited situational awareness [23]. In an operational setting, however, analysts want actionable insights into the exact path followed by the attacker.

We propose a data-driven approach to automatically discover attack campaigns captured by intrusion alert logs. Figure 1.3a shows an excerpt of intrusion alerts generated during an incident. While the alerts capture the actions of the attacker, it is time-consuming to manually identify the relationship between the alerts and the compromised assets. Our data-driven agent aggregates correlated alerts, discovers dependencies between them, and displays the strategy in an alert-driven attack graph¹. Figure 1.3b shows an attack campaign, demonstrating that the attacker host 10.0.254.204 performed data manipulation using an SQL injection attack after conducting a Nikto vulnerability scan, and penetrating the vulnerable host using the CodeRed, ColdFusion, and Shellshock exploits. Thus, our data-driven agent automates the time-consuming aspect of alert investigation: It reduces analyst workload by efficiently triaging critical alerts and providing actionable threat intelligence regarding attacker strategies.

EXAMPLE 2: MALWARE CAPABILITY ASSESSMENT

Another example is from the malicious software (malware) domain where analysts want to characterize the capabilities of malware samples. The information about malware capabilities is then utilized to study malware phylogeny [24], [25], and to create behavior catalogs, *e.g.*, the MITRE's Malware Behavior Catalog². The behavior catalog is particularly useful for creating behavior-based signatures.

To this end, malware analysts manually dissect malware executables, *e.g.*, using tools like Ghidra³, and investigate software logs resulting from malware executions. Figure 1.4a shows an excerpt of network traffic logs generated from the execution of several malware samples. It is time-consuming to analyze the logs, and identify the captured behaviors, especially for novel capabilities. In contrast, we propose a data-driven agent for automatically discovering the capabilities and creating a behavior catalog⁴. This behavior catalog can be refreshed as new malware samples are received. We can then describe previously unseen malware samples based on the discovered behaviors, *e.g.*, see Figure 1.4b. The behavior profiles help to identify interesting relationships between malware samples. For instance, the samples #29 and #17 exhibit identical capabilities and belong to the same malware family, *i.e.*, ZeuS-VM-AES. In contrast, the Gozi-ISFB samples #229

¹Details about this approach are given in Chapter 3.

²MITRE MBC: <https://github.com/MBCProject>

³Ghidra: <https://ghidra-sre.org/>







⁴Details about this approach are given in Chapter 5.

No., Time, Source, Destination, Protocol, Length, Info

```

1260, 2023-09-12 20:56:47.811358, 74.125.34.46, 192.168.178.36, TLV1.3, 1466,
Application Data
1261, 2023-09-12 20:56:47.811373, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=465138 Win=525856 Len=0
1262, 2023-09-12 20:56:47.811416, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=467962 Win=525856 Len=0
1263, 2023-09-12 20:56:47.811486, 2a00:1450:400e:801::2003,
2001:1c00:c04:a200:18b3:a103:7ba0:7d75, TCP, 86, 443 > 64601 [SYN, ACK] Seq=0
Ack=1 Win=65535 Len=0 MSS=1440 SACK_PERM WS=256
1264, 2023-09-12 20:56:47.811499, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=470706 Win=525856 Len=0
1265, 2023-09-12 20:56:47.811538, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=473610 Win=1053184 Len=0
1266, 2023-09-12 20:56:47.811585, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=476434 Win=1053184 Len=0
1267, 2023-09-12 20:56:47.811616, 2001:1c00:c04:a200:18b3:a103:7ba0:7d75,
2a00:1450:400e:801::2003, TCP, 74, 64601 > 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
1268, 2023-09-12 20:56:47.811636, 192.168.178.36, 74.125.34.46, TCP, 54, 64597 >
443 [ACK] Seq=2384 Ack=479258 Win=1053184 Len=0
1269, 2023-09-12 20:56:47.811882, 2a00:1450:400e:801::2003,
2001:1c00:c04:a200:18b3:a103:7ba0:7d75, TCP, 86, 443 > 64600 [SYN, ACK] Seq=0
Ack=1 Win=65535 Len=0 MSS=1440 SACK_PERM WS=256
1270, 2023-09-12 20:56:47.811977, 2001:1c00:c04:a200:18b3:a103:7ba0:7d75,
2a00:1450:400e:801::2003, TCP, 74, 64600 > 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
1271, 2023-09-12 20:56:47.813045, 2001:1c00:c04:a200:18b3:a103:7ba0:7d75,
2a00:1450:400e:801::2003, TLV1.3, 591, Client Hello
1272, 2023-09-12 20:56:47.813052, 2001:1c00:c04:a200:18b3:a103:7ba0:7d75,
2a00:1450:400e:801::2008, TLV1.3, 591, Client Hello
1273, 2023-09-12 20:56:47.817705, 2a00:1450:400e:801::2003,
2001:1c00:c04:a200:18b3:a103:7ba0:7d75, TCP, 74, 443 > 64599 [ACK] Seq=4485
Ack=1171 Win=67840 Len=0

```

Malware sample	Behavior profile
 ZeuS-VM-AES-29	<ul style="list-style-type: none"> Systematic port scan Randomized port scans Malicious subnet
 ZeuS-VM-AES-17	<ul style="list-style-type: none"> Systematic port scan Randomized port scan Malicious subnet
 Blackmoon-77	<ul style="list-style-type: none"> C&C reuse
 Zeus-Panda-770	<ul style="list-style-type: none"> C&C reuse
 Gozi-ISFB-229	<ul style="list-style-type: none"> Broadcast traffic LLMNR traffic Systematic port scan Randomized port scan
 Gozi-ISFB-86	<ul style="list-style-type: none"> Connection spam HTTPs traffic

(a) An excerpt from network traffic logs

(b) Behavior profiles derived by a data-driven agent

Figure 1.4: Example 2: Discovering the capabilities of malware from software logs is time-intensive. A data-driven approach automatically identifies the behaviors observed in software logs, and describes a malware sample using its behavior profile.

and #86 exhibit significantly different capabilities. Peculiar still, the Blackmoon sample #77 and the Zeus-Panda sample #770 receive instructions from the same Command and Control (C&C) server, which is strange since malware authors typically do not share C&C servers. In summary, these insights derived from a data-driven approach dramatically enhance the capacity of malware analysts to handle large volumes of malware samples.

1.2. CHALLENGES OF LEARNING ADVERSARY BEHAVIOR

In this section, we describe the challenges pertaining to the design of an effective data-driven agent capable of extracting insights regarding adversary behavior. While some of these challenges arise from the lack of representative datasets, it is mainly the modeling assumptions that determine the quality of the behavioral insights.

I. EXPLAINABLE ADVERSARY BEHAVIOR MODELING

Complex (black-box) ensembles of machine learning pipelines are difficult to understand and validate. This is particularly relevant for data-driven agents meant for modeling adversary behavior since their primary goal is *to provide behavioral insights to security analysts*. Existing research has shown that machine learning models learn biases from training data, making it difficult to trust them if they are black boxes [6]. This is why it is critical to develop white-box (interpretable) settings for security analysts so they can understand what the model is actually learning [26]. The field of explainable artificial intelligence (XAI) is actively developing methods to convert black-box models into their white-box counterparts by explaining their inner workings [27]. Although explain-

able agents for modeling adversary behavior can improve practitioner trust in machine learning, the design of such agents remains an active area of research [28].

II. UNLABELED DATASETS

Cyber data is rarely labeled in real-world settings. In addition, open-source datasets are typically not accompanied with granular behavior-related labels, which makes it really difficult to understand the observable behaviors in the datasets. For instance, although the scenarios described in the CTU-13 dataset [29] state when the hosts are infected with botnets, the network flows (Netflows) themselves are only labeled as benign or malicious without any indication of the exact behavior they correspond to. For the datasets where labels are available, they are often noisy, which makes it difficult to validate learning approaches. For example, the malware family names linked to open-source malware datasets have repeatedly been shown to be noisy and unreliable [30]. This unreliable nature of ground truth makes supervised learning very challenging, since the model is learning from faulty labels. Learning paradigms with limited reliance on ground truth are thus more realistic. Although semi-supervised and unsupervised learning techniques are more suitable paradigms for learning adversary behavior, it is significantly more challenging to design and evaluate such approaches [31].

III. INFREQUENT PATTERN MINING

Cyber data typically follows a skewed class distribution, such that the majority of the data is associated to benign activities. For instance, the majority of the traces in network induced observables, such as intrusion alerts and network traffic, often reflect benign/low-risk behavior, while those related to malicious activities are rare. Frequency-based learning algorithms end up discarding the rare malicious activities, which defeats the purpose of modeling adversary behavior. Learning with infrequent data remains an open problem in the machine learning community [32].

IV. FEATURE REPRESENTATION FOR BEHAVIOR CHARACTERIZATION

The quality of the behavioral insights that can be obtained from data is only as good as the data itself. The lack of good quality datasets for adversary behavior modeling is a well-known problem in the security community [33], [34]. In the presence of low quality datasets, it is important to consider feature selection and representation for effective behavior modeling. An important aspect of behavior characterization is context modeling. Sequential (temporal) features are effective for capturing the context in which an observable appears. For example, while a single incorrect login attempt seems innocuous, a series of irrational access attempts with mismatched port numbers point to an ongoing brute force attack. Nevertheless, sequential features are expensive to process and have limited algorithmic support. Effectively exploiting sequential features to learn contextually meaningful adversary behaviors is an active area of research [35], [36].

V. EVOLVING THREAT LANDSCAPE

In cybersecurity, there is a continual arms-race between attackers and defenders, which causes the threat landscape to evolve rapidly, making it near-impossible for data-driven agents to rely on supervised learning, or much of a priori expert knowledge. A data-

driven agent is expected to discover intricate behavioral patterns while adapting to the evolving landscape. For instance, anomaly detectors commonly learn from evolving data streams. They model the normal state of a system in order to detect deviations from it. Over time, the normal system behavior may evolve, either due to system upgrades or new features, which can trigger the anomaly detector to raise false alarms for normal behavior that no longer fits its criteria of normality [37], [38]. Hence, the agent must detect when the data distribution has changed sufficiently (known as concept drift), and relearn what the “new” normal behavior looks like. Continually learning from evolving data (in real-time) can become prohibitively expensive especially for sequential features, and thus remains an open problem [39], [40].

VI. EVALUATION APPROACHES FOR BEHAVIOR MODELING

It is challenging to evaluate data-driven agents meant for modeling adversary behavior from both quantitative and qualitative perspectives. From a quantitative point of view, there is a scarcity of metrics that measure model interpretability and the quality of the derived behavioral insights, specifically in the absence of ground truth [41]. Standard performance metrics like accuracy and precision may not adequately capture the true performance of a model [42]. This is why it is imperative that practitioners attempt to understand the inner workings of the data-driven agent. This requires qualitative approaches, *e.g.*, expert evaluation and case studies. To this aim, application-grounded evaluation (*i.e.*, user studies with security practitioners) is the gold-standard [43]. However, it is expensive to conduct user studies with a small pool of highly specialized personnel. Besides, security practitioners already face demanding workloads and may not have time to participate in such evaluations. The low response rate of security practitioners in user studies is a well-observed phenomenon in the security literature (discussed in Chapter 2). As such, a combination of appropriate metrics and qualitative analysis of diverse case studies is required, which should be designed with care.

1.3. EXPLAINABLE SEQUENTIAL MACHINE LEARNING

There are three key challenges related to modeling adversary behavior in terms of modeling assumptions (as described in Section 1.2): unsupervised setting, feature representation, and explainability. Since behavior has a temporal element (*e.g.*, in Figure 1.3b), we adopt the *sequential machine learning* paradigm to model adversary behavior in an *unsupervised setting*. We also pay special attention to making these sequential pipelines *explainable* to make it easier to extract threat intelligence about adversary behavior.

UNSUPERVISED SEQUENTIAL MACHINE LEARNING

Existing research typically assumes conditional independence between the features of network-induced observables, *e.g.*, intrusion alerts and network traffic. This enables them to collapse the structural and temporal properties of the observables into statistical features, which are cheap and efficient to process. Although statistical methods can efficiently handle large volumes of observables, neglecting their temporal dependence negatively impacts the quality of discovered patterns, since such observables are inherently linked due to unobserved latent factors [44]. Moreover, subtle differences in adver-

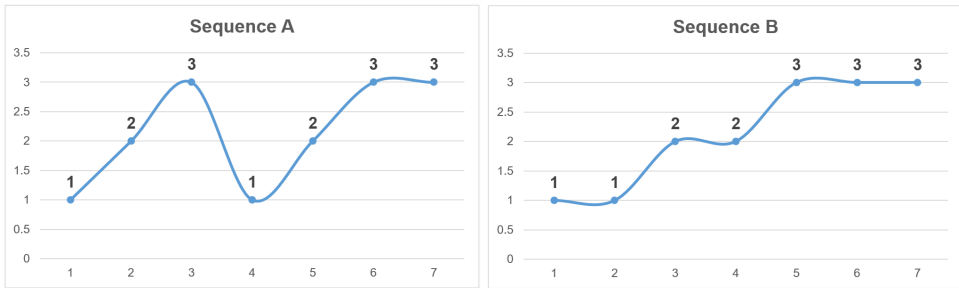


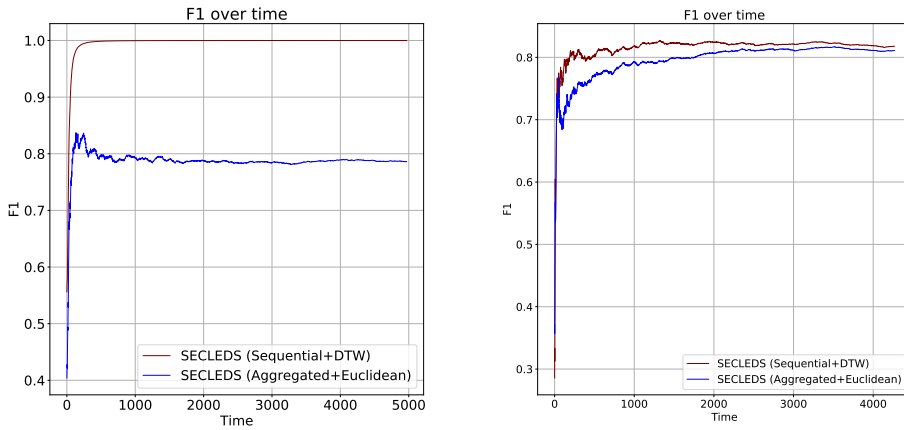
Figure 1.5: Sequential vs. statistical modeling: Sequence A is oscillating and Sequence B is monotonically increasing. Both sequences have the same statistical aggregates, *i.e.*, $\mu = 2.1$, $Me = 2$, $\sigma = 0.83$, and $\sigma^2 = 0.69$.

sary behavior can get lost when these features are collapsed into statistical aggregates, which would have been otherwise visible in sequential features. Figure 1.5 presents a toy example of two sequences (from [45]) with clear temporal differences that are lost when collapsed into statistical aggregates. The sequences have different temporal characteristics, *i.e.*, one is oscillating while the other is monotonically increasing. Both sequences can be represented by the same mean of 2.1, median of 2, standard deviation of 0.83, and variance of 0.69.

Sequential machine learning refers to learning algorithms that support sequential (temporal) features. Sequential features are a rich resource for behavior analysis since the order of events often provides meaningful insights regarding the nature of events. Sequential features also enable modeling the context (or semantics) of adversary actions. The neighboring activities (*i.e.*, what happens before and after an action) can be used to model the context. This context can be used for differentiating between similar actions that result in significantly different outcomes. For instance, recent work has shown that sequential features are more effective in detecting intrusions than statistical features [46]. Similarly, Figure 1.6 shows the impact of feature representation on the performance of two clustering tasks (discussed later in Chapter 6), where sequential features clearly outperform statistical aggregates.

Sequence learning has only recently started getting attention in the literature. This interest can be explained, in part, due to the ubiquity of deep learning methods, such as the Long Short-Term Memory (LSTM) and Transformer networks. The typical usage of these methods is prominent within supervised learning scenarios (where ground truth is available). They are also usually fed entire executables or network streams without explicit feature selection optimized for behavior modeling [47], [48]. Within the unsupervised learning paradigm, Markov chains, and finite state automata are popular sequence learning models. They have been sparingly used within cybersecurity, *e.g.*, for reverse engineering network protocols [49], and detecting attacks on cyber-physical systems [50]. However, many of these works focus on attack detection, and do not utilize the full potential of machine learning for analyzing adversary behavior.

Nevertheless, as powerful as sequential features are, it is difficult to work with them: First, they may be misaligned, requiring specialized distance measures to determine the similarity between them. These distance measures often have a quadratic run-time



(a) Univariate sine curve sequences

(b) Multivariate network traffic sequences

Figure 1.6: Sequence clustering (using the algorithm presented in Chapter 6) achieves higher F1 scores versus collapsing the temporal features into aggregates. (a) Clustering 5000 univariate sine curves from 4 classes. (b) Clustering 4276 multivariate sequences of network traffic from 2 classes. All sequences have length 100.

complexity in the length of the sequence, making their use in real-time applications nearly impossible. Second, sequential features have high dimensionality, making it difficult to visualize them, and consequently understand the results. Besides, many sequence learning models are considered black boxes because it is difficult for an analyst to understand how the input data led to a model prediction.

EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI)

Explainable Artificial Intelligence (XAI) refers to the tools and techniques that aim to improve the understandability of machine learning models for humans [27]. There are two key paradigms within XAI: explaining a black-box model using post-hoc explainability, and directly learning a white-box (interpretable) model. See Chapter 2 for details.

In summary, post-hoc explanations either identify influential features for the black-box model [51], or learn a simpler interpretable model called a “surrogate model” [52]. While promising for verifying the correctness of black-box models, the fidelity and trustworthiness of post-hoc explanations themselves can be subjected to attacks [53].

Alternatively, one can directly learn an interpretable model. These models are widely believed to be less performant than black-box models [6]. Recent studies show that there does not necessarily have to be a trade-off between explainability and performance, *i.e.*, interpretable models can sometimes even achieve better performance than their black-box counterparts [54]. In fact, when a model is interpretable, it allows humans to learn from it, which ultimately also elevates human performance. There is an increasing emphasis on interpretable models that enable human-in-the-loop setting for reliable decision support [6].

Note that keeping a data-driven model interpretable requires an innate understanding of the problem domain and the machine learning methods, especially for complex structured data, *e.g.*, sequential features. As such, there is limited algorithmic support for interpretable sequence learning algorithms that can operate in an unsupervised setting.

1.4. PROBLEM STATEMENT

This thesis addresses the open problem of *designing explainable sequence learning methods for understanding adversary behavior that do not rely on ground truth*. Below, we motivate the research questions answered in this dissertation:

Data-driven agents that are meant to provide insights into adversary behavior are expected to be explainable. Although Explainable Artificial Intelligence (XAI) methods have recently gained traction within cybersecurity as a means to improve the understandability of machine learning models, they are not yet commonly used to understand adversary behavior. Additionally, the available literature is fragmented across several research communities, including machine learning, computer graphics, and cybersecurity. The first research question systematizes the state-of-the-art literature that uses XAI for security applications in order to *consolidate the notion of explainability within cybersecurity, i.e.*, in terms of key stakeholders (explanation users), application objectives (explanation uses), task domains, and preferred explanation methods.

Q1: *What are the challenges and opportunities for explainable artificial intelligence (XAI) within cybersecurity?*

From the broad field explainable adversary behavior analysis, we focus on modeling the strategies employed by human threat actors, and the capabilities exhibited by automated adversaries (malware). We limit ourselves to analyzing sequences of network-induced observable data (*e.g.*, intrusion alerts, network logs) since they can be acquired remotely, and are more cost-effective to obtain compared to system logs.

1.4.1. ATTACKER STRATEGY DISCOVERY

Practitioners in Security Operations Centers (SOC) struggle to keep up with the dramatic volumes of intrusion alerts. Once a network intrusion is detected, the process to identify and reverse engineer attacker strategies (attack campaigns) is largely manual and time-consuming. While existing alert correlation techniques reduce the volume of alerts to analyze, they do not show attack progression. Meanwhile, traditional attack graphs (AG) show attack progression, but rely heavily on expensive expert knowledge that is not always available, resulting in graphs that only show a static view of the network. Learning attack graphs from data is a longstanding open problem in cybersecurity and formal methods [55], [56]. It is challenging to discover sequential constraints within data while mitigating issues related to a state-space explosion. Existing efforts have so far managed to summarize alert datasets using sequential machine learning without explicitly extracting attack graphs [57], [58]. The next question investigates the possibility of using sequential machine learning to construct attack graphs directly from intrusion alerts as a way of bridging the gap between dynamic alert management and static attack graphs.

Q2: *How can we learn attack graphs directly from intrusion alerts using interpretable unsupervised sequence learning?*

Provided that attack graphs can be learned from intrusion alerts, we hypothesize that these ‘alert-driven’ attack graphs will capture attacker strategies reflected in the intrusion alerts. The next question investigates whether the alert-driven attack graphs provide more threat intelligence regarding attacker strategies compared to raw alert investigation, *e.g.*, related to similarities and differences between different attack campaigns, and fingerprintable attacker behavior.

Q3: *What kind of threat intelligence can be extracted from alert-driven attack graphs for discovering similarities and differences between attacker strategies?*

1.4.2. NETWORK ATTACK CHARACTERIZATION

Malware analysts identify the capabilities of malware by manually tagging known behaviors during the investigation of decompiled malware code and dynamically created system logs. Anecdotally, it can take them anywhere from a few minutes to several days to investigate the capabilities of a malware sample, and automating malware capability assessment remains an open problem. To this end, the next question investigates whether it is possible to use unsupervised machine learning to discover the capabilities of malware samples from their execution traces, *i.e.*, by clustering sequences of execution traces to identify distinct behavioral groups. One key consideration in answering this question is feature selection – the choice of features determines the granularity of the behaviors that are visible in the clusters. We limit ourselves to identifying the capabilities of malware visible in their network logs, since it is believed that network traffic shows the core behavior of malware [59].

Q4: *How can we leverage unsupervised sequence clustering to characterize the network behavior of malware in order to discover similarities and differences between malware capabilities?*

The previous question considers an offline setting for malware capability assessment. However, malware behavior evolves rapidly in order to evade detection. Network traffic must be continuously monitored to accurately characterize malware capabilities at any given time. This creates the need for a real-time sequence clustering approach that can handle evolving data distributions (*i.e.*, due to concept drift). Existing real-time clustering algorithms have limited support for sequential data [60]. Besides, a highly efficient algorithm is required to cluster the large volumes of network traffic required for real-time behavioral analytics. Furthermore, the algorithms that support sequential data do not provide interpretability, which is a problem in itself since the analysts need to be able to understand the evolving behaviors captured by a cluster. To the best of our knowledge, there is currently no way to cluster large sequential datasets in real-time

while supporting interpretability and concept drift. The next question fills this gap, and investigates whether such a real-time sequence clustering algorithm improves the quality of malware capability assessment compared to standard approaches.

Q5: *How can we characterize the network behavior of malware in real-time using interpretable unsupervised sequence clustering?*

1.5. CONTRIBUTIONS OF THIS THESIS

This thesis tackles the challenges outlined in Section 1.2 and the questions posed in Section 1.4 by following a two-pronged approach: 1) We develop learning algorithms capable of handling the unique challenges of cyber data, *e.g.*, algorithms with inherent support for sequences [61], algorithms that learn from infrequent data [62], and algorithms that learn from evolving data [39]. 2) We develop interpretable tool-chains that utilize these learning algorithms for modeling adversary behavior. These tool-chains aim to reduce practitioner workload by automatically triaging large volumes of observable data, and extracting actionable intelligence regarding adversary behavior.

We show, for multiple scenarios, that it is possible to develop interpretable machine learning models (even for complex sequential data in an unsupervised setting) that provide more insights than just prediction probabilities, while achieving competitive performance. Consequently, this thesis challenges the status quo, and makes fundamental contributions to the fields of machine learning and cybersecurity.

The specific contributions of this thesis are as follows:

1. In **Chapter 2**, we propose the first comprehensive taxonomy that defines the notion of explainability within cybersecurity in terms of stakeholders and application objectives. We distill key takeaways and recommendations from the literature that should streamline and stimulate further research in the field.
2. We also provide a tutorial that (i) shows how model designers can utilize commonplace explanation tools to discover weaknesses in their models (*e.g.*, spurious correlations), and (ii) warns about the dangers of misinterpreting post-hoc explanations that may steer practitioners towards misleading conclusions.
3. We propose a myriad of explanation techniques for analysis tasks, as opposed to the dominant paradigm of prediction tasks. The explanations elucidate the input data and the behavioral patterns discovered by the model, *e.g.*, by summarizing attacker strategies in intrusion alerts (**Chapter 3**), by explaining the behavioral relationships among malware samples (**Chapter 5**), and by explaining the meaning of multivariate sequential clusters in real-time (**Chapter 6**).
4. In **Chapter 3**, we propose a suffix-based probabilistic deterministic finite automaton (S-PDFA) – an interpretable sequence learning model that accentuates infrequent intrusion alerts and models the semantic meaning behind the alerts.
5. We develop SAGE – the first tool-chain to learn succinct and interpretable attack graphs directly from intrusion alerts using the S-PDFA. The attack graph generator is available publicly as a docker container.

6. We also develop a web-based dashboard with querying and prioritization capabilities to further consolidate the ‘alert-driven’ attack graphs.
7. In **Chapter 4**, we demonstrate the extraction of threat intelligence enabled by the ‘alert-driven’ attack graphs (pertaining to strategic differences, scripted attacks, and fingerprintable attempts) that would have been much more costly to acquire with a raw corpus of intrusion alerts.
8. To the best of our knowledge, we develop the first method to automatically characterize the network behavior of malware samples using explainable sequence clustering in **Chapter 5**. We have made our machine learning pipeline, MalPaCA, open-source.
9. In **Chapter 6**, we propose SECLEDS – the first real-time interpretable algorithm for clustering sequences in evolving data streams. Our algorithm significantly improves the state-of-the-art, and is a lightweight algorithm that efficiently clusters large volume of sequences.
10. We utilize SECLEDS as a novel network traffic sampling technique that preserves the temporal relationships between network packets. Our sampling technique improves the quality of real-time malware behavior characterization compared to standard sampling techniques.

1.5.1. OUR APPROACH TO LEARNING ADVERSARY BEHAVIOR

This thesis addresses the challenges described in Section 1.2 as follows: The machine learning tool-chains proposed in Chapters 3 – 6 predominantly model adversary behavior in an interpretable way by intelligently summarizing large volumes of network-induced observables in order to reduce practitioner workload (*Challenge I*). These tool-chains go beyond accuracy scores by providing actionable insights into adversary behavior. SAGE, MalPaCA, and SECLEDS also do not rely on ground truth or expert knowledge to discover adversary behaviors (*Challenge II*). To encompass a variety of data-collection methods, we utilize industry-acquired data in Chapter 5, data generated from security competitions in Chapters 3 and 4, and multiple open-source datasets in Chapter 6. Each of these tool-chains extract contextually meaningful behaviors by modeling the temporal relationship between observables (*Challenge IV*). Moreover, the S-PDFA model proposed in Chapter 3 accentuates infrequent patterns in intrusion alerts (*Challenge III*), while the SECLEDS clustering algorithm proposed in Chapter 6 summarizes the concepts present in evolving data streams (*Challenge V*).

Finally, the challenges around evaluation are addressed by conducting quantitative and qualitative analyses. Since the tool-chains are aimed at reducing practitioner workload, one way to evaluate their utility is to measure the fraction of data that is summarized by the methods. For instance, the utility of SAGE is quantitatively evaluated by measuring the quality of the S-PDFA model and its alert compression power, and it is qualitatively evaluated by investigating the attacker strategies in multiple datasets and conducting informal user studies with senior security analysts. Similarly, MalPaCA and SECLEDS are evaluated by first comparing their performance against existing approaches, and then analyzing case studies to delve deeper into their added utility.

1.5.2. THESIS OUTLINE

Most technical chapters of this thesis contain an integral copy of a publication sometimes with minor changes. The chapters are independent and can be read on their own. The publication details are given on the first page of each chapter. Because we preserve the technical details of the chapters as their original publications, there may be overlap between various chapters, *e.g.*, in the introduction, preliminaries, and datasets. The thesis is organized as follows:

CHAPTER 2: A TAXONOMY OF EXPLAINABLE MACHINE LEARNING IN CYBERSECURITY

This chapter answers **Q1** by systematizing the fragmented body of literature that uses XAI for cybersecurity applications. We identify three main stakeholders (*i.e.*, model users, model designers, adversaries) and four application objectives (XAI-enabled user assistance, XAI-enabled model verification, explanation verification & robustness, and offensive use of explanations). We observe that explainability is predominantly used with black-box models for prediction tasks. We note that the security literature sometimes fails to disentangle the role of the various stakeholders, which makes it harder to provide explanations to model users and designers without also exposing them to adversaries. We provide a tutorial that shows the utility of off-the-shelf XAI methods for exposing weaknesses in a machine learning model, but also warns model designers of the potential pitfalls of these explanations that may steer them towards misleading conclusions. We provide recommendations for streamlining future research efforts.

This chapter has been published as: “*SoK: Explainable Machine Learning for Computer Security Applications*” by **Nadeem, A.**, Vos, D., Cao, C., Pajola, L., Dieck, S., Baumgartner, R., & Verwer, S. in IEEE European Symposium on Security and Privacy (Euro S&P), 2023 (pp. 221-240).

PART I: INTERPRETABLE SEQUENTIAL LEARNING FOR ATTACKER STRATEGY DISCOVERY

Part I of this thesis presents approaches to learn attacker strategies from intrusion alerts.

CHAPTER 3: ALERT-DRIVEN ATTACK GRAPH GENERATION

This chapter answers **Q2** regarding learning attack graphs. We propose a novel solution for ‘alert fatigue’ by developing an interpretable sequence learning tool, SAGE, that learns attacker strategies directly from intrusion alerts (without any expert input), and displays them as succinct attack graphs. At the heart of SAGE lies a suffix-based probabilistic deterministic finite automaton (S-PDFA) – an interpretable model that accentuates infrequent severe alerts, and models the context of alerts to differentiate between similar strategies leading to different outcomes. We also study the impact of various modeling decisions and alert datasets on the quality of the attack graphs. We show that SAGE compresses over 1.4 million alerts into ~400 attack graphs, dramatically reducing the workload of security analysts.

Parts of this chapter have been published as: “*Alert-driven Attack Graph Generation using S-PDFA.*” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in IEEE Transactions on

Dependable and Secure Computing (TDSC), 2021, 19(2), 731-746, and “Enabling Visual Analytics via Alert-driven Attack Graphs” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in ACM SIGSAC Conference on Computer and Communications Security (CCS), 2021 (pp. 2420-2422).

CHAPTER 4: ENABLING VISUAL ANALYTICS VIA ALERT-DRIVEN ATTACK GRAPHS

In addition to triaging alerts, the alert-driven attack graphs help visualize and compare attacker strategies. This chapter answers **Q3** by conducting a qualitative analysis of the alert-driven attack graphs. We show that the attack graphs i) enable forensic analysis of prior attacks, and ii) enable proactive defense by providing threat intelligence regarding attacker strategies, *e.g.*, pertaining to strategic differences, scripted attacks, and fingerprintable attempts. We also notice that while SAGE does the heavy lifting in terms of discovering attacker strategies, it is infeasible for security practitioners to visualize each attack graph for finding global patterns. Thus, we build an interactive web-based visual analytics dashboard to consolidate the alert-driven attack graphs. The dashboard is equipped with filtering and prioritization capabilities to highlight alerts that might require the urgent attention of security practitioners, further reducing their workload.

Parts of this chapter have been published as: “Alert-driven Attack Graph Generation using S-PDFA.” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in IEEE Transactions on Dependable and Secure Computing (TDSC), 2021, 19(2), 731-746, and “Critical Path Exploration Dashboard for Alert-driven Attack Graphs” by **Nadeem, A.**, Diaz, S. L., & Verwer in IEEE Symposium on Visualization for Cyber Security (VizSec), 2022.

PART II: EXPLAINABLE SEQUENTIAL LEARNING FOR NETWORK ATTACK ANALYSIS

Part II of this thesis presents approaches to characterize the network behavior of malicious software (malware/botnets).

CHAPTER 5: BEYOND LABELS: AUTOMATED BEHAVIOR PROFILING OF MALWARE

This chapter answers **Q4** regarding malware behavior characterization. We follow the intuition that similar malware capabilities/behaviors generate similar network traffic. We utilize abstract features extracted only from network packet headers since malware analysts do not always have access to network packet payloads. We develop MalPaCA – an explainable sequence clustering approach that partitions the unique behaviors present in a network traffic dataset. It uses the cluster membership information to build behavioral profiles for malware samples that describe their capabilities. A global dendrogram of the behavioral profiles enables comparison between the capabilities of different malware samples. This way, MalPaCA reduces the time required to analyze a malware sample by automatically explaining its capabilities.

This chapter has been published as: “Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families” by **Nadeem, A.**, Hammerschmidt, C., Gañán, C. H., & Verwer, S. in Malware Analysis Using Artificial Intelligence and Deep Learning 2021 (pp. 381-409), Springer.

CHAPTER 6: REAL-TIME SEQUENCE CLUSTERING FOR NETWORK ATTACK SUMMARIZATION

This chapter answers **Q5** by designing an interpretable sequence clustering algorithm, SECLEDS, that efficiently clusters large data streams with concept drift in almost linear time. We demonstrate that SECLEDS outperforms several state-of-the-art real-time clustering algorithms, especially in the presence of concept drift. SECLEDS is an on-line variant of the popular k-medoids algorithm that represents the clusters with actual data instances (known as medoids). We develop an explanation method that utilizes the medoids to incrementally explain the clusters as new data arrives. We utilize SECLEDS to summarize network traffic in order to reduce the burden on downstream tasks. We use SECLEDS as an intelligent temporal pattern-preserving traffic sampling technique. We show that such a sampling technique can support network bandwidths of over 1 GB/s. Finally, we combine SECLEDS with MalPaCA in order to support real-time malware capability assessment. By sampling traffic from real botnets, we show that SECLEDS enables MalPaCA to build better behavioral profiles of bot-infected devices in real-time compared to standard sampling techniques.

Parts of this chapter have been published as: “*SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting*” by **Nadeem, A.**, & Verwer, S. in European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), 2022 (pp. 157-173).

CHAPTER 7: CONCLUSIONS, OUTLOOK, AND SOCIETAL RELEVANCE

This chapter presents a detailed reflection on the research questions and concludes the thesis. We also discuss future research directions that we identified during this thesis.

1.5.3. LIST OF EXCLUDED PUBLICATIONS

The following papers were published during the Ph.D. study but were not included in the dissertation because they do not contribute significantly to the story line of the thesis.

1. **Nadeem, A.** (2024). *Cybersecurity as a Crosscutting Concept Across an Undergrad Computer Science Curriculum: An Experience Report*. ACM Technical Symposium on Computer Science Education (SIGCSE).
2. Mouwen, D., Verwer, S., & **Nadeem, A.** (2022). *Robust attack graph generation*. Learning and Automata workshop (LearnAut).
3. **Nadeem, A.**, Rimmer, V., Joosen, W., & Verwer, S. (2022). *Intelligent malware defenses*. Security and Artificial Intelligence, Springer, pp. 217-253.
4. Rimmer, V., **Nadeem, A.**, Verwer, S., Preuveneers, D., & Joosen, W. (2022). *Open-World network intrusion detection*. In Security and Artificial Intelligence, Springer, pp. 254-283.
5. **Nadeem, A.**, Verwer, S., & Yang, S. J. (2021). *SAGE: Intrusion Alert-driven Attack Graph Extractor*. IEEE Symposium on Visualization for Cyber Security (VizSec) (pp. 36-41). IEEE.
6. Verwer, S., **Nadeem, A.**, Hammerschmidt, C., Blik, L., Al-Dujaili, A., & O'Reilly, U. M. (2020). *The robust malware detection challenge and greedy random accelerated multi-bit search*. ACM Workshop on Artificial Intelligence and Security (AISec), pp. 61-70.

7. Roeling, M. P., Nadeem, A., & Verwer, S. (2020). *Hybrid connection and host clustering for community detection in spatial-temporal network data*. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), Springer, pp. 178-204.

REFERENCES

- [1] A. Nadeem, S. J. Yang, and S. Verwer, “Learning about the adversary”, *Autonomous intelligent agents for cyber defense*, 2023.
- [2] N. James, *160 cybersecurity statistics 2023 [updated]*, Accessed: 22-Sept-2023, Sep. 2023. [Online]. Available: <https://www.getastra.com/blog/security-audit/cyber-security-statistics/>.
- [3] T. Casey, *Survey: 27 Percent of IT professionals receive more than 1 million security alerts daily*, Accessed: 08-Jul-2021, 2018. [Online]. Available: <https://www.imperva.com/blog/27-percent-of-it-professionals-receive-more-than-1-million-security-alerts-daily>.
- [4] N. James, *35 cyber security vulnerability statistics, facts in 2023*, Accessed: 22-Sept-2023, Aug. 2023. [Online]. Available: <https://www.getastra.com/blog/security-audit/cyber-security-vulnerability-statistics/>.
- [5] R. Kohani and F. Provos, “Glossary of terms”, *Machine Learning*, vol. 30, no. 2, pp. 271–274, 1998.
- [6] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”, *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.
- [7] A. Surkov, V. Srinivas, and J. Gregorie, *Unleashing the power of machine learning models in banking through Explainable Artificial Intelligence (XAI)*, Jun. 2022. [Online]. Available: <https://www2.deloitte.com/us/en/insights/industry/financial-services/explainable-ai-in-banking.html>.
- [8] G. Apruzzese, P. Laskov, E. M. de Oca, *et al.*, “The role of machine learning in cybersecurity”, *Digital Threats: Research and Practice*, 2022.
- [9] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why should I trust you?” Explaining the predictions of any classifier”, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.
- [10] Z. Zhu and T. Dumitras, “Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports”, in *IEEE European symposium on security and privacy (EuroS&P)*, IEEE, 2018, pp. 458–472.
- [11] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains”, *Leading Issues in Information Warfare & Security Research*, 2011.
- [12] D. Bianco, “The pyramid of pain”, *Enterprise Detection & Response*, 2013.
- [13] M. McFate, “The military utility of understanding adversary culture”, Office of naval research, Arlington VA, Tech. Rep., 2005.

- [14] A. N. Steinberg, “An approach to threat assessment”, in *2005 7th International Conference on Information Fusion*, vol. 2, Jul. 2005, p. 8.
- [15] A. Steinberg, “Predictive modeling of interacting agents”, in *2007 10th International Conference on Information Fusion*, Jul. 2007, pp. 1–6.
- [16] S. Michie, M. M. van Stralen, and R. West, “The behaviour change wheel: A new method for characterising and designing behaviour change interventions”, *Implement. Sci.*, vol. 6, p. 42, Apr. 2011.
- [17] 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.
- [18] W. U. Hassan, S. Guo, D. Li, *et al.*, “Nodoze: Combatting threat alert fatigue with automated provenance triage”, in *Network and Distributed System Security*, 2019.
- [19] J. Dykstra and C. L. Paul, “Cyber Operations Stress Survey COSS: Studying fatigue, frustration, and cognitive workload in cybersecurity operations”, in *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, 2018.
- [20] S. Moskal and S. J. Yang, “Framework to Describe Intentions of a Cyber Attack Action”, *arXiv preprint arXiv:2002.07838*, 2020.
- [21] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O’Hare, and K. Prole, “Advances in topological vulnerability analysis”, in *CATCH*, IEEE, 2009.
- [22] S. Jha, O. Sheyner, and J. Wing, “Two formal analyses of attack graphs”, in *CSFW*, IEEE, 2002.
- [23] R. P. Lippmann and K. W. Ingols, “An annotated review of past papers on attack graphs”, *Massachusetts Institute of Technology, Lincoln Laboratory Lexington, MA, USA*, 2005.
- [24] P. Black, I. Gondal, and R. Layton, “A survey of similarities in banking malware behaviours”, *Computers & Security*, 2017.
- [25] J. Moubarak, M. Chamoun, and E. Filiol, “Comparative study of recent MEA malware phylogeny”, in *Computer and Communication Systems (ICCCS)*, IEEE, 2017, pp. 16–20.
- [26] T. J. Sejnowski, “The unreasonable effectiveness of deep learning in artificial intelligence”, *Proc. Natl. Acad. Sci. U. S. A.*, vol. 117, no. 48, pp. 30 033–30 038, Dec. 2020.
- [27] T. Miller, “Explanation in artificial intelligence: Insights from the social sciences”, *Artificial intelligence*, vol. 267, pp. 1–38, 2019.
- [28] A. Nadeem, D. Vos, C. Cao, *et al.*, “SoK: Explainable Machine Learning for Computer Security Applications”, in *IEEE European Symposium on Security and Privacy (Euro S&P)*, IEEE, 2023.
- [29] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods”, *Computers & Security*, vol. 45, pp. 100–123, 2014.
- [30] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, “AVClass: A tool for massive malware labeling”, in *RAID*, Springer, 2016, pp. 230–253.

- [31] A. Okutan and S. J. Yang, “ASSERT: Attack synthesis and separation with entropy redistribution towards predictive cyber defense”, *Cybersecurity*, vol. 2, no. 1, pp. 1–18, 2019.
- [32] Y. Lu, F. Richter, and T. Seidl, “Efficient Infrequent Pattern Mining Using Negative Itemset Tree”, in *Complex Pattern Mining: New Challenges, Methods and Applications*, Cham: Springer International Publishing, 2020, pp. 1–16.
- [33] P. Du, Z. Sun, H. Chen, J.-H. Cho, and S. Xu, “Statistical Estimation of Malware Detection Metrics in the Absence of Ground Truth”, *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 12, pp. 2965–2980, Dec. 2018.
- [34] A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer, “Intelligent Malware Defenses”, in *Security and Artificial Intelligence*, Springer, 2022, pp. 217–253.
- [35] 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.
- [36] A. Alsaheel, Y. Nan, S. Ma, *et al.*, “ATLAS: A Sequence-based Learning Approach for Attack Investigation”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 3005–3022.
- [37] C. Hammerschmidt, S. Marchal, R. State, and S. Verwer, “Behavioral clustering of non-stationary IP flow record data”, in *CNSM*, IEEE, 2016, pp. 297–301.
- [38] R. Jordaney, K. Sharad, S. K. Dash, *et al.*, “Transcend: Detecting concept drift in malware classification models”, in *Proc. of the USENIX Security Symposium*, 2017, pp. 625–642.
- [39] A. Nadeem and S. Verwer, “SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, 2022.
- [40] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. d. Carvalho, and J. Gama, “Data stream clustering: A survey”, *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, pp. 1–31, 2013.
- [41] M. L. Leavitt and A. Morcos, “Towards falsifiable interpretability research”, *arXiv preprint arXiv:2010.12016*, 2020.
- [42] R. Jordaney, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, “Misleading metrics: On evaluating machine learning for malware with confidence”, *Tech. Rep.*, 2016.
- [43] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning”, *arXiv preprint arXiv:1702.08608*, 2017.
- [44] M. P. Roeling, A. Nadeem, and S. Verwer, “Hybrid connection and host clustering for community detection in spatial-temporal network data”, in *Workshop on Machine Learning for Cyber Security (MLCS)*, Springer, 2020, pp. 178–204.
- [45] A. Nadeem, “Clustering Malware’s Network Behavior using Simple Sequential Features”, Delft University of Technology, 2018. DOI: <http://resolver.tudelft.nl/uuid:c8a221b9-9289-4978-a356-af64d8f2c5e0>.

- [46] A. Corsini, S. J. Yang, and G. Apruzzese, “On the evaluation of sequential machine learning for network intrusion detection”, in *International Conference on Availability, Reliability and Security*, 2021, pp. 1–10.
- [47] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, “Malware detection by eating a whole exe”, in *Workshops at AAAI Conference on Artificial Intelligence*, 2018.
- [48] S. Jain and Y. K. Meena, “Byte level n-gram analysis for malware detection”, in *International Conference on Information Processing*, Springer, 2011, pp. 51–59.
- [49] C. Y. Cho, D. Babić, E. C. R. Shin, and D. Song, “Inference and analysis of formal models of botnet command and control protocols”, in *Proceedings of the 2010 ACM SIGSAC Conference on Computer and Communications Security*, 2010, pp. 426–439.
- [50] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, “TABOR: A graphical model-based approach for anomaly detection in industrial control systems”, in *AsiaCCS*, 2018.
- [51] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni, “Deep Reinforcement Adversarial Learning Against Botnet Evasion Attacks”, *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 1975–1987, 2020.
- [52] M. Szczepański, M. Choraś, M. Pawlicki, and R. Kozik, “Achieving explainability of intrusion detection system by hybrid oracle-explainer approach”, in *IJCNN*, IEEE, 2020, pp. 1–8.
- [53] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, “Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods”, in *Proceedings of the AAAI Conference on AI, Ethics, and Society*, New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 180–186.
- [54] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan, “Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model”, *Ann. Appl. Stat.*, vol. 9, no. 3, pp. 1350–1371, Sep. 2015.
- [55] A. Nadeem, S. Verwer, and S. J. Yang, “SAGE: Intrusion Alert-driven Attack Graph Extractor”, in *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2021.
- [56] K. Sowka, V. Palade, H. Jadidbonab, P. Wooderson, and H. Nguyen, “A Review on Automatic Generation of Attack Trees and Its Application to Automotive Cybersecurity”, *Artificial Intelligence and Cyber Security in Industry 4.0*, pp. 165–193, 2023.
- [57] 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.
- [58] S. Haas and M. Fischer, “GAC: Graph-based alert correlation for the detection of distributed multi-step attacks”, in *SAC*, 2018.
- [59] L. Cavallaro, C. Kruegel, and G. Vigna, “Mining the network behavior of bots”, *Technical Report 2009-12*, 2009.

- [60] A. Khaleghi, D. Ryabko, J. Mary, and P. Preux, “Online Clustering of Processes”, in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, vol. 22, PMLR, Apr. 2012, pp. 601–609.
- [61] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, “Beyond labeling: Using clustering to build network behavioral profiles of malware families”, in *Malware Analysis Using Artificial Intelligence and Deep Learning*, Springer, 2021, pp. 381–409.
- [62] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, “Alert-driven Attack Graph Generation using S-PDFA”, *IEEE Transactions on Dependable and Secure Computing*, 2021.

2

A TAXONOMY OF EXPLAINABLE MACHINE LEARNING IN CYBERSECURITY

Explainable Artificial Intelligence (XAI) improves the transparency of machine learning (ML) pipelines. In this chapter, we consolidate the notion of explainability within cybersecurity, and identify the challenges and opportunities for the use of XAI in cybersecurity.

We systematize the increasingly growing (but fragmented) microcosm of studies that develop and utilize XAI methods for defensive and offensive cybersecurity tasks. We identify 3 cybersecurity stakeholders (users) who utilize XAI for 4 distinct objectives (uses) within an ML pipeline. Our analysis of the literature indicates that many of the XAI-enabled user assistance applications remove the model user from the equation. The security literature sometimes also fails to disentangle the role of the various stakeholders – the role of model designers is particularly minimized in the security literature. As a solution, we present an illustrative tutorial for model designers, demonstrating how off-the-shelf XAI tools can help with model verification. We also discuss scenarios where interpretability by design may be a better alternative. We hope that the discussion helps to shape the future of XAI research within cybersecurity.

This chapter is based on the paper “SoK: Explainable Machine Learning for Computer Security Applications” by **Nadeem, A.**, Vos, D., Cao, C., Pajola, L., Dieck, S., Baumgartner, R., & Verwer, S. in IEEE European Symposium on Security and Privacy (Euro S&P), 2023 (pp. 221–240) [1].

2.1. INTRODUCTION

Cybersecurity applications predominately use machine learning (ML) to maximize attack detection, while minimizing false alarms and detection times. At the same time, security practitioners are interested in high-performing ML models that can also explain their decisions [2]. However, despite the unprecedented performance achieved by prevailing ML systems, they have been slow to materialize in the security industry [3]–[5]. This is because these systems are considered ‘black boxes’ due to their lack of transparency – they are notoriously difficult to understand for humans because of their complex configurations and large model sizes. In addition to the lack of understandability, their correctness and robustness can also not be easily verified. For instance, the model might learn incorrect associations (*i.e.*, spurious correlations) from the input data, giving it the illusion of being performant without being able to generalize in practice¹. The model might also have fatal weaknesses that can be exploited by an adversary to evade detection². For the safety-critical environment of cybersecurity, the usage of such models is not ideal. In fact, black-box models are not even allowed in regulated fields unless they are supplemented with explanations [10], *e.g.*, courts do not consider model outputs as admissible evidence unless a forensic analyst is able to justify how the output links to the case [11]. Moreover, the “right to explanation” in the GDPR AI act also makes it tricky to deploy black-box models [12].

Explainable artificial intelligence (XAI)³ has been proposed to open the proverbial ‘black box’ by making the model internals more human understandable [13]. The first mention of XAI can be traced back to van Lent *et al.* [14] in 2004, while the field really started growing drastically after DARPA announced its XAI program in 2014 [15].

In this chapter, we systematize the increasingly growing microcosm of studies that develop and utilize XAI methods for security-specific target domains. We argue that the applications of XAI within cybersecurity are intrinsically different from other domains because cybersecurity works with practical use cases for safety-critical and high-stakes decision-making under adversarial settings. The lack of explainability is also an obstacle for deployment in cybersecurity [3]. In fact, explainability has arguably always been a core tenet in the design of ML pipelines for security [2], [16]. Even the seminal work by van Lent *et al.* uses XAI to explain the behaviour of AI-controlled entities in *military simulation games* [14].

In recent years, the security community has actively adopted XAI as a means to increase practitioners’ trust [17]. Numerous studies have applied existing XAI methods to security applications [18], [19]. However, recent studies have recognized their shortcomings in addressing the unique pain points of the security domain [20]. To this end, several security-specific XAI methods [21]–[23], and evaluation criteria [24], [25] have been proposed.

These recent developments have made XAI research within cybersecurity a fast-growing field: While there were only 42 articles about ‘*explainability*’, ‘*learning*’, and ‘*cybersecurity*’ in 2015, that number has since skyrocketed to 2600+ in 2021, according to

¹This is often referred to as the Clever Hans phenomenon [6].

²Adversarial machine learning studies these cases, see *e.g.*, [7]–[9].

³The terms ‘explainable artificial intelligence’, ‘explainable machine learning’, and ‘interpretable machine learning’ are used interchangeably in the literature.

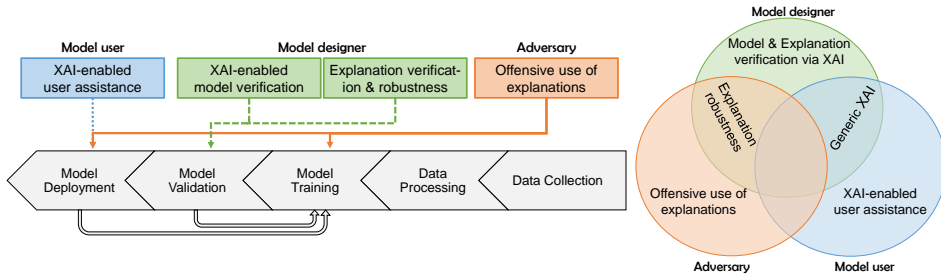


Figure 2.1: (Left): The interplay between application objectives and an ML pipeline. (Right): The interplay between stakeholders and application objectives.

Google Scholar. This literature is fragmented across several research communities (including machine learning, security, graphics, and software engineering) with no unified overview. Additionally, existing works often use different terminologies interchangeably, *e.g.*, explicable, accountable, transparent, and understandable, making it more difficult to track research developments.

To the best of our knowledge, this is the first SoK on explainable ML for cybersecurity⁴. By taking a step back, we synthesize insights from the vast body of fragmented literature and identify open areas to stimulate further research in this field.

Following the XAI definitions set forth by Roscher *et al.* [28], we identify three cybersecurity stakeholders, *i.e.*, (i) *model users*, (ii) *model designers*, and (iii) *adversaries* who utilize XAI for four distinct objectives within the security literature: (i) *XAI-enabled user assistance*, (ii) *XAI-enabled model verification*, (iii) *explanation verification & robustness*, and (iv) *offensive use of explanations*. The interplay between the stakeholders, objectives, and the stages of a typical ML pipeline are given in Figure 2.1. Particularly, the stakeholders remain central throughout our discussions. We also categorize the literature *w.r.t* the targeted domain (*e.g.*, intrusion detection), ML model, and XAI method. This taxonomy serves as a guide for finding related literature on XAI for cybersecurity.

After carefully reviewing 300+ papers, we found that XAI has been most commonly used for providing decision support to model users – 58% of the works are classified under *XAI-enabled user assistance*. User evaluation is a critical aspect of these studies to ensure that they are usable, and are sufficiently aligned with existing analyst workflows. However, user studies are conducted in only 14% of the cases, which is alarming since these methods aim to work directly with model users. We propose ideas for mitigating the lack of user studies in Section 2.5.

In addition, the stakeholders we identify have different competencies, and thus require tailored explanations [29], *e.g.*, model designers are typically experts in ML while model users are not. However, we identify several cases that either do not distinguish between model users and designers or do not specify any stakeholder. In contrast, model users and adversaries interact with the explanations in similar ways, but with opposing intent, requiring special manoeuvres to limit adversary access.

⁴Although Hariharan *et al.* [26] present a survey on XAI4cybersecurity, it covers only a small fraction of the literature. Also, the Explainable Security (XSec) framework proposed by Vigano *et al.* [27] is a non-conventional take on explainability, and does not embed the traditional XAI concepts within the security context.

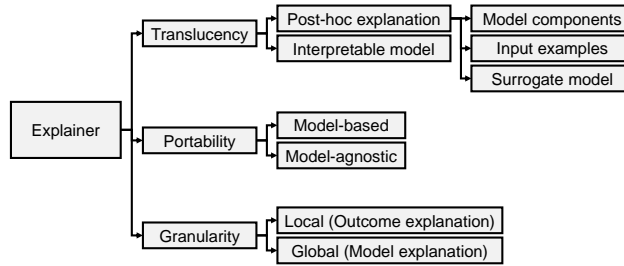


Figure 2.2: Ontology showing the key concepts within Explainable Artificial Intelligence (XAI).

Furthermore, the role of model designers is substantial in cybersecurity for ensuring the security of the model and its explanations. Yet, the reviewed literature only provides decision support to model designers in 22.3% of the cases. We tease out the role of model designers in Section 2.4: We present a walk-through tutorial of how model designers can utilize off-the-shelf XAI methods to detect and remove spurious features in a network attack detection scenario. The tutorial serves as a practical and easy starting point for security practitioners by showing three concrete ways to debug a black-box model via XAI. We show cases where the explanations are helpful, and cases where model designers may draw misleading conclusions instead. We discuss what can go wrong when explaining black-box models, and advocate for interpretability by design.

Organization. In Sections 2.2 and 2.3, we describe the scope and the proposed taxonomy. In Sections 2.3.1–2.3.4, we elaborate on the main takeaways from the reviewed literature. In Section 2.4, we demonstrate how model designers can use XAI to debug their models. In Section 2.5, we present open problems and recommendations for further XAI research within cybersecurity.

2.2. BACKGROUND AND METHODOLOGY

Explainable machine learning. ML pipelines either use white-box models, which are inherently *interpretable*, or use black-box models that are explained via *post-hoc explainability*. For instance, linear regression and decision trees are considered white-box, while neural networks and random forests are considered black-box. The output of a post-hoc explainability method is either a *surrogate model*, which is an interpretable model that approximates the black-box model, or is an explanation of the black-box model in terms of *model components* (e.g., feature importance) or *input examples* (e.g., counterfactuals). Additionally, an explanation can either elucidate how the model prediction is affected by a single data point (*local methods*), or by all data points (*global methods*). Note that interpretable models provide both local and global explanations. Furthermore, most post-hoc methods can be applied to any ML model, making them *model-agnostic*, while interpretable models are also referred to as *model-based explanations*. Figure 2.2 shows the various XAI terminologies, adapted from [30].

Method and Scoping. We synthesize available literature that uses XAI for (offensive and defensive) cybersecurity tasks. To this aim, we collect relevant literature, apply a

reflexive/inductive thematic analysis [31] to construct a taxonomy based on common themes (*i.e.*, stakeholders, application objectives), and classify the literature into those themes. The literature was collected by seven researchers. Each paper was investigated by at least two researchers independently and discussed with all authors during weekly meetings. The code books were updated as new categories emerged.

We collected peer-reviewed literature that has used explainable models to address cybersecurity problems since 2014, *i.e.*, post-DARPA, by searching popular scientific repositories (*e.g.*, IEEE Xplore) and top security conference proceedings (*e.g.*, Usenix). We used known search terms, *e.g.*, ‘*explainable*’, ‘*interpretable*’, ‘*artificial intelligence*’, ‘*cybersecurity*’, ‘*robust*’, ‘*offensive*’, ‘*attacks*’, and ‘*trustworthiness*’. To handle the fragmented literature, we expanded our search to include synonyms of the search terms at smaller security and non-security venues (see appendix 2.7.1 for the full list of venues). We also included older popular works that try to explain their model without explicitly using XAI, see *e.g.*, [16], [32], [33]. After carefully reviewing 300+ papers, we select 75 cybersecurity studies to build the taxonomy. Since it is impossible to cover all the available literature in the limited space, we chose representative works from each problem area. As such, there is some overlap with usable security, safety, and robustness literature, but we mainly focus on the use of XAI within cybersecurity.

2.3. SYSTEMATIZATION

Given an ML pipeline from data collection to model deployment, explainable ML is applied once a model becomes available. In the literature, XAI has been used to explain the model output to a human, either for supporting them in decision-making or for helping them understand whether the model works as intended. In addition, the adversarial threat landscape of cybersecurity suggests that XAI can also be used by an adversary to gain actionable information about the model in order to strengthen their attacks. This implies the existence of three stakeholders who interact with different phases of an ML pipeline and accomplish distinct objectives using XAI. We classify the literature based on the stakeholders, application objectives, target domain, model and explainer class. Figure 2.1 shows an overview of the stakeholder objectives that can be accomplished by applying XAI on a typical ML pipeline.

Stakeholders. We identify three stakeholders (explainees or users) who have different intents and expertise, and thus consume explanations for distinct objectives, even when interacting with the same ML model. The explanations are hence tailored to the specific needs of the stakeholders.

a) Model user is defined as a broad class of personnel who utilize the ML pipeline to improve the defence capacity of an organization, such as an analyst, developer, operator, domain expert, practitioner, or end-user. To this aim, a model user utilizes XAI techniques to better understand the output of a deployed model and to make informed decisions, *e.g.*, a security analyst uses XAI to analyze attack campaigns present in an intrusion alert dataset [34], or a malware analyst uses explanations to gain insights into why a binary was classified as malicious [35].

b) Model designer is responsible for engineering the ML pipeline used for a security

application, and consequently has a more intimate relationship with the model. A model designer utilizes XAI techniques during model training and validation to verify that the model works as intended, *e.g.*, a malware analyst uses explanations to investigate the causes of misclassifications, and to ensure that the model employs meaningful features [36]. Moreover, since the ML pipeline exists in an adversarial threat landscape, the model designer also ensures the safety and robustness of the model and its explanations [37].

c) Adversary is a human or an automated agent that intends to harm an organization by compromising the ML pipeline. An adversary exploits XAI techniques to formulate more efficient attacks, *e.g.*, by discovering weaknesses in the model [38]. In addition, an adversary may attack the XAI component of the ML pipeline itself to alter the generated explanations [39]. Depending on the attacker model, the adversary may interact with the explanations either during model training or after the model is deployed.

Application objectives. We classify the literature under four application objectives based on the intended use of the XAI technique – XAI is used to provide decision support to model users in (1); model designers in (2) & (3), and adversaries in (4).

(1) XAI-enabled user assistance covers techniques that are developed and utilized to support *model users* in making informed decisions, usually with the help of visual analytics dashboards. The explanations are meant to give control back to the user by helping them understand how/why an attack took place [40], and analyze attacker behavior [41]. Since it is the model designers who typically develop the explanations for model users, it is essential to include model users during the evaluation process to understand the explanation efficacy.

(2) XAI-enabled model verification studies techniques that are developed and utilized to help *model designers* debug and validate the correctness of the ML model. These explanations are usually more technical in nature. In the literature, XAI has primarily been used to discover spurious/faulty features by investigating a given black-box model using, *e.g.*, feature importance [36], [42] or surrogate models [22], [43].

(3) Explanation verification & robustness studies techniques that are developed and utilized to help *model designers* debug and validate the correctness & robustness of the XAI component in the ML pipeline. These methods test the correctness of post-hoc explanations under natural settings [44], and the robustness of explanations produced by post-hoc methods [37] and interpretable models [45] under adversarial settings [46].

(4) Offensive use of explanations studies how *adversaries* exploit insights provided by XAI techniques for enhancing their capabilities, *e.g.*, i) by using explanations to compromise the privacy of the model, and ii) by using explanations to compromise the integrity and availability of the model. These attacks can be deployed during model training (*e.g.*, poisoning attacks [47]), and model deployment (*e.g.*, privacy attacks [48]).

Target domain. We further classify the literature according to the cybersecurity target domain. In terms of defensive security domains, we cover: *malware detection, anomaly detection, intrusion detection, alert management, vulnerability discovery, asset prioritization, phishing detection, reverse engineering, traffic classification, and privacy protection*. In terms of offensive security domains, we cover: *privacy attacks (e.g., membership inference, model inversion, and model extraction), poisoning attacks (e.g., backdoor injec-*

Table 2.1: Code book for ML model classes. ‘w.b’ represents white-box, and ‘b.b’ represents black-box models.

Model class	Machine learning algorithms	w.b	b.b
CNN	Convolutional neural networks for image data, <i>e.g.</i> , ResNet, VGGnet, RPN, and inception network		✓
DNN	Deep neural networks for tabular data, <i>e.g.</i> , MLP, and autoencoder		✓
GNN	Graph neural networks, <i>e.g.</i> , GCN, and graph attention network		✓
SeqNN	Sequential neural networks, <i>e.g.</i> , RNN, LSTM, and transformers		✓
Kernel-SVM	Support vector machine with non-linear kernel		✓
Ensemble	Ensemble of models, <i>e.g.</i> , random forest, gradient boosting trees, and neural network ensembles		✓
LM	Linear models, <i>e.g.</i> , logistic (rule) regression, linear rank regression, and linear SVM	✓	
RBC	Rule-based classifiers, <i>e.g.</i> , decision trees, regular expressions, and BRCC	✓	
NB	Naive Bayes and its gaussian variant	✓	
Automata	Abstract computing machines, <i>e.g.</i> , Markov chains, and probabilistic deterministic finite automata	✓	
kNN	K-nearest neighbors	✓	✓
Unsupervised	Clustering algorithms, <i>e.g.</i> , HDBSCAN, kmeans, with(out) dimensionality reduction, <i>e.g.</i> , self-organizing maps, PCA, t-SNE	✓	✓

Table 2.2: Code book for explainer classes.

Explainer class	Explanation methods
SHAP	SHAP and its variants, <i>e.g.</i> , kernelSHAP
LIME	LIME and its variants, <i>e.g.</i> , graphLIME
LEMNA	Non-linear LIME variant for security applications
GNNExplainer	Explanation method for graph neural networks
Grad-based	Gradient-based methods, <i>e.g.</i> , GradCAM, saliency map, integrated gradients, and layer-wise relevance propagation
Activation	Neuron activations, activation maps and attention
Importance	Feature importance computed using tree-based splitting, feature permutation, and SOM-based dimensionality reduction
Exemplar	Example-based explanations, <i>e.g.</i> , kNN, prototypes, protoDash
Contrastive	Contrastive explanations, <i>e.g.</i> , counterfactuals
Anomaly-score	Custom metric capturing deviation from normalcy, <i>e.g.</i> , decoder reconstruction loss
Visual-explanation	Explanation based on visualizing model components or model output for human perception
Sur-RBC	Surrogate rule-based classifiers, <i>e.g.</i> , decision trees, decision lists, and rule sets
Sur-Mixture	Surrogate mixture linear regression model
Sur-Automata	Surrogate automaton model

tion), and *evasion attacks* (*e.g.*, *test-time adversarial perturbations*). The papers addressing generic non-security concepts, such as *anomaly detection* are further categorized according to the data sources they use, *e.g.*, image, binary, and network traffic. To the best of our knowledge, this is the first SoK to cover such a broad range of target domains.

Model & explainer class. Finally, we specify the ML models and XAI techniques used in the literature. The models are grouped according to the algorithm and the input data type accepted by the model (*e.g.*, tabular, images). The models are further classified coarsely as either black-box or white-box models, following the consensus of the ML community, see Table 2.1 for the model code book. The XAI techniques (called explainers henceforth) are categorized according to their underlying mechanism (*e.g.*, model components, examples, surrogate), see Table 2.2 for the explainer code book.

Table 2.3 provides a summary of the reviewed literature, which also showcases the

co-occurrence of certain models and explainers. Note that the classification in Table 2.1 reflects the general level of understanding provided by the model class, while Table 2.3 shows the actual usage of the model: Some studies explicitly treat white-box models as black-box for a model-free approach, see *e.g.*, [18], [49]. Other studies utilize an incomprehensible feature set (*e.g.*, by replacing feature names with integers), turning an interpretable model into a black box, see *e.g.*, [50].

The overview also helps us identify the misleading usage of certain terminologies. For instance, some works report their methods as being ‘interpretable’ while utilizing post-hoc explainers for black-box models, see *e.g.*, [22], [51], [52]. Strictly speaking, black-box models cannot be interpretable [28]. Thus, we categorize such works under post-hoc explainability. Note that it is possible to have an interpretable model that also uses a post-hoc explainer, but when a black-box model is explained via an interpretable model, it is called a surrogate model.

2.3.1. XAI-ENABLED USER ASSISTANCE

The fundamental objective for employing (explainable) ML methods in security workflows is to provide decision support to model users. In fact, practitioners have been trying to make their models understandable since before the popularity of XAI, see *e.g.*, [16], [53], [54]. Over the past decade, numerous XAI applications have arisen to support model users in their decision-making when interacting with a deployed model. The prominence of this objective is evident from the distribution of the available literature – 58% of the reviewed studies provide decision support to model users.

Within the reviewed literature, the explanations are generated for distinct purposes at different levels of expertise even when considering a single stakeholder. For instance, to assist *software developers* in understanding vulnerable code, some approaches simply highlight the lines of code that the model thinks are vulnerable [55]–[57], while others extract human-understandable rules from the vulnerable code that can serve as actionable intelligence for periodic scanning and control [53], [58]. Often, the goal of XAI-enabled user assistance is to explain the input data that captures attacker behavior dynamics. As such, these methods fall under two broad application scenarios: i) XAI is employed to provide assistance to model users for understanding model decisions and reducing their workload (*i.e.*, threat prioritization, false alarm reduction, user awareness), or ii) XAI is employed for the synthesis of new information (*i.e.*, expert knowledge creation, reverse engineering). Below we provide examples from the literature showing the different uses of explanations for different model users.

Triaging and threat prioritization. *Security practitioners* receive an enormous influx of cyber data that needs to be analyzed. XAI-enabled triaging techniques have been proposed to reduce analyst workload by redirecting their attention to critical events. This is crucial for Security Operations Center (SOC) analysts who often suffer from ‘alert fatigue’ caused by investigating large volumes of intrusion alerts on a daily basis [34]. Black-box methods can often not be applied since the analysts are under contractual obligation to review all alerts [59]. Instead, XAI techniques can reduce their workload via intelligent alert management while enabling them to justify the model’s decisions. The intuition is that a security analyst can use the explainable ML model as a ‘virtual assistant’ that

discovers meaningful patterns in large datasets and presents them to the analyst who can then make informed decisions about which data to triage and what actions to take next [60]. For example, Nadeem *et al.* [34] propose alert-driven attack graphs that show attacker strategies learned from intrusion alerts (discussed in Chapter 3). They utilize an interpretable suffix-based automaton model to learn contextually meaningful attack paths. *Security analysts* can triage critical alerts by selecting one of the attack graphs. Van Ede *et al.* [61] use an LSTM to learn the contextual meaning of alerts by capturing the correlation between them in an attention vector. Their system clusters attention vectors, capturing attack campaigns. *Security analysts* only need to analyze outlier and sampled events from emerging clusters, drastically reducing their workload. Similar approaches have been proposed to triage critical syslog entries for the forensic analysis of cyber attacks in a federated learning setup [51], and to efficiently allocate cyber resources for advanced persistent threat (APT) detection [62].

False alarm reduction. XAI can help *security practitioners* and other *model users* quickly disregard false alarms by explaining why the model made a prediction. For instance, Sopan *et al.* [63] propose a visual analytics dashboard to understand why an alert was raised. The dashboard provides an explanation of the alert in the form of an approximated decision path followed by the model and a list of important features. A similar approach is proposed in [70], [72]. Other works only show feature importance to help *security analysts* understand model predictions, *e.g.*, for malware detection [79]–[81], and anomaly detection [19], [50], [66], [68], [69]. In contrast, instead of explaining the predictions, de Bie *et al.* [49] have proposed a metric to help *security analysts* weed out false or untrustworthy predictions in regression models. They follow the intuition that instances close to each other typically have similar predictions. Thus, by comparing the prediction of a given instance with those of its *k*-nearest neighbours, a *model user* can identify whether the prediction can be trusted.

A handful of works have used anomaly scores to automatically discard anomalous events, thus reducing the cognitive load on *general model users*. For instance, Ardito *et al.* [67] use anomaly scores to support *medical staff* in detecting when an attack has occurred on a patient's e-health telemonitoring device. The autoencoder-based system avoids processing anomalies that can otherwise have devastating effects on a patient's health. Instead, it sends out a validation request to the medical staff. Similarly, Akerman *et al.* [103] use image reconstruction loss as an indication of whether artefacts in ADS-B video frames are false alarms. ADS-B is a protocol used by air traffic controllers to communicate with pilots regarding surrounding objects. By highlighting what might be false alarms, *pilots* can efficiently focus on the mission at hand.

User awareness & education. XAI has been utilized to increase the general awareness of different *model users* for insecure behaviour deterrence. For example, to keep *android users* safe, multiple works display warning signs with explanations for why an app was blocked or marked as malicious. The explanations are constructed from influential features extracted from apps' permission usage [16], [78], and network traffic [54].

XAI has also been used for warning *end-users* when they land on potential phishing websites to improve their overall Internet browsing behaviour [82]–[84]. For instance,

Phishpedia [84] employs logo detection to generate visual explanations in the form of insightful annotations on the websites. Chai *et al.* [83] take one step further by developing a multi-modal learning setup for more accurate phishing website detection. Their attention-based explanations highlight the URL characters, website text, and images that were relevant for the detection.

Finally, in order to raise awareness among *security analysts* regarding the impact of adversarial examples on a given ML model, Norton *et al.* [87] develop a visualization suite that lets them investigate the effect of various gradient-based adversarial attacks on image classifiers.

Expert knowledge creation. XAI can be used to synthesize human-understandable knowledge from black-box models. For instance, Mahdavifar *et al.* [75] extract a surrogate rule set from a pre-trained neural network that substitutes the knowledge base of their expert system. *Security analysts* interact with the expert system, which uses the rule sets to explain classification decisions. These rules are then used to classify unseen security incidents (*e.g.*, malware attacks and phishing attempts).

In order to address the lack of interpretability in vulnerability discovery methods [104], Zou *et al.* [58] and Yamaguchi *et al.* [53] extract human-understandable rules from code snippets that the model thinks are vulnerable. These rules are then used by *software developers* to detect vulnerabilities in previously unseen code bases. Next to this, counterfactual explanations have been used to automatically generate patches for vulnerable code. Wijekoon *et al.* [88] discover vulnerabilities in source code and proactively correct them with minimal changes necessary. To this end, they use LIME to find the nearest unlike neighbour as the most similar code snippet that is not vulnerable, which is then used as a patch.

Reverse engineering. Reverse engineering is commonly used in software engineering to convert black-box systems into white-box alternatives. However, there is a key difference between surrogate model learning and reverse engineering: The former extracts an interpretable model from a *black-box model*, while the latter either learns an interpretable model or uses a post-hoc explainer to *provide insights into the input data*. In this sense, reverse engineering methods can be considered as standalone tools that provide decision support to *model users* regarding input data (that capture behavior dynamics).

The most common application of reverse engineering is to consider a live system as a black box, collect traces from it, and learn an interpretable model from these traces. This model can be relatively easily visualized for model-based explanations about the black-box system. For instance, Fiterau *et al.* [86] apply protocol state fuzzing on servers that use the Datagram Transport Layer Security (DTLS) protocol in order to discover functional and non-conformance issues in several implementations. Discoverer [32] and Prospex [105] are two other popular systems for reverse engineering application-level specifications of network protocols. Similarly, Cho *et al.* [33] learn an automaton from botnet traffic to understand its Command and Control (C&C) channels; Lin *et al.* [65] learn an automaton from sensors of a water treatment plant to detect potential sensor malfunction, and Cao *et al.* [40] learn an automaton from the network traffic of a Kubernetes cluster to identify misbehaving pods.

Alternatively, a black-box model can be learned from the traces, and post-hoc explainers can be used to explain the properties of the traces. For instance, Gulmezoglu *et al.* [85] want to understand the type of web requests that leak side-channel information, such as performance counters and cache occupancy. This leakage enables website fingerprinting attacks in which users can be tracked by monitoring the unique combination of websites visited by their browsers. To this aim, they collect side-channel information leaked from different browsers, use it to learn several ML models, and use LIME and saliency maps to identify the leakiest web requests. Similarly, *malware analysts* can use reverse engineering to understand the relationship between malware samples. For instance, Nadeem *et al.* [41] build behavioural profiles of malware samples by clustering their network activities (discussed in Chapter 5). They visualize the overlap in the malware profiles in order to discover interesting malware capabilities. Similarly, Iadarola *et al.* [77] use gradient-based saliency maps to construct cumulative heatmaps for individual malware families that show visual differences between their disassembled code.

THE ROLE OF VISUALIZATIONS IN XAI

Visual explanations are the most common way to explain the inner workings of a black-box model. This is because human cognition prefers visual information over text for providing decision support [106]. The reviewed literature proposes several types of visual explanations, *e.g.*, a graph-based interpretable automaton model proposed by Lin *et al.* [65] that can directly be visualized for anomaly detection, and a context-based visual analytics dashboard proposed by Alperin *et al.* [18] that uses LIME and t-SNE for triaging vulnerabilities.

Visualizing the structure of tree-based models is another popular explanation method [63], [70], [74]. Sopan *et al.* [63] report that the security analysts found their visualization of an approximated decision path generally helpful. However, this is not always the case. Angelini *et al.* [74] propose a visual analytics system to explain the reason for malware detection by showing geo-locations of downloaded files and allowing a *malware analyst* to drill deeper into the individual paths of a random forest. However, simultaneously exploring the paths of ~ 100 decision trees does not make it any easier to decipher what the model is doing. Instead, it is preferable to provide different explanations based on the user's trust, *e.g.*, by providing less explanation when trust is high, and more explanation when trust is low [107].

Usability is an important consideration when designing decision-support tools for human analysts. Every explanation method has an associated cost in terms of its adaptation time. Even a simple XAI tool that plots reconstruction errors and lists top-k anomalies can cost analysts a full day to get used to it [19]. *Generally, simpler explanations are preferred, otherwise they can make the original task even more time-consuming* [108]. In other terms, complex visualizations contribute to cognitive load, subverting effective explanations. This is why the knowledge of existing analyst workflows is an important predictor in the successful deployment of XAI tools [64].

In addition, visualizations are not always equivalent to effective explanations. A model does not become interpretable just by virtue of visualizing it. For instance, the automaton model presented in [65] requires some level of expert knowledge to correctly interpret it. Similarly, the decision tree proposed in [72] is claimed to be interpretable

by default since it mimics human-level decision making, while it does not appear to be size-limited to actually be considered interpretable [109]. Furthermore, the explanations provided by DeltaPhish [82] are incomplete because they are limited to the linear coefficients of a single SVM, while it uses an ensemble of SVMs for different features.

Takeaway 1: *Visualization is not equivalent to effective explanation. XAI should reduce complexity, not add another layer of complex visualizations.*

EXPLANATION EVALUATION VIA USER STUDIES

XAI-enabled user assistance tools can be evaluated along several dimensions, *e.g.*, fidelity, understandability, efficiency, and construction cost [110]. Most of these criteria can be evaluated without human involvement. However, understandability involves multiple usability factors that can only be suitably evaluated with model users. This is tricky because analyst time is expensive [22]. Thus, many existing works focus on evaluating other aspects of explanations instead, *e.g.*, their fidelity and efficiency [19], [71], [93]. However, an explanation is unlikely to be used in practice if it is not understandable, even if it is robust and correct. Therefore, we advise bringing humans back in the loop by evaluating XAI-enabled user assistance tools with application-grounded (with experts) or human-grounded (with lay persons) user studies [111].

In order to subvert costs, qualitative analyses are often conducted in place of user studies [23], [34], [53], [57], [74]. This is problematic because of the involvement of multiple stakeholders – the XAI tools are typically developed by *model designers* for *model users*. In practice, these stakeholders have different expertise. We recommend avoiding qualitative analyses that only investigate the happy flows (successful explanations) in order to circumvent the possibility of cherry-picking [112].

Takeaway 2: *User studies are necessary to evaluate the usability of decision support tools. Yet, only 14% of the reviewed literature performs user studies with a median of 8 participants.*

THE IMPORTANCE OF STAKEHOLDER SPECIFICATION

We identified six cases within the reviewed literature where the roles of model users and designers were entangled [21], [22], [35], [71], [74], [76]. These methods assume that the same person is both, the designer and the user: 1) Angelini *et al.* [74] propose a visual analytics system for helping *malware analysts* handle ‘grey cases’ where a model produces a classification with low confidence. The intuition is that the explanations can either enhance the analyst’s confidence in the system if the explanations make sense, or *can trigger model improvement if they do not*. 2) Kyadige *et al.* [76] and 3) Mathews [35] explain the output of a malware detector to help *analysts* understand why a binary was classified as malicious, and *evaluate whether the model uses meaningful features*. 4) Wang *et al.* [71] use SHAP to help *security analysts* recognize the relationship between specific features and attack types, which *can guide the design of a more efficient intrusion detection system*. 5) LEMNA [21] and 6) DeepAID [22] are specialized XAI methods that address the unique challenges of the cybersecurity domain, *e.g.*, non-linear decision boundaries and concept drift [113]. LEMNA is a non-linear variant of LIME, while

DeepAID learns a surrogate automaton model that allows users to understand the back-box model and *improve it, if necessary*.

We also identified 17 cases where the intended stakeholder was left unspecified [33], [42], [55], [62], [66], [68], [79]–[82], [85], [86], [89], [91]–[94]. These methods appear to heavily focus on the fidelity of the explanations instead of their understandability, removing the human from the loop and potentially limiting their deployability.

Disentangling stakeholders is necessary for assessing how the proposed method translates to industry, since model users and designers are often distinct parties, working in different departments or even different organizations. Moreover, since model users and designers interact with distinct phases of the ML pipeline, they often have different expertise and require disparate explanations. For example, while both Russell *et al.* [55] and Chakraborty *et al.* [42] use activation to highlight vulnerable code, the former is intended for *model users* (explaining why a code snippet was considered vulnerable), and the latter is intended for *model designers* (making sure the model highlights correct code snippets). Even within the same domain and for the same stakeholder, the explanations can have contrasting uses, *e.g.*, within malware detection, some works use explanations to warn *smartphone users* of malicious apps on their phones [16], [54], [78], while other works provide more technical explanations to *malware analysts* regarding classifier decisions [79]–[81]. Thus, explanations meant for one type of user might be too vague or too technical for another user [29].

Takeaway 3: *Effective explanations are tailored to a specific user. Model users and designers usually have different expertise, and thus require disparate explanations. We encourage the community to specify their intended explanation stakeholders.*

2.3.2. XAI-ENABLED MODEL VERIFICATION

In fields other than cybersecurity, humans interact with AI systems with the assumption that they are near-perfect [114]. Thus, *faith* or *fidelity* is a major constituent of trust in the beginning, which is eventually replaced by reliance and predictability. The reverse is true for the adversarial threat landscape of cybersecurity: Reliance and predictability are important constituents of trust since these systems can be attacked. To this aim, *model designers* have a vital role in validating the safety and correctness of the ML pipeline in order to build trust with practitioners.

A defensive security model designer is generally concerned with two aspects of model verification: (i) The model is robust to adversarial perturbations. (ii) The model is generalizable and works as intended. The former is covered by the adversarial learning literature that aims to limit the possibility of evasion by making models robust to adversarial perturbations [7], [8]. Recent works have started to investigate the relationship between robustness and interpretability – early evidence suggests that robust models may be more interpretable than their non-robust counterparts [115], [116]. The intuition here is that robust models are smoother and can thus be more easily interpreted by humans. Nevertheless, further research is warranted to explore how XAI can guide the search for tamper-proof features used to train robust models.

The latter aspect of model verification fundamentally scrutinizes the generalizability of the model. Generalization is a highly desirable property in learning-based security

systems as they are meant to detect previously unseen threats. XAI has been used to detect spurious correlations – artefacts unrelated to the security task that allow the learning algorithm to create shortcuts for separating the classes, instead of actually solving the task. These artefacts make the model *seem performant* without being able to generalize in practice [101], [117]. In this systematization, we expand the traditional definition of spurious features to also include faulty features whose distributions are not representative of real-world cases [113]. In the literature, spurious/faulty feature detection is done via conformance checking, influential feature analysis, and surrogate model analysis.

Conformance checking. Comparing classifier decisions with some notion of ground truth can be used for model debugging. For instance, Kyadige *et al.* [76] and Chua *et al.* [89] compare model outputs with expert knowledge as a sanity check to ensure that the model works correctly. Specifically, given an RNN that recovers function types and signatures from decompiled binaries, Chua *et al.* [89] use post-hoc explainers to verify that the model is able to learn concepts comparable to an expert’s domain knowledge. To this aim, they use t-SNE to visualize semantically similar word embeddings, and saliency maps to understand which instructions are relevant for the recovery of the input functions. Nevertheless, many security applications struggle with obtaining ground truth, making conformance checking difficult in practice [113].

Influential feature analysis. Feature importance can be employed to investigate whether the model uses meaningful features. For example, Chakraborty *et al.* [42] use LEMNA to check whether the highlighted tokens meaningfully communicate why a code snippet was classified as vulnerable. Similarly, Reyes *et al.* [94] use SHAP and Ahn *et al.* [92] use feature permutation to select meaningful features for intrusion detection and network traffic classification, respectively.

Feature importance can also be used to investigate causes of misclassifications in order to improve the model. For example, Becker *et al.* [36] propose a visual analytics system that enables *malware analysts* to explore how the model views malware samples at different layers by clustering neuron activations. By visualizing the internal components of black-box models, malware analysts can identify sources of bias and misclassifications. Another example is from the domain of voice assistants: Chen *et al.* [91] design a more robust voice assistant by first using SHAP to identify the type of fuzzy words that cause a given tree ensemble-based wake-up word detector to become falsely triggered, and then proposing countermeasures to avoid it from happening. Within continual learning settings, CADE [23] explains the cause of performance degradation of a malware detector by reporting the features that are most affected by concept drift. It uses the contrastive explanation method: It perturbs features to see which combination increases the distance to the training data the most and thus is responsible for causing drift. Finally, Marino *et al.* [90] identify and correct the cause of missed detections and false alarms in IDS. They use adversarial examples that naturally serve as counterfactual explanations, showing the minimal changes required in feature values to correctly classify the (misclassified) security events. They expect that these insights will further improve their IDS performance. However, it is unclear how they avoid over-fitting since they utilize the knowledge of the test set to improve their model performance.

Surrogate model analysis. Interpretable surrogate models are inferred from black-box models that can directly be inspected for defects, *e.g.*, Han *et al.* [22] learn a surrogate automaton model, while Dolejš *et al.* [43] learn a rule-based surrogate model. In addition, Dolejš *et al.* [43] measure the interpretability of the surrogate model in terms of its behavioural similarity to the black-box model, *i.e.*, by checking whether they make similar mistakes.

THE RISKS OF POST-HOC EXPLAINABILITY

As it stands, the security literature heavily relies on performance metrics (*e.g.*, F1 score) as a means to conduct model verification. Goodhart's law dictates that when a measure becomes a target, it ceases to be a good measure [118]. This is evident from the abundant literature on adversarial learning, suggesting that merely relying on performance metrics is a dangerous strategy as the model might have fatal weaknesses that an adversary can exploit. Moreover, high performance on experimental data does not imply that these methods would generalize in practice. This is because the analysis is rarely conducted in operational settings due to excessive costs. Furthermore, security papers often skip details on the operationalization of the ML pipeline, making it difficult to know if any spurious/faulty features have been used. As such, feature attribution can be used for identifying spurious features [117]. In fact, spurious feature detection and removal should become more commonplace before deploying new models. We also recommend that publicly available models be supplemented with a verification analysis to enhance trust among practitioners. To assist *model designers*, we provide an illustrative tutorial of how they can debug their models using commonplace XAI tools in Section 2.4.

Having said that, model designers must also be aware of the risks of using post-hoc XAI for model verification: Post-hoc explanations are approximations of black-box models that either hide away details or learn different concepts altogether. For example, explanations based on feature importance often disagree on the same model prediction [119], suggesting that there is a mismatch between the explanations and what the model actually does. A similar observation has been made for surrogate models [120]. In fact, it is even possible to extract fair explanations from known unfair models. In regulated environments where companies are required to supplement their black-box models with explanations, they can be abused to perform '*fairwashing*' – promoting the false perception that a model is fair when it is actually not [120]. Therefore, it is advisable to opt for interpretable models. Where that is not possible, it is critical to establish an equivalence relationship between a model and its explanation, *e.g.*, by learning a certifiably equivalent surrogate model. For instance, Weiss *et al.* [121] and Koul *et al.* [122] extract equivalent deterministic finite automata from black-box RNNs. These works fall under the safety verification literature, see *e.g.*, [123]–[126].

Takeaway 4: *Regardless of the XAI method used to validate a model, it is vital for safety-critical applications to establish an equivalence relation between the model and its explainer. However, this is not yet common practice within cybersecurity.*

2.3.3. EXPLANATION VERIFICATION & ROBUSTNESS

Whilst using XAI for model verification, the explanations themselves need to be verified for correctness and robustness. *Model designers* are thus also responsible for conduct-

ing explanation verification to ensure the safety of the ML pipeline. This is an important line of work because XAI methods can sometimes trigger on input data patterns rather than on meaningful model behaviour. For instance, Adebayo *et al.* [95] reset the weights of a neural network to their initial random values and show that some gradient-based methods still use information from the input. Therefore, evaluating the fidelity of explanations becomes vital. Yet, it has sometimes been overlooked within the security literature, see *e.g.*, [36], [70]. In addition, qualitative analysis alone does not provide sufficient test coverage, and may even lead to cherry-picking [112]. Knowing that XAI methods can generate arbitrary explanations, objective evaluation criteria are required to ensure that (a) the explanation methods work [44], [93], [95], and (b) the explanations are robust to adversarial attacks [37], [45], [127]–[130]. Warnecke *et al.* [24] and Ganz *et al.* [25] are excellent starting points for evaluation criteria for a wide variety of post-hoc explainers under security settings. Their criteria include descriptive accuracy, sparsity, completeness, stability, efficiency, and robustness.

Fidelity evaluation. The explanation fidelity can be evaluated in several ways: Wickramasinghe *et al.* [93] test the fidelity of their attribution method by perturbing feature values and analyzing their impact on the explanations. Lin *et al.* [44] test the correctness of different saliency explanations by deliberately injecting artefacts in the input data to see if the explanations detect them. Specifically, they inject backdoor trigger patterns in input images that would naturally result in misclassifications by a CNN. These backdoor triggers serve as ground truth, *i.e.*, the backdoor features are primarily responsible for causing misclassifications, so a faithful explainer must be able to identify them.

Adversarial robustness. More importantly, XAI forms an additional attack vector for *adversaries* within the context of cybersecurity – both post-hoc explainers [39], [46], [52], [96]–[98] and interpretable models [131], [132] are sensitive to small adversarial perturbations. For instance, Ghorbani *et al.* [39] investigate the effect of adversarial perturbations on exemplars. Exemplars are samples from the training set whose features most resemble the instance to be explained. They find that while keeping the prediction equal, they can cause the top-3 exemplars to be entirely different for perturbed samples, implying that the perturbed samples enter a part of the model with drastically different latent features. Moreover, Dombrowski *et al.* [46] exploit the fragility of explanations to perform targeted attacks. They show that by adding imperceptible perturbations to the input image, the adversary can completely control the generated explanation. These studies identify three problematic traits of post-hoc explainers:

- Predictions and explanations can change tremendously under small perturbations;
- While keeping the explanation fixed, input samples can be perturbed to cause misclassifications;
- While keeping the prediction fixed, input samples can be perturbed to change the explanation.

The fact that models and explainers can be attacked independently opens up a new range of attacks. For instance, malware authors can evade detection while masking the

features that they used for evasion. In this case, the generated feature importance maps do not represent the features that are actually important for classification. Therefore, it is imperative that *model designers* robustify explainers against adversarial perturbations.

Recent works have started to investigate the robustness of post-hoc explainers: Alvarez-Melis *et al.* [37] investigate the smoothness of explanations around data points as a measure of robustness. Based on a local version of the Lipschitz constant, they show that the smoothness of model-agnostic explainers, *e.g.*, LIME and SHAP, can vary across datasets. They also show that gradient-based explanations are approximately four times smoother than LIME, suggesting that model-based explanations are more robust than their model-agnostic counterparts. For counterfactual explanations, Fokkema *et al.* [127] show that robust explainers cannot also be recourse sensitive⁵. This means that there will always be model inputs for which the explanations suggest modifications that do not end up changing the model's prediction⁶. As a solution, they suggest using multiple counterfactual explanations pointing in different directions.

A handful of works have also proposed robust variants of interpretable models, such as linear models and decision trees. For instance, Vos *et al.* [45] learn efficient and robust decision trees, while Hayes *et al.* [130] learn robust and differentially private logistic regression. Note that decision trees and logistic regression are considered interpretable as long as they are size-limited [109].

Takeaway 5: Along with ML models, explainers can also be attacked. In addition to fidelity testing, we recommend either using a robust explainer or conducting explanation verification under adversarial settings.

2.3.4. OFFENSIVE USE OF EXPLANATIONS

From the offensive security perspective, XAI can also provide decision support to *adversaries* for better attack formulation. As outlined by Papernot *et al.* [9], adversaries can target multiple phases of an ML pipeline, *e.g.*, the training phase for poisoning attacks, and the deployment phase for evasion and privacy attacks. XAI can further strengthen these capabilities by exposing sensitive details about the model. Adjusting the definitions proposed in [9] for XAI, we organize the nefarious uses of explainers through the lens of the classical confidentiality, integrity, and availability (CIA) triad [133]. Considering the added utility of XAI, attacks on *confidentiality* utilize explanations to expose the model structure or the data on which the model was trained. Attacks on *integrity* and *availability* utilize explanations to discover knowledge that adversaries can use to induce specific model outcomes of the adversary's choosing and thwart legitimate users from accessing meaningful model outputs.

Confidentiality attacks. Explanations provide additional information to *adversaries* about the inner workings of a deployed model, making it easier to reconstruct the model and the training data. This is why explanations are seen as privacy vulnerabilities [134].

⁵Recourse refers to a description of feature modifications required to change the model outcome.

⁶Note that the inputs for which this happens might not occur in practice and that this problem does not exist in linear unbounded models.

Yet, little work is done to generate privacy-preserving explanations [135]. In the literature, XAI has been used to strengthen model inversion [48], membership inference [100], and model extraction attacks [38].

Model inversion attacks enable adversaries to reconstruct training data from model predictions [9]. Adversaries can reproduce the model more accurately by utilizing explanations, *e.g.*, Zhao *et al.* [48] use an XAI-aware model inversion attack to successfully recover images from the training data. They show that feature importance maps generated from gradients and layer-wise relevance propagation (LRP) helped improve image reconstruction and led to an increase in model inversion performance compared to only using predicted probabilities.

Membership inference attacks assume that an adversary has some inputs and they want to predict whether they were used during training [9]. Shokri *et al.* [100] utilize gradient-based explanations to perform stronger membership inference attacks. They use the variance of saliency maps as a feature to infer membership and show that it works better than mere random guessing. The performance further improves when using the full explanation instead of only the variance.

Finally, model extraction attacks enable adversaries to recover the model's structure and parameters from predictions [9]. Kuppa *et al.* [38] utilize counterfactual explanations to improve their model extraction and membership inference attacks. They perform the model extraction attack by learning a surrogate model from known predictions and explanations. In addition, they perform membership inference by comparing the predictions of the target and counterfactual models to infer whether an input belonged to the training data.

Integrity and Availability attacks. Explanations provide additional knowledge to *adversaries* about the features to perturb in order to alter the correct functioning of the model. This can be done while the model is already deployed (*i.e.*, evasion attacks) or when the model is training (*i.e.*, poisoning and backdoor attacks).

Demetrio *et al.* [101] use integrated gradients to explain the importance assigned to the various fields of binary executables. They use this information to identify a few bytes in the malware header that need to be perturbed in order to successfully evade detection.

Poisoning attacks are specialized adversarial attacks where an adversary injects a small percentage of perturbed data to get some desired change in the learned model. Kuppa *et al.* [38] use counterfactual explanations to find the malware features that most heavily impact the classifier decision. They use this knowledge to craft adversarial training samples that efficiently poison the model.

Backdoor attacks are specialized poisoning attacks where the adversary makes the model sensitive to a pre-specified trigger. Severi *et al.* [47] use SHAP to craft backdoor triggers in malware detectors. Utilizing the explanation, they determine which features to poison, resulting in a success rate of up to three times higher than that of a greedy algorithm that does not use XAI. Similarly, Xu *et al.* [99] inject backdoors into GNNs by leveraging XAI techniques. They employ GNNExplainer to identify the parts of the graph to attack, and GraphLIME to identify the node features and values to change.

In two-player competitive games, Wu *et al.* [102] utilize XAI to exploit the weakness of an adversarial reinforcement learning agent. In such games, the agents take optimal

actions according to their policy function, which is often learned using self-play. Using saliency maps, the proposed adversarial agent observes which of their actions the opponent pays the most attention to, and alters them in the next time stamp, thus confusing and manipulating the opponent's actions.

Takeaway 6: *Uniquely attributed to the security domain, adversaries may abuse explanations to bolster their capabilities. Meanwhile, research on privacy-preserving explanations that are also robust to evasion attacks is almost non-existent.*

2.4. USE CASE: DEBUGGING A MALICIOUS NETWORK TRAFFIC DETECTOR VIA XAI

Sections 2.3.2 and 2.3.3 elucidate the critical role of model designers in ensuring the correctness and robustness of an ML pipeline. While XAI has been commonly directed towards model users, we argue that model designers can also greatly benefit from it. In this section, we present an illustrative tutorial on how model designers might use XAI for model verification. Specifically, we demonstrate how the investigation of influential features and misclassifications can identify problematic or spurious features. The experiments necessitate a sufficient understanding of post-hoc explainers for correct interpretation and highlight the expressive power of interpretable models.

We consider a *model designer* who learns an ML model to detect malicious botnet traffic on their company's network. They have some intuition of how a potential botnet-infected device might behave, and thus use XAI to validate whether the model follows that intuition, e.g., by checking whether it uses any spurious features or any strange artefacts from the training data. Note that we are only interested in finding spurious features: While the selection of tamper-resistant features is also an important problem, using XAI to discover such features remains an open problem, to the best of our knowledge.

Dataset selection. We use the open-source CTU-13 [136] as our experimental dataset. It has 13 scenarios, each containing both benign and malicious Netflow data. The malicious Netflows in each scenario are collected by monitoring virtual machines (VM) infected with real malware. Each Netflow has the following features: start time (StartTime), duration (Dur), protocol (Proto), source port (Sport), Netflow direction (Dir), destination port (Dport), state (State), source type of service (sTos), destination type of service (dTos), total packets (TotPkts), total bytes (TotBytes), and source bytes (SrcBytes). The dataset contains 64,855,215 benign and 1,535,374 malicious Netflows.

Experimental setup. We have developed a modular XAI pipeline in Python with six models and four explainers. Implementation details are given in appendix 2.7.2. We release the code for reproducibility⁷.

For the experiments, we train a gradient boosting machine (GBM) over all the features of the Netflow data, as described in [136]. The GBM achieves a balanced accuracy of 86.4%. While this model is arguably not state-of-the-art for detection purposes, it is

⁷XAI pipeline: <https://github.com/tudelft-cda-lab/xai-pipeline>

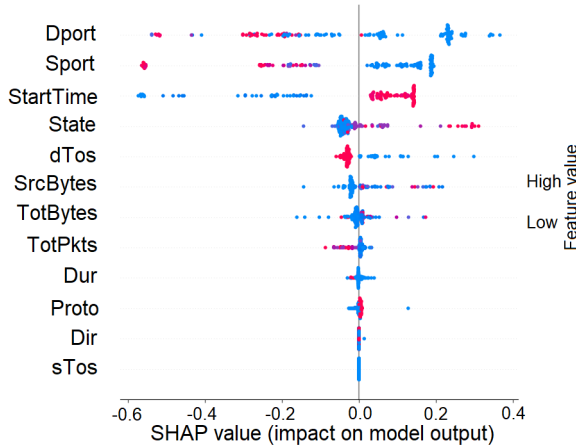


Figure 2.3: Global SHAP summary plot for the GBM. The y-axis shows the features ordered by their importance. For each feature, the Netflows are shown as dots with color representing the feature’s contribution towards model output. It shows the GBM uses Dport, Sport, and StartTime as the most important features.

a black box that concretely shows how improvements can be obtained via XAI. As such, the analysis described in this section can be applied to any black-box model⁸.

We use SHAP, LIME, and LEMNA to explain the predictions of the GBM. The global SHAP summary plot is given in Figure 2.3. The LEMNA explanations⁹ for the GBM are given in appendix 2.7.4. We also learn an interpretable decision tree (see Figure 2.4) to verify whether similar conclusions can be drawn from model-based and model-agnostic explanations. The decision tree has nine nodes and achieves a balanced accuracy of 83.6%, which is only slightly worse than the GBM. We generate explanations for 140 Netflows from the test set: 50 true positives (malicious), 50 true negatives (benign), 20 false positives (not malicious), and 20 false negatives (not benign).

2.4.1. XAI FOR DISCOVERING SPURIOUS CORRELATIONS

It is evident from the SHAP summary plot (Figure 2.3) that the GBM exhibits a strong reliance on the destination port, source port, and start time features. We also see this trend in the interpretable decision tree in Figure 2.4.

The reliance on the start time and source port features is problematic: Start time is problematic because it represents Unix time, so each new Netflow will have a vastly different feature value compared to the ones seen in the training data, negatively impacting the test accuracy and the model’s generalizability. For instance, a benign Netflow that the GBM considers as malicious with a probability of 65% suddenly becomes benign with a probability of 94% if we artificially perturb the start time to four weeks earlier. This implies that the model learns to predict *when a Netflow is generated*, rather than

⁸We recognize that the tutorial discusses a simple use case and that the features may have more complex relationships in reality. However, even this simple case occurs frequently in practice, as shown in [137].

⁹LEMNA is excluded from the analysis since it provides remarkably fewer insights for class distinction compared to SHAP and LIME.

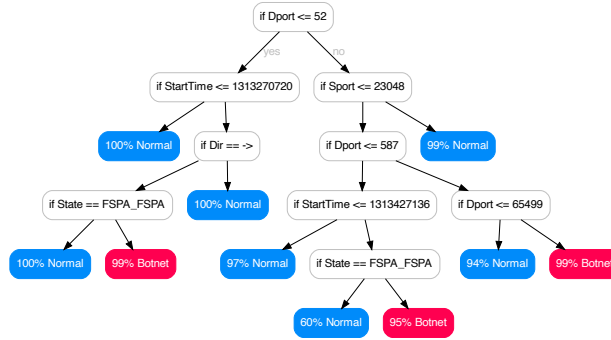


Figure 2.4: Decision tree for the CTU-13 dataset. It predominantly considers StartTime and Sport to differentiate between benign and malicious Netflows.

Feature	SHAP Value	Feature	Value	LIME Rule	Weight	Feature	SHAP Value
State = 54	0.2339	Sport	1703	Sport=1703	0.17	Dport = 3389	0.5387
SrcBytes = 186	-0.1756	Dport	80	Dport = 80	0.12	State = 16	0.0495
StartTime = 1313593252	-0.1210	TotPkts	8	TotPkts > 4.00	0.05	Sport = 4505	-0.0488
Dport = 80	0.1121	Proto	0	Proto=0	0.03	StartTime = 1313571534	-0.0475
Sport = 1703	-0.1113	TotBytes	492	271.50 < TotBytes <= 4...	0.02	dTos = 0	0.0274
dTos = 0	0.0465	State	54	State=54	0.01	TotPkts = 10	0.0209
TotPkts = 8	0.0408	Dir	2	Dir = 2	0.01	TotPkts = 1076	-0.0077
TotBytes = 492	-0.0295	Dur	8.96	0.13 < Dur <= 9.01	0.01	SrcBytes = 437	-0.0057
Proto = 0	0.0266	SrcBytes	186	83.50 < SrcBytes <= 1...	0.0	Dur = 60.95	0.0032
Dur = 8.96	-0.0247					Proto = 0	0.0
Dir = 2	0.0					Dir = 2	0.0
sTos = 0	0.0					sTos = 0	0.0

Figure 2.5: (Left): SHAP explanation for a false positive Netflow. (Middle): LIME explanation for a false positive Netflow. (Right): SHAP explanation for a false negative Netflow. Orange rows contribute positively, and blue rows contribute negatively towards the malicious label.

the Netflow's maliciousness.

Source port is problematic because it typically gets arbitrarily assigned by the operating system, and as such should not be indicative of malicious behaviour. However, the CTU-13 dataset uses only a small subset of VM-related port numbers [138], which inadvertently becomes indicative of malicious behaviour. This is a common shortcoming of lab-collected datasets [117]. Thus, it can also be considered an artefact of the experimental data.

It is noteworthy that start time and source port are perfectly valid features if the test set comes from CTU-13. Since we cannot expect real data to follow the same patterns as CTU-13, we consider them spurious features. This type of analysis is not common practice in the security literature: Several recent and relatively popular works utilize the identified faulty features, see *e.g.*, [139]–[141]. Since these features are tightly coupled with the prediction label, standard feature selection methods are unlikely to get rid of them. This is where XAI can help to detect spurious features, and even improve data collection methods [142].

The next logical step is to retrain the model without the spurious features. Doing so lowers the balanced accuracy of the GBM and decision tree to 74.4% and 58.1%, respec-

tively. We argue that this is an improvement since the faulty features were making the classifier *appear performant* without being able to generalize in practice. Because cyber data is often noisy, sole reliance on performance metrics is generally meaningless, especially when spurious features are involved. Therefore, we recommend that like ablation studies, the identification and removal of spurious features should become a fundamental step in the design of ML pipelines.

2.4.2. XAI FOR FINDING CAUSES OF MISCLASSIFICATIONS

We find that post-hoc explanations must be supplemented with input data statistics to make meaningful inferences regarding the causes of misclassifications. For instance, we analyze a randomly sampled false positive Netflow. The local SHAP explanation (see Figure 2.5a) shows a heavy reliance on the state value of 54 and source bytes of 186. This information in itself is likely insufficient for an analyst to understand why the model made this mistake. However, combining this information with an analysis of the training data reveals that these feature values appear almost exclusively in malicious samples, thus identifying the cause of the false positive.

In another example, we analyze a randomly sampled false negative Netflow. The local SHAP explanation (see Figure 2.5c) shows a substantial reliance on the destination port 3389, which is associated with the remote desktop protocol (RDP). Internet-facing RDP servers commonly fall victim to cyber attacks¹⁰, making it a likely indicator of suspicious activity. Yet strangely, the port 3389 has contributed heavily towards the opposite. Analyzing the training data reveals that RDP is mostly used by benign hosts in the CTU-13 dataset, due to which the model incorrectly classifies a malicious Netflow as benign. These examples reveal what appear to be sampling and confounding biases in the CTU-13 dataset.

Takeaway 7: *Feature importance explanations do not provide the full picture in isolation. Instead, actionable insights can be obtained by combining the input data together with post-hoc explanations.*

2.4.3. UTILITY OF DIFFERENT XAI TYPES

All explanations are not created equal. Since XAI is meant to explain the behaviour of a model, testing the predictability of the model on a new (previously unseen) data instance, given a few explanations provides an estimate of the explanation's utility. In this sense, there is a clear divide between interpretable models and post-hoc explanations.

For a given interpretable model, such as the decision tree in Figure 2.4, it is almost trivial to predict how a new instance will be classified by following the decision path. However, since post-hoc explanations are mere approximations of the black-box model, it is difficult to predict how the GBM would classify a new instance, given its LIME and SHAP explanations. For instance, the local SHAP explanations provide feature importance with equality relationships (*e.g.*, see Figure 2.5a), which makes it impossible to predict how a new instance will be classified, even if it resembles the instances for which explanations are already available. This is because the explanations do not reveal the impact of slight feature perturbations on the classification. We encountered almost the

¹⁰<http://darktrace.com/blog/botnet-malware-remote-desktop-protocol-rdp-attack>

same problem for LIME even though it considers a local neighbourhood to prevent this very issue.

Moreover, post-hoc explainers compute their local neighbourhoods differently, causing explanations for the same model prediction to differ. Going back to the false positive example, SHAP (Figure 2.5a) heavily relies on the state feature, while LIME (Figure 2.5b) assigns a very low importance to it. Also, while SHAP considers dTos to be important, it does not even appear in LIME. This disagreement problem between feature attribution methods has recently been discussed by Krishna *et al.* [119]. Based on their metric, there is a 25.5% disagreement rate between the top-3 features of SHAP and LIME for our 140 Netflows. This exemplifies the mismatch between the black-box model and its explanations and makes a strong case for learning interpretable models from the get-go.

The correct interpretation of post-hoc explanations often relies on how well the explainee understands the underlying mechanisms of the method, reiterating the importance of user studies in explanation evaluation. For instance, while both local-SHAP and LIME show feature importance, their explanation interpretation can be very different. We found that LIME assigns very low weights to all features for almost all the Netflows. This does not imply that similar Netflows should have the same label, as one would intuitively expect, but rather that LIME has low confidence about the prediction given its local surroundings. Thus, an unsuspecting analyst might draw misleading conclusions by overly relying on intuition rather than the understanding of the method [112].

Takeaway 8: *LIME and local-SHAP are both feature attribution methods but their interpretations can be different. Working knowledge of post-hoc explainers is cardinal for correctly interpreting the explanations.*

2.5. DISCUSSION AND OPEN PROBLEMS

While more than half of the reviewed literature focuses on model users, there is limited research on methods that model adversary behavior – the vast majority of the literature aims to explain model decisions, instead of generating threat intelligence regarding adversary behavior. We believe that this is a missed opportunity, and that there should be more research into this topic. We present several solutions to fill this gap in Chapters 3 – 6. Below, we provide broader recommendations for future research directions:

User study crisis. The lack of qualitative validation among decision support papers is alarming. While *model users* are the most common consumers of explanations in the security literature, they have regularly been excluded from the evaluation process (*Takeaways 1-3, 8*). The evaluation of robustness and fidelity does not guarantee usability, which is arguably an equally important trait of good explanations. However, usability is rarely taken into account when designing evaluation criteria for effective security explanations, see *e.g.*, [24], [25]. Since analyst time is expensive, it may be beneficial to develop proxy tasks and metrics on which to evaluate new research instead. A handful of studies have incorporated human cognition in their metric definition. For example, Islam *et al.* [143] quantify the complexity of post-hoc explanations in terms of cognitive chunks, and Dolejš *et al.* [43] quantify the added opaqueness of explanations *w.r.t.* known interpretable models. Alternatively, in the absence of security practitioners,

newly developed tools could be peer-reviewed regarding their usability, *e.g.*, during conference artefact evaluation sessions. Furthermore, disentangling and specifying stakeholders should also provide clarity regarding the intended subjects for user studies.

Robustness vs. interpretability. The role of *model designers* is minimized in the security literature with merely 22.3% of the literature focused on model & explainer verification (*Takeaways 4-5, 7-8*). Since trust manifests inherently differently in the security domain, specialized XAI methods are needed to bolster practitioner trust in ML pipelines. While the tutorial in Section 2.4 helps model designers get started with XAI-enabled model verification, the role of XAI in tamper-resistant feature selection and robust model learning remains unclear. Another related question is regarding the relationship between robustness and interpretability: Initial research eludes to robust models being more interpretable than non-robust models [115], requiring further research in this direction.

Price of interpretability. If interpretable surrogate models are to be used for model verification, they must be certifiably equivalent to their black-box parent models for the evaluation to be meaningful. In this sense, directly learning a robust interpretable model (as opposed to learning a black-box model explained by a surrogate) may prove more helpful in establishing trust. Yet, only 25.9% of the studies we reviewed adopt interpretable models, while the majority of them focus on applying post-hoc explainability. The discussion regarding the '*price of interpretability*' (measuring the trade-off between explainability and performance) [144] requires special considerations in cybersecurity. We believe that the presence of an adversary and the prevalence of spurious features will likely make this trade-off less pronounced compared to other fields. However, further research is warranted in this area. Furthermore, it may be possible to exploit the power of post-hoc explanations without losing interpretability: Black-box models may be used as a benchmark to guide the search for better interpretable models. Post-hoc explanations can provide actionable intelligence regarding features and parameters for interpretable models. In this way, we view post-hoc explainers and interpretable models as complementary methods rather than as alternatives.

Privacy-preserving explanations. In addition to attacking the XAI module, adversaries can also utilize explanations, much like *model users*, but with a devious intent (*Takeaway 6*). This makes it difficult to provide explanations to model designers and model users without the adversaries also taking advantage of them. There is some preliminary work that studies the trade-off between explainability and privacy in order to select privacy-preserving explanations [135]. However, such explanations could still be used to bolster attacks on model integrity and availability. Therefore, this is also an urgent avenue for future research.

2.6. CONCLUSIONS

We systematize available research that utilizes explainable models for solving security problems. We identify 3 cybersecurity stakeholders (users) that employ XAI for 4 re-

search objectives (uses) within a typical ML pipeline. Distilled from a diverse body of literature, this overview streamlines existing research on explainability within cybersecurity, and provides a starting point for practitioners.

While most of the reviewed literature provides XAI methods for model users, they are usually eliminated from the evaluation of the explanations. Also, there are limited studies that attempt to understand adversary behavior for the generation of threat intelligence. We also found evidence that the security literature does not always disentangle model users and designers. In addition, only 22.3% of the security literature focuses on model & explanation verification. This is problematic because model designers have a critical role in ensuring the correctness and security of an ML pipeline. With regards to model correctness, we specifically provide a walk-through tutorial of how model designers can successfully detect and discard spurious features using SHAP & LIME. At the same time, the example also exposed the disagreement problem between local explanations and showed that SHAP & LIME have different interpretations. Thus, model designers must have a working knowledge of the explanation method in order to draw correct conclusions.

Moreover, adversaries can not only attack the XAI component but can also utilize explanations to compromise the confidentiality, integrity, and availability of a model. Meanwhile, research on limiting these abuses is almost non-existent. Finally, the lack of user validation in XAI-enabled user assistance, the shortage of automated threat intelligence generation studies, and the scarcity of interpretability by design shows the substantial margins of improvement within the field of XAI for cybersecurity.

Acknowledgments. We thank Daniël Meinsma for his contributions to the literature review, and the anonymous reviewers for their feedback. This work was made possible by TTW VIDi project 17541 (LIMIT) and EU H2020 project 952647 (AssureMOSS).

2.7. SUPPLEMENTARY MATERIAL

2.7.1. SELECTED LITERATURE

The literature related to *explainability* and *cybersecurity* has increased dramatically since 2014, see Figure 2.6. This literature is fragmented across various Computer Science domains. Table 2.4 provides a list of venues used for literature selection.

2.7.2. XAI PIPELINE DESIGN

We develop a modular XAI pipeline in Python (since it has in-built support for many popular models and explainers). The pipeline has three components: (1) The parser parses the input data (train and test) in either CSV or NumPy array format. The user can specify to the parser which feature fields should be read by means of providing a configuration file for the parser. (2) The classifiers are implemented as a wrapper over the ML algorithms provided by `scikit-learn`. We currently support decision trees, logistic regression, explainable boosting machines, random forests, gradient boosting machines, and SVMs. The wrapper specifies the ML algorithm and its hyperparameters. (3) Similarly, the explainers are also implemented as wrapper functions and currently provide support for SHAP, LIME, LEMNA, and ELI5. The modules can be extended for added

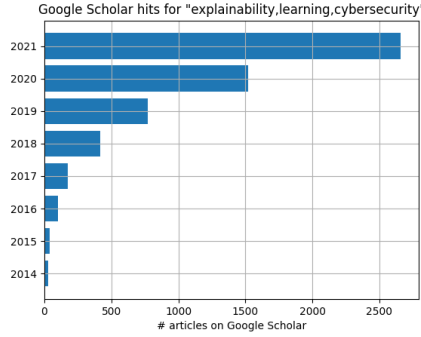


Figure 2.6: Prevalence of XAI literature in cybersecurity from 2014-2021.

Feature	SHAP Value	Feature	Value	LIME Rule	Weight	Feature	Coefficient
State = 54	0.2339	Sport	1703	Sport=1703	0.17	dTos	1.0
SrcBytes = 186	-0.1756	StartTime	1313593252	1313537772.00 < Start...	0.14	sTos	2.46E-10
StartTime = 1313593252	-0.1210	Dport	80	Dport = 80	0.12	StartTime	4.53E-11
Dport = 80	0.1171	TotPkts	8	TotPkts > 4.00	0.05	Sport	-4.33E-11
Sport = 1703	-0.1113	Proto	0	Proto=0	0.03	Dir	-4.06E-11
dTos = 0	0.0465	TotBytes	492	271.50 < TotBytes <= 4...	0.02	State	3.77E-11
TotPkts = 8	0.0408	State	54	State=54	0.01	Dur	2.07E-11
TotBytes = 492	-0.0295	Dir	2	Dir = 2	0.01	Proto	-1.93E-11
Proto = 0	0.0266	Dur	8.96	0.13 < Dur <= 9.01	0.01	Dport	-5.96E-12
Dur = 8.96	-0.0247	sTos = 0	0.0				
Dir = 2	0.0	SrcBytes	186	83.50 < SrcBytes <= 1...	0.0		
sTos = 0	0.0						

Figure 2.7: Post-hoc explanations for a false positive Netflow (including spurious features). (Left): SHAP. (Middle): LIME. (Right): LEMNA.

Feature	SHAP Value	Feature	Value	LIME Rule	Weight
State = 54	-0.3583	TotPkts	8	TotPkts > 4.00	0.13
Dur = 8.96	-0.2450	State	54	State=54	0.11
TotBytes = 492	-0.1873	Dport	80	Dport = 80	0.10
SrcBytes = 186	-0.1518	Dur	8.96	0.13 < Dur <= 9.01	0.10
Dport = 80	0.0850	TotBytes	492	271.50 < TotBytes <= 4...	0.06
dTos = 0	0.0569	SrcBytes	186	83.50 < SrcBytes <= 1...	0.05
TotPkts = 8	0.0359	sTos	0	sTos <= 0.00	0.0
Proto = 0	0.0023	dTos	0	dTos <= 0.00	0.0
Dir = 2	0.0023	Proto	0	Proto=0	0.0
sTos = 0	0.0	Dir	2	Dir = 2	0.0

Figure 2.8: Post-hoc explanations for a false positive Netflow (after removing spurious features). (Left): SHAP. (Right): LIME.

Feature	SHAP Value	Feature	Value	LIME Rule	Weight	Feature	Coefficient
Dport = 3389	0.5387	Dport	3389	Dport = 3389	0.18	Dir	0.2574
State = 16	0.0495	StartTime	1313571534	1313537772.00 < Start...	0.13	Proto	0.2440
Sport = 4505	-0.0488	Sport	4505	Sport=4505	0.09	Dport	0.2404
StartTime = 1313571534	-0.0475	TotPkts	10	TotPkts > 4.00	0.07	Sport	0.2035
dTos = 0	0.0274	State	16	State=16	0.04	Dur	0.1680
TotPkts = 10	0.0209	Proto	0	Proto=0	0.03	State	0.1116
TotBytes = 1076	-0.0077	SrcBytes	437	SrcBytes > 186.0	0.03	StartTime	0.0810
SrcBytes = 437	-0.0057	TotBytes	1076	TotBytes > 494.25	0.02	sTos	0.0241
Dur = 60.95	0.0032	Dir	2	Dir = 2	0.01	dTos	-0.0179
Proto = 0	0.0	Dur	60.95	Duration > 9.01	0.01		
Dir = 2	0.0						
sTos = 0	0.0						

Figure 2.9: Post-hoc explanations for a false negative Netflow (including spurious features). (Left): SHAP. (Middle): LIME. (Right): LEMNA.

Feature	SHAP Value	Feature	Value	LIME Rule	Weight
Dport = 3389	0.1542	Dport	3389	Dport=3389	0.20
SrcBytes = 437	0.0707	TotBytes	1076	TotBytes > 494.25	0.15
TotBytes = 1076	-0.0591	TotPkts	10	TotPkts > 4.00	0.14
dTos = 0	0.0394	State	16	State=16	0.04
State = 16	0.0312	SrcBytes	437	SrcBytes > 186.0	0.03
TotPkts = 10	0.0200	Dir	2	Dir=2	0.01
Dur = 60.95	-0.0153	sTos	0	sTos <= 0.0	0.0
Proto = 0	-0.0014	dTos	0	dTos <= 0.0	0.0
Dir = 2	0.0003	Proto	0	Proto=0	0.0
sTos = 0	0.0	Dur	60.95	Duration > 9.01	0.0

Figure 2.10: Post-hoc explanations for a false negative Netflow (after removing spurious features). (Left): SHAP. (Right): LIME.

Table 2.4: Venues explored for literature discovery from various domains.

Domain	Type	Venue
Cybersecurity	Conference	ACM Conference on Computer and Communications Security (CCS)
Cybersecurity	Conference	Asia Conference on Computer and Communications Security (AsiaCCS)
Cybersecurity	Conference	European Symposium on Research in Computer Security (ESORICS)
Cybersecurity	Conference	European Symposium on Security and Privacy (Euro S&P)
Cybersecurity	Conference	IEEE Symposium on Security and Privacy (S&P)
Cybersecurity	Conference	International Conference on Security and Privacy
Cybersecurity	Conference	Italian Conference on Cybersecurity (ITASEC)
Cybersecurity	Conference	Network and Distributed System Security (NDSS)
Cybersecurity	Conference	USENIX Security Symposium
Cybersecurity	Journal	Computers and Security
Cybersecurity	Journal	IEEE Transactions on Dependable and Secure Computing (TDSC)
Cybersecurity	Journal	IEEE Transactions on Information Forensics and Security (TIFS)
Cybersecurity	Journal	IEEE Transactions on Networks and Systems Management (TNSM)
Cybersecurity	Workshop	ACM workshop on Artificial Intelligence and Security (AISeC) @ CCS
Cybersecurity	Workshop	ACM workshop on Wireless Security and Machine Learning
Cybersecurity	Workshop	AI-enabled Cybersecurity Analytics and Deployable Defense (AI4Cyber)
Cybersecurity	Workshop	IEEE Symposium on Visualization for Cybersecurity (VizSec) @ VIS
Cybersecurity	Workshop	Machine Learning for Cyber Security (MLCS) @ ECML/PKDD
Cybersecurity	Workshop	Workshop on Artificial Intelligence and Cybersecurity (AI-Cybersec)
Cybersecurity	Book	Malware Analysis using Artificial Intelligence and Deep learning (Springer)
Machine learn.	Conference	AAAI Conference on Artificial Intelligence
Machine learn.	Conference	ACM Conference on Knowledge Discovery & Data Mining (SIGKDD)
Machine learn.	Conference	Conference on Neural Information Processing Systems (NeurIPS)
Machine learn.	Conference	International Conference on Computer Vision
Machine learn.	Conference	International Conference on Intelligence Virtual Agents
Machine learn.	Conference	International Conference on Machine Learning (ICML)
Machine learn.	Conference	International Conference on Pattern Recognition
Machine learn.	Conference	International Joint Conference on Artificial Intelligence (IJCAI)
Machine learn.	Conference	International Joint Conference on Neural Networks (IJCNN)
Machine learn.	Journal	Advances in Intelligent Systems and Computing (Springer)
Machine learn.	Journal	Human-Intelligent Systems Integration (Springer)
Machine learn.	Journal	Nature Machine Learning
Machine learn.	Journal	Neural Computing and Applications (Springer)
Computer Sci.	Conference	ACM Symposium on High-Performance
Computer Sci.	Conference	Parallel and Distributed Computing (HPDC)
Computer Sci.	Conference	Conference on Human Factors in Computing Systems (CHI)
Computer Sci.	Conference	International Conference on Enabling Technologies (WETICE)
Computer Sci.	Conference	International Conference on Human System Interactions (HSI)
Computer Sci.	Journal	ACM Computing Surveys
Computer Sci.	Journal	Annual Conference on Industrial Electronics Society (IECON)
Computer Sci.	Journal	Electronics (MDPI)
Computer Sci.	Journal	Expert Systems with Applications (Science Direct)
Computer Sci.	Journal	IEEE Access
Computer Sci.	Journal	Lancet Digital Health (Elsevier)
Computer Sci.	Journal	Procedia Computer Science (Science Direct)
Computer Sci.	Journal	Quality and Reliability Engineering (Wiley)
Software Engg.	Conference	ACM Joint European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE)
Software Engg.	Conference	ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)
Software Engg.	Journal	ACM Transactions on Software Engineering
Software Engg.	Journal	ACM Transactions on Software Engineering and Methodology
Software Engg.	Workshop	International workshop on Continuous Software Evaluation and Certification (IWCSE) @ ARES

support of custom parsers, models, and explainers. For the sake of reproducibility, the pipeline saves the model, predictions, and explanations in a file.

2.7.3. LEMNA IMPLEMENTATION

We based our implementation of LEMNA on the code by Warnecke *et al.* [24]. For the explanation generation, we use the following settings: $N = 500$, $K = 6$, $S = 10$. The values of N and K are based on the original LEMNA paper. We do not need fused Lasso since our features do not have a temporal structure. Therefore we set S to a high value, effectively turning off the fusing effect. We expect LEMNA to perform better on tabular data when using feature discretization. However, optimizing LEMNA is out of the scope of this work as we are only using existing methods for model debugging.

2.7.4. EXPLANATIONS FROM THE TUTORIAL

Figures 2.7 and 2.8 show the post-hoc explanations for the false positive Netflow with and without the identified spurious features, respectively. Figures 2.9 and 2.10 show the post-hoc explanations for the false negative Netflow with and without the identified spurious features, respectively.

REFERENCES

- [1] A. Nadeem, D. Vos, C. Cao, *et al.*, “SoK: Explainable Machine Learning for Computer Security Applications”, in *IEEE European Symposium on Security and Privacy (Euro S&P)*, IEEE, 2023.
- [2] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, “Learning and classification of malware behavior”, in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Springer, 2008, pp. 108–125.
- [3] A. Surkov, V. Srinivas, and J. Gregorie, *Unleashing the power of machine learning models in banking through Explainable Artificial Intelligence (XAI)*, Jun. 2022. [Online]. Available: <https://www2.deloitte.com/us/en/insights/industry/financial-services/explainable-ai-in-banking.html>.
- [4] G. Apruzzese, P. Laskov, E. M. de Oca, *et al.*, “The role of machine learning in cybersecurity”, *Digital Threats: Research and Practice*, 2022.
- [5] B. Hale, D. L. Van Bossuyt, N. Papakonstantinou, and B. O’Halloran, “A zero-trust methodology for security of complex systems with machine learning components”, in *International design engineering technical conferences and computers and information in engineering conference*, American Society of Mechanical Engineers, vol. 85376, 2021.
- [6] L. Samhita and H. J. Gross, “The “Clever Hans phenomenon” revisited”, *Communicative & integrative biology*, vol. 6, no. 6, 2013.
- [7] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, “Adversarial machine learning attacks and defense methods in the cyber security domain”, *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.
- [8] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning”, *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

- [9] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, “SoK: Security and privacy in machine learning”, in *IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2018, pp. 399–414.
- [10] P. Giudici and E. Raffinetti, “Explainable AI methods in cyber risk management”, *Quality and Reliability Engineering International*, 2021.
- [11] F. Casino, T. K. Dasaklis, G. P. Spathoulas, *et al.*, “Research Trends, Challenges, and Emerging Topics in Digital Forensics: A Review of Reviews”, *IEEE Access*, vol. 10, pp. 25 464–25 493, 2022.
- [12] B. Goodman and S. Flaxman, “European Union regulations on algorithmic decision-making and a “right to explanation””, *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.
- [13] T. Miller, “Explanation in artificial intelligence: Insights from the social sciences”, *Artificial intelligence*, vol. 267, pp. 1–38, 2019.
- [14] M. Van Lent, W. Fisher, and M. Mancuso, “An explainable artificial intelligence system for small-unit tactical behavior”, in *Proceedings of the national conference on artificial intelligence*, 2004, pp. 900–907.
- [15] G. D. A. DW, “DARPA’s explainable artificial intelligence program”, *AI Mag*, vol. 40, no. 2, p. 44, 2019.
- [16] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket”, in *Network and Distributed System Security*, vol. 14, 2014, pp. 23–26.
- [17] S. Samtani, M. Kantarcioglu, and H. Chen, “Trailblazing the artificial intelligence for cybersecurity discipline: A multi-disciplinary research roadmap”, *ACM Transactions on Management Information Systems (TMIS)*, vol. 11, no. 4, pp. 1–19, 2020.
- [18] K. B. Alperin, A. B. Wollaber, and S. R. Gomez, “Improving Interpretability for Cyber Vulnerability Assessment Using Focus and Context Visualizations”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2020, pp. 30–39.
- [19] L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, “Explaining anomalies detected by autoencoders using Shapley Additive Explanations”, *Expert Systems with Applications*, vol. 186, p. 115 736, 2021.
- [20] J. N. Paredes, J. C. L. Teze, G. I. Simari, and M. V. Martinez, “On the Importance of Domain-specific Explanations in AI-based Cybersecurity Systems (Technical Report)”, *arXiv preprint arXiv:2108.02006*, 2021.
- [21] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, “LEMNA: Explaining Deep Learning Based Security Applications”, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, 2018, pp. 364–379.
- [22] D. Han, Z. Wang, W. Chen, *et al.*, “DeepAID: Interpreting and improving deep learning-based anomaly detection in security applications”, in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3197–3217.

- [23] L. Yang, W. Guo, Q. Hao, *et al.*, “CADE: Detecting and explaining concept drift samples for security applications”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 2327–2344.
- [24] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, “Evaluating Explanation Methods for Deep Learning in Security”, in *IEEE European Symposium on Security and Privacy (Euro S&P)*, Sep. 2020, pp. 158–174.
- [25] T. Ganz, M. Härterich, A. Warnecke, and K. Rieck, “Explaining Graph Neural Networks for Vulnerability Discovery”, in *ACM Workshop on Artificial Intelligence and Security*, 2021, pp. 145–156.
- [26] S. Hariharan, A. Velicheti, A. Anagha, C. Thomas, and N. Balakrishnan, “Explainable Artificial Intelligence in Cybersecurity: A Brief Review”, in *International Conference on Security and Privacy*, IEEE, 2021, pp. 1–12.
- [27] L. Viganò and D. Magazzeni, “Explainable security”, in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 293–300.
- [28] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, “Explainable machine learning for scientific insights and discoveries”, *IEEE Access*, vol. 8, pp. 42 200–42 216, 2020.
- [29] M. Blumreiter, J. Greenyer, F. J. C. Garcia, *et al.*, “Towards self-explainable cyber-physical systems”, in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, IEEE, 2019, pp. 543–548.
- [30] A. Adadi and M. Berrada, “Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)”, *IEEE Access*, 2018.
- [31] V. Braun and V. Clarke, “Using thematic analysis in psychology”, *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.
- [32] W. Cui, J. Kannan, and H. J. Wang, “Discoverer: Automatic Protocol Reverse Engineering from Network Traces”, in *Proc. of the USENIX Security Symposium*, 2007, pp. 1–14.
- [33] C. Y. Cho, D. Babić, E. C. R. Shin, and D. Song, “Inference and analysis of formal models of botnet command and control protocols”, in *Proceedings of the 2010 ACM SIGSAC Conference on Computer and Communications Security*, 2010, pp. 426–439.
- [34] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, “Alert-driven Attack Graph Generation using S-PDFA”, *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [35] S. M. Mathews, “Explainable artificial intelligence applications in NLP, biomedical, and malware classification: A literature review”, in *Intelligent computing-proceedings of the computing conference*, Springer, 2019, pp. 1269–1292.
- [36] F. Becker, A. Drichel, C. Müller, and T. Ertl, “Interpretable Visualizations of Deep Neural Networks for Domain Generation Algorithm Detection”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2020, pp. 25–29.
- [37] D. Alvarez-Melis and T. S. Jaakkola, “On the robustness of interpretability methods”, *arXiv preprint arXiv:1806.08049*, 2018.

- [38] A. Kuppa and N.-A. Le-Khac, “Adversarial XAI methods in cybersecurity”, *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4924–4938, 2021.
- [39] A. Ghorbani, A. Abid, and J. Zou, “Interpretation of neural networks is fragile”, in *AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3681–3688.
- [40] C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, “Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster”, in *International Conference on Availability, Reliability and Security*, 2022.
- [41] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, “Beyond labeling: Using clustering to build network behavioral profiles of malware families”, in *Malware Analysis Using Artificial Intelligence and Deep Learning*, Springer, 2021, pp. 381–409.
- [42] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, “Deep Learning based Vulnerability Detection: Are We There Yet?”, *IEEE Transactions on Software Engineering*, vol. PP, pp. 1–1, Jun. 2021.
- [43] J. Dolejš and M. Jureček, “Interpretability of Machine Learning-Based Results of Malware Detection Using a Set of Rules”, in *Cybersecurity for Artificial Intelligence*, Springer, 2022, pp. 107–136.
- [44] Y.-S. Lin, W.-C. Lee, and Z. B. Celik, “What do you see? Evaluation of explainable artificial intelligence (XAI) interpretability through neural backdoors”, *arXiv preprint arXiv:2009.10639*, 2020.
- [45] D. Vos and S. Verwer, “Efficient training of robust decision trees against adversarial examples”, in *International Conference on Machine Learning*, PMLR, 2021, pp. 10 586–10 595.
- [46] A.-K. Dombrowski, M. Alber, C. Anders, M. Ackermann, K.-R. Müller, and P. Kessel, “Explanations can be manipulated and geometry is to blame”, *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [47] G. Severi, J. Meyer, S. Coull, and A. Oprea, “Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 1487–1504.
- [48] X. Zhao, W. Zhang, X. Xiao, and B. Lim, “Exploiting Explanations for Model Inversion Attacks”, in *IEEE/CVF International Conference on Computer Vision*, 2021, pp. 682–692.
- [49] K. de Bie, A. Lucic, and H. Haned, “To Trust or Not to Trust a Regressor: Estimating and Explaining Trustworthiness of Regression Predictions”, *arXiv preprint arXiv:2104.06982*, 2021.
- [50] R. R. Karn, P. Kudva, H. Huang, S. Suneja, and I. M. Elfadel, “Cryptomining detection in container clouds using system calls and explainable machine learning”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 674–691, 2020.
- [51] G. D. L. T. Parra, L. Selvera, J. Khoury, H. Irizarry, E. Bou-Harb, and P. Rad, “Interpretable Federated Transformer Log Learning for Cloud Threat Forensics”, in *Network and Distributed System Security*, 2022.

- [52] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang, “Interpretable deep learning under fire”, in *Proc. of the USENIX Security Symposium*, 2020.
- [53] F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, “Automatic inference of search patterns for taint-style vulnerabilities”, in *IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 797–812.
- [54] S. Wang, Z. Chen, L. Zhang, *et al.*, “TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic”, in *IEEE/ACM International Symposium on Quality of Service*, IEEE, 2016, pp. 1–6.
- [55] R. Russell, L. Kim, L. Hamilton, *et al.*, “Automated Vulnerability Detection in Source Code Using Deep Representation Learning”, Dec. 2018, pp. 757–762.
- [56] X. Duan, J. Wu, S. Ji, *et al.*, “VulSniper: Focus Your Attention to Shoot Fine-Grained Vulnerabilities”, in *International Joint Conference on Artificial Intelligence*, Aug. 2019, pp. 4665–4671.
- [57] Y. Li, S. Wang, and N. Tien, “Vulnerability detection with fine-grained interpretations”, in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Aug. 2021, pp. 292–303.
- [58] D. Zou, Y. Zhu, H. Jin, *et al.*, “Interpreting Deep Learning-based Vulnerability Detector Predictions Based on Heuristic Searching”, *ACM Transactions on Software Engineering and Methodology*, vol. 30, Aug. 2020.
- [59] A. Nadeem, S. J. Yang, and S. Verwer, “Learning about the adversary”, *Autonomous intelligent agents for cyber defense*, 2023.
- [60] E. Holder and N. Wang, “Explainable artificial intelligence (XAI) interactively working with humans as a junior cyber analyst”, *Human-Intelligent Systems Integration*, pp. 1–15, 2021.
- [61] T. van Ede, H. Aghakhani, N. Spahn, *et al.*, “DeepCASE: Semi-Supervised Contextual Analysis of Security Events”, in *IEEE Symposium on Security and Privacy*, May 2022.
- [62] H. Li, J. Wu, H. Xu, G. Li, and M. Guizani, “Explainable Intelligence-Driven Defense Mechanism Against Advanced Persistent Threats: A Joint Edge Game and AI Approach”, *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 757–775, 2021.
- [63] A. Sopan, M. Berninger, M. Mulakaluri, and R. Katakam, “Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2018, pp. 1–8.
- [64] M. Nyre-Yu, E. S. Morris, B. C. Moss, C. Smutz, and M. Smith, “Explainable AI in Cybersecurity Operations: Lessons Learned from XAI Tool Deployment”, in *Usable Security and Privacy (USEC) Symposium*, 2022.
- [65] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, “TABOR: A graphical model-based approach for anomaly detection in industrial control systems”, in *AsiaCCS*, 2018.

- [66] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, “Recurrent neural network attention mechanisms for interpretable system log anomaly detection”, in *Workshop on Machine Learning for Computing Systems*, 2018, pp. 1–8.
- [67] C. Ardito, T. Di Noia, E. Di Sciascio, D. Lofù, A. Pazienza, and F. Vitulano, “An artificial intelligence cyberattack detection system to improve threat reaction in e-health”, in *Italian Conference on Cybersecurity*, 2021.
- [68] C. Ardito, Y. Deldjoo, E. Di Sciascio, and F. Nazary, “Revisiting security threat on smart grids: Accurate and interpretable fault location prediction and type classification”, in *Italian Conference on CyberSecurity*, 2021.
- [69] C. Hwang and T. Lee, “E-SFD: Explainable Sensor Fault Detection in the ICS Anomaly Detection System”, *IEEE Access*, vol. 9, pp. 140 470–140 486, 2021.
- [70] M. Szczepański, M. Choraś, M. Pawlicki, and R. Kozik, “Achieving explainability of intrusion detection system by hybrid oracle-explainer approach”, in *IJCNN*, IEEE, 2020, pp. 1–8.
- [71] M. Wang, K. Zheng, Y. Yang, and X. Wang, “An Explainable Machine Learning Framework for Intrusion Detection Systems”, *IEEE Access*, vol. 8, pp. 73 127–73 141, 2020.
- [72] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, “Explainable artificial intelligence (XAI) to enhance trust management in intrusion detection systems using decision tree model”, *Complexity*, vol. 2021, 2021.
- [73] H. Liu, C. Zhong, A. Alnusair, and S. R. Islam, “FAIXID: a framework for enhancing ai explainability of intrusion detection results using data cleaning techniques”, *Journal of Network and Systems Management*, vol. 29, no. 4, pp. 1–30, 2021.
- [74] M. Angelini, L. Aniello, S. Lenti, G. Santucci, and D. Ucci, “The goods, the bads and the uglies: Supporting decisions in malware detection through visual analytics”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2017, pp. 1–8.
- [75] S. Mahdavifar and A. A. Ghorbani, “DeNNeS: deep embedded neural network expert system for detecting cyber attacks”, *Neural Computing and Applications*, vol. 32, no. 18, pp. 14 753–14 780, 2020.
- [76] A. Kyadige, E. M. Rudd, and K. Berlin, “Learning from Context: A Multi-View Deep Learning Architecture for Malware Detection”, in *IEEE Security and Privacy Workshops (SPW)*, 2020, pp. 1–7.
- [77] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, “Towards an interpretable deep learning model for mobile malware detection and family identification”, *Computers & Security*, vol. 105, p. 102 198, 2021.
- [78] B. Wu, S. Chen, C. Gao, *et al.*, “Why an android app is classified as malware: Toward malware classification interpretation”, *ACM TOSEM*, vol. 30, no. 2, pp. 1–29, 2021.

- [79] N. Ben Rabah, B. Le Grand, and M. K. Pinheiro, “IoT Botnet Detection using Black-box Machine Learning Models: The Trade-off between Performance and Interpretability”, in *IEEE International Conference on Enabling Technologies*, 2021, pp. 101–106.
- [80] M. Kinlead, S. Millar, N. McLaughlin, and P. O’Kane, “Towards explainable CNNs for Android malware detection”, *Procedia Computer Science*, vol. 184, pp. 959–965, 2021.
- [81] V. Koutsokostas, N. Lykousas, T. Apostolopoulos, *et al.*, “Invoice #31415 attached: Automated analysis of malicious Microsoft Office documents”, *Computers & Security*, vol. 114, p. 102 582, 2022.
- [82] I. Corona, B. Biggio, M. Contini, *et al.*, “DeltaPhish: Detecting phishing webpages in compromised websites”, in *European Symposium on Research in Computer Security*, Springer, 2017, pp. 370–388.
- [83] Y. Chai, Y. Zhou, W. Li, and Y. Jiang, “An explainable multi-modal hierarchical attention model for developing phishing threat intelligence”, *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 790–803, 2021.
- [84] Y. Lin, R. Liu, D. M. Divakaran, *et al.*, “Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 3793–3810.
- [85] B. Gulmezoglu, “XAI-based Microarchitectural Side-Channel Analysis for Website Fingerprinting Attacks and Defenses”, *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021.
- [86] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. De Ruiter, K. Sagonas, and J. Somorovsky, “Analysis of DTLS Implementations Using Protocol State Fuzzing”, in *Proc. of the USENIX Security Symposium*, 2020, pp. 2523–2540.
- [87] A. P. Norton and Y. Qi, “Adversarial-Playground: A visualization suite showing how adversarial examples fool deep learning”, in *IEEE symposium on visualization for cyber security (VizSec)*, IEEE, 2017, pp. 1–4.
- [88] A. Wijekoon and N. Wiratunga, “Reasoning with counterfactual explanations for code vulnerability detection and correction”, in *AI-Cybersec@ SGAI*, 2021, pp. 1–13.
- [89] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, “Neural nets can learn function type signatures from binaries”, in *Proc. of the USENIX Security Symposium*, 2017, pp. 99–116.
- [90] D. L. Marino, C. S. Wickramasinghe, and M. Manic, “An adversarial approach for explainable ai in intrusion detection systems”, in *Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2018, pp. 3237–3243.
- [91] Y. Chen, Y. Bai, R. Mitev, K. Wang, A.-R. Sadeghi, and W. Xu, “FakeWake: Understanding and Mitigating Fake Wake-up Words of Voice Assistants”, in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, 2021, pp. 1861–1883.

- [92] S. Ahn, J. Kim, S. Y. Park, and S. Cho, “Explaining deep learning-based traffic classification using a genetic algorithm”, *IEEE Access*, vol. 9, pp. 4738–4751, 2020.
- [93] C. S. Wickramasinghe, K. Amarasinghe, D. L. Marino, C. Rieger, and M. Manic, “Explainable unsupervised machine learning for cyber-physical systems”, *IEEE Access*, vol. 9, pp. 131 824–131 843, 2021.
- [94] A. A. Reyes, F. D. Vaca, G. A. Castro Aguayo, Q. Niyaz, and V. Devabhaktuni, “A Machine Learning Based Two-Stage Wi-Fi Network Intrusion Detection System”, *Electronics*, vol. 9, no. 10, 2020.
- [95] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, “Sanity checks for saliency maps”, *Advances in neural information processing systems*, vol. 31, 2018.
- [96] A. Kuppa and N.-A. Le-Khac, “Black box attacks on explainable artificial intelligence (XAI) methods in cyber security”, in *IJCNN*, IEEE, 2020, pp. 1–8.
- [97] A. Galli, S. Marrone, V. Moscato, and C. Sansone, “Reliability of explainable artificial intelligence in adversarial perturbation scenarios”, in *International Conference on Pattern Recognition*, Springer, 2021, pp. 243–256.
- [98] M. Ghassemi, L. Oakden-Rayner, and A. L. Beam, “The false hope of current approaches to explainable artificial intelligence in health care”, *The Lancet Digital Health*, vol. 3, no. 11, 2021.
- [99] J. Xu, M. Xue, and S. Picek, “Explainability-based backdoor attacks against graph neural networks”, in *ACM Workshop on Wireless Security and Machine Learning*, 2021, pp. 31–36.
- [100] R. Shokri, M. Strobel, and Y. Zick, “On the privacy risks of model explanations”, in *AAAI/ACM Conference on AI, Ethics, and Society*, 2021, pp. 231–241.
- [101] L. Demetrio, B. Biggio, L. Giovanni, F. Roli, and A. Alessandro, “Explaining vulnerabilities of deep learning to adversarial malware binaries”, in *Italian Conference on Cyber Security*, vol. 2315, 2019.
- [102] X. Wu, W. Guo, H. Wei, and X. Xing, “Adversarial policy training against deep reinforcement learning”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 1883–1900.
- [103] S. Akerman, E. Habler, and A. Shabtai, “VizADS-B: Analyzing sequences of ADS-B images using explainable convolutional LSTM encoder-decoder to detect cyber attacks”, *arXiv preprint arXiv:1906.07921*, 2019.
- [104] T. Le, H. Chen, and M. Ali Babar, “A Survey on Data-driven Software Vulnerability Assessment and Prioritization”, *ACM Computing Surveys (CSUR)*, Jul. 2021.
- [105] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, “Prospex: Protocol specification extraction”, in *IEEE Symposium on Security and Privacy*, IEEE, 2009.
- [106] L. M. Padilla, S. H. Creem-Regehr, M. Hegarty, and J. K. Stefanucci, “Decision making with visualizations: A cognitive framework across disciplines”, *Cognitive research: principles and implications*, vol. 3, no. 1, pp. 1–25, 2018.

- [107] S. Anjomshoae, A. Najjar, D. Calvaresi, and K. Främling, “Explainable agents and robots: Results from a systematic literature review”, in *AAMAS, International Foundation for Autonomous Agents and Multiagent Systems*, 2019, pp. 1078–1088.
- [108] C. Panigutti, A. Beretta, F. Giannotti, and D. Pedreschi, “Understanding the impact of explanations on advice-taking: A user study for AI-based clinical Decision Support Systems”, in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–9.
- [109] Z. C. Lipton, “The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery”, *Queue*, vol. 16, no. 3, pp. 31–57, 2018.
- [110] L. K. Hansen and L. Rieger, “Interpretability in intelligent systems—A new concept?”, in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, Springer, 2019, pp. 41–49.
- [111] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning”, *arXiv preprint arXiv:1702.08608*, 2017.
- [112] M. L. Leavitt and A. Morcos, “Towards falsifiable interpretability research”, *arXiv preprint arXiv:2010.12016*, 2020.
- [113] A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer, “Intelligent Malware Defenses”, in *Security and Artificial Intelligence*, Springer, 2022, pp. 217–253.
- [114] C. Nicodeme, “Build confidence and acceptance of AI-based decision support systems—Explainable and liable AI”, in *International Conference on Human System Interaction*, IEEE, 2020, pp. 20–23.
- [115] A. Ross and F. Doshi-Velez, “Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients”, in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [116] M. Moshkovitz, Y.-Y. Yang, and K. Chaudhuri, “Connecting interpretability and robustness in decision trees through separation”, in *International Conference on Machine Learning*, PMLR, 2021, pp. 7839–7849.
- [117] D. Arp, E. Quiring, F. Pendlebury, *et al.*, “Do’s and don’ts of machine learning in computer security”, in *Proc. of the USENIX Security Symposium*, 2022.
- [118] J. Clear, *Atomic habits: An easy & proven way to build good habits & break bad ones*. Penguin, 2018.
- [119] S. Krishna, T. Han, A. Gu, *et al.*, “The Disagreement Problem in Explainable Machine Learning: A Practitioner’s Perspective”, *arXiv preprint arXiv:2202.01602*, 2022.
- [120] U. Aivodji, H. Arai, O. Fortineau, S. Gambs, S. Hara, and A. Tapp, “Fairwashing: The risk of rationalization”, in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, Jun. 2019, pp. 161–170.
- [121] G. Weiss, Y. Goldberg, and E. Yahav, “Extracting automata from recurrent neural networks using queries and counterexamples”, in *International Conference on Machine Learning*, PMLR, 2018, pp. 5247–5256.

- [122] A. Koul, S. Greydanus, and A. Fern, “Learning finite state representations of recurrent policy networks”, *arXiv preprint arXiv:1811.12530*, 2018.
- [123] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, “Safe reinforcement learning via shielding”, in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [124] W. Xiang, P. Musau, A. A. Wild, *et al.*, “Verification for machine learning, autonomy, and neural networks survey”, *arXiv preprint arXiv:1810.01989*, 2018.
- [125] X. Huang, D. Kroening, W. Ruan, *et al.*, “A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability”, *Computer Science Review*, vol. 37, p. 100 270, 2020.
- [126] T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, “Ai2: Safety and robustness certification of neural networks with abstract interpretation”, in *IEEE Symposium on Security and Privacy*, IEEE, 2018, pp. 3–18.
- [127] H. Fokkema, R. de Heide, and T. van Erven, “Attribution-based Explanations that Provide Recourse Cannot be Robust”, *arXiv preprint arXiv:2205.15834*, 2022.
- [128] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, “Robust decision trees against adversarial examples”, in *International Conference on Machine Learning*, PMLR, 2019, pp. 1122–1131.
- [129] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, “Treat: training evasion-aware decision trees”, *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1390–1420, 2020.
- [130] J. Hayes, B. Balle, and M. P. Kumar, “Learning to be adversarially robust and differentially private”, *arXiv preprint arXiv:2201.02265*, 2022.
- [131] B. Biggio, I. Corona, D. Maiorca, *et al.*, “Evasion attacks against machine learning at test time”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 387–402.
- [132] D. Lowd and C. Meek, “Adversarial learning”, in *ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 641–647.
- [133] C. P. Pfleeger and S. L. Pfleeger, *Analyzing computer security: A threat/vulnerability/countermeasure approach*. Prentice Hall Professional, 2012.
- [134] P. Hall, N. Gill, and N. Schmidt, “Proposed Guidelines for the Responsible Use of Explainable Machine Learning”, *arXiv preprint arXiv:1906.03533*, 2019.
- [135] S. Biswal, “System Preserving Explainability and Confidentiality (SPEC)”, in *4th workshop on machine learning for cybersecurity*, Springer, 2022.
- [136] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods”, *Computers & Security*, vol. 45, pp. 100–123, 2014.
- [137] L. D’hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, “Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models”, in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Springer, 2022, pp. 23–41.

- [138] C. Cao, A. Panichella, S. Verwer, A. Blaise, and F. Rebecchi, “Encoding NetFlows for State-Machine Learning”, *arXiv preprint arXiv:2207.03890*, 2022.
- [139] M. Nicolau and J. McDermott, “Learning neural representations for network anomaly detection”, *IEEE transactions on cybernetics*, vol. 49, no. 8, pp. 3074–3087, 2018.
- [140] R. U. Khan, X. Zhang, R. Kumar, A. Sharif, N. A. Golilarz, and M. Alazab, “An adaptive multi-layer botnet detection technique using machine learning classifiers”, *Applied Sciences*, vol. 9, no. 11, p. 2375, 2019.
- [141] R. Priyadarshini and R. K. Barik, “A deep learning based intelligent framework to mitigate DDoS attack in fog environment”, *Journal of King Saud University-Computer and Information Sciences*, 2019.
- [142] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger, “In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems”, in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, 2023, pp. 2217–2231.
- [143] S. R. Islam, W. Eberle, and S. K. Ghafoor, “Towards quantification of explainability in explainable artificial intelligence methods”, in *The thirty-third international flairs conference*, 2020.
- [144] D. Bertsimas, A. Delarue, P. Jaillet, and S. Martin, “The price of interpretability”, *arXiv preprint arXiv:1907.03419*, 2019.

I

INTERPRETABLE SEQUENTIAL LEARNING FOR ATTACKER STRATEGY DISCOVERY

3

ALERT-DRIVEN ATTACK GRAPH GENERATION

Learning about attacker strategies, such as their tactics, techniques, and procedures (TTPs) is largely a manual task. Attacker strategies are typically represented using attack graphs (AG). Existing techniques for AG generation cannot directly be used to monitor ongoing attacks in Security Operations Centers (SOCs) since they do not show the dynamic strategies being employed by the attackers. Meanwhile, SOC analysts defend against cyber attacks by monitoring millions of intrusion alerts on a daily basis, often leading to ‘alert fatigue’ and reduced productivity.

Instead of deriving AGs based on system vulnerabilities, this work advocates the direct use of intrusion alerts. We propose SAGE, an interpretable 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. We show that SAGE compresses over 1.4 million alerts into ~400 AGs (a reduction of 99.97%), in under 5 minutes.

This chapter is based on the papers “Alert-driven Attack Graph Generation using S-PDFA” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in IEEE Transactions on Dependable and Secure Computing (TDSC), 2021, 19(2), 731-746 [1], and poster “Enabling Visual Analytics via Alert-driven Attack Graphs” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in ACM SIGSAC Conference on Computer and Communications Security (CCS), 2021 (pp. 2420-2422) [2].

3.1. 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 [3]. 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 [4]. *Alert correlation* reduces the volume of alerts by grouping alerts from the same attack stage [5]–[7]. 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, which are commonly used for visual analytics [8]–[10] and forensic analysis [11], [12]. An attack graph (AG) displays all the paths an attacker can exploit to penetrate a network. Vertices in a typical attack graph represent privileges gained by an attacker, and the edges represent the vulnerabilities that enabled the attacker to gain those privileges [13]. Existing AG generation approaches fall under the Topological Vulnerability Analysis (TVA) [14] that requires extensive amount of expert knowledge and published vulnerability reports [15], [16]. 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 [17]. Additionally, shared threat intelligence reports are often not directly relevant to one's own network [18]. 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 chapter, we formally define our proposed system, SAGE (IntruSion alert-driven Attack Graph Extractor) [19]. SAGE generates AGs directly from intrusion alerts without a priori vulnerability and network topology information. It adopts an interpretable 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. Figure 3.1 shows the boxology diagram for SAGE, according to the modular design patterns by van Bekkum *et al.* [20].

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 [21]. 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 [22]. We tune the learning algorithm and transform the alert data such that the resulting model accentuates infrequent severe alerts, without discarding any low-severity alerts. The model summarizes attack paths leading to severe attack stages. It can distinguish between alerts with the same signature but different contexts, *i.e.*, scanning at the start and scanning midway through an

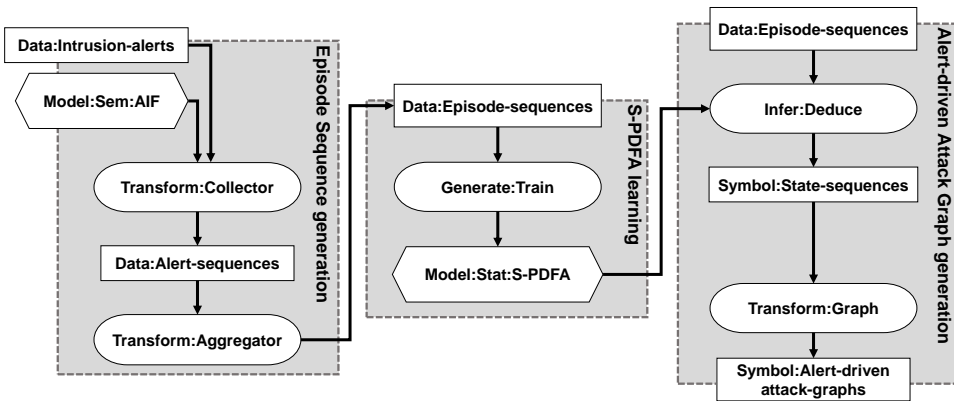


Figure 3.1: SAGE takes intrusion alerts as input to generate attack graphs. Intrusion alerts are transformed into episode sequences (Section 3.3.1). An interpretable S-PDFA model is learned from those sequences (Section 3.3.2). The sequences are replayed through the S-PDFA and transformed into targeted attack graphs (Section 3.3.3).

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 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 [23] and blue team exercises [24] that have significantly different statistical properties and target infrastructures.

On all the datasets combined, SAGE compresses over 1.4 million alerts into 401 AGs in under 5 minutes. Even with an imperfect IDS, the AGs capture the strategies used by the participating teams. We show that 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 a suffix-based probabilistic deterministic finite automaton (S-PDFA) – an interpretable sequence learning 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 apply SAGE on alerts from three security testing competition datasets. We show that SAGE is generalizable, and that the AGs capture attacker strategies.

Organization. We provide a brief overview of the related works in Section 3.2. The architecture of SAGE is illustrated in Section 3.3, and its explainability aspect is described in Section 3.4. Sections 3.5 and 3.6 describe the experimental setup and a thorough analysis of alert-driven attack graphs. We discuss the limitations of SAGE in Section 3.7 and conclude in Section 3.8.

3.2. RELATED WORK

Alert management. 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 [5]–[7], [25]–[29] and alert prioritization [30], [31]. 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 [10], [17]. Specifically in the network security domain, Kaynar *et al.* [13] have proposed a taxonomy of the existing AG generation approaches. Many of them fall under the topological vulnerability analysis (TVA) [14], which relies heavily on a priori knowledge about the topology of, and vulnerabilities in a network, making them unsuitable for zero-day attacks. MulVAL [15] and NetSPA [16] 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 [32], [33], AG complexity reduction [34], [35], and what-if analyses [8], [9].

Learning from observables. Cyber data from prior security incidents can be utilized to gain insights into attacker behavior, *e.g.*, using log data [36]–[38], sensor data [39], and network traffic [40]. 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 [41], [42]. 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.* [43] 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 chapter, we propose SAGE, which is a purely alert-driven approach for generating attack graphs. We borrow initial ideas from Moskal *et al.* [43]. 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 and the S-PDFA instead. Because the explainability aspect of SAGE is an important design consideration, we do not consider inherently black-box models, such as neural networks [44]. While attention mechanisms [45] and linear surrogate models [46] help explain the decisions of such black-box models, they typically offer post-hoc explainability 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.* [47], 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 provide examples of these concepts in Section 3.4.

3.3. INTRUSION ALERT-DRIVEN ATTACK GRAPH EXTRACTOR – SAGE

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 Figure 3.1. It takes raw intrusion alerts as input and aggregates them into sequences of attacker actions. An automaton model is learned using these sequences, summarizing attacker strategies. Finally, attack graphs are extracted from the model on a per-victim, per-objective basis. SAGE is released as open-source¹. It is implemented in Python and released as a docker container for cross-platform support.

In this section, we use the Collegiate Penetration Testing Competition dataset from 2018 [48], *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 3.5 for details).

3.3.1. ATTACK EPISODE AGGREGATION

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.

¹SAGE: <https://github.com/tudelft-cda-lab/SAGE>

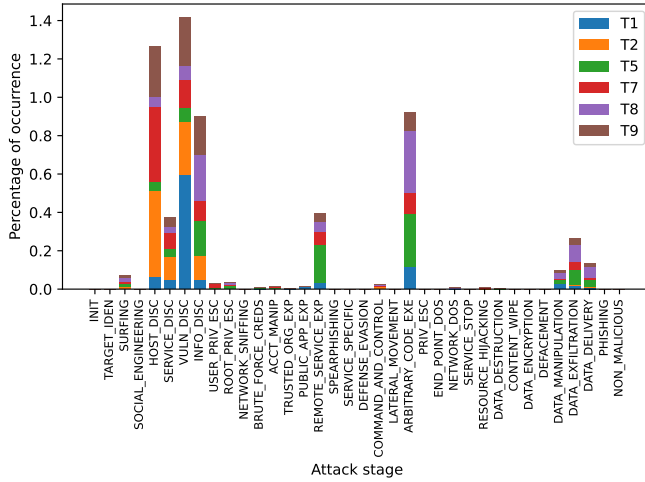


Figure 3.2: The alert distribution per attack stage for the CPTC-2018 teams (T1-T9), showing that scanning-alerts are significantly more frequent than exploitation-alerts.

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 the open-source IANA mapping [49]. (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 chapter, we use $t = 1.0$ sec following [7], [43]. (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.* [50] 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 [51], 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 Table 3.9). (iv) Finally, the filtered set \tilde{O} of intrusion alerts \tilde{o} is a 5-tuple $\tilde{o} = \langle sIP, dIP, tServ, ts, mcat \rangle$ for each $o \in O_F$. Figure 3.2 shows the distribution of the attack stages across all six teams in the filtered CPTC-2018 dataset.

Gathering alerts into Alert Sequences (AS). There are three main methods for converting discrete observables into sequences with 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. We implement an alert sequence as a windowed list of alerts, *i.e.*, it is a list of multi-sets between a unique (attacker, victim) pair, where each multi-set contains time-ordered alerts occurring within a time window.

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 $AS_{avc} = \ddot{o}_1^{avc} \dots \ddot{o}_n^{avc}$, where $(a, v) \in A \times V$, $c \in C$. Here, $\ddot{o}_i^{avc} = \{\delta_1^{avc} \dots \delta_\omega^{avc}\}$ is a multi-set of alerts for $1 \leq i \leq n$. For a window w and given $\tilde{o}_j = \langle a, v, tServ, ts, c \rangle \in \tilde{O}$, we define $\hat{o}_j^{avc} = \langle tServ, ts, c \rangle$ such that $\Pi_{ts}(\hat{o}_1^{avc}) = i \cdot w$, $\Pi_{ts}(\hat{o}_\omega^{avc}) - \Pi_{ts}(\hat{o}_1^{avc}) \leq w$, and $\Pi_{ts}(\hat{o}_j^{avc}) \leq \Pi_{ts}(\hat{o}_{j+1}^{avc})$, for $1 \leq j \leq \omega$.*

Here, $\Pi_X(\hat{o}_j^{avc})$ is the projection of the X attribute of \hat{o}_j^{avc} . In contrast to other works that use *sIP* and *dIP* as explicit features [26]–[28], we only use them to construct sequences. This allows identification of related alerts originating from different sources.

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* or a *hyper-alert* [43]. 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 Figure 3.3). Episodes are the building block of SAGE. All extracted 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, $ES_{av} = ep_1^{av} \dots ep_m^{av}$. An episode is a 4-tuple $ep_j^{av} = \langle st^{av}, et^{av}, mcat^{av}, mServ^{av} \rangle$ for $1 \leq j \leq m$, where $st^{av}, et^{av} \in \mathbb{R}$ denote the start time 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, episode sequences aggregate bursts of alerts. We construct ES_{av} from windowed alert sequences with different attack stages c , *i.e.*, $AS_{avc} = \ddot{o}_1^{avc} \dots \ddot{o}_n^{avc}$. For each $1 \leq s \leq e \leq n$, the start time is $st^{av} = \min(\Pi_{ts}(\ddot{o}_s^{avc}))$ if $f'(|\ddot{o}_s^{avc}|) = 0$ and $f'(|\ddot{o}_{s+1}^{avc}|) > 0$; the end time is $et^{av} = \max(\Pi_{ts}(\ddot{o}_e^{avc}))$ if $f'(|\ddot{o}_e^{avc}|) = 0$ and $f'(|\ddot{o}_{e-1}^{avc}|) < 0$; the attack stage is $mcat^{av} = c$, and the most frequently targeted service is $mServ^{av} = \arg \max_{mServ} |\{\Pi_{tServ}(\ddot{o}_i^{avc}) = mServ : s \leq i \leq e\}|$. Here, $f'(|\ddot{o}_i^{avc}|)$ is the slope derived from the number of alerts in each window, *i.e.*, $\frac{\Delta|\ddot{o}_i^{avc}|}{\Delta i}$.

Algorithm 1: Convert alert sequences into episode sequences

Input: Alert sequence: as
Output: Episode sequence: es

```

1 function CONVERT_TO_ES( $as$ )
2    $es \leftarrow []$ 
3   forall ( $mcat_x, as_x$ ) in SPLIT_ON_MCAT( $as$ ) do
4      $timed\_as \leftarrow \text{LEN}(window)$  for all  $window \in as_x$ 
5      $slope \leftarrow f'(x)$  for all  $x \in timed\_as$ 
6      $ups \leftarrow \text{GET\_POSITIVE\_SLOPES}(slope)$ 
7      $downs \leftarrow \text{GET\_NEGATIVE\_SLOPES}(slope)$ 
8      $episodes \leftarrow \text{GET\_EPISODES}(ups, downs)$ 
9     Append ( $mcat_x, ep$ ) to  $es$  for all  $ep \in episodes$ 
10  Sort  $es$  by episode start time
11  yield  $es$ 
12 function GET_EPISODES( $ups, downs$ )
13   $episodes \leftarrow []$ 
14  for  $i \leftarrow 0 \dots \text{LEN}(ups) - 1$  do
15    if IS_DOWN_BETWEEN_UPS( $i, i + 1, downs$ ) then
16       $down \leftarrow \text{GET\_LAST\_DOWN}(i, i + 1, downs)$ 
17      Append ( $ups[i], down$ ) to  $episodes$ 
18  yield  $episodes$ 

```

3.3.2. SUFFIX-BASED PROBABILISTIC DETERMINISTIC FINITE AUTOMATON

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 important 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.*, [52]–[55].

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; $P : \Delta \rightarrow [0, 1]$ is the transition probability function, and $q_0 \in Q$ is the final state (for 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 $\sum_{q,a} P(\langle q, q', a \rangle) = 1$. 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.

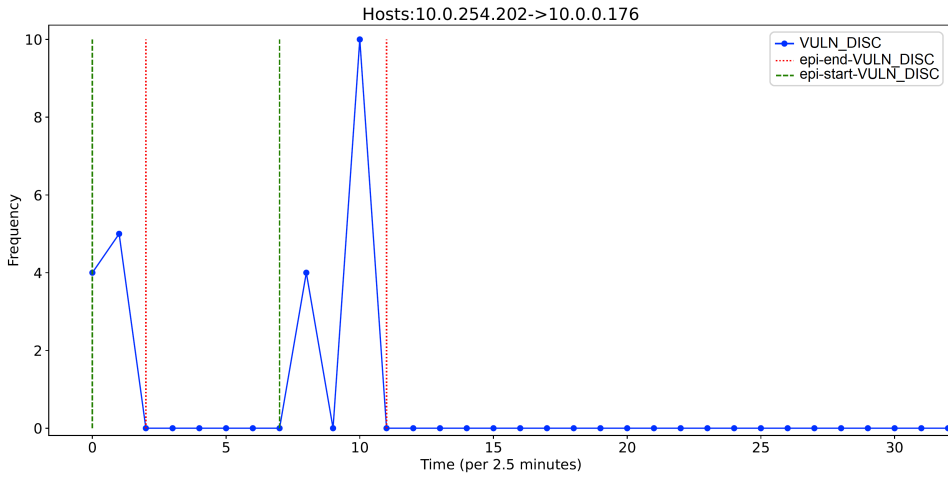


Figure 3.3: For each (attacker, victim) pair, bursts of alerts from each attack stage are aggregated into episodes. The figure shows an attack sequence related to vulnerability scanning that is aggregated into two episodes.

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 $1 \leq n$. 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 $P(s) = \prod_{0 \leq i < n} P(\langle q_i, q_{i+1}, a_{n-i} \rangle)$, where \prod denotes a product. For any trace, there exists a unique run due to suffix determinism. The Flexfringe automaton learning framework [22] can be used to learn suffix models. Flexfringe implements several automaton learning heuristics within the well-known state merging algorithms, such as state merging [56] and DFASAT [57] (see [58] for details).

Note that the S-PDFA learns only from positive traces that are composed of observed events. This is done by modeling the probability distribution of the the input traces, and using it as evidence for state merging. Essentially, the merge criteria tests a null-hypothesis that the probability distribution followed by two different states is equivalent, *i.e.*, the trace prefixes that can occur after two different states have been reached following the same distribution, and thus the two states can be modeled using a single state. If the null-hypothesis is rejected with sufficient confidence provided by the evidence, it prevents a merge from happening, since the two states lead to significantly different outcomes (see [59] for details).

Input trace construction. Whereas an episode sequence may contain multiple attempts to exploit a victim host, an S-PDFA models each attempt separately to find partial overlap in attacker strategies. To this end, an ES is partitioned into episode subsequences (ESS) when a low-severity episode follows a high-severity one. *Severity(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 sever-

```

Attack stage Targeted service
  ↓           ↓
CnC|http infoD|http infoD|http
serD|fis vulnD|ahsp serD|cpdlc vulnD|ahsp hostD|http serD|fis vulnD|ahsp hostD|http
serD|unknown vulnD|ms-sql-s serD|ssh hostD|http serD|unknown vulnD|ahsp serD|ssh hostD|http
serD|rpas-c2 vulnD|ncube-1m serD|unknown vulnD|ahsp hostD|http serD|unknown vulnD|ncube-1m
serD|ssh hostD|http
bfCred|Unknown serD|ssh vulnD|ncube-1m serD|ssh hostD|http
bfCred|pop3s serD|unknown vulnD|ahsp hostD|http
rPrivEsc|smtp serD|ssh hostD|http serD|unknown vulnD|ahsp serD|ssh
serD|ssh hostD|Unknown serD|unknown vulnD|ahsp hostD|Unknown serD|ssh vulnD|ncube-1m
serD|ssh hostD|http
exfil|Unknown infoD|Unknown hostD|wap-wsp serD|unknown vulnD|ahsp hostD|wap-wsp
serD|unknown vulnD|ncube-1m serD|ssh hostD|http
exfil|http bfCred|http hostD|http serD|unknown vulnD|ahsp serD|ssh

```

Figure 3.4: An excerpt from the input traces used for learning the S-PDFA. Each symbol in the trace is a tuple $\langle mcat, mServ \rangle$ or $\langle attack\ stage, targeted\ service \rangle$. The traces have been reversed to learn a suffix-based model.

ity (see appendix Table 3.9).

Definition 4 Given an $ES_{av} = epi_1^{av} \dots epi_m^{av}$, define a break-point as an index i such that $Severity(epi_{i+1}^{av}) = low$ and $Severity(epi_i^{av}) = medium/high$. 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*. Figure 3.4 shows an excerpt from the CPTC-2018 traces. One trace is constructed per ESS. The symbols signify the most apparent intent of episodes, defined by $\langle mcat, mServ \rangle$. $Theme(mServ)$ groups services based on their functionality (see appendix Table 3.10). For CPTC-2018, this gives 664 traces, which is small but sufficient to learn insightful S-PDFAs.

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. A suffix-automaton model is thus used to predict the past, instead of predicting the future. Each state in an S-PDFA model can then 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 [60] because of limited data. For reversed traces, the algorithm constructs a suffix tree (see Figure 3.5 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. Figure 3.6 shows the S-PDFA for CPTC-2018, learned from all 664 traces to enable behavior comparison.

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

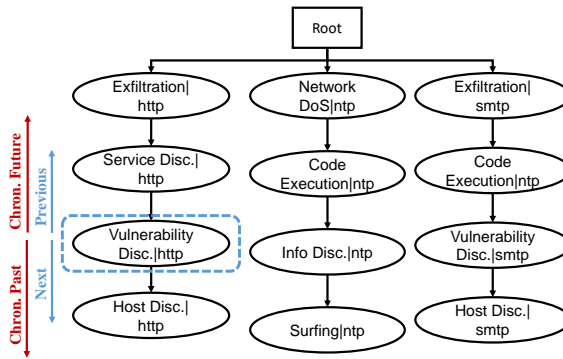


Figure 3.5: A suffix tree for three example traces. For any vertex, the previous vertex happens chronologically in the future, and the next vertex corresponds to the past.

it does not find sufficient evidence that the states are different, *i.e.*, accepting the null-hypothesis. 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.*, $\langle d', q_1, a \rangle$ and $\langle d', q_2, a \rangle$. It enforces that the incoming transition label for states is unique, which makes the model easier to interpret. (iii) We utilize *sink states*. Sink states are non-final states that typically have no outgoing transitions. We utilize sink states to model extremely infrequent actions. The core algorithm continues merging until all states have either been merged or added to the model. For infrequent (sink) 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 for CPTC-2018, which otherwise would not have appeared in the attack graphs.

The state merging algorithm ensures that only the states with similar pasts are merged. The Markovian property, in addition, enforces that the immediate-future is identical. Thus, the occurrence of identical episodes leading to different states shows semantic differences, *e.g.*, *data exfiltration* may either be reached by *service discovery* \rightarrow *arbitrary code execution*, or by *vulnerability discovery* \rightarrow *privilege escalation*. Separate states will be learned for these types of data exfiltration, capturing their context.

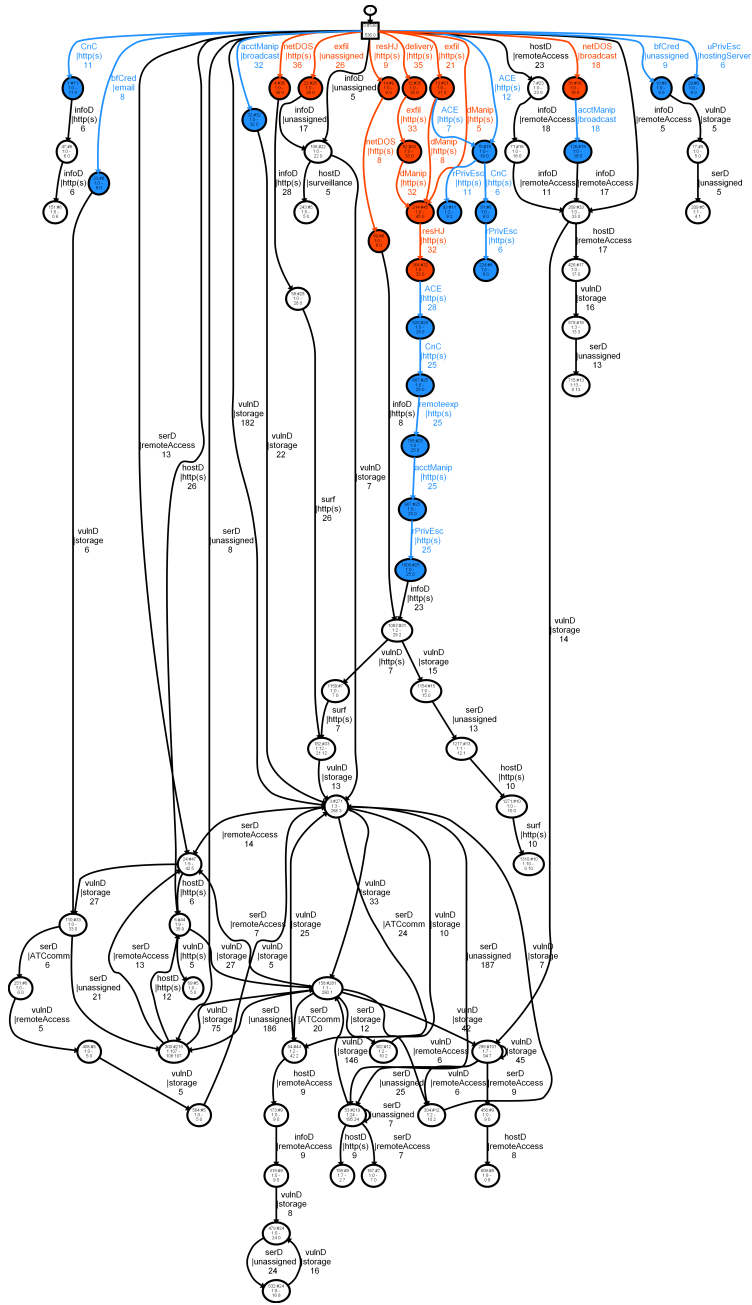


Figure 3.6: The S-PDFA model for the six teams of CPTC-2018. The states are colored according to the severity of the incoming symbol's attack stage: Red is high, blue is medium, white is low. Sink states are excluded.

3.3.3. 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.

Adding context to Episode Sequences (ES). The states of an S-PDFA provide contextual meaning to the episodes' attack stages. Existing work by Lin *et al.* [61] 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 $1 \leq i \leq k$.

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 Table 3.9). 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 1330 (a reduction of 93%). Each AG compresses over 500 alerts in less than 16 vertices, on average.

Figure 3.7 shows an alert-driven attack graph's anatomy. Somewhat deviating from a traditional AG, the vertices of an alert-driven AG represent attacker actions, and the edges represent the temporal relationship between them. Specifically, the root of an alert-driven AG is $\langle mcat, mServ \rangle$. Other vertices are the unique items in $path$. We remove the state identifiers of low-severity vertices to reduce the total vertices, and to further highlight the infrequent severe vertices. 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 the strategy comparison easier, all teams that achieve an objective are shown in one graph, distinguishable by their edge color.

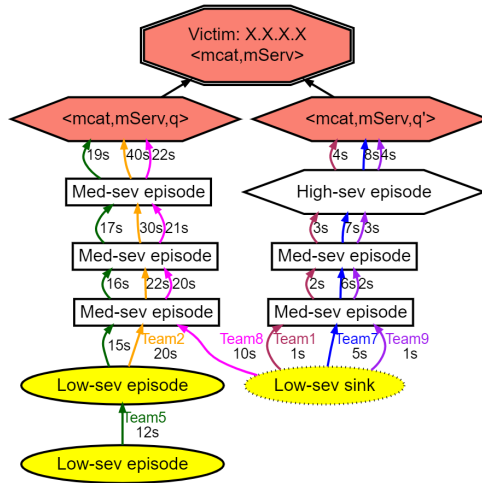


Figure 3.7: An alert-driven attack graph: Vertices: Labels show $\langle \text{attack stage}, \text{targeted service}, \text{state identifier} \rangle$. Low-severity episodes are *oval*, medium-severity are *boxes*, high-severity are *hexagons*. The first episode in a *path* is *yellow* & the objective is *red*. Sinks are *dotted*. Edges: Labels show seconds since the first alert. Colors show team affiliation: T1 (Maroon), T2 (Orange), T5 (Green), T7 (Blue), T8 (Magenta), T9 (Purple).

3.4. 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 [62].

Figure 3.7 shows the composition of an alert-driven AG. An AG for a given $\langle v, obj \rangle$ 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) pair, and (b) *other attack attempts* by modeling each one as a separate path. A path can be traced starting from a yellow vertex, following the time progression of the edge labels, and 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, make the *model components interpretable*. The deterministic nature

Table 3.1: Summary of experimental datasets.

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

of the model makes it *algorithmically transparent*. The parameter settings are guided by the input data, making the model *design transparent*.

3.5. DATASET AND EXPERIMENTAL SETUP

Datasets. Security testing competitions provide an ideal setting for distributed multi-stage attacks in a controlled environment. In this chapter, we use three open-source intrusion alert datasets: two datasets from the Collegiate Penetration Testing Competition (CPTC) [63] for showing SAGE’s efficacy, and one dataset from the Collegiate Cyber Defense Competition (CCDC) [64] for showing SAGE’s generalizability. A summary of the datasets is given in Table 3.1.

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 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 [65], 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 manually investigate how an attack transpired.

Parameters. In this chapter, we set $t = 1.0$ seconds to filter repeated alerts [7], [43]. 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.

Experiments. We perform three experiments pertaining to the quality of the S-PDFA model, and the efficacy of SAGE in reducing the number of alerts to analyze.

Table 3.2: Quality evaluation (via Perplexity) of four suffix models on the CPTC-2018 traces. Suffix tree and SAGE S-PDFA are the best on training and test data, respectively. Best values* are marked.

	Suffix tree	Markov chain	Default S-PDFA	SAGE S-PDFA
Training set	1265.4*	13,659.6	15,136.5	2397.8
Holdout test set	13,020.7	11,617.8	11,241.5	9884.6*

1. **S-PDFA quality evaluation.** We evaluate the quality of the S-PDFA model learned for CPTC-2018 compared to alternative modeling techniques (Section 3.6.1).
2. **S-PDFA 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 (Section 3.6.2).
3. **Alert triaging.** We investigate the extent to which SAGE triages alerts for the three experimental datasets, and analyze the statistical properties of the alert-driven attack graphs (AGs). Specifically, we use the CCDC-2018 dataset to demonstrate that SAGE can learn attack graphs (and triage alerts) even on datasets with no prior information (Section 3.6.3).

3.6. RESULTS AND DISCUSSION

SAGE addresses a long-standing open problem in the security domain regarding data-driven attack graphs by discovering sequential constraints that do not lead to a state-space explosion [66]. As such, SAGE generates purely alert-driven attack graphs without expert input. It has an explainable architecture, and can directly augment existing intrusion detection systems. It is released in a docker container for cross-platform support. 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.

In this section, we rigorously evaluate SAGE with diverse datasets and against alternative modeling approaches.

- We evaluate the quality of the S-PDFA for modeling attacker strategies, compared to well-known alternative models, *e.g.*, Markov chains.
- We take a deep-dive into infrastructural differences captured by the S-PDFA models for the different datasets. Figures 3.6 and 3.8 show the S-PDFA models for the CPTC-2018 and CPTC-2017 datasets, respectively.
- We investigate the extent to which SAGE helps in triaging the alert volume for CPTC-2017, CPTC-2018, and CCDC-2018. Particularly, SAGE compresses 99.97% of the alerts into AGs in under 5 minutes.

3.6.1. S-PDFA MODEL QUALITY EVALUATION

The S-PDFA model discovers attacker strategies in an alert dataset, and represents them graphically. Because the S-PDFA is not used for classification, typical evaluation metrics

(such as, F1-score) cannot be used to evaluate its quality.

Evaluating model quality without ground truth is a hard problem in grammatical inference [58], [67]. 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 [68] to quantify model quality. It measures the prediction power of a model, and has been used in grammatical inference competitions [69], [70]. 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 generalizes to unseen test data.

We compute Perplexity 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, and (iv) SAGE S-PDFA: an S-PDFA learned using the settings in this chapter. Table 3.2 shows the perplexity for each variant on the CPTC-2018 dataset. 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.

In addition, the AGs generated from each of the models show a different perspective: The Markov chain-AGs do not model the context and make vast over-generalizations, thus producing no added-benefit of the modeling step, *i.e.*, the state identifiers do not differentiate between contextually different objectives. The suffix tree-AGs and the S-PDFA-AGs are highly similar, except the S-PDFA-AGs are smaller due to the state merging algorithm. The real benefit of the S-PDFA becomes apparent in larger graphs because similar paths are merged in an S-PDFA. Thus, repeated (sub-)strategies are displayed using already-existing vertices. In contrast, the suffix tree-AGs are larger because of several similar vertices, which are not merged since no learning is applied. This analysis raises the question: When is learning (*i.e.*, making generalizations) a good idea, and when does simply showing raw data suffice?

3.6.2. COMPARING INFRASTRUCTURAL DIFFERENCES ACROSS DATASETS

We analyze the extent to which an S-PDFA model summarizes infrastructure-related nuances present in an alert dataset. We learn two S-PDFA models, one for CPTC-2018 (Figure 3.6) and the other for CPTC-2017 (Figure 3.8) using the same method and parameter settings. Both models summarize the various paths taken by the teams to reach high-severity states. Table 3.3 shows the number of states and transitions utilized to model the intrusion alerts from CPTC-2017 and CPTC-2018. The 2017 model is larger (considering also sink states) 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 and infrequent sub-behaviors than the 2018 teams.

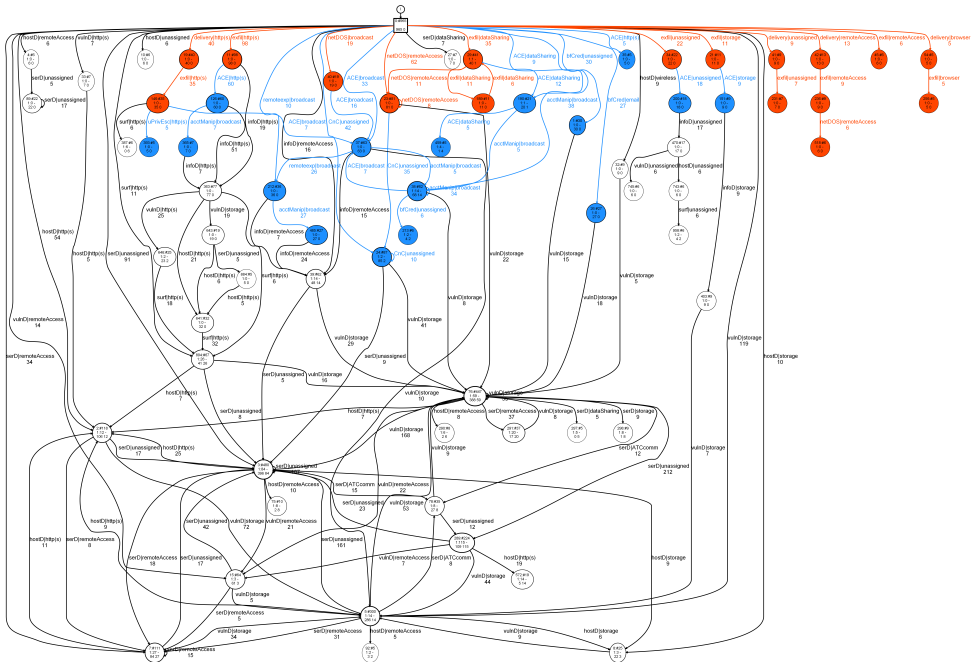


Figure 3.8: The S-PDFA model for the nine teams of CPTC-2017. The states are colored according to the severity of the incoming symbol's attack stage: Red is high, blue is medium, white is low. Sink states are excluded.

Table 3.3: Model statistics for the S-PDFA learned from CPTC-2017 and CPTC-2018.

Dataset	Alerts	Traces	S-PDFA states (incl. sinks)	S-PDFA transitions (incl. sinks)
CPTC-2018	330,270	664	74 (262)	106 (293)
CPTC-2017	43,611	1039	73 (351)	143 (420)

Table 3.4 shows an exhaustive comparison between the two models in terms of the services used to carry out the objectives. It shows the number of unique objectives exploited by the teams via a particular service. This includes the different ways of reaching the same objective, as identified by the S-PDFA model. The most striking difference between the models is that there are, on average, more paths leading to severe states in the 2017 model than in the 2018 one. This means that a control could be more easily placed in the 2018 network, making it impossible for attackers 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 3.4 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

Table 3.4: Differences between the S-PDFAs for CPTC-2017 and CPTC-2018 as the number of unique objective states modeled, *i.e.*, (high-severity attack stage, targeted service).

Service	S-PDFA for dataset	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	-			n/a			
	CPTC-2018	9		✓	✓	✓	✓	
email	CPTC-2017	1				✓		
	CPTC-2018	-				n/a		
authentication	CPTC-2017	2	✓			✓		
	CPTC-2018	-				n/a		
dataSharing	CPTC-2017	2				✓		
	CPTC-2018	-				n/a		
nameserver	CPTC-2017	1			✓			
	CPTC-2018	-			n/a			
browser	CPTC-2017	2				✓	✓	
	CPTC-2018	-				n/a		
clocksync	CPTC-2017	2	✓					
	CPTC-2018	-				n/a		
storage	CPTC-2017	3				✓	✓	
	CPTC-2018	-				n/a		
unassigned	CPTC-2017	8				✓	✓	
	CPTC-2018	7		✓	✓	✓	✓	

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.

3.6.3. ALERT TRIAGING VIA ALERT-DRIVEN ATTACK GRAPHS

Alert compression. SAGE compresses intrusion alerts into a handful of *alert-driven attack graphs* (AGs). Instead of investigating tabular alerts, analysts can triage alerts based

Table 3.5: Breakdown of the CPTC-2018 teams: Raw alerts and their compression* at each step of SAGE. *Note that this table was regenerated after making modifications to the code. There may be discrepancies compared to the published manuscript [1].

CPTC-2018 Teams	Alerts (raw)	Alerts (filtered)	Episodes	ES/ESQ	ESS	Traces	AGs
T1	81,373	26,651	738	113	184	123	36
T2	42,474	4922	671	95	119	94	6
T5	52,550	11,918	668	76	137	118	51
T7	47,101	8517	664	88	120	81	23
T8	55,170	9037	623	93	176	122	22
T9	51,602	10,081	1115	82	163	126	29

Table 3.6: Breakdown of the CPTC-2017 teams: Raw alerts and their compression at each step of SAGE.

CPTC-2017 Teams	Alerts (raw)	Alerts (filtered)	Episodes	ES/ESQ	ESS	Traces	AGs
T2	2923	2904	411	142	142	56	9
T3	3353	3293	520	120	131	84	22
T4	7801	7232	1539	213	462	276	120
T5	1912	1890	433	138	138	56	11
T6	8413	7485	980	136	258	192	72
T7	4712	4220	767	141	176	114	26
T8	7150	4944	1019	132	203	123	35
T9	2233	2199	511	122	147	75	36
T10	5114	4949	496	146	148	63	11

on a few AGs of interest. SAGE compresses $\sim 330k$ alerts into 93 AGs for CPTC-2018, and compresses $\sim 43k$ alerts into 169 AGs for CPTC-2017. Tables 3.5 and 3.6 show a team-level break-down of the reduction in the volume of alerts by each component of SAGE for CPTC-2018 and CPTC-2017, respectively. They show the number of raw intrusion alerts triggered by each team, which are then filtered for duplicate alerts and aggregated into episodes. The episodes are converted into episode sequences, which are then broken into individual attack attempts, and are then converted into input traces for the S-PDFA. Note that the S-PDFA is learned on traces with a minimum length of 3 symbols. Hence, the number of traces is often fewer than the number of episode sub-sequences.

Note that the last column of Tables 3.5 and 3.6 shows the number of AGs each team participates in, *i.e.*, the number of unique (victim, objective) combinations attacked by a team. The number of AGs per dataset (*i.e.*, 93 and 169) is hence an upper-bound for the last column. For instance, we observe that while no team participates in all of the AGs, T4 in CPTC-2017 comes close – T4 exploits 120 out of 169 combinations of victim hosts and objectives.

Attack graphs for CCDC-2018. The CCDC-2018 dataset is used to understand the generalizability of SAGE. Table 3.7 demonstrates that SAGE can successfully generate AGs for over a million alerts with no prior information about attacker/victim IPs and the underlying infrastructure. Having information about attacker/victim IPs (or even internal/external hosts) definitely helps to create targeted AGs, but it is not necessary. SAGE is also

Table 3.7: Statistical summary of the alert-driven attack graphs for each experimental dataset. The granularity of the objectives is set to the default (*attack stage, targeted service, state identifier*).

Dataset	Alerts	AGs	Average objectives per AG	Average paths per AG	Average path length per AG	Generation time (s)
CPTC-2018	330,270	93	1.3	4.1	8.6	49.1
CPTC-2017	43,611	169	1.4	4.7	9.1	20.9
CCDC-2018	1,052,281	139	1.2	2.0	2.1	141.0

not limited to AG generation for penetration testing competition datasets – CCDC-2018 contains alerts from a blue teaming exercise.

We observe that several traces in CCDC-2018 pertain to the same IP as both, source and destination. This may indicate a spoofing attempt or misconfiguration of the IDS. This unique property of the dataset results in several more AGs compared to, *e.g.*, the CPTC-2018 dataset. Furthermore, the average size of the AGs for CCDC-2018 is smaller compared to the AGs generated for the CPTC datasets.

Statistical properties of attack graphs. Table 3.7 shows that for the three experimental datasets combined, SAGE compresses 1,426,162 alerts into 401 AGs. Even though CPTC-2017 has significantly fewer alerts compared to CPTC-2018, more AGs are generated because there are more hosts and services present in the dataset. This naturally follows the observation we made for the CPTC-2017 S-PDFA model in Section 3.6.2. Table 3.7 also summarizes the average number of unique (contextual) objectives exploited in each AG, the average number of paths in each AG, the average length of these paths, and the time it takes to construct these AGs per dataset.

We observe that the S-PDFA identifies, on average, 1.2 to 1.4 unique objectives per AG. Typically, the AGs show a minimum of 1 and a maximum of 4 semantic ways of exploiting an objective for all three datasets. There are, on average, 4 to 5 paths in each AG for the CPTC-2017 and CPTC-2018 datasets, which also follow a long-tailed distribution. These paths contain, on average, 8 to 9 episodes. Note that it is possible for multiple paths to belong to the same attacker that are split in different sub-attempts. CCDC-2018 is an exception with fewer average paths per AG and shorter average path lengths. This is because the dataset contains a lot of hosts that maintain short-lived interactions with other hosts. This phenomenon may be explained by the fact that the alerts are only related to the defensive blue teaming exercise, and the ones related to the offensive red teaming are excluded.

Finally, the table also lists the execution time of SAGE for each dataset. Combined for the three datasets, it takes SAGE 3.5 minutes to process over 1.4M alerts (including parsing the alert files and rendering the AGs). It is important to note that a significant fraction of the execution time goes towards parsing the alerts, which is why CPTC-2017 has the lowest and CCDC-2018 has the highest execution time.

Attack graph complexity analysis. We evaluate the complexity of the AGs using the model simplicity metric proposed by De Alvarenga *et al.* [41] for process mining, *i.e.*, $Simplicity(AG) = \frac{|V|}{|E|}$, where $|V|$ and $|E|$ are the number of vertices and edges, respectively.

Table 3.8: Simplicity of the alert-driven attack graphs computed for each experimental dataset.

Dataset	Average vertices per AG	Average edges per AG	Average Simplicity
CPTC-2018	16.0	33.2	0.48
CPTC-2017	20.2	36.6	0.55
CCDC-2018	4.9	5.4	0.91

3

We use this metric as a proxy for interpretability – we assume that simpler AGs produce lower cognitive load for understanding attacker strategies.

Table 3.8 shows the average simplicity of the alert-driven AGs for the three experimental datasets. For each dataset, the average number of vertices and edges are significantly lower than those reported by [41], *i.e.*, AGs with more than 30 vertices are considered complex. These AGs also show the paths for all teams, making strategy comparison much easier. Furthermore, while the average simplicity of CCDC-2018 is higher than the other datasets, the AGs are significantly smaller, making it easier to analyze them.

3.7. LIMITATIONS AND FUTURE WORK

SAGE leverages interpretable sequence learning to compress thousands of alerts into a few objective-oriented attack graphs (AG). Further research is required to investigate the attacker behavior dynamics captured by these AGs, and to what extent they can be utilized in operational settings. In addition, attack graph querying and prioritization is an important direction for future work since it will enable analysts to reach the most interesting attack paths quicker. We address these limitations in Chapter 4.

The current method for episode sequence construction does not show distributed attacks in the same AG. Although changing the granularity of the sequence construction is a simple fix, it produces considerably larger AGs. Thus, a trade-off is required between sequence granularity and AG size.

Learning from infrequent data is a difficult problem, which is exacerbated by the unavailability of labeled data. 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.

It is also currently unclear how to empirically measure interpretability and the usefulness of an AG. Metrics like AIC, BIC, and Perplexity produce arbitrary values for models learned on different parameters, making the comparison meaningless. Further research is required into the design of a metric to measure AG quality.

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, and leave it for future work.

Another open question for SAGE is its handling of on-going attacks. Currently, only the state sequences that reach an objective are part of its corresponding AG. If the AGs can be generated in real-time, evolving attacks can be monitored and highlighted. The AGs can potentially even be used to predict next attack steps, thus enabling proactive

defense and dynamic risk assessment.

3.8. CONCLUSIONS

Intrusion alerts play a critical role in extracting intelligence about attacker strategies, which is 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.

Our experiments show that i) the S-PDFA we learn for SAGE accurately models the underlying data and can also generalize to unseen data, and ii) SAGE is agnostic to host and network properties: SAGE is capable of producing insightful AGs even when no ground truth about attackers and the target network is available. SAGE compresses over 1.4M alerts in exactly 401 AGs in under 5 minutes. The resulting AGs are also simpler than those proposed in existing works. Analysts can triage critical attacks by investigating a few of these AGs, drastically reducing their workload. SAGE is released in a docker container for cross-platform support.

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.

3.9. SUPPLEMENTARY MATERIAL

3.9.1. ATTACK STAGE DEFINITION

Table 3.9 shows the micro attack stages defined by the Action-Intent Framework (AIF), and their severity levels as defined in this chapter.

3.9.2. TARGETED SERVICE MAPPING

Table 3.10 lists the categorization of the targeted services (in CPTC-2018) according to their functionality.

3.9.3. S-PDFA MODEL FOR CCDC-2018

Figure 3.9 shows the S-PDFA model learned for the CCDC-2018 dataset using the parameters specified in the chapter.

REFERENCES

- [1] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, "Alert-driven Attack Graph Generation using S-PDFA", *IEEE Transactions on Dependable and Secure Computing*, 2021.

Table 3.9: Attack stages from Moskal *et al.* and their severity levels.

Acronym	Micro attack stage	Severity
surf	Surfing	Low
hostD	Host Discovery	Low
SerD	Service Discovery	Low
vulnD	Vulnerability Discovery	Low
infoD	Information Discovery	Low
uPrivEsc	User Privilege Escalation	Med
rPrivEsc	Root Privilege escalation	Med
bfCred	Brute force Credentials	Med
acctManip	Account Manipulation	Med
PAexp	Public Application Exploitation	Med
remoteexp	Remote Service Exploitation	Med
CnC	Command and Control	Med
lateral	Lateral movement	Med
ACE	Arbitrary code execution	Med
privEsc	Privilege escalation	Med
netDOS	Network Denial of Service	High
resHJ	Resource hijacking	High
dManip	Data manipulation	High
exfil	Data exfiltration	High
delivery	Data delivery	High
dDestruct	Data destruction	High

Table 3.10: Service to *Theme()* mapping according to functionality.

Category/Theme	Services
http(s)	'http', 'https', 'ddi-udp-1', 'radan-http'
wireless	'wap-wsp'
voip	'sip', 'sips'
browser	'vrml-multi-use'
searchEng	'search-agent'
broadcast	'ssdp', 'snmp', 'complex-main', 'icmpd', 'wsdapi'
nameserver	'domain', 'netbios-ns', 'menandmice-dns'
	'ssh', 'rftb', 'us-cli', 'ahsp', 'spt-automation', 'asf-rmcp', 'xdmcp', 'pcanywherestat', 'esmagent', 'irdmi',
remoteAccess	'epmap', 'wsman', 'icslap', 'ms-wbt-server', 'sunrpc'
	'appiq-mgmt', 'mosaicssysvc1'
surveillance	'remoteware-cl', 'ads-c', 'syslog', 'websm', 'distinct'
hostingServer	'cslister', 'etlservicemgr', 'web2host'
printService	'pharos', 'ipps'
email	'smtp', 'imaps', 'pop3', 'imap', 'pop3s', 'submission'
authentication	'kerberos', 'nv-video'
ATCComm	'cpdlc', 'fis'
storage	'http-alt', 'ncube-lm', 'postgresql', 'mysql', 'cm', 'ms-sql-s', 'ms-sql-m'
dataSharing	'ftp', 'pcsync-https', 'ndmp', 'netbios-ssn', 'microsoft-ds', 'profinet-rt', 'instantia'
clocksync	'ntp'
unassigned	'unknown', 'Unknown'

- [2] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, "Enabling Visual Analytics via Alert-driven Attack Graphs", in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2021.
- [3] W. U. Hassan, S. Guo, D. Li, *et al.*, "Nodoze: Combatting threat alert fatigue with automated provenance triage", in *Network and Distributed System Security*, 2019.
- [4] T. Casey, *Survey: 27 Percent of IT professionals receive more than 1 million security alerts daily*, Accessed: 08-Jul-2021, 2018. [Online]. Available: <https://www.imperva.com/blog/27-percent-of-it-professionals-receive-more-than-1-million-security-alerts-daily>.
- [5] R. Sadoddin and A. Ghorbani, "Alert correlation survey: Framework and techniques", in *PST*, 2006.
- [6] S. Salah, G. Maciá-Fernández, and J. E. DiAz-Verdejo, "A model-based survey of alert correlation techniques", *Computer Networks*, 2013.
- [7] F. M. Alserhani, "Alert correlation and aggregation techniques for reduction of security alerts and detection of multistage attack", *IJASCSE*, 2016.
- [8] L. Williams, R. Lippmann, and K. Ingols, "GARNET: A graphical attack graph and reachability network evaluation tool", in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, Springer, 2008.
- [9] M. Chu, K. Ingols, R. Lippmann, S. Webster, and S. Boyer, "Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR", in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2010.

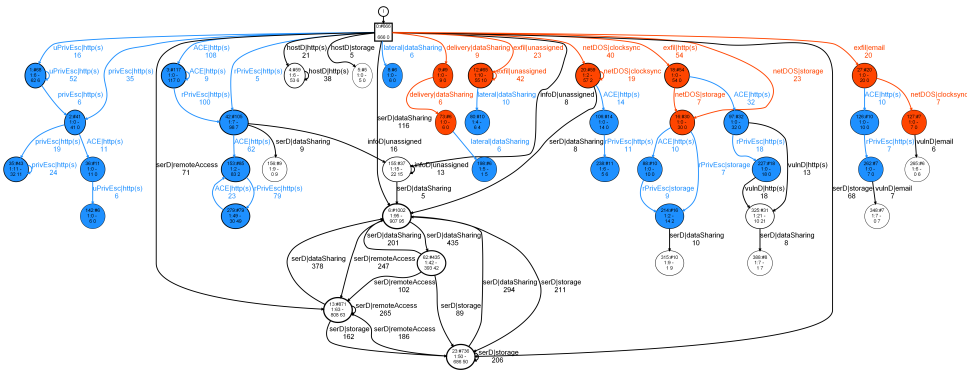


Figure 3.9: The S-PDFA model for CCDC-2018.

- [10] M. Angelini, N. Prigent, and G. Santucci, “Percival: proactive and reactive attack and response assessment for cyber incidents using visual analytics”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2015.
- [11] S. Roschke, F. Cheng, and C. Meinel, “A new alert correlation algorithm based on attack graph”, in *CISIS*, Springer, 2011.
- [12] C. Liu, A. Singhal, and D. Wijesekera, “Using attack graphs in forensic examinations”, in *International Conference on Availability, Reliability and Security*, IEEE, 2012.
- [13] K. Kaynar, “A taxonomy for attack graph generation and usage in network security”, *JISA*, 2016.
- [14] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O’Hare, and K. Prole, “Advances in topological vulnerability analysis”, in *CATCH*, IEEE, 2009.
- [15] X. Ou, S. Govindavajhala, and A. W. Appel, “MulVAL: A Logic-based Network Security Analyzer”, in *Proc. of the USENIX Security Symposium*, Baltimore, MD, 2005.
- [16] M. L. Artz, “NetSPA: A network security planning architecture”, Ph.D. dissertation, Massachusetts Institute of Technology, 2002.
- [17] S. Jha, O. Sheyner, and J. Wing, “Two formal analyses of attack graphs”, in *CSFW*, IEEE, 2002.
- [18] C. Sillaber, C. Sauerwein, A. Mussmann, and R. Brey, “Data quality challenges and future research directions in threat intelligence sharing practice”, in *WISCS*, 2016.
- [19] A. Nadeem, S. Verwer, and S. J. Yang, “SAGE: Intrusion Alert-driven Attack Graph Extractor”, in *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2021.
- [20] 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.

- [21] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation", *IEEE Transactions on Dependable and Secure Computing*, 2004.
- [22] S. Verwer and C. A. Hammerschmidt, "Flexfringe: A passive automaton learning package", in *ICSME*, IEEE, 2017.
- [23] *Global Collegiate Penetration Testing Competition*, Accessed: 08-Jul-2021, 2021. [Online]. Available: <https://globalcptc.org/>.
- [24] *National Collegiate Cyber Defense Competition*, Accessed: 08-Jul-2021, 2021. [Online]. Available: <http://www.nationalccdc.org/>.
- [25] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts", in *Proceedings of the 2002 ACM SIGSAC Conference on Computer and Communications Security*, 2002.
- [26] X. Qin and W. Lee, "Discovering novel attack strategies from INFOSEC alerts", in *ESORICS*, Springer, 2004.
- [27] B. Zhu and A. A. Ghorbani, "Alert correlation for extracting attack strategies", *IJ Network Security*, 2006.
- [28] C.-H. Wang and Y.-C. Chiou, "Alert correlation system with automatic extraction of attack strategies by using dynamic feature weights", *IJCCE*, 2016.
- [29] S. Haas and M. Fischer, "GAC: Graph-based alert correlation for the detection of distributed multi-step attacks", in *SAC*, 2018.
- [30] 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.
- [31] S. McElwee, J. Heaton, J. Fraley, and J. Cannady, "Deep learning for prioritizing and responding to intrusion detection alerts", in *MILCOM*, IEEE, 2017.
- [32] P. Ning, D. Xu, C. G. Healey, and R. S. Amant, "Building Attack Scenarios through Integration of Complementary Alert Correlation Method", in *Network and Distributed System Security*, 2004.
- [33] 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.
- [34] J. Homer, A. Varikuti, X. Ou, and M. A. McQueen, "Improving attack graph visualization through data reduction and attack grouping", in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, Springer, 2008.
- [35] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs", in *ACSAC*, IEEE, 2009.
- [36] J. Navarro, V. Legrand, S. Lagraa, *et al.*, "HUMA: A multi-layer framework for threat analysis in a heterogeneous log environment", in *FPS*, Springer, 2017.
- [37] 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.

- [38] 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.
- [39] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, “TABOR: A graphical model-based approach for anomaly detection in industrial control systems”, in *AsiaCCS*, 2018.
- [40] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, “Beyond labeling: Using clustering to build network behavioral profiles of malware families”, in *Malware Analysis Using Artificial Intelligence and Deep Learning*, Springer, 2021, pp. 381–409.
- [41] 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.
- [42] Y. Chen, Z. Liu, Y. Liu, and C. Dong, “Distributed Attack Modeling Approach Based on Process Mining and Graph Segmentation”, *Entropy*, 2020.
- [43] 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.
- [44] 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.
- [45] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, “Attention models in graphs: A survey”, *Transactions on Knowledge Discovery from Data*, 2019.
- [46] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why should I trust you?” Explaining the predictions of any classifier”, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.
- [47] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, “Explainable machine learning for scientific insights and discoveries”, *IEEE Access*, vol. 8, pp. 42 200–42 216, 2020.
- [48] N. Munaiah, A. Rahman, J. Pelletier, L. Williams, and A. Meneely, “Characterizing Attacker Behavior in a Cybersecurity Penetration Testing Competition”, in *ESEM*, IEEE, 2019.
- [49] *Service names to port number mapping*, Accessed: 08-Jul-2021, 2021. [Online]. Available: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.
- [50] S. Moskal and S. J. Yang, “Framework to Describe Intentions of a Cyber Attack Action”, *arXiv preprint arXiv:2002.07838*, 2020.
- [51] 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.
- [52] W. Cui, J. Kannan, and H. J. Wang, “Discoverer: Automatic Protocol Reverse Engineering from Network Traces”, in *Proc. of the USENIX Security Symposium*, 2007, pp. 1–14.
- [53] J. De Ruiter and E. Poll, “Protocol State Fuzzing of TLS Implementations”, in *Proc. of the USENIX Security Symposium*, 2015.

- [54] C. Y. Cho, D. Babić, E. C. R. Shin, and D. Song, “Inference and analysis of formal models of botnet command and control protocols”, in *Proceedings of the 2010 ACM SIGSAC Conference on Computer and Communications Security*, 2010, pp. 426–439.
- [55] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, “Prospex: Protocol specification extraction”, in *IEEE Symposium on Security and Privacy*, IEEE, 2009.
- [56] 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.
- [57] M. J. Heule and S. Verwer, “Software model synthesis using satisfiability solvers”, *Empirical Software Engineering*, 2013.
- [58] C. De la Higuera, *Grammatical inference: Learning automata and grammars*. Cambridge University Press, 2010.
- [59] S. Verwer, M. de Weerd, and C. Witteveen, “A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data”, in *Grammatical Inference: Theoretical Results and Applications*, Springer, 2010, pp. 203–216.
- [60] R. C. Carrasco and J. Oncina, “Learning stochastic regular grammars by means of a state merging method”, in *ICGI*, Springer, 1994.
- [61] Q. Lin, Y. Zhang, S. Verwer, and J. Wang, “MOHA: A multi-mode hybrid automaton model for learning car-following behaviors”, *T-ITS*, 2018.
- [62] A. Adadi and M. Berrada, “Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)”, *IEEE Access*, 2018.
- [63] *CPTC-2018 dataset*, Accessed: 08-Jul-2021, 2018. [Online]. Available: <https://mirror.rit.edu/cptc/>.
- [64] F. Hassanabad, *CCDC-2018 dataset*, Accessed: 08-Jul-2021, 2019. [Online]. Available: <https://github.com/FrankHassanabad/suricata-sample-data>.
- [65] *Suricata IDS*, Accessed: 08-Jul-2021, 2019. [Online]. Available: <https://suricata.readthedocs.io/en/suricata-6.0.3/>.
- [66] K. Sowka, V. Palade, H. Jadidbonab, P. Wooderson, and H. Nguyen, “A Review on Automatic Generation of Attack Trees and Its Application to Automotive Cybersecurity”, *Artificial Intelligence and Cyber Security in Industry 4.0*, pp. 165–193, 2023.
- [67] R. Parekh and V. Honavar, “Learning DFA from simple examples”, *Machine Learning*, 2001.
- [68] T. Cover and J. Thomas, *Elements of information theory*. John Wiley & Sons, 1991.
- [69] S. Verwer, R. Eyraud, and C. De La Higuera, “PAutomaC: A probabilistic automata and hidden Markov models learning competition”, *Machine learning*, vol. 96, no. 1-2, pp. 129–154, 2014.
- [70] 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.

4

ENABLING VISUAL ANALYTICS VIA ALERT-DRIVEN ATTACK GRAPHS

The knowledge of attacker strategies that is specific to a network under observation serves as actionable cyber threat intelligence (CTI). 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). Recently, alert-driven AGs have emerged as a novel paradigm of attack graphs that reverse engineer attacker strategies directly from the actions observed through intrusion alerts without any expert input. However, a comprehensive qualitative analysis of these attacker strategies remains unexplored.

In this chapter, we learn alert-driven attack graphs from two diverse intrusion alert datasets and investigate the attacker behavior dynamics they capture. Specifically, we show how these AGs (a) enable forensic analysis of prior attacks, and (b) enable proactive defense by providing relevant threat intelligence regarding attacker strategies. We also build a dashboard for the alert-driven AGs that provides querying and prioritization capabilities to analysts, further reducing their workload. We believe that alert-driven AGs can play a key role in AI-enabled cyber threat intelligence as they open up new avenues for attacker strategy analysis whilst reducing analyst workload.

Parts of this chapter are based on the paper “Alert-driven Attack Graph Generation using S-PDFA.” by **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. in IEEE Transactions on Dependable and Secure Computing 2021, 19(2), 731-746 [1], and poster “Critical Path Exploration Dashboard for Alert-driven Attack Graphs” by **Nadeem, A.**, Diaz, S. L., & Verwer in IEEE Symposium on Visualization for Cyber Security (VizSec), 2022 [2].

4.1. MOTIVATION & RELATED WORK

Analysts working in Security Operations Centers (SOC) are responsible for managing, triaging, and analyzing large volumes of intrusion alerts for the forensic analysis of security incidents. The process of reverse engineering attacker objectives and strategies from intrusion alerts is often manual and time-intensive [3].

Extensive research has been conducted to reduce analyst workload by introducing ML-powered visual analytics tools in their workflows that visualize large datasets, identify unique patterns, and provide actionable intelligence. Graphical dashboards are popular for providing statistical insights into intrusion alerts [4]–[9]. There is also a growing interest in utilizing alert sequences for classifying critical security events [10], conducting root-cause analysis of security events [11], predicting next attack steps [12], and modeling attack campaigns [13]. The focus of these studies is typically on detecting attacks and triaging intrusion alerts. Existing literature rarely conducts qualitative analyses to understand the extent to which attacker strategies are captured by the machine learning models. This is an important question because such analyses can discover and visualize hidden patterns in input data for augmenting human intelligence. For instance, the analysis can equip analysts to compare attacker strategies, distinguish novice from expert attackers, fingerprint different styles of strategies, or detect *unknown unknowns*.

Recently, ‘alert-driven attack graphs’ (AGs) have been proposed as a novel paradigm of attack graphs that reverse engineer attacker strategies from the actions observed through intrusion alerts [1]. The attack graph generator, SAGE [14], automatically discovers and summarizes attacker strategies as individual attack paths in objective-oriented AGs. However, a comprehensive qualitative analysis assessing the effectiveness of these AGs in capturing attacker strategies is currently lacking. We hypothesize that by analyzing alert-driven AGs, we can identify and characterize the behavior dynamics of attackers, which would enable security analysts to differentiate between novice and expert attackers. This is an interesting finding for criminologists, and could have significant military applications. For example, the AGs could be used to determine how vulnerable an asset is based on the complexity of the strategies that exploit it. Additionally, we could even create profiles of attackers based on their unique style of strategies.

Another challenge is that while SAGE does the heavy lifting in terms of discovering and displaying attacker strategies, it is infeasible and time-consuming for security analysts to visualize each AG separately for finding global patterns. Even though the average complexity of SAGE AGs is lower than alternative modeling approaches (see Chapter 3), the AGs of relatively common objectives (e.g., data exfiltration on HTTP) can be significantly more complex with hundreds of vertices. The AGs also lack interactivity, *i.e.*, attack paths cannot be filtered by time or attack stage of interest. Preliminary interviews with security analysts revealed that such large graphs with no interaction capabilities are unhelpful in an operational setting. Finally, SAGE does not prioritize critical attack paths¹ that might need the urgent attention of security analysts.

To this end, we first conduct a rigorous qualitative analysis of the attacker strategies modeled by the alert-driven AGs. Specifically, our investigation focuses on identifying the attacker behavior dynamics present in the AGs, and assessing whether we can obtain

¹A critical path is a series of actions that leads to critical event(s).

actionable threat intelligence regarding potentially scripted attack attempts and fingerprintable paths. In addition to the qualitative analysis, we build a *critical path exploration dashboard* to consolidate the alert-driven AGs in a single location, and to further empower security analysts. Specifically, we develop a web-based visual analytics dashboard for alert-driven AGs with querying and prioritization capabilities for critical attack paths. The dashboard reduces analyst workload by i) discovering and extracting attacker strategies from intrusion alerts (done by the SAGE module); ii) consolidating the discovered attacker strategies in a dashboard with filtering capabilities, and iii) prioritizing critical events that might require an analyst's urgent attention.

We utilize two security testing competition datasets to extensively evaluate the AGs generated by SAGE, and to test the efficacy of the dashboard. We show that SAGE is generalizable, and that the AGs unlock a new means to derive intelligence regarding attacker strategies without having to investigate thousands of intrusion alerts. 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. On one of the experimental datasets, they reveal that attackers follow a shorter path to re-exploit an objective after they have already discovered a longer path in 84.5% of the cases. We also provide an exemplary use case for the dashboard demonstrating how the querying and prioritization functionalities further help triage attack paths of interest. In summary, our main contributions are:

1. We evaluate the behavioral analytics enabled by alert-driven attack graphs, *i.e.*, in terms of attacker strategy comparison and fingerprintable path discovery.
2. We demonstrate the generalizability of the attack graphs by investigating three use cases generated from an unexplored blue teaming exercise.
3. We build a web-based dashboard to consolidate all the alert-driven attack graphs by providing querying and prioritization capabilities to security analysts.

Deployability of Alert-driven Attack Graphs. 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 for the following two types of end-users:

1. **SOC analysts.** The primary use case explored in this chapter is about enabling SOC analysts to 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. Section 4.4.1 discusses concrete examples of interpreting and comparing attacker strategies. The occurrence of certain paths in an AG can also serve as fingerprints. Additionally, attacker hosts can be ranked based on the severity of alerts they raise.
2. **Red teams.** As an adversarial use case, SAGE acts as a monitoring intermediary during red team training. After a training session, the teams can review alert-driven AGs for gaining insights, such as (a) identifying the shortest path to an objective that was discovered by a team member, or (b) showing redundant paths,

for instance, due to lack of communication between the team members. Enumerating all paths toward an objective can help the teams develop creative strategies, see Section 4.4.2. The teams can use such feedback to improve their performance.

Organization. Section 4.2 provides a summary of the alert-driven attack graphs and the design details of the dashboard. Section 4.3 explains the experimental setup. Section 4.4 presents the results pertaining to attacker strategy analysis, generalization of SAGE, and an exemplary use case of how the dashboard helps investigate attacks. We discuss the limitations and the practical implications for cyber threat intelligence in Sections 4.5 and 4.6. Finally, we conclude in Section 4.7.

4

4.2. METHODOLOGY

In this section, we first present a brief summary of SAGE and our method for validating the alert-driven attack graphs. Then, we explain the design of the critical path exploration dashboard.

4.2.1. ALERT-DRIVEN ATTACK GRAPHS

SAGE² discovers attacker strategies in intrusion alerts using unsupervised machine learning and represents them as attack graphs. The choice of the machine learning algorithm is driven by the following challenges:

- Class imbalance between severe (*e.g.*, exploitation) and non-severe (*e.g.*, scanning) alerts presents a huge difficulty. Severe alerts are infrequent, while non-severe alerts reflect an important aspect of an attacker's strategy. A solution that keeps both type of alerts, while highlighting infrequent alerts is required. This is a tricky problem because most ML solutions discard infrequent events.
- The future and past of a given alert captures important contextual cues about the intent of an attacker. Thus, the proposed solution must model this context to distinguish between similar alerts that lead to different attacks.
- Black-box solutions that security analysts cannot understand are undesirable, thus calling for an explainable approach.

SAGE is an interpretable, unsupervised, sequential learning pipeline that compresses thousands of intrusion alerts into AGs without a priori expert knowledge about existing vulnerabilities and network topology. It starts by aggregating raw intrusion alerts into episode (hyper-alert) sequences. The temporal and probabilistic dependence between alerts is leveraged using a suffix-based probabilistic deterministic finite automaton (S-PDFA) – it is an interpretable and deterministic graphical model of all attack paths present in an alert dataset. A suffix-based PDFA is utilized to accentuate infrequent severe alerts and to model context. The model distinguishes between episodes with different contexts but identical signatures so as to differentiate between similar attacker

²SAGE: <https://github.com/tudelft-cda-lab/SAGE>

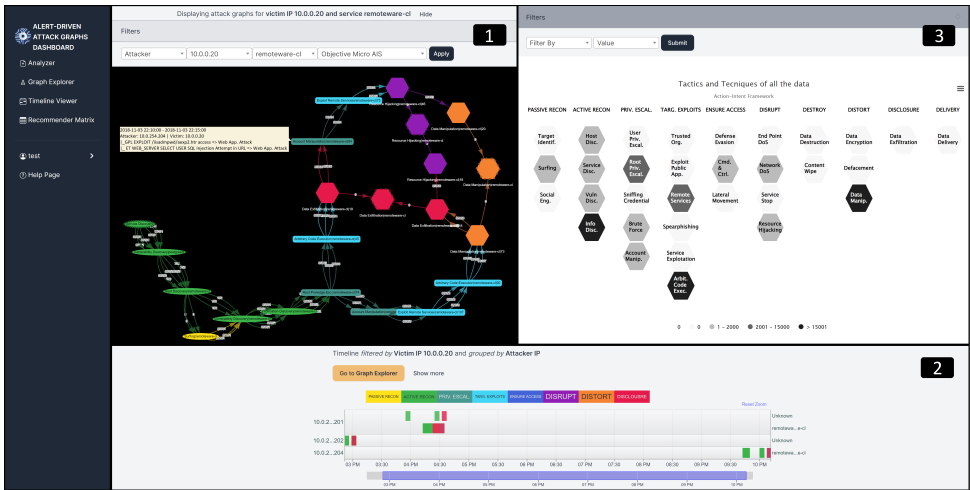


Figure 4.2: The critical path exploration dashboard for alert-driven attack graphs. Attacker strategies extracted from the uploaded intrusion alerts are visualized as: (1) *Graph explorer* showing a unified view of all strategies, (2) *Timeline viewer* allowing the analysis of individual attacker actions, and (3) *Recommender matrix* highlighting the alerts that require the urgent attention of an analyst.

We validated the completeness of the AGs by matching them against the teams' self-reported claims. We found that most of the AGs supported at least one of the claims. In fact, the AGs provided significantly more detail into attacker strategies than the steps described by the teams. Some claims did not have corresponding attack paths, which could indicate that those actions did not trigger any alerts. Further investigation is required to understand what causes such missing paths.

We also conducted an informal user study with two senior security researchers regarding the correctness and usability of the AGs, whose responses suggested that SAGE is a promising tool for extracting insights from intrusion data. They also indicated that large AGs with no interaction capabilities might be unhelpful in an operational setting.

4.2.2. CRITICAL PATH EXPLORATION DASHBOARD FOR ALERT-DRIVEN ATTACK GRAPHS

The proposed critical path exploration dashboard is a Django application, implemented as a wrapper around SAGE, responsible for visualizing the attacker strategies extracted by SAGE. Figure 4.2 shows the components of the proposed dashboard. Intrusion alert files can be uploaded in JSON format to the dashboard. This triggers the execution of the SAGE module, which processes the alerts, applies unsupervised sequence learning, extracts temporal and probabilistic patterns from the alerts, and produces state sequences that reflect attacker strategies. The discovered attacker actions and their temporal relationships are then stored in a relational SQLite database that is later queried to populate the three visualizations, *i.e.*, graph explorer, timeline viewer, and recommender matrix.

In summary, the *graph explorer* presents a consolidated view of all attacker strategies discovered by SAGE, and interactively visualizes them in order to find relationships

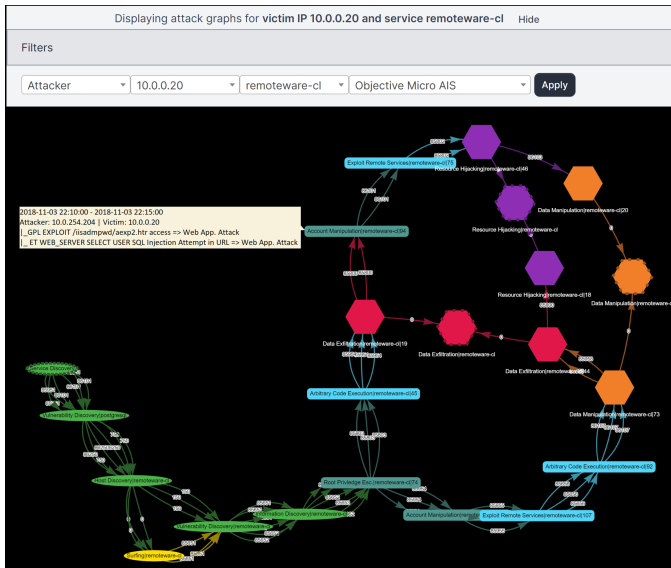


Figure 4.3: The graph explorer allows analysts to view composite attack graphs based on selected filters, and also allows them to view relevant alert signatures by selecting vertices of interest.

Arbitrary Code Execution http 96						
Start Time	End Time	Attacker	Victim	Signature	Category	Frequency
Sat, 03 Nov 2018 15:20:31 GMT	Sat, 03 Nov 2018 15:25:23 GMT	10.0.254.204	10.0.1.5	ET WEB_SERVER Possible CVE-2014-6271 Attempt	Web Application Attack	122
Sat, 03 Nov 2018 15:25:23 GMT	Sat, 03 Nov 2018 18:37:19 GMT	10.0.254.204	10.0.1.5	ET WEB_SPECIFIC_APPS Horde type Parameter Local File Inclusion Attempt	Web Application Attack	3

Figure 4.4: An alert signature table is displayed upon selecting a vertex in the graph explorer. The table can be exported in various formats for reporting purposes.

between attacker objectives. The *timeline viewer* enables the analyst to investigate temporal correlations between attacker actions, and compare the tactics/techniques used by the attackers at different timestamps. The *recommender matrix* shows a condensed version of the MITRE ATT&CK stages that assists the analyst in prioritizing critical events based on their prevalence and urgency in the alert dataset.

GRAPH EXPLORER

A global, unified view of all attacker strategies is shown in the graph explorer (see Figure 4.3). Analysts can use filters to query for specific attack paths based on the attacker host, victim host, targeted service, and attack stage of interest. This view enables analysts to draw conclusions about the most frequently used pathways towards an objective, and the temporal inter-dependency between them. The graph explorer is implemented using the *vis.js* Network chart that enables efficient interactions with large graphs. The

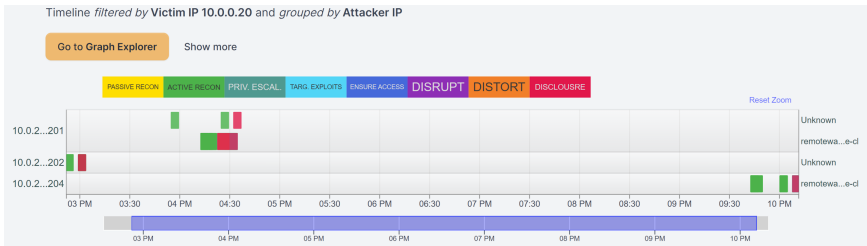


Figure 4.5: The timeline viewer allows analysts to view and analyze individual attacker actions/episodes by setting filters and zooming in on time windows of interest.

4

vertices and edges in the graph represent the same information as the SAGE AGs, *i.e.*, the vertices correspond to the attacker actions (showing the attack stage, targeted service, and state identifier), while the edges show the temporal relation between the vertices. A single-click on a vertex displays a signature table that shows the alert signatures that are triggered by the corresponding attacker action (see Figure 4.4). The table shows the start and end times of the attacker action (*i.e.*, episode), the attacker and victim IP addresses, and the frequencies of the different alert signatures. This table can be exported in multiple formats to be used for reporting.

TIMELINE VIEWER

The timeline viewer (see Figure 4.5) enables analysts to focus on specific attacker actions that occurred during a user-selected time-window to determine, for instance, if a victim is targeted by numerous attackers at the same time. The timeline viewer is implemented using the D3 `timelines-chart`, which supports several *swimlanes*. Each swimlane corresponds to the various actions taken by an attacker on a victim using a specific service. The segments in a swimlane correspond to the vertices in the graph explorer (*i.e.*, attacker actions). The segment color corresponds to the action's attack stage. The swimlanes are either grouped by attacker or victim IP and can be filtered by time. Once the interesting actions are narrowed down, clicking on 'Go to Graph Explorer' redirects to the graph explorer and shows the corresponding attack paths.

RECOMMENDER MATRIX

The recommender matrix (see Figure 4.6) shows a condensed version of the MITRE ATT&CK framework, similar to the ATT&CK Navigator³, where the different attack stages are highlighted based on the urgency of the observed alerts. The urgency score for each attack stage is calculated based on two factors – severity and prevalence (see Eq. 4.1). First, each attack stage is assigned a default severity level according to the original SAGE paper [1]. We quantify the severity levels with weights between [0, 1], *i.e.*, low-severity (0.25), medium-severity (0.5), and high-severity (1.0). The severity levels and weights can be adjusted based on the risk appetite of a SOC. Second, we compute the prevalence of each attack stage based on their occurrence frequency. We normalize the prevalence of attack stages to accommodate for the imbalance in the alert types, shown by Eq. 4.2.

³ATT&CK Navigator: <https://mitre-attack.github.io/attack-navigator/>

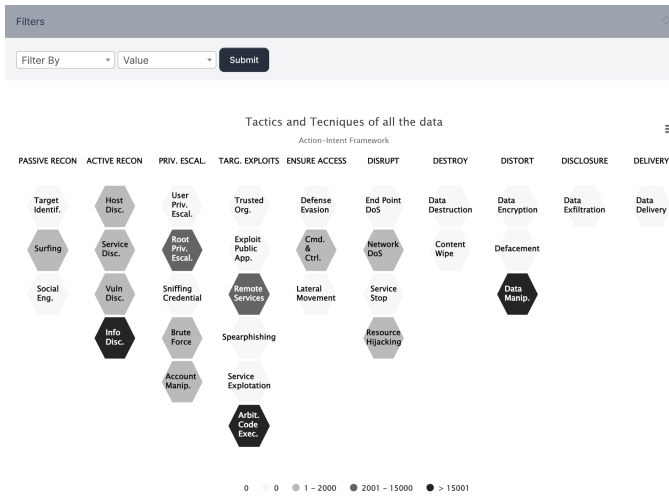


Figure 4.6: The recommender matrix allows analysts to triage critical attack stages by computing their urgency based on the observed alerts, and filtering on hosts of interest. By clicking on a highlighted attack stage, the dashboard redirects to the graph explorer with only the attack paths that contain the selected attack stage.

Here, V contains all attack graph vertices, and $Count(V, mcat)$ gives the count of vertices with the specific attack stage $mcat$.

$$urgency(mcat) = severity(mcat) * prevalence(mcat) \quad (4.1)$$

$$prevalence(mcat) = \frac{Count(V, mcat)}{|V|} \quad (4.2)$$

The recommender matrix is implemented using the Highcharts Honeycomb tile map. Each hexagonal tile in the tile map shows an attack stage from the Action-Intent Framework (AIF) [15], while its color shows how urgently it needs to be assessed (darker \rightarrow more urgent). The tiles from left to right increase in their severity, *i.e.*, the left tiles typically show the start of an attack and the right tiles show adversary objectives. Clicking on a tile with a non-zero urgency score redirects the user to the graph explorer, which shows the specific attack paths where the particular attack stage occurs (which is colored white for easier identification). This allows the analysts to gain a broader perspective on the different kinds of attacks that were enabled due to the selected technique.

4.3. DATASET AND EXPERIMENTAL SETUP

Datasets. We use two open-source intrusion alert datasets: one dataset from the Collegiate Penetration Testing Competition (CPTC) [16] for investigating the attacker behavior dynamics, and one dataset from the Collegiate Cyber Defense Competition (CCDC) [17] for showing the attack graphs' generalizability. Table 4.1 shows the summary of the experimental datasets.

Table 4.1: Summary of the experimental datasets used to learn attacker strategies.

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

The alert datasets are generated by different student teams who are tasked to compromise/defend a common fictitious network. The CPTC-2018 dataset contains alerts by six teams (T1, T2, T5, T7, T8, T9) targeting an automotive company. 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 [18], 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.

The CCDC-2018 dataset 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. This reinforces the claim that SAGE does not need any expert input to produce insightful AGs.

Parameters. In this chapter, we set $t = 1.0$ seconds to filter repeated alerts [13], [19]. 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.

Experiments. We perform three experiments pertaining to the behavior dynamics captured by alert-driven attack graphs.

1. **Behavioral analytics.** We analyze the attack graphs generated from CPTC-2018 for explaining and comparing attacker strategies, discovering fingerprintable paths, and ranking attackers based on their observed actions (Section 4.4.1).
2. **Replication use case.** We investigate the usefulness of the attacker strategies captured by the attack graphs generated from an unexplored blue team exercise, *i.e.*, CCDC-2018 (Section 4.4.2).

3. **Dashboard usage.** We show the added utility of the critical path exploration dashboard by showing an exemplary attack investigation use case (Section 4.4.3).

4.4. RESULTS AND DISCUSSION

In this section, we show that alert-driven attack graphs (AG) reflect the actual pathways taken by the attacker teams. The AGs are powerful because they not only enable forensic analysis of prior attacks (*i.e.*, displaying and comparing attack paths), but they also provide relevant threat intelligence about attacker strategies (*i.e.*, insights into behavior dynamics, fingerprinting paths for attacker re-identification, and ranking attackers based on the uniqueness and severity of their actions). The AGs are also succinct, interpretable, and generalizable.

4.4.1. EXPLAINING ATTACKER STRATEGIES IN CPTC-2018

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 49 seconds, where most of the time is spent parsing the intrusion alerts. Below, we discuss the visual analytics enabled by SAGE for attack path interpretation, and strategic difference analysis.

COMPARING INDIVIDUAL ATTACK PATHS

Alert-driven AGs provide insights into the paths exploited by attackers. Figure 4.7 shows the strategies of three teams (the absence of other teams indicates that they were unable to achieve this objective or did not trigger alerts of this type). This graph compresses 300 alerts into 25 vertices, enabling SOC analysts to follow the attack progression.

Figure 4.7 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 alert signatures displayed by the different vertices of the AG concretely show that this was achieved by exploiting a combination of the *CodeRed v2*, *ColdFusion*, and *ShellShock* exploits.

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 $\langle \text{data_exfiltration}, \text{remoteware-cl}, 17 \rangle$ and $\langle \dots, 116 \rangle$ are reached later in the competition with fewer steps, implicitly capturing attackers' increasing experience.

Interestingly, an AG of data manipulation (Figure 4.8) results in a partial sub-graph of the AG from Figure 4.7, due to overlap in paths that attain both objectives. It shows three variants of data manipulation, two of which are also present in the exfiltration graph, *i.e.*, $\langle \text{data_manipulation}, \text{remoteware-cl}, 95 \rangle$ and $\langle \dots, 288 \rangle$. T5 finds one additional path to reach $\langle \dots, 18 \rangle$ right after it has reached $\langle \text{data_exfiltration}, \text{remoteware-cl}, 17 \rangle$ from the previous AG. These type of insights provide actionable intelligence to disrupt the cyber kill-chain [20].

4

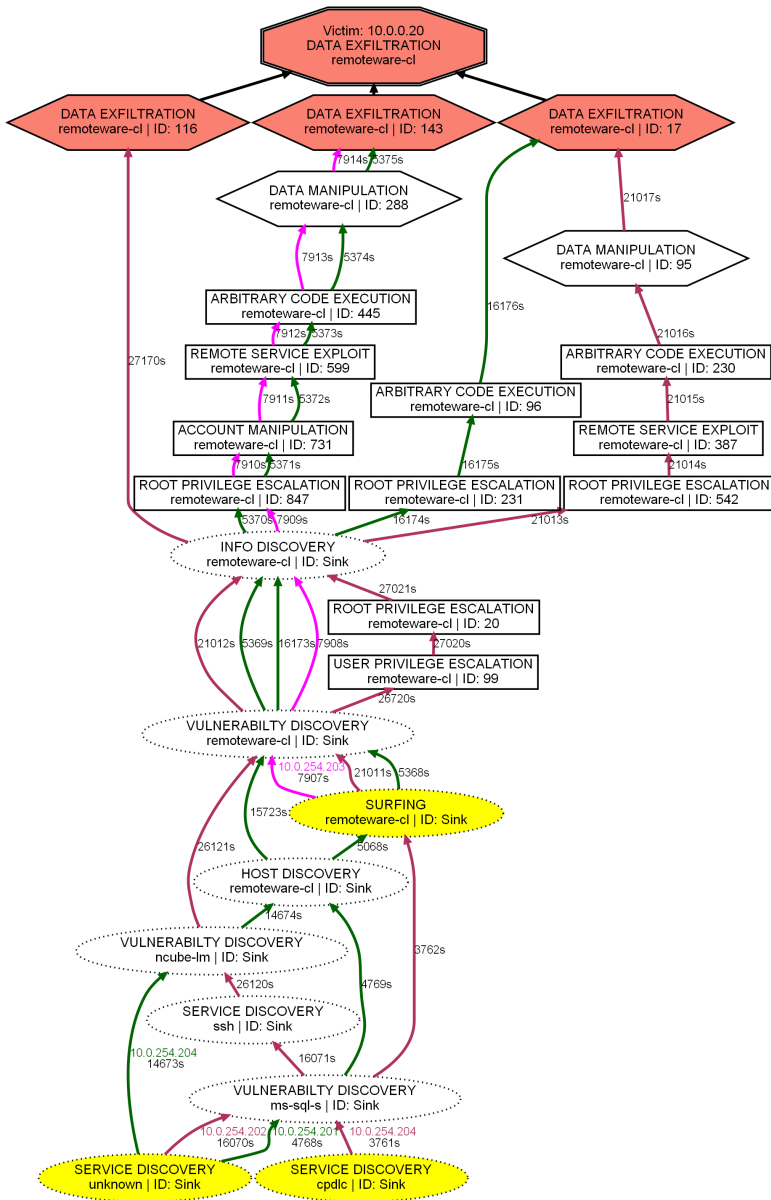


Figure 4.7: An alert-driven attack graph for 10.0.0.20 – data exfiltration over remoteware-cl (IDs are state identifiers, capturing the context). Three attacker teams exploit it: Teams 1 (maroon) and 5 (green) exploit it twice, where subsequent attempts are shorter than the previous ones. There are three ways of exploiting the objective, based on the actions that lead up to it, as determined by the S-PDFA. Sinks are states that are too infrequent for the S-PDFA to learn from (dotted vertices). Edge labels show time progression in seconds.

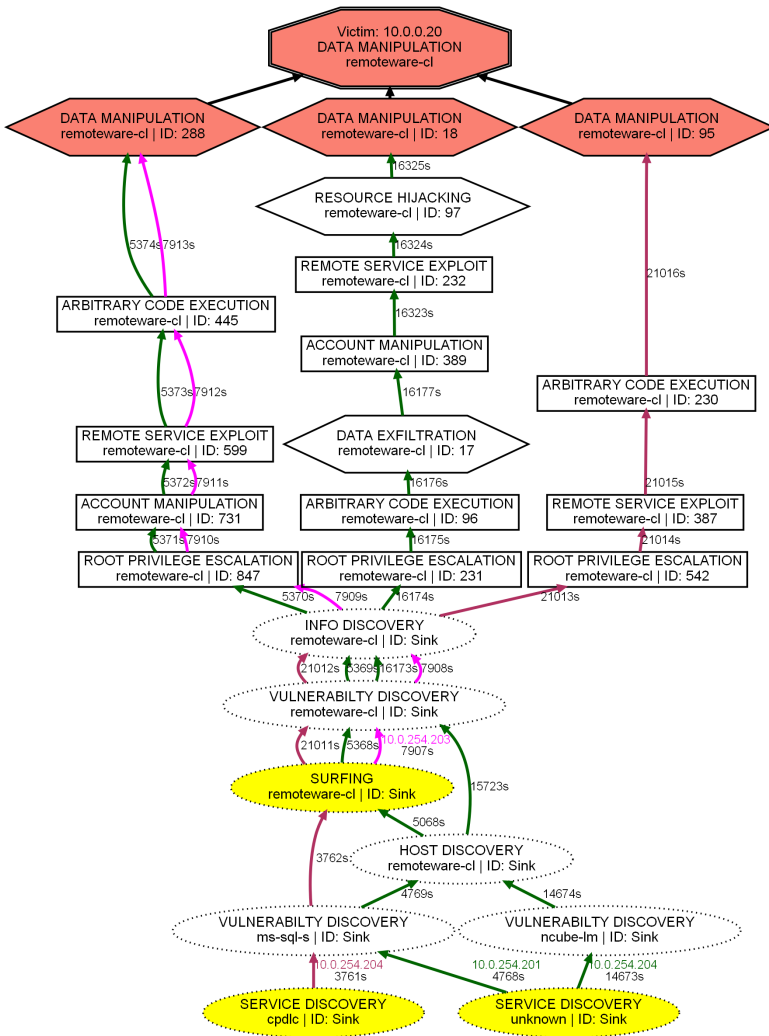


Figure 4.8: A data manipulation attack graph is a partial sub-graph of Figure 4.7 due to overlapping paths.

EXPLAINING STRATEGIC DIFFERENCES ACROSS AGS

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 Figure 4.9). 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. A simple graph edit distance is enough to automate the detection of such identical AGs.

Figure 4.10 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. This finding is backed by common-sense intuition that if an attacker already knows how to exploit an objective, they would skip the unnecessary hit-and-trial steps for re-exploitation.

4

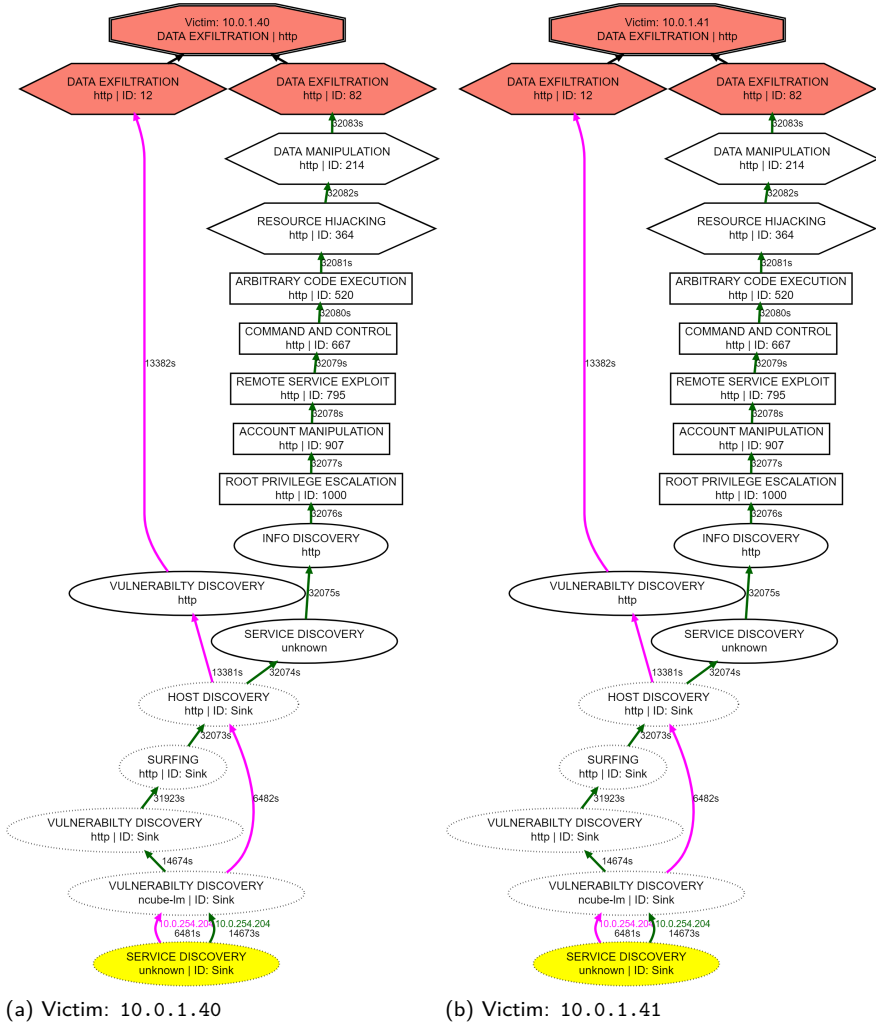


Figure 4.9: Identical and simultaneous attacks targeting multiple victim hosts result in identical attack graphs.

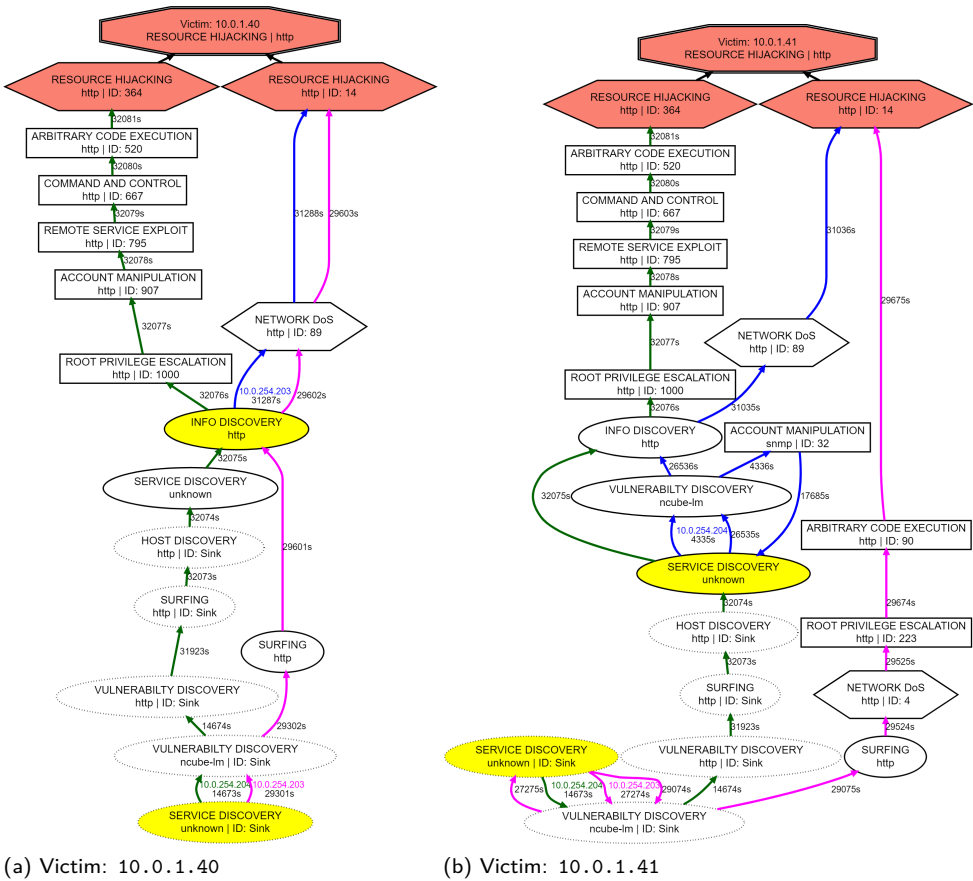


Figure 4.10: Similar attacks targeting multiple victim hosts result in overlapping attack graphs.

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* (from Chapter 3), 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 4.2 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.

Table 4.2: The 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						Fingerprint?
	T1	T2	T5	T7	T8	T9	
Data Delivery							
http	8		3	2	5	5	✓
complex-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	✓
complex-main	3*						
cslistener						1*	
wap-wsp			1*				✓
remoteware-cl	2		2		1		✓
us-cli			1		2	1	
etlservicemgr	2*						✓
unassigned	2	7	1	6	1	7	
Data Manipulation							
http	14		7	3	8	6	✓
complex-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	✓
complex-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*						✓

RANKING ATTACKER PERFORMANCE

Each vertex in an alert-driven AG (corresponding to a unique S-PDFA state identifier) signifies a new milestone achieved by an attacker. We argue that the fraction of unique

milestones discovered by an attacker provides a metric for their performance, which can be used by SOC analysts and red teams to rank *interesting attacker hosts*. A medium-severity episode serves as a stepping-stone towards a high-severity episode. Hence, we propose that high-severity vertices hold twice the weight of medium-severity vertices, *i.e.*, $\frac{(2*high)+(1*medium)}{3}$.

Table 4.3 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 4.2, which shows T2 being unsuccessful in discovering many of the objectives.

Table 4.3: CPTC-2018 team ranking based on the fraction of unique severe vertices discovered, and the number of attacker hosts responsible for discovering them.

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

4.4.2. USE CASE A: APPLYING SAGE ON AN UNEXPLORED BLUE TEAM EXERCISE

The Collegiate Cyber Defense Competition (CCDC) dataset is given as input to SAGE to verify whether the AGs provide the same interpretability and succinctness on a dataset that is not related to penetration testing. 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 can produce generalizable attack graphs, and is agnostic to host, dataset, and infrastructure properties. The cases discussed in this section verify that the alert-driven AGs require no expert knowledge to be insightful.

CASE 1 - PATH ENUMERATION

The AG in Figure 4.11 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.

4

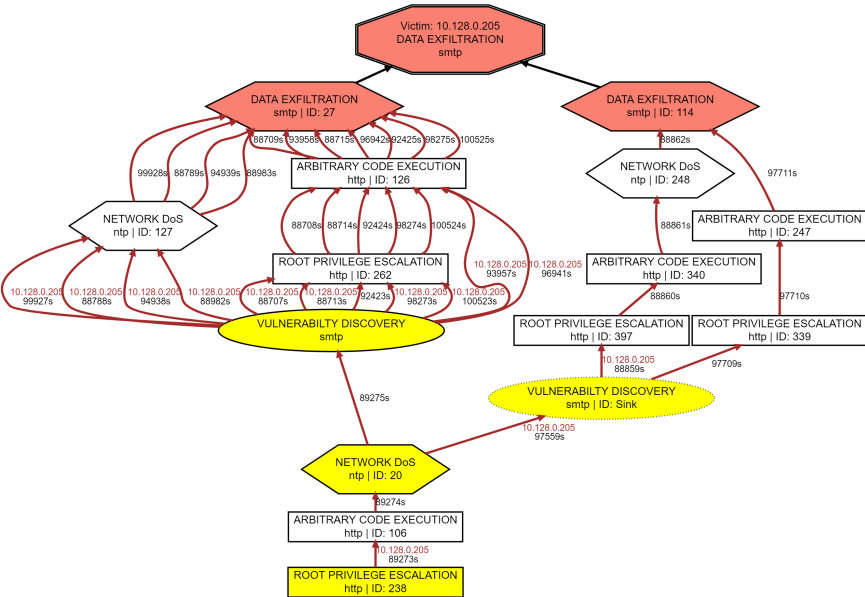


Figure 4.11: A data exfiltration over smtp attack graph in CCDC-2018 showing one attacker making 13 attempts. Paths starting from severe attack stages are possible as these attack graphs show partial attack campaigns.

CASE 2 - SHORTEST PATH

Figure 4.12 shows the AG for performing network DoS using ntp. It shows two possible variants, starting from six different vertices. Various services are targeted along the way, including http and microsoft-ds (data sharing protocol). The different attacker hosts are highlighted by different edge colors. This AG shows that it is possible to obtain this objective with just two actions, *i.e.*, data exfiltration and network DoS. This happens at the 4-hour mark. About 30 minutes later, root privilege escalation is 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.

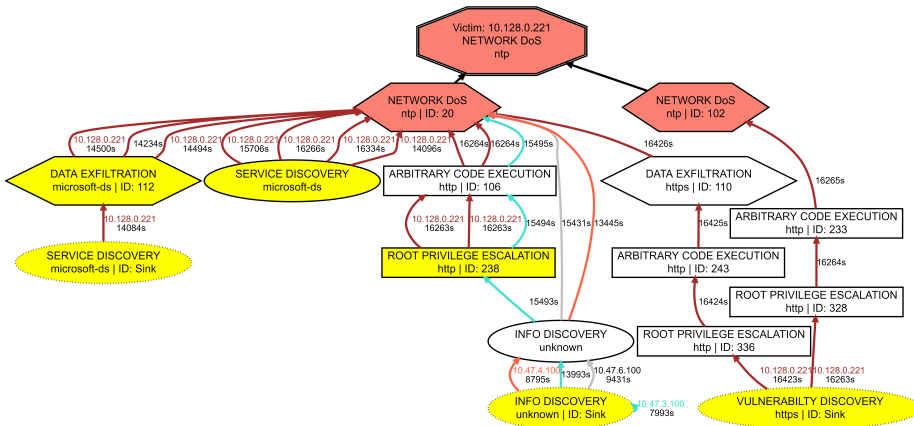
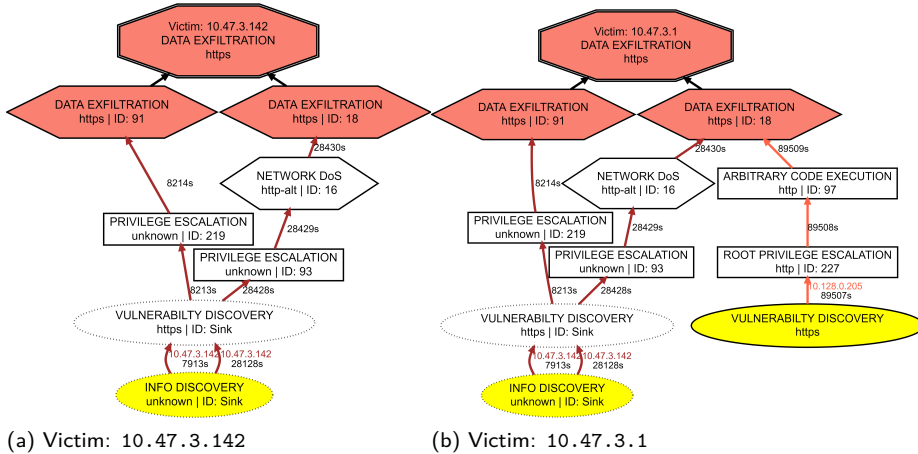


Figure 4.12: An attack graph of network DoS over ntp in CCDC-2018 showing six possible starting actions and two possible ways to reach the objective.

CASE 3 - AN EXTRA ATTEMPT

Figure 4.13 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.



(a) Victim: 10.47.3.142

(b) Victim: 10.47.3.1

Figure 4.13: Highly similar attack graphs from two victims in CCDC-2018 showing that the graphs are identical, except for an additional attack attempt by 10.128.0.205 in the second graph.

4.4.3. USE CASE B: ATTACK INVESTIGATION VIA THE DASHBOARD

We populate the critical path exploration dashboard with the CPTC-2018 dataset. We use the intrusion alerts of teams 5 and 9 (resulting in 104,152 alerts) to compare the proposed dashboard against SAGE AGs. SAGE compresses the alerts into 63 AGs, while the unified dashboard allows to explore and reason about the discovered attacker strategies.

The recommender matrix shows that alerts associated to *data manipulation*, *arbitrary code execution*, and *information discovery* require the urgent attention of security analysts due to their prevalence and severity scores (see Figure 4.6). Information discovery is being highlighted because of the sheer frequency of raised alerts.

DATA EXFILTRATION ATTEMPTS

A security analyst can view the most common strategies used by attackers to exfiltrate data over http by using the *Data Exfiltration* and *HTTP* filters in the graph explorer. All the filtered pathways do *root privileged escalation* and *data manipulation* before exfiltration. By selecting one of the privilege escalation vertices, the alert signatures show, e.g., “*GPL EXPLOIT CodeRed v2 root.exe access*”, providing actionable intelligence that the CodeRed exploit was used to carry out this attack. Ensuring that the network is not vulnerable to CodeRed helps mitigate similar privilege escalation attacks in the future.

ATTACKS AFTER WORKING HOURS

An analyst may investigate if any anomalous activity occurred at specific times, such as after working hours or on holidays. The timeline viewer can be used to investigate what transpired on a victim host. For example, the timeline viewer shows that the victim 10.0.0.22 was targeted by the attacker 10.0.254.204 until 9:44:42 PM. The analyst can view the strategies employed during this time by redirecting to the graph explorer. The graph explorer illustrates that *arbitrary code execution* enabled *resource hijacking*, which further enabled *data manipulation* and *data exfiltration*. The analyst can use this information to determine which resources were compromised during these attacks.

4.5. PRACTICAL IMPLICATIONS FOR CYBER THREAT INTELLIGENCE

Cyber threat intelligence (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 facilitates attacker strategy analysis via advanced visualizations. The analysis presented in this chapter 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 show that the AGs indeed model the teams’ self-reported claims. As demonstrated in Section 4.4.2, SAGE is agnostic to network, host, and alert properties: With no

ground truth about any aspect of the dataset, SAGE produces succinct and interpretable AGs, capable of actionable insights.

In addition, the proposed critical path exploration dashboard consolidates the AGs produced by SAGE. The unified view provided by the graph explorer and timeline viewer helps analysts discover temporal relationships between different objectives. The complexity of the global graph is managed via several filters. We opt for two different views for investigating attack paths (graph explorer) and attacker actions (timeline viewer) to reduce the cognitive load on analysts. The recommender matrix highlights critically urgent attack stages. Based on the unique circumstances of a SOC, analysts can adjust the threshold for what is considered *urgent*. In addition, analysts can filter the urgency of critical events based on specific attacker hosts, victim hosts, and targeted services. Finally, since the dashboard is a web application, analysts can investigate cyber threats remotely, without having to run SAGE locally.

4.6. LIMITATIONS AND FUTURE WORK

The alert-driven attack graphs provide a comprehensive overview of attacker behavior dynamics, and the dashboard enhances their deployability for operational settings. We aim to conduct a deeper investigation into the concept of fingerprints, particularly assessing their potential to be translated into signatures and integrated back into an IDS in an active learning setup. We also want to investigate whether such signatures hold across different datasets/infrastructures.

It is important to note that SAGE currently displays *all* alerts, even if they are considered false positives. This is because SAGE operates in a completely unsupervised setting without access to ground truth. Further research is needed to understand how false positives manifest in the alert-driven AGs, and whether we can design a heuristic to automatically discard them.

Additionally, a more rigorous validation study is required to measure the correctness of the AGs, and to understand which design decisions enable the analysts to reach correct conclusions when interacting with the dashboard. For instance, one technical limitation that may mislead analysts pertains to concurrent actions – due to the nature of sequential modeling in the S-PDFA, actions that may have been triggered simultaneously are placed in a sequential order, giving the false perception that one action precedes the other. To avoid this confusion, it may be beneficial to explicitly state the start and end-time of each episode (vertex) in the AGs.

4.7. CONCLUSIONS

SAGE is utilized to generate alert-driven attack graphs (AG) for two open-source alert datasets. We show that SAGE generates insightful AGs even with no prior knowledge about an alert dataset. We demonstrate that alert-driven AGs can provide insights into past attacks and intelligence for future attacks. 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 show exactly how specific attacks transpired and reveal that attackers follow shorter paths to re-exploit objectives 84.5% of the time. We

discover 29 uniquely identifiable attack paths, composed of 15.8 episodes on average. We also 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.

Furthermore, the critical path exploration dashboard provides a consolidated view of the attacker strategies discovered by SAGE. The graph explorer and timeline viewer help discover global attack patterns, while the recommender matrix highlights the most critical events to analyze. The filtering capabilities in each visualization help analysts drill down to more targeted attacks and provide greater flexibility than SAGE did alone.

Acknowledgments. We thank Profs. Bill Stackpole and Daryl Johnson for their feedback on the attack graphs. This effort is partially supported by United States NSF Award 1742789 and RIT Global Cybersecurity Institute.

REFERENCES

- [1] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, “Alert-driven Attack Graph Generation using S-PDFA”, *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [2] A. Nadeem, S. L. Diaz, and S. Verwer, “Critical Path Exploration Dashboard for Alert-driven Attack Graphs”, *IEEE symposium on visualization for cyber security (VizSec)*, 2022.
- [3] W. U. Hassan, S. Guo, D. Li, *et al.*, “Nodoze: Combatting threat alert fatigue with automated provenance triage”, in *Network and Distributed System Security*, 2019.
- [4] A. Sopan, M. Berninger, M. Mulakaluri, and R. Katakam, “Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC”, in *IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2018, pp. 1–8.
- [5] L. Hao, C. G. Healey, and S. E. Hutchinson, “Flexible web visualization for alert-based network security analytics”, in *Proceedings of the Tenth Workshop on Visualization for Cyber Security*, 2013, pp. 33–40.
- [6] B. C. Cappers and J. J. van Wijk, “Understanding the context of network traffic alerts”, in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2016, pp. 1–8.
- [7] L. Franklin, M. Pirrung, L. Blaha, M. Dowling, and M. Feng, “Toward a visualization-supported workflow for cyber alert management using threat models and human-centered design”, in *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2017, pp. 1–8.
- [8] A. Okutan and S. J. Yang, “ASSERT: Attack synthesis and separation with entropy redistribution towards predictive cyber defense”, *Cybersecurity*, vol. 2, no. 1, pp. 1–18, 2019.
- [9] S. Moskal and S. J. Yang, “Heated Alert Triage (HeAT): Network-agnostic extraction of cyber attack campaigns”, in *Proceedings of the Conference on Applied Machine Learning for Information Security*, Hybrid, 2021.

- [10] T. van Ede, H. Aghakhani, N. Spahn, *et al.*, “DeepCASE: Semi-Supervised Contextual Analysis of Security Events”, in *IEEE Symposium on Security and Privacy*, May 2022.
- [11] A. Alsaheel, Y. Nan, S. Ma, *et al.*, “ATLAS: A Sequence-based Learning Approach for Attack Investigation”, in *Proc. of the USENIX Security Symposium*, 2021, pp. 3005–3022.
- [12] P. Holgado, V. A. Villagra, and L. Vazquez, “Real-time multistep attack prediction based on hidden markov models”, *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 134–147, 2017.
- [13] 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.
- [14] A. Nadeem, S. Verwer, and S. J. Yang, “SAGE: Intrusion Alert-driven Attack Graph Extractor”, in *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, 2021.
- [15] S. Moskal and S. J. Yang, “Framework to Describe Intentions of a Cyber Attack Action”, *arXiv preprint arXiv:2002.07838*, 2020.
- [16] *CPTC-2018 dataset*, Accessed: 08-Jul-2021, 2018. [Online]. Available: <https://mirror.rit.edu/cptc/>.
- [17] F. Hassanabad, *CCDC-2018 dataset*, Accessed: 08-Jul-2021, 2019. [Online]. Available: <https://github.com/FrankHassanabad/suricata-sample-data>.
- [18] *Suricata IDS*, Accessed: 08-Jul-2021, 2019. [Online]. Available: <https://suricata.readthedocs.io/en/suricata-6.0.3/>.
- [19] F. M. Alserhani, “Alert correlation and aggregation techniques for reduction of security alerts and detection of multistage attack”, *IJASCSE*, 2016.
- [20] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains”, *Leading Issues in Information Warfare & Security Research*, 2011.

II

EXPLAINABLE SEQUENTIAL LEARNING FOR NETWORK ATTACK ANALYSIS

5

BEYOND LABELS: AUTOMATED BEHAVIOR PROFILING OF MALWARE

The capability assessment of malware samples is typically a manual and time-intensive task. This chapter proposes a novel unsupervised sequence clustering tool-chain called MalPaCA, which automates capability assessment by clustering the observable behavior in malware's network traces. The behavioral profiles are generated based on the cluster membership of malware's network traces. A directed acyclic graph (DAG) shows the similarities and differences between malware behavioral profiles. The behavioral profiles together with the DAG provide a more meaningful characterization of malware than current family designations, which are known to be inconsistent and noisy.

We apply MalPaCA on a financial malware dataset collected in-the-wild that comprises of 1196 malware samples resulting in 3.6M packets. Our experiments show that (i) MalPaCA successfully identifies capabilities, such as port scans and the reuse of Command and Control servers by clustering only 20 packet headers; (ii) it uncovers multiple discrepancies between behavioral clusters and family labels, and (iii) it demonstrates the effectiveness of clustering temporal features by producing an error rate of 8.3%, compared to 57.5% obtained from statistical features.

This chapter is based on the paper “Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families” by **Nadeem, A.**, Hammerschmidt, C., Gañán, C. H., & Verwer, S. in *Malware Analysis Using Artificial Intelligence and Deep Learning 2021* (pp. 381-409), Springer [1]. Section 5.4 was added afterward.

5.1. INTRODUCTION

The first malware was discovered over thirty years ago. Yet, malware infections remain one of the leading threats in cybersecurity¹. AV-test, a security research institute, reported detecting over 1000M malware samples in 2019². Anti Virus (AV) companies play a pivotal role in identifying newly discovered malware samples by labeling them. The family labels for these malware samples are either taken from the malware code itself, or are assigned by a malware analyst. These family labels typically do not represent the capabilities of malware samples. Also, the overall process of assigning malware labels is often inconsistent, and multiple AV vendors can end up assigning different names to the same sample [2], [3]. This black-box (uninterpretable) nature of malware labeling makes it impossible to verify assigned family labels, causing the evaluation of newer detection methods to depend on unreliable ground truth [4].

In this chapter, we address the limited interpretability of malware family labels by augmenting them with their explainable behavioral profiles. To this aim, we propose MalPaCA (Malware Packet Sequence Clustering and Analysis) – an unsupervised sequence clustering tool-chain for automated capability assessment of malware samples.

We challenge the current research paradigm in two ways: First, the behavioral profiles are constructed using manual capability assessment³ [5]–[7], which causes them to become quickly outdated. We investigate the usage of unsupervised machine learning (ML) for automated capability assessment in order to address the growing volume of malware samples. Second, the behavior analysis of malware is heavily based on its static and system-level activity analysis. Network traffic is rarely used to analyze malware behavior because of the scarcity of ground truth and the non-stationarity of its data distribution [8]. As a result, malware samples that exhibit identical network behavior but have different code attributes end up in different families, see *e.g.*, Perdisci *et al.* [9]. Meanwhile, existing research suggests that network traffic shows malware’s core behavior by capturing direct interactions with the attacker or the command and control (C&C) server [10]. Network traffic analysis can also be performed remotely, which presents a lower overhead than many popular system-activity solutions. Thus, we place an emphasis on building network behavioral profiles.

Existing learning-based behavior analysis approaches typically construct a single model that either describes the whole network or describes each protocol usage individually [11]. However, the network traffic originating from even a single host can be so complex that these models fail to correctly represent malicious behaviors [12]. This is why MalPaCA splits the network traffic between hosts into *unidirectional connections* and considers them as discrete behaviors (or *capabilities*). MalPaCA clusters similar connections based on their temporal similarity, where each cluster represents a unique capability. A malware sample is then represented by its *behavioral profile* – an enumeration of the cluster membership of its connections. We represent these behavioral profiles in a directed acyclic graph (DAG) that shows the overlapping behaviors of different samples. The graph also shows malware samples from different families behaving identically, showing potentially inconsistent family labels.

¹<https://www.cybersecurity-insiders.com/top-15-cyber-threats-for-2019/>

²<https://www.av-test.org/en/statistics/malware/>

³The goal of capability assessment is to discover the behaviors a malware sample exhibits.

MalPaCA's novelty lies in its sequential features that keep the temporal nature of the traffic intact. It utilizes only 20 packets to characterize the network behavior of any given connection. It also utilizes only the packet header features that are available even when traffic is encrypted. MalPaCA utilizes a combination of dynamic time warping and ngrams to measure the distance between network connections for the clustering task. The clusters are visualized using *temporal heatmaps* that provide a data-driven approach to investigate the quality of MalPaCA's clusters by clearly showing multivariate sequences of network connections that are grouped together. The heatmaps also enable analysts to assign capability labels to clusters, without having to manually investigate thousands of network traces. Thus, the temporal heatmaps provide a visual means to understand MalPaCA's rationale for finding behavioral similarity. In doing so, we address the interpretability problem of typical black-box analysis methods.

We evaluate MalPaCA's performance on 1196 financial malware samples (resulting in 3.6M packets) coming from 15 families collected in-the-wild. We also compare the effectiveness of sequence clustering by comparing with an existing method based on frequently-used statistical (aggregate) features [13]. We show that (i) MalPaCA's capability assessment works on low quality datasets with as low as 20 packets in each connection, though longer/additional connections result in more thorough profiles; (ii) it successfully discovers several attacking capabilities, such as port scans and reuse of C&C servers; (iii) MalPaCA demonstrates the effectiveness of sequence clustering by producing an error rate of 8.3% compared to 57.5% obtained from statistical features, and (iv) MalPaCA uncovers multiple discrepancies between behavioral clusters and family labels. We believe this happens either because the labels are incorrect or because the overlapping samples share significant behavior. In summary, our contributions are as follows:

1. We present a comprehensive survey of machine learning use for malware analysis in the available literature.
2. We build MalPaCA – a tool-chain that automatically builds network behavioral profiles for malware samples collected in-the-wild.
3. We introduce temporal heatmaps as a data-driven and visualization-based cluster evaluation method for multivariate sequential features. We also demonstrate the utility of the directed acyclic graph in uncovering discrepancies between behavioral clusters and traditional family labels.
4. We show the superiority of clustering sequential features over statistical features.

THE PROBLEM WITH AV LABELS

This section presents an analysis of our experimental dataset (*i.e.*, 1196 malware samples) to emphasize the problem of inconsistent AV labels, and to motivate the need for explainable behavioral profiles. We compare the *agreement rate* of two popular malware labeling practices, *i.e.*, YARA rules⁴ and VirusTotal⁵. The malware collection process is given in Section 5.5. Table 5.2 shows the number of binaries in each malware family.

⁴YARA: <https://virustotal.github.io/yara/>

⁵VirusTotal: <https://www.virustotal.com/>

YARA labels	Dridex-Loader	-	-	-	-	-	-	-	-	-	-	3	12	
	Zeus-OpenSSL	-	-	-	-	-	-	-	-	-	-	1	1	
	DridexRAT	-	-	-	-	-	-	-	-	-	-	-	7	
	Dridex	-	-	-	-	-	-	-	-	-	-	3	3	
	Zeus-VM-AES	-	-	-	-	-	-	10	-	-	-	15	4	
	Zeus	-	-	-	-	-	-	-	-	-	-	3	-	
	Gozi-ISFB	-	-	-	14	6	-	-	-	45	-	37	20	
	Zeus-Panda	-	-	-	-	-	-	-	-	-	-	10	-	
	Ramnit	-	-	-	-	-	-	12	-	-	-	3	7	
	Zeus-v1	-	-	-	-	-	-	-	-	-	-	10	-	
	Zeus-Action	-	-	-	-	-	-	-	-	-	-	2	-	
	Blackmoon	77	-	700	-	-	31	-	41	-	-	11	11	16
	Gozi-EQ	-	-	-	-	-	-	-	-	-	-	-	7	-
	Zeus-P2P	-	-	-	-	-	-	-	-	-	-	-	4	-
	Citadel	-	1	-	-	-	-	-	-	-	26	-	12	31
			banbra	citadel	dinwod	gamarue	gozi	qzonit	ramnit	razy	urshif	zbot	zusy	OTHERS
		AVClass labels												

Figure 5.1: Label agreement rate among AV vendors for malware binaries. Rows show YARA labels, and columns show AVClass labels.

The malware binaries in the dataset are labeled using YARA rules. Each malware binary also has a VirusTotal (VT) scan report. On average, there are 61 AV vendors for each malware sample, out of which 25.8% vendors per malware sample return a null detection, *i.e.*, unable to detect it as malicious. The rest assign various labels to each malware binary.

Since each AV vendor has its own vocabulary, a trivial filtering attempt on a VT report cannot identify the true underlying family label. Sebastian *et al.* [3] have developed an open-source tool, called AVClass, that takes VT reports as input and returns the most likely family label. If, after all the filtering steps, AVClass is unable to identify the family name, it declares the malware as a “SINGLETON”. We use AVClass to reduce a VT report into its representative VT family label. In the experimental dataset, AVClass returns “SINGLETON” for 101/1196 (8.4%) VT reports, while assigning 42 unique family labels to the rest 1095 malware binaries.

Figure 5.1 shows the agreement rate between the YARA and VT labels. The y-axis shows the YARA labels. The x-axis shows the VT labels as aggregated by AVClass. For brevity, “OTHERS” category contains all samples for which *counts* < 10. Only 3 family names co-exist in both YARA and VT labels, *i.e.*, Citadel, Gozi, and Ramnit. Also, although Ramnit is detected under the same name by both YARA and VT, 10 malware samples are still labeled differently. In fact, YARA family labels are assigned 4.2 distinct VT labels on average, while VT labels are assigned 1.5 distinct YARA labels on average. One example demonstrating this is: YARA: Zeus-VM-AES (29 samples) are predicted as

VT: razy (10 samples), gamarue (6 samples), cerber (3 samples), upatre (3 samples), farfli (1 samples), locky (1 samples), hpcerber (1 samples), and SINGLETON (4 samples). This makes it very hard to understand the collected malware. One fair conclusion is that some VT labels can be considered as sub-families of the popular YARA malware family. For example, Dinwod and Banbra seem to be sub-families of Blackmoon, but the names alone do not explain which attributes set them apart from each other.

5.2. SURVEY OF ML-BASED MALWARE DEFENSE

The field of malware analysis has existed since the first malware was discovered over thirty years ago. Since then, multiple machine learning based approaches have been proposed to automate malware detection and analysis. In this section, we present a brief survey of the major research challenges targeted by prior work. In doing so, we highlight how our work fills the gaps across various research themes.

5.2.1. CHALLENGES IN MALWARE LABELING

Existing research has repeatedly shown that malware family labels are noisy and inconsistent. Popular tools, such as VirusTotal, run multiple AV scanners and return an array of labels predicted by each scanner, without any indication as to which is correct. There is also an absence of a common vocabulary that all security companies can follow to label malware samples. Maggi *et al.* [14] propose a method to find inconsistencies in malware family labels generated by Anti Virus (AV) scanners. Mohaisen *et al.* [15] are the first to measure the accuracy, consistency, and completeness of AV scanners. Their results show that AV vendors produce inconsistent labels 50% of the time, on average. These findings resulted in research that found ways to deal with the inconsistencies in the family labels. Kantchelian *et al.* [2] proposed an algorithm based on expectation maximization and Bayesian models that assigns weights to each vendor's trustworthiness. Sebastián *et al.* [3] developed a useful open-source tool, called AVClass, that determines the likely family name after performing heavy filtering on all the predicted labels. However, these methods do not address the key underlying issue – malware family labels are black-box and have a limited link with malware capabilities.

Behavioral profiles complement family names in that they also describe the behavior of a sample. *Capability assessment* is done to characterize a malware sample, which has primarily been a manual effort resulting in behavioral profiles that are quickly outdated. Also, most of the prior works in capability assessment utilize information extracted from the static analysis of malware executables: Black *et al.* [5] bridge the semantic gap between low-level API calls and high-level behaviors in order to build a taxonomy of banking malware. They extract API calls by statically analyzing a banking malware dataset, and map them to high-level behaviors manually with the help of domain experts. Sharma *et al.* [7] recently proposed a method to automatically build behavioral profiles. They select a few high-level capabilities possessed by malware by investigating the literature, and map them to low-level behaviors extracted from the static analysis of 56 malware samples. *In contrast, we propose MalPaCA that automatically builds dynamic (network) behavioral profiles.*

5.2.2. RESEARCH OBJECTIVES: DETECTION VS. ANALYSIS

Existing research on malware comes in two strains: detection-based and analysis-based. Malware detection and signature generation dominates existing literature, with the end-goal of optimizing metrics [9], [13], [16]–[26], while only a few of these studies provide qualitative analysis of the obtained results [12], [27]. Recently, however, several malware analysis approaches have been proposed that aim to improve malware understandability rather than optimizing detection rates. Black *et al.* [5] perform an in-depth analysis of the key behaviors of banking malware families and how they have evolved over time. Moubarak *et al.* [6] discuss malware evolution and the structural relationship between several potentially state-sponsored malware. In [28], the authors cluster android malware samples, and build a dendrogram of the malware families showing overlapping code snippets. Sharma *et al.* [7] build behavioral profiles for malware samples using static analysis. *In this chapter, we also build an analysis method, MalPaCA, that groups similar network connections.*

Although clustering is an unsupervised technique, existing literature often uses some notion of ground truth (family labels) to evaluate cluster quality. Bayer *et al.* [22] evaluate their malware clustering approach using labels obtained by the majority voting of 6 AV vendors. Perdisci *et al.* [9] evaluate their malware clustering approach by introducing a notion of AV graphs that depict the agreement between AV vendors as a measure of cluster cohesion and separation. In [17], the authors report over 95% precision and recall for their malware clustering approach. They use the majority voted family labels from 25 AV vendors as their ground truth. Li *et al.* [4] have advised caution against deciphering highly accurate clustering results as they may be impacted by spatial bias: Relying on majority voting with AV-provided labels is hazardous since widespread agreement among AV vendors indicates that the families are already easily identifiable. *In this chapter, we propose a data-driven cluster evaluation method without using family labels. Instead of optimizing clustering accuracy, our emphasis is on explainability of the results.*

5.2.3. CHALLENGES IN MALWARE BEHAVIOR MODELING

Modeling malware behavior is challenging since malware authors specifically try to evade detection [29]. Static analysis of malware binaries and disassembled code has been a popular malware analysis approach in the literature [5], [17], [21], [30], [31]. Increasingly more malware uses obfuscation techniques to evade analysis, causing difficulties for statically analyzing malware. The obfuscation attempts gave rise to dynamic analysis of malware that executes a malware sample in a sandbox and collects execution traces from it. Dynamic analysis is generally divided in two strains: system activity and network traffic analysis. Network traffic analysis collects traces of malware samples remotely using existing network monitoring infrastructures [9], making it much easier to apply. However, the behavior analysis and signature generation literature is heavily focused on system activity analysis, *e.g.*, see [7], [22], [32], [33]. Research suggests that network traffic shows the core behavior of malware [10]. Although sometimes encrypted, network traffic contains the direct interaction with the attacker. In this section, we discuss three major challenges of modeling malware behavior via traffic analysis.

1. **Feature selection.** Network traffic analysis is typically utilized for network intrusion detection systems (NIDS) that either detect anomalous traffic [23] or generate

signatures for malware families [34]–[36]. Deep packet inspection (DPI) is a common approach to extract information from packet payloads. For example, Rafique *et al.* [26] use DPI for automated signature generation of malware families. Although effective, DPI-based approaches are privacy-intrusive, operationally expensive, and do not work out-of-the-box for encrypted traffic. There are also approaches that use specific headers to detect attacks. For example, HTTP-based malware can be detected using specific features from the application header [9]. Similar approaches exist for DNS-based malware [37], [38], and HTTPs-based malware [39]. Several works use *coarse* or high-level features that are protocol-agnostic and work out-of-the-box even with encrypted traffic. For example, Conti *et al.* [19] use sequences of packet sizes to characterize the network behaviors generated by android applications. Aioli *et al.* [20] use various statistical features computed over packet sizes to detect bitcoin wallet application functionality. Acar *et al.* [18] use network traffic direction and packet lengths to identify commands issued to smart home IoT devices. These works aim to characterize benign network behaviors. In the malware domain, Tegeler *et al.* [13] use average packet size, average packet inter arrival time, average connection duration, and the fast fourier transform (FFT) of C&C communication to detect bot infected hosts. Garcia [12] builds a behavioral intrusion detection system by using the size, duration, and periodicity of Netflows. *In this chapter, we also use high-level features from packet headers to characterize malware's network behavior.*

- 2. Feature representation.** Machine learning methods take a feature vector as input, which can represent anything ranging from a single behavior to a complete malware sample. Multiple observations for a single feature are aggregated into *statistical features*, *e.g.*, average bytes in a Netflow. Existing literature is filled with approaches that use such statistical features, *e.g.*, see [12], [13], [16], [40]. Although they are computationally efficient, they lose local behavioral details, which can be a problem when the goal is to characterize that behavior. Another approach that is gaining momentum is the use of *sequential features*. Numeric sequential features are typically used in two ways: *discretized* and *raw sequences*. A raw sequence (or a continuous time-series) is composed of the original observations, while a discretized sequence encodes the observations into a finite set of bins. Discretizing sequences is typically faster and makes measuring distances easier. Pellegrino *et al.* [27] learn state machines from discretized Netflow data in order to detect bot-infected traffic, while Hammerschmidt *et al.* [41] use it to cluster host behavior over time. Lin *et al.* [24] detect anomalies in industrial water treatment plant by using discretized sequences from sensor readings. In practice, malware-related data is often scarce and noisy. In this case, discretization can lose important information. Raw sequences are rarely used for modeling network traffic because it is non-stationary and contains noise (*e.g.*, empty acknowledgment packets or retransmissions), and delays (due to varying network latency) [8]. Ntlangu *et al.* [42] provide a brief overview of time-series approaches to model network traffic. As noted in [42], due to the nature of network traffic and their distributions, (auto-)regressive models struggle to accurately capture them. Kim *et al.* [43] use a multivariate time-series regression model on host-based resource consumption, such

as CPU and memory usage to identify android malware. The closest approach to ours is proposed by Conti *et al.* [19] who use raw sequential features to characterize user actions on android applications. *MalPaCA*, however, uses significantly shorter sequences to characterize malware network behavior.

- 3. Distance measure.** The notion of behavioral similarity necessitates the measurement of distance between objects. The choice of the distance measure is directly dependent on the data type of the feature set (*e.g.*, numeric or categorical) and the way the features are represented (*e.g.*, statistical or sequential). For statistical features, Euclidean distance is most commonly used. For instance, Chan *et al.* [33] use Euclidean distance to determine similar android processes. Calculating the distance between sequential features is more challenging because they may not always be properly aligned. For categorical (or discretized) sequences, there exist bioinformatics-inspired solutions using sequence alignment [44]. They require pre-computed substitution matrices, which currently do not exist for malware. There also exist string matching solutions typically used in natural language processing. Baysa *et al.* [45] use Levenshtein, or edit distance, to measure the similarity between two malware binary files. A sliding window can also be utilized to obtain a vector encoding of a sequence, called ngrams, which have been used to model genomic sequences [46] and to match files [47]. They have also been used to classify malware families in [48]. Longest common subsequence (LCS) with k-gaps can also be used to measure distances between sequences. The gaps account for the occasional noise. Chan *et al.* [33] use LCS to group similar resource access patterns in android applications. A few distance measures exist for continuous sequences. Verwer *et al.* [49] have used Kullback-Leibler divergence to measure the distance between two probability distributions of sequences for learning probabilistic automata. However, it requires substantial amount of data to measure the similarity with a high confidence, which is not always available for malware. Another promising distance measure is dynamic time warping (DTW). DTW has been used in fingerprint verification [50], characterizing DDoS attacks [51], and measuring similarity in android application behavior [19]. *MalPaCA* uses a combination of DTW and ngrams to measure the distance between network connections.

5.3. MALWARE PACKET SEQUENCE CLUSTERING AND ANALYSIS – MALPACA

MalPaCA (Malware Packet Sequence Clustering and Analysis) aims to construct a behavioral profile for each malware sample that explains its observable capabilities. To this end, MalPaCA first clusters the various observed network behaviors, and then represents malware samples using their behavioral information. The profiles are built using observed behavior since only the executed functionality is relevant for behavior profiling. MalPaCA does not assume any a priori knowledge about the malware's family name or its capabilities, and hence can be used out-of-the-box for other malware datasets. The profiles for individual samples can be enriched further by observing additional traffic.

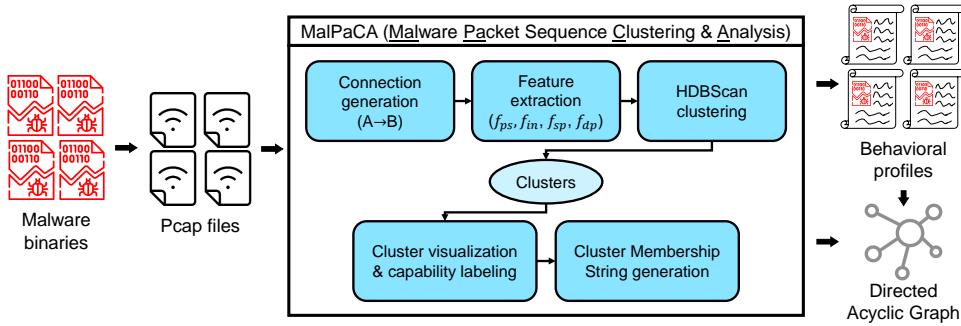


Figure 5.2: MalPaCA workflow: The network behavior of malware binaries is stored in Pcap files. Network connections are clustered based on behavioral similarity, and malware binaries are described using the cluster membership of their network connections.

We release MalPaCA as open-source⁶.

Figure 5.2 illustrates the architecture of MalPaCA. Network traces (Pcap files) are given as input to the system, which are split into unidirectional packet streams (or *connections*) that are clustered based on temporal similarity. Each cluster is assigned a capability label by visualizing temporal heatmaps that show multivariate feature values of the connections. Each malware sample (and its associated Pcap file) is then described by a *cluster membership string* (CMS), forming a descriptive behavioral profile. The CMS for all Pcaps are represented in a directed acyclic graph (DAG), which shows the behavioral relationships between different malware samples. The temporal heatmaps together with the DAG are intended for human-in-the-loop exploration – they actively support malware behavior analysis and provide more insightful characterization of malware than current family labels.

Connection generation. A *connection* is defined as an uninterrupted unidirectional list of packets sent from source IP to destination IP. We characterize connections as either *outgoing* and *incoming* based on their direction with respect to the localhost.

Ideally, a connection captures one complete capability. The connection length can vary significantly depending upon the behavior and network delays. Since the network delay is an artifact of the network, not of the malware behavior, it is important to reduce its impact when measuring behavioral similarity. MalPaCA does so by capping the sequence length to a fixed threshold. Existing research suggests that it is possible to identify behavioral differences from a *handshake*⁷. Wang *et al.* [52] use the first 3 to 12 bytes of packet headers in order to identify the different so-called protocol format messages. MalPaCA builds upon this idea and utilizes the first few packets of a connection to identify the capability. This threshold is a tunable parameter, *len*. It should be large enough to allow the handshake to be modeled, but not so large that the sequence includes noise artifacts and increases MalPaCA’s computational costs.

⁶MalPaCA: <https://github.com/tudelft-cda-lab/malpacapub>

⁷Handshake traffic refers to the introductory packets of a connection.

Feature extraction. The choice of features is crucial for determining the kind of behaviors that are identified by MalPaCA. Two considerations motivate our choice: 1) MalPaCA should be generalizable to more than one type of malware, and 2) the feature set is small and easy to extract. Hence, we do not use features extracted from the packet payload itself as they limit the applicability of the method. We also do not use IP addresses as they are easy to spoof and are considered personally identifiable information⁸ in countries like the Netherlands. Furthermore, the use of IP addresses also limits the discovery of unintuitive behavioral similarities.

We use four sequential features: (i) packet size, (ii) time interval, (iii) source port, and (iv) destination port. All four features are independent of the protocol type, making them available for every packet. Although these features are simplistic, we demonstrate that their sequential nature captures malware behavior effectively. *Packet size* (f_{ps}) measures the size of the *IP datagram* of each packet in bytes. *Time interval* (f_{in}) captures the inter packet arrival time in milliseconds⁹. We use time interval because malware tends to show a periodic behavior, *e.g.*, bots send periodic heartbeat packets¹⁰ to inform the C&C server about the infected host. We use both *source* (f_{sp}) and *destination* (f_{dp}) *port numbers* because the connections are unidirectional. Each connection F is represented by four sequences, one per feature, $F = (f_{ps}, f_{in}, f_{sp}, f_{dp})$.

HDBScan clustering with sequential features. A key strength of MalPaCA is its clustering algorithm. There exists a familial structure among malware behaviors [28], [36]. Therefore, it makes sense to use hierarchical clustering to model the relationships between them. To this aim, we have used hierarchical density-based spatial clustering of applications with noise (HDBScan) [53]. HDBScan is powerful because it automatically determines the optimal number of clusters, and generates high-quality clusters that remain stable over time. It does not force data points to become part of clusters – all data points whose membership to a cluster cannot be determined are considered to be *noise*. In our context, *noise* refers to behaviors that are either too different or are an amalgamation of different behaviors. An ideal dataset with clear cluster boundaries has no noise. In the presence of a less ideal dataset, noise is discarded to extract high-quality clusters. This is based on a minimal set of tunable parameters: the parameter *min_cluster_size* specifies the minimum cluster size required, and *min_samples* specifies how conservative the clusters should be. Keep in mind that discarding excessive connections can also be counterproductive, as discussed in Section 5.7.

We provide a pre-computed pairwise distance matrix to HDBScan for the clustering task. This is computed using a combination of dynamic time warping (DTW) and ngrams to cater for numeric and categorical feature sequences. DTW is particularly used to produce clustering results that are resilient to delays and noise, which are common characteristics of network traces.

Dynamic time warping (DTW) [54] is used to measure distances between numeric time-series (packet size and time interval). It minimizes the distance between two se-

⁸<https://www.enterprisetimes.co.uk/2016/10/20/ecj-rules-ip-address-is-pii/>

⁹Note that MalPaCA is meant to be used on a single network since using inter-arrival time makes connections collected on different latency networks incomparable.

¹⁰<https://www.ixiacom.com/company/blog/mirai-botnet-things>

quences by aligning similar local substructures of one sequence to those of the other sequence. Given a set of sequences S , the DTW distance $d_{dtw}(a, b)$ for any two sequences $a = [a_0, \dots, a_n]$, $b = [b_0, \dots, b_m] \in S$ is given by Eq. 5.1. $d_{dtw}(a, b)$ is a dissimilarity score, which is normalized using Eq. 5.2.

$$d_{dtw}(a, b) = \sum_{i=1}^{n+1} \sum_{j=1}^{m+1} \|a_i - b_j\| + \min \begin{cases} d(a_{i-1}, b_j), \\ d(a_i, b_{j-1}), \\ d(a_{i-1}, b_{j-1}) \end{cases} \quad (5.1)$$

$$d_{ndtw}(a, b) = \frac{d_{dtw}(a, b) - \arg\min_{x, y \in S \text{ \& } x \neq y} d_{dtw}(x, y)}{\arg\max_{x, y \in S \text{ \& } x \neq y} d_{dtw}(x, y) - \arg\min_{x, y \in S \text{ \& } x \neq y} d_{dtw}(x, y)} \quad (5.2)$$

Ngrams with *cosine similarity* are used to measure the distance between categorical sequences (source and destination ports). An *ngram profile* is a vector encoding of a sequence that is derived by running a sliding window of length n along it, and counting the frequency of each ngram. The larger the value of n , the more sequence structure is captured. An example for bigrams, *i.e.*, $n = 2$ is shown in Table 5.1, where A, B, C, D are hypothetical port numbers. Let G be the set of all unique bigrams occurring in the dataset. For each sequence a , we compute the vector $a_g = [Count(a, g_1), \dots, Count(a, g_{|G|})]$ that contains the occurrence frequencies $Count(a, g_i)$ of each ngram $g_i \in G$. The distance between the vectors is computed using cosine similarity, which has shown promise in measuring similarity between categorical sequences [55]. It is defined by the angle between two non-zero vectors. The similarity value lies between $[0, 1]$, where 1 means the vectors are identical (parallel to each other) and 0 means they are totally different (orthogonal to each other). For two sequences in their vector representations $a = [v_1, \dots, v_{|G|}]$, $b = [v'_1, \dots, v'_{|G|}]$, the cosine distance $d_{cos}(a, b)$ is computed using Eq. 5.3.

Table 5.1: Example: Distance measurement using ngram analysis and cosine similarity.

Input	Ngram profiles	G=[AB,BC,CB,DA,CA]	Cosine distance $d_{cos}(a, b)$
$a = ABCBC$	AB, BC, CB, BC	$a_g = [1, 2, 1, 0, 0]$	0.3876
$b = DABCA$	DA, AB, BC, CA	$b_g = [1, 1, 0, 1, 1]$	

$$d_{cos}(a, b) = 1 - \frac{\sum_{i=1}^{|G|} a_i \times b_i}{\sqrt{\sum_{i=1}^{|G|} a_i^2} \times \sqrt{\sum_{i=1}^{|G|} b_i^2}} \quad (5.3)$$

Finally, the DTW and cosine distances are combined with equal weights to compute the pairwise distance matrix for all connections using Eq. 5.4.

$$d_{conn}(A, B) = \frac{d_{ndtw}(a_{ps}, b_{ps}) + d_{ndtw}(a_{in}, b_{in}) + d_{cos}(a_{sp}, b_{sp}) + d_{cos}(a_{dp}, b_{dp})}{4} \quad (5.4)$$

where $A = (a_{ps}, a_{in}, a_{sp}, a_{dp})$ and $B = (b_{ps}, b_{in}, b_{sp}, b_{dp})$ are connections and their features: packet sizes $\{a|b\}_{ps}$, intervals $\{a|b\}_{in}$, source port ngram profiles $\{a|b\}_{ps}$, and destination port ngram profiles $\{a|b\}_{dp}$.

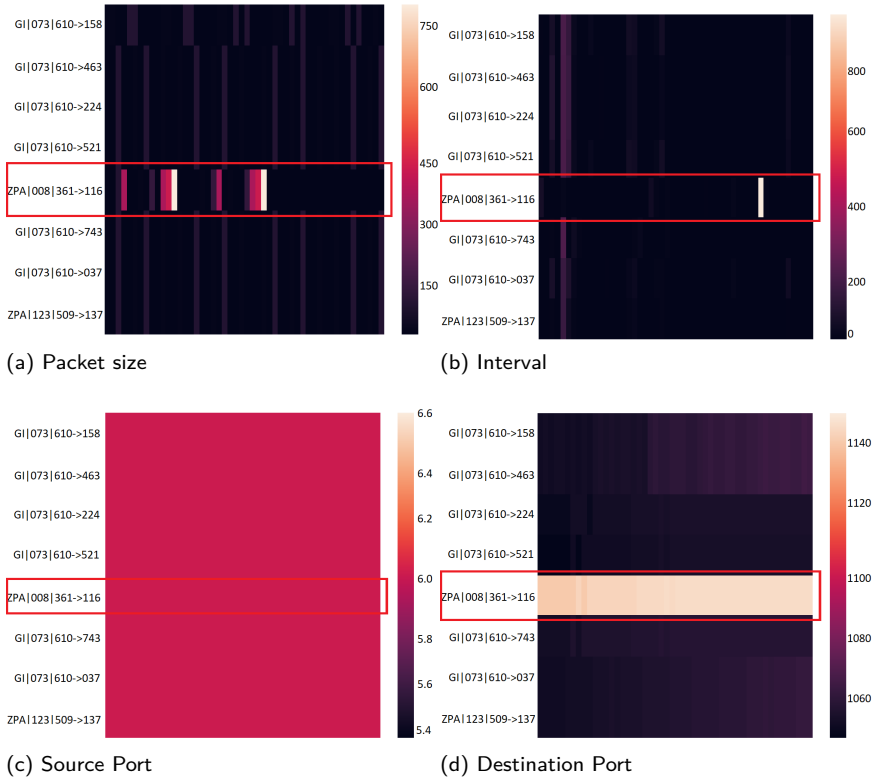


Figure 5.3: Temporal heatmaps for investigating the cluster content of multivariate sequential features. The highlighted connection is a clustering error, *i.e.*, it is much too different from other connections in this cluster.

Cluster visualization via temporal heatmaps. We use *temporal heatmaps* as post-hoc explanations for the obtained clusters. Four temporal heatmaps are associated to each cluster, one corresponding to each feature. Each row in a heatmap shows the corresponding feature sequence of the first len packets in a connection. Figure 5.3 shows the temporal heatmaps for a cluster containing eight connections, where one is highlighted due to its dissimilarity. Analysts can inspect the heatmaps to determine the cluster label, *i.e.*, which behavior is captured in a cluster. In addition to the insights provided by the heatmaps, the analysts can also investigate open-source intelligence (OSINT) regarding the involved IP addresses. The cluster labels then serve as an inventory of the observed capabilities of malware samples. Note that MalPaCA's goal is to identify different behaviors in the network traffic, and it does so regardless of their maliciousness and origin. Hence, the resulting clusters contain both, benign and malicious behaviors. The common clusters can be discarded if they contain known-benign behaviors, drastically reducing the number of connections to analyze.

Cluster validation and robustness. Formalizing cluster quality without ground truth is a fundamental challenge in clustering. Although some metrics exist that capture cluster quality (*i.e.*, Silhouette index [56] and DB index [57]), they require a notion of distance from a cluster centroid, which is difficult to obtain for sequences. In MalPaCA, each connection is represented by four sequences and collapsing these into a single cluster quality measure loses important local behavior. As a result, we observe that the clustering which was optimal according to standard clustering metrics (*e.g.*, DB index) does not match the clustering that followed our intuition. Instead, we define the following properties to be indicative of good clustering:

- Cluster homogeneity is high – a cluster contains only similar connections.
- Cluster separation is high – each cluster captures a unique capability.
- Clusters are small and specific so they only capture the core capability.

The first two properties ensure that we obtain meaningful capability-based clusters, the third ensures that only the core capabilities are captured. Visualizing the cluster content can help to identify the connections that belong in a cluster. To this end, we utilize temporal heatmaps and rely on human visualization to determine cluster quality.

A *clustering error* (CE) is defined as a connection that is placed in cluster X despite 50% of its features being different from other connections in the cluster. Since each feature holds equal weight, we only consider a connection as CE if more than two features differ. We consider two features *different* if more than 50% of their sequences differ so significantly that a different color appears on the temporal heatmap. This is where human visualization skills play a key role in determining feature similarity. Figure 5.3 shows a cluster containing one CE, highlighted in red. It shows that three out of four feature values of this connection are different from other connections. The clustering error rate is calculated as $\frac{\# \text{CE}}{\text{cluster size}}$, *i.e.*, $\frac{1}{8}$. We measure the error rate of each cluster, and calculate the *average percentage of errors per cluster* as a notion of clustering quality.

In practice, we first establish the *common majority* by finding two or more connections that are the most similar to one another, *i.e.*, the ones that have the least mutual distance. The pre-computed pairwise distance matrix is used as a lookup table for finding such connections. Figure 5.3 shows a simple case where the *rightful owners of a cluster* are easily visible since seven out of eight connections are very similar. The rest of the connections are compared with the rightful owners and are either considered as true positives or clustering errors, depending on how many feature sequences differ.

In terms of robustness, a common assumption is that malware can easily evade detection by adding random delays and padding to packets. However, there is a limit to what an attacker can change. For example, a TCP handshake needs to happen in a certain way because this is how the protocol dictates it. Also, padding-related provisions are already standardized by some commonly used protocols, such as TLS making it difficult to hide “coarse” features like packet sizes and inter-arrival times [58]. We expect that MalPaCA is evasion resilient, *e.g.*, since MalPaCA only uses coarse features, evading it is not a trivial task. Moreover, the usage of dynamic time warping distance makes it resilient to random delays [59] and due to the relative distance measures used in HDBScan, randomized port numbers are already clustered together, as shown in Section 5.6.1. If,

after all this, attackers still manage to evade MalPaCA, the malware sample will end up with a new behavioral profile, making analysts more prone to analyze it. Nevertheless, more study is needed to strengthen these claims.

Directed acyclic graph generation. Once the clustering is complete, the cluster membership of a malware sample's connections can be used to compute its *cluster membership string (CMS)*. The cluster membership string can be regarded as the behavioral profile for the malware. Precisely, for a malware sample x , $CMS_x = b^n$, where $b \in \{0, 1\}$, n is the number of behavioral clusters, and b^i indicates whether x 's connections are present in the i^{th} cluster. In this work, we consider binary CMSs because we are only interested in the behavior overlap of different malware samples. Non-binary $CMS_x = z^n$, for connection counts $z \in \mathbb{Z}$, is an interesting avenue to investigate. We model the behavioral relationships between the malware samples by considering the cluster membership strings as a *set membership* problem. It dictates that, e.g., Set $A = \{0, 1, 1\}$ is a *subset* of Set $B = \{1, 1, 1\}$ because Set B encapsulates all of Set A 's behaviors and more. Similarly, Set $C = \{0, 0, 0\}$ is a subset of every other set in this domain. Set C represents a malware sample for which all of its connections were discarded as noise. We represent these relationships between malware samples using a directed acyclic graph (DAG). Each vertex in the DAG represents a unique CMS. Different malware samples can share a single CMS *iff* their behaviors overlap completely. The vertices with the minimum hamming distance are connected using edges. This method allows multiple parents, *i.e.*, a CMS of "111" may be reached by both "110" and "101". Note that this graph is constructed purely from a data-driven approach without using any knowledge of family labels. In combination with human intelligence, we believe that it can serve as a powerful tool in understanding malware's network behavior.

5

5.4. EXPLAINABILITY ANALYSIS OF MALPaCA

MalPaCA was designed for explainability – it enables malware analysts to understand the capabilities of malware by investigating the cluster contents (via temporal heatmaps), and visualizing the similarities and differences between malware samples (via the directed acyclic graph). In doing so, MalPaCA enables the creation of new knowledge, which would be difficult to obtain using manual approaches [60].

MalPaCA is (partially) *transparent* and *explainable*. Transparency has two components: *design* and *algorithmic transparency*. A machine learning tool-chain is considered *design transparent* if the design decisions can be motivated from the application domain. A tool-chain is considered *algorithmically transparent* if the solution is *deterministic*, and if the tool-chain allows a user to reverse the learning pipeline to obtain the input data that led to modeling decisions. A tool-chain is considered *interpretable* if the model can be understood by a human on its own. If the model needs to be simplified or requires additional (post-hoc) explanations, then the tool-chain is considered *explainable*. These notions are described by Roscher *et al.* [61].

MalPaCA is *algorithmically transparent* and *partially design transparent*. The HDB-Scan algorithm is considered deterministic (for a fixed seed). The clusters together with the temporal heatmaps make it possible to reverse the input network connections from

the model, making it algorithmically transparent. Moreover, MalPaCA's design decisions are motivated by domain knowledge, making it design transparent – feature selection (coarse features for privacy reasons), feature representation (sequences for modeling temporal behavior), distance measure (DTW and ngram for robustness), and clustering algorithm (HDBScan for modeling the familial relationship between various behaviors). The parameter settings of the HDBScan algorithm may be considered partially design transparent since they were chosen heuristically on a configuration dataset.

MalPaCA is *partially interpretable*. The input feature processing can be explained by a human. However, the HDBScan clustering algorithm is not interpretable. Not only is the algorithm non-linear, the density computations and the agglomerative aspects of the algorithm make it difficult to understand the clustering decisions.

MalPaCA produces *local and global post-hoc explanations*. The temporal heatmaps explain why specific connections are clustered together, and can be considered as *batched local explanations*. The directed acyclic graph utilizes the clustering results and the input data to explain the *global relationships* among malware samples. Note that it may be possible to derive a counterfactual explanation from the temporal heatmaps for why a connection A was placed in cluster X instead of cluster Y based on their visual appearance. However, it is not so straightforward since the range of feature values in each heatmap is local, *i.e.*, the color bars are set on local ranges. In contrast, if the ranges were set based on global minimum and maximum, the heatmaps would lose the subtle intricacies in feature changes locally (within a single cluster). Further research is warranted in this direction.

5.5. DATASET AND EXPERIMENTAL SETUP

In this section, we describe the dataset used for the experiments and the configuration details of MalPaCA's parameters.

Dataset. MalPaCA was evaluated on financial malware samples collected in-the-wild. We collaborated with a security company that provided 1196 malware samples collected in 2018. These 1196 malware samples belong to 15 famous financial malware families, labeled using the company's proprietary YARA rules. Table 5.2 summarizes the dataset. Each malware sample was executed in a sandboxed environment containing several virtual machines. The resulting network traffic was stored in a Pcap file. This resulted in a total of 1196 Pcap files. Unidirectional connections based on IP addresses were extracted, resulting in a total of 8997 connections containing 3.6M packets.

Parameters. MalPaCA has four parameters, *i.e.*, the size of the n grams used for port numbers, len of packet sequences for features, and the two parameters of HDBScan clustering algorithm. In our experiments, we have used trigrams ($n = 3$) for port numbers because they form a good trade-off between performance and data sparsity [62]. The HDBScan algorithm uses $min_cluster_size = 7$ and $min_samples = 7$. The specificity of the identified behaviors is highly dependent on the length of sequences, *i.e.*, len ¹¹. In the experimental dataset, the length of connections is highly skewed towards

¹¹ len can be adjusted based on the required behavior specificity.

Table 5.2: Summary of experimental dataset: Malware binaries and their associated YARA family labels.

Family name (YARA)	# Malware binaries
Blackmoon (B)	887 (74.10%)
Gozi ISFB (GI)	122 (10.19%)
Citadel (C)	70 (5.85%)
Zeus VM AES (ZVA)	29 (2.42%)
Ramnit (R)	22 (1.83%)
Dridex Loader (DL)	15 (1.25%)
Zeus v1 (Zv1)	10 (0.83%)
Zeus Panda (ZPa)	10 (0.83%)
Gozi EQ (GE)	7 (0.58%)
Dridex RAT Fake Pin	7 (0.58%)
Dridex (D)	6 (0.50%)
Zeus P2P (ZP)	4 (0.33%)
Zeus (Z)	3 (0.25%)
Zeus OpenSSL	2 (0.17%)
Zeus Action	2 (0.16%)
Total	1196 (100%)

shorter sequences, with a mean of 20 packets. Based on preliminary experiments with $len = \{5, 10, 20, 50\}$, we found that $len = 20$ provided the optimal trade-off between behavior characterization and data loss. For smaller values, the connections were too generic. For larger values, connections with slight behavioral differences were considered very different. For example, at $len = 50$ several clusters capture slightly different variations of port scans, while at $len = 20$ those variations merge to form a few strong clusters. 733 out of 8997 connections were longer than $len = 20$, belonging to 12 malware families. The parameters were selected by tuning MalPaCA on a configuration dataset (5% of the usable data). The experiments were run on Intel Xeon E3-12xx v2 processor, 8 cores and 64GB RAM.

5.6. RESULTS AND DISCUSSION

5.6.1. MALWARE CAPABILITY ASSESSMENT VIA *Behavioral Profiles*

MalPaCA produces 18 clusters from the dataset. There are, on average, 25 connections in each cluster. The algorithm discards 284 connections as noise. The remaining 449 connections originate from 216 Pcap files. We successfully assigned labels to 12 clusters based on their temporal heatmaps. Table 5.3 shows an inventory of the observed network behaviors in the malware dataset, and enumerates the cluster attributes. In terms of cluster validation, e.g., for connection spam (c6), the whole cluster is filled with almost identical connections originating from the same host. We validate this observation by investigating the network traffic of these connections. We also left six clusters unlabeled since we could not identify the captured capability simply by exploring their temporal heatmaps. These particular clusters were also the source of clustering errors. The temporal heatmaps show that on average, 8.3% connections per cluster are CEs – their feature sequences are different from the other connections in the cluster.

Table 5.3: The connections and families in each cluster, including its capability label, and traffic direction.

Cluster	# Conns	# Families	Behavior	Direction
c1	39	9	SSDP traffic (<i>Common</i>)	Out
c2	90	9	Broadcast traffic (<i>Common</i>)	Out
c3	9	4	LLMNR traffic	Out
c4	49	5	Systematic port scan	In
c5	56	5	Randomized port scan	Out
c6	25	1	Connection spam (<i>Rare</i>)	In
c7	23	1	Connection spam (<i>Rare</i>)	Out
c8	16	1	Malicious subnet (<i>Rare</i>)	Out
c9	11	1	Connection spam (<i>Rare</i>)	Out
c10	9	2	HTTPs traffic	Out
c11	8	2	C&C Reuse	In
c12	18	4	HTTPs traffic	In
c13	25	5	Misc.	In
c14	10	3	Misc.	In
c15	20	3	Misc.	In
c16	12	3	Misc.	Out
c17	19	3	Misc.	Out
c18	10	4	Misc.	Out

INVESTIGATING INTERESTING CAPABILITIES

MalPaCA identifies interesting network behaviors using four coarse features from only 20 packet headers. We describe a few examples below. Note how host-based black-listing [13], [63] would not have been able to pick up these behaviors since each host behaves in many different ways.

1. **Connection Direction Identification.** MalPaCA successfully identifies the direction of traffic flow even though no such feature is used. The clusters and their traffic direction are listed in Table 5.3. We continue to see this pattern even when port-related features are removed. Thus, the sequence of packet sizes and their inter-arrival time are collectively indicative of the flow direction. This important trait identifies whether the suspicious behavior is originating from inside or outside the network.
2. **Split-personality C&C Servers.** In several instances, an infected host was observed responding differently to the same request, so much so that the resulting connections ended up in different clusters. For example, two connections of Gozi-ISFB contact 46.38.238.XX, which has been reported as a malicious server located in Germany. The outgoing connections are identical as they both request for the same resource. However, the responses received are very different – the first response contains a small packet followed by a series of 1200-byte packets, while the second one contains a periodic list of small and large packets in the range of 600 to 1800 bytes. A blacklist would have simply grouped these connections since they belong to the same host. In contrast, MalPaCA provides a better understating of the C&C's behavior.
3. **Port Scan Detection.** Some clusters capture a *port scan*¹², which is used for identifying open ports on a network host. Port scans are usually a part of the recon-

¹²<https://whatismyipaddress.com/port-scan>

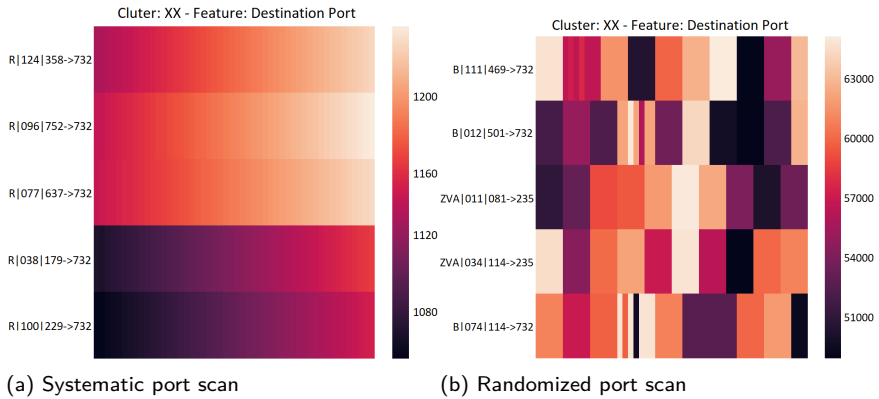


Figure 5.4: Two clusters capturing systematic and randomized port scans.

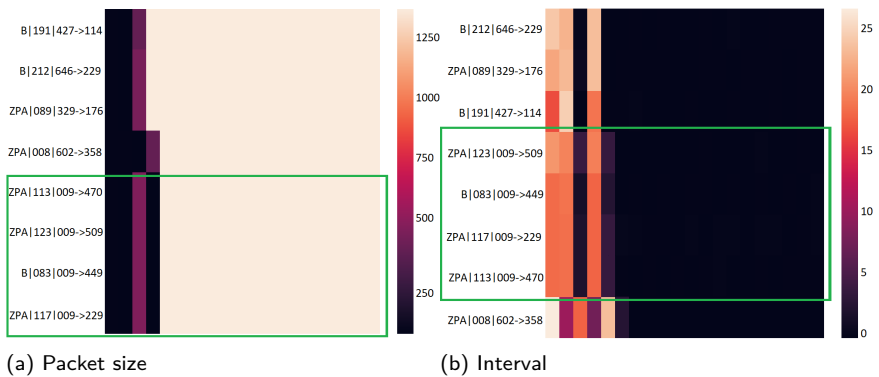


Figure 5.5: A cluster showing Zeus-Panda and Blackmoon connections communicating with IP encoded as 009 and behaving similarly.

naissance phase in the attack kill chain [64]. Utilizing sequences of port numbers enables us to detect any suspicious temporal behavior before an attack happens. The clusters (c4, c5) identify two types of port scans: (i) *systematic port scan* where ports are swept incrementally, which is seen as a gradient in the corresponding temporal heatmap, and (ii) *randomized port scan* where ports are contacted randomly, which shows up in the heatmap as a checkered pattern (see Figure 5.4). Port scans carried out by different connections are clustered together if they contact the same range of port numbers, which increases their mutual similarity. This result is in direct contrast with Mohaisen *et al.* [21] who conclude that port numbers are the least useful features in distinguishing malware families.

4. **C&C Reuse by Multiple Families.** One cluster (c11) contains connections from different families that contact the same C&C server, and their temporal heatmaps look identical (see Figure 5.5). The cluster includes three Zeus-Panda (ZPA) con-

nections and one Blackmoon (B) connection who contact a single IP address (encoded as 009), which has been reported as malicious. The suspicious connections are highlighted in green. This observation suggests that either the YARA rules mislabeled the samples or that the authors share the C&C servers.

5. **Device Probing.** Some clusters (c1, c2, c3) capture connections that connect to the same host. For example, one cluster contains all connections broadcasting to 239.255.255.250, which is used by the SSDP protocol to find plug and play devices. Another cluster captures all connections broadcasting to 224.0.0.252, which is used by the link-local multicast name resolution (LLMNR) protocol to find local network computers.
6. **Malicious Subnet Identification.** In some instances (e.g., c8), several connections contact IP addresses that fall in the same subnet. For example, two Zeus-VM-AES connections contact one host from 62.113.203.XX subnet, while another connection detected 15 days later contacts another host in the same subnet. Similarly, two Zeus-Panda connections and one Blackmoon connection contact two hosts in 88.221.14.XX subnet. This gives actionable intelligence to ISPs to investigate if other IPs in these subnets are also hosting C&C servers.

Table 5.3 shows that *SSDP* (c1) and *broadcast traffic* (c2) are the most common behaviors. Since the dataset is composed of Windows-based malware, it explains why 9 out of 12 families have connections in these two clusters. In contrast, *connection spam* (c6, c7, c9) and *malicious subnet* (c8) are the rarest behaviors. *Malicious subnet* is only exhibited by Zeus-VM-AES. In addition, Gozi-ISFB opens numerous connections, creating a *connection spam*. The incoming connections are stored in one cluster (c6), while the outgoing traffic is split into two clusters due to the difference in the type of requests (c7, c9). This detailed behavior analysis enables the identification of interesting clusters to analyze further.

BEHAVIORAL SIMILARITIES ACROSS MALWARE FAMILIES

Table 5.4 lists the composite behavioral profiles for the 12 malware families in the dataset – each family is represented as the union of its samples' CMSs. Dridex, Gozi-EQ, Zeus-P2P and Zeus-v1 only generate *SSDP* and *broadcast traffic*. Since this traffic is obtained from standard Windows services, it is likely that the malware was not activated when the associated Pcap files were recorded. Gozi-ISFB has the most diverse profile, with connection in 16 out of 18 clusters, which exhibit attacking capabilities such as *port scans* and *connection spamming*. Specifically, the *connection spamming* behavior is never exhibited by any other malware in the dataset. There are two reasons for Gozi-ISFB's diversity: (i) Gozi-ISFB is the largest family under consideration, and (ii) Gozi-ISFB opens more connections per sample compared to other families. For example, one sample of Gozi-ISFB opens 111 connections, while the average number of connections for other malware samples is 3.

5.6.2. COMPARATIVE ANALYSIS WITH EXISTING METHODS

We show MalPaCA's results in relation to existing work by conducting two comparisons: (i) comparing MalPaCA's behavioral profiles against YARA family labels, and (ii) compar-

Table 5.4: Composite behavioral profiles of malware families based on the identified cluster labels.

Cluster labels	B	C	D	DL	GE	GI	R	Z	ZP	ZPa	Zv1	ZVA
SSDP traffic	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓
Broadcast traffic	✓	✓		✓		✓	✓		✓		✓	✓
LLMNR traffic	✓	✓		✓		✓						
System. port scan	✓	✓				✓	✓					✓
Random. port scan	✓	✓				✓	✓					✓
In conn spam						✓						
Out conn spam						✓						
Malicious Subnet												✓
In HTTPs		✓		✓		✓				✓		
Out HTTPs						✓				✓		
C&C reuse	✓									✓		
Misc.	✓	✓		✓		✓		✓		✓		✓
# Clusters	7	11	1	8	1	16	4	2	1	7	1	7

ing MalPaCA's cluster quality with an existing approach based on statistical features.

5

COMPARISON WITH TRADITIONAL FAMILY LABELS VIA DAG

Figure 5.6 shows the DAG that represents the behavioral relationships between the 216 malware samples (and their associated Pcap files). We use this DAG to compare the malware family labels against MalPaCA's behavioral profiles, and identify discrepancies.

Each vertex in the DAG shows a unique CMS, and the number of malware samples that share it. For example, the vertex with the CMS of "00000000000001010" is labeled as "Citadel(2), Gozi-ISFB(7)" because 2 Citadel Pcaps and 7 Gozi-ISFB Pcaps show the same behavior – their connections are co-located in c15 and c17. The root (on the left most side) contains the Pcaps for which all connections were discarded as noise. These Pcaps require further investigation as they may contain zero-day attacks. Pcaps showing increasingly more behaviors are placed towards the right of the graph, with the right most vertex "111110000001100000 Citadel(1)" containing one Citadel Pcap that shows the most diverse behavior.

The graph shows four major partitions (denoted by G1-G4), indicating that there are four high-level behavioral sub-groups present in the dataset. The G2 group containing only one vertex stands out. It contains Pcaps from Zeus-Panda and Blackmoon, and are the only malware samples that share a C&C server. This observation makes a strong case that these particular Pcap files, albeit originating from two families, are behaviorally alike. The G3 group contains Pcaps from various families that are observed doing port scans and broadcasting behaviors. Some servers from this group also form malicious subnets. The G4 group, on the other hand, is the largest group that uses HTTPs traffic along with broadcasting behaviors. The G1 group is highly dominated by Gozi-ISFB and is observed doing connection spamming, along with using HTTPs traffic. Some connections from these Gozi-ISFB Pcaps were placed in the behavioral clusters that we failed to label (c13-c18).

The vertex location for some malware samples is intriguing. For example, most of the Zeus-VM-AES Pcaps that are associated with malicious subnets are located in the G3 group, together with Ramnit files that are associated with port scans. Dridex-Loader is only observed in group G4, where Citadel Pcaps are also co-located. Blackmoon

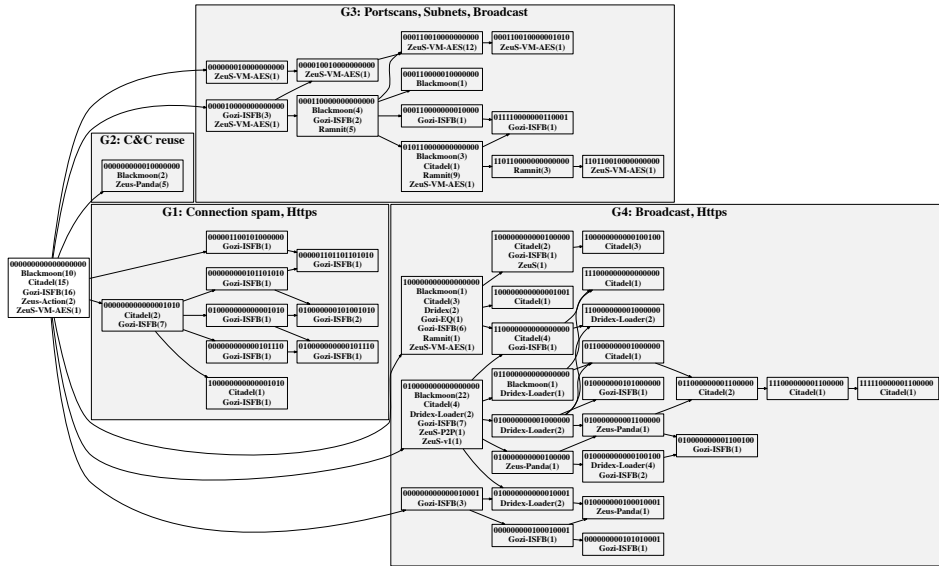


Figure 5.6: The DAG shows the relationship among the behavioral profiles of the 216 malware samples. Each vertex shows a CMS and the YARA labels of the malware samples that share it.

and Gozi-ISFB Pcaps are distributed over all of the behavioral sub-groups. However, Gozi-ISFB is seen dominating the G1 group, while Backmoon dominates the G4 group. Furthermore, as observed from Table 5.4, Gozi-ISFB's Pcaps collectively show 16 discrete behaviors and Citadel's Pcaps show 11 behaviors. However, Citadel shows more discrete behaviors per Pcap versus Gozi-ISFB – Gozi-ISFB's Pcaps contain more (behaviorally similar) connections on average, while the Pcaps themselves are more behaviorally dissimilar than Citadel's Pcaps.

Zeus-Panda's Pcaps are divided into two behavioral sub-groups – one in G2 group with Backmoon samples and the other in the G4 group. Zeus-v1, Zeus-P2P, ZeuS, Gozi-EQ, and Dridex are only seen at the left side of the graph, indicating that none of their distinguishing behaviors were present in the dataset.

COMPARISON WITH STATISTICAL FEATURES

Baseline setup. We compare the cluster quality of using sequential versus statistical features. We use the existing method by Tegeler *et al.* [13] (called baseline, henceforth) to compare our results since they not only use statistical features, but also incorporate periodic behavior using fourier transform to detect bot-infected network traffic. Although the goal of their study diverges from ours, their feature selection approach is similar. For objectivity, we keep the rest of the pipeline as explained in Section 5.3. Taking guidelines from Tegeler *et al.* [13] and adapting them to our problem statement, each connection in the baseline is characterized by 1) average packet size, 2) average interval between packets, 3) average duration of a connection, and 4) the maximum power spectral density (PSD) of the FFT obtained by the binary sampling approach by Tegeler *et al.* [13] –

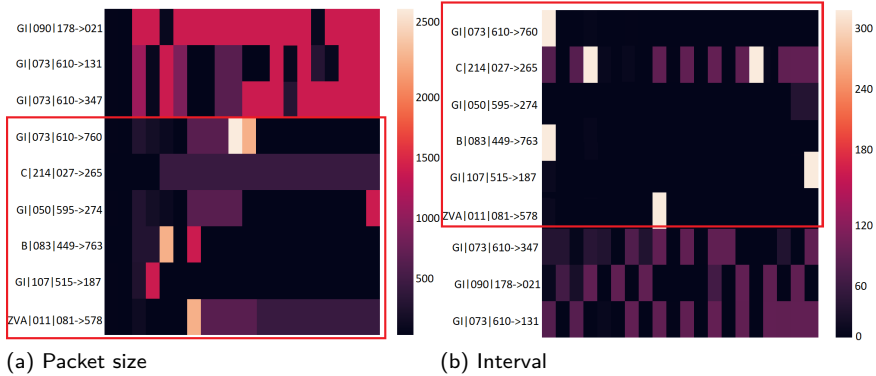


Figure 5.7: Baseline clusters: 6/9 behaviorally different connections are clustered in the baseline.

5

the signal is 1 when a packet is present in the connection and is 0 in between.

Cluster quality comparison. The baseline method results in 22 clusters, with an average of 21.2 connections per cluster. 265 connections are discarded as noise. These results are in contrast to sequence clustering – 18 clusters; 25 connections per cluster, on average, and 284 connections discarded as noise.

The baseline seems to perform better with smaller cluster size on average and discarding fewer connections as noise. However, a deeper analysis shows that the clusters lack quality:

1. With statistical features, the connections present in most clusters appear very different from other connections in the cluster. On average, 57.5% connections per cluster have visually different temporal heatmaps, compared to 8.3% for sequential features. Figure 5.7 shows a cluster from the baseline. It has nine connections, out of which six are errors based on their behavior. The *rightful owners* of the cluster are the connections that have the least mutual distance, *i.e.*, GI|090|178→021, GI|073|610→131, GI|073|610→347. The other six connections have minor differences in all features, except the source port (which is 6/TCP). They were clustered together because their statistical features had the least mutual distance, *i.e.*, $average_time_interval = 19.77 \pm 3.11$; $fft = 0.07 \pm 0.05$; $average_duration = 397.7 \pm 61.7$; $average_bytes = 573.3 \pm 113.8$. The temporal heatmaps show behavioral differences in nearly all clusters.
2. Statistical features are also unable to identify the direction of network traffic. In the cluster shown in Figure 5.7, there is one incoming connection in the cluster along with eight outgoing ones. A similar trend is observed for 19 out of 22 clusters. In contrast, sequences of packet size and inter-arrival time are enough to identify traffic direction in sequence clustering.

In summary, while statistical features may be simple to use, they lose behavioral information that plays a crucial role in accurately determining similarities in network be-

havior. Sequence clustering obtains significantly better clusters. Given that modeling behavioral profiles is already challenging for short sequences, it is remarkable that MalPaCA can identify network behaviors using only 20 packets and 4 coarse features.

5.7. LIMITATIONS AND FUTURE WORK

First, performance optimizations are needed to make sequence clustering more efficient and scalable to perform automated capability assessment in (near) real-time. In MalPaCA, DTW forms the main bottleneck as the length of sequences grows longer. There exist streaming versions of DTW that compute results in real-time. One such technique is presented by Oregi *et al.* [65]. Another alternative is to replace the offline clustering algorithm with an online, adaptive clustering algorithm with inherent support for sequences and evolving data distributions.

Second, density-based clustering discards rare events as noise. This makes sense if the dataset is noisy. However, in the presence of a purely malicious dataset, the connections that lie in lower-density regions may represent rare attacking capabilities, which are discarded in the current implementation.

In the future, we will work on (i) combining MalPaCA with a streaming sequence clustering approach to create behavioral profiles in real-time; (ii) automating the cluster labeling by building a directory of observed behaviors; (iii) integrating additional behavioral data sources in MalPaCA so as to create holistic profiles using static, system-level, and network behavior, and (iv) testing and improving MalPaCA's adversarial robustness.

5.8. CONCLUSIONS

In this chapter, we propose MalPaCA, an intuitive network traffic-based tool-chain to perform malware capability assessment: It groups capabilities using sequence clustering, and uses the cluster membership to build network behavioral profiles. We also propose a visualization-based cluster evaluation method, which serves as a post-hoc explanation method for the clusters, allowing malware analysts to investigate, understand, and even correct labels, if necessary. We implement MalPaCA and evaluate it on real-world financial malware samples collected in-the-wild. MalPaCA independently identifies attacking capabilities. We build a DAG to show overlapping malware behaviors, and discover a number of samples that do not adhere to their family names, either because of incorrect labeling by black-box solutions or extensive overlap in the samples' behavior. We also show that sequence clustering outperforms an existing statistical method by making only 8.3% errors, as opposed to 57.5%. MalPaCA, with its visualizations and capability assessment, can actively support the understanding of malware samples. The resulting behavioral profiles give malware researchers a more informative and actionable characterization of malware than current family designations.

REFERENCES

- [1] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, "Beyond labeling: Using clustering to build network behavioral profiles of malware families", in

- Malware Analysis Using Artificial Intelligence and Deep Learning*, Springer, 2021, pp. 381–409.
- [2] A. Kantchelian, M. C. Tschantz, S. Afroz, *et al.*, “Better malware ground truth: Techniques for weighting anti-virus vendor labels”, in *AISec*, 2015.
 - [3] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, “AVClass: A tool for massive malware labeling”, in *RAID*, Springer, 2016, pp. 230–253.
 - [4] P. Li, L. Liu, D. Gao, and M. K. Reiter, “On challenges in evaluating malware clustering”, in *RAID*, Springer, 2010, pp. 238–255.
 - [5] P. Black, I. Gondal, and R. Layton, “A survey of similarities in banking malware behaviours”, *Computers & Security*, 2017.
 - [6] J. Moubarak, M. Chamoun, and E. Filiol, “Comparative study of recent MEA malware phylogeny”, in *Computer and Communication Systems (ICCCS)*, IEEE, 2017, pp. 16–20.
 - [7] A. Sharma, E. Gandotra, D. Bansal, and D. Gupta, “Malware Capability Assessment using Fuzzy Logic”, *Cybernetics and Systems*, pp. 1–16, 2019.
 - [8] B. Anderson and D. McGrew, “Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity”, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1723–1732.
 - [9] R. Perdisci, W. Lee, and N. Feamster, “Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces”, in *NSDI*, vol. 10, 2010.
 - [10] L. Cavallaro, C. Kruegel, and G. Vigna, “Mining the network behavior of bots”, *Technical Report 2009-12*, 2009.
 - [11] K. Rieck, P. Trinius, C. Willems, and T. Holz, “Automatic analysis of malware behavior using machine learning”, *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.
 - [12] S. Garcia, “Modelling the Network Behaviour of Malware To Block Malicious Patterns. The Stratosphere Project: A Behavioural IPS”, *Virus Bulletin*, 2015.
 - [13] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, “Botfinder: Finding bots in network traffic without deep packet inspection”, in *CoNEXT*, ACM, 2012, pp. 349–360.
 - [14] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero, “Finding non-trivial malware naming inconsistencies”, in *ICISS*, 2011, pp. 144–159.
 - [15] A. Mohaisen, O. Alrawi, M. Larson, and D. McPherson, “Towards a methodical evaluation of antivirus scans and labels”, in *ISA workshop*, Springer, 2013, pp. 231–241.
 - [16] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, “Disclosure: Detecting botnet command and control servers through large-scale netflow analysis”, in *ACSAC*, ACM, 2012, pp. 129–138.
 - [17] Y. Li, J. Jang, X. Hu, and X. Ou, “Android malware clustering through malicious payload mining”, in *RAID*, Springer, 2017, pp. 192–214.

- [18] A. Acar, H. Fereidooni, T. Abera, *et al.*, “Peek-a-Boo: I see your smart home activities, even encrypted!”, *arXiv preprint arXiv:1808.02741*, 2018.
- [19] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, “Can’t you hear me knocking: Identification of user actions on android apps via traffic analysis”, in *CODASPY*, ACM, 2015, pp. 297–304.
- [20] F. Aioli, M. Conti, A. Gangwal, and M. Polato, “Mind your wallet’s privacy: Identifying Bitcoin wallet apps and user’s actions through network traffic analysis”, in *SIGAPP*, ACM, 2019, pp. 1484–1491.
- [21] A. Mohaisen, O. Alrawi, and M. Mohaisen, “AMAL: High-fidelity, behavior-based automated malware analysis and classification”, *Computers & Security*, vol. 52, 2015.
- [22] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, “Scalable, behavior-based malware clustering”, in *Network and Distributed System Security*, vol. 9, 2009, pp. 8–11.
- [23] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges”, *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, 2009.
- [24] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, “TABOR: A graphical model-based approach for anomaly detection in industrial control systems”, in *AsiaCCS*, 2018.
- [25] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning”, in *2017 International Conference on Information Networking (ICOIN)*, IEEE, 2017, pp. 712–717.
- [26] M. Z. Rafique and J. Caballero, “Firma: Malware clustering and network signature generation with mixed network behaviors”, in *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2013, pp. 144–163.
- [27] G. Pellegrino, Q. Lin, C. Hammerschmidt, and S. Verwer, “Learning behavioral fingerprints from netflows using timed automata”, in *IFIP*, IEEE, 2017, pp. 308–316.
- [28] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, “Dendroid: A text mining approach to analyzing and classifying code structures in android malware families”, *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104–1117, 2014.
- [29] S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi, “A survey on malware analysis and mitigation techniques”, *Computer Science Review*, vol. 32, pp. 1–23, 2019.
- [30] Y. Feng, S. Anand, I. Dillig, and A. Aiken, “Apposcopy: Semantics-based detection of android malware through static analysis”, in *SIGSOFT*, ACM, 2014, pp. 576–587.
- [31] A. Azab, R. Layton, M. Alazab, and J. Oliver, “Mining malware to detect variants”, in *Cybercrime and Trustworthy Computing Conference*, IEEE, 2014, pp. 44–53.
- [32] M. Sun, X. Li, J. C. Lui, R. T. Ma, and Z. Liang, “Monet: A user-oriented behavior-based malware variants detection system for android”, *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1103–1112, 2016.

- [33] N. W. H. Chan and S. J. Yang, "SCANNER: Sequence clustering of android resource accesses", in *IEEE DSC 2017*, 2017.
- [34] A. A. Ghorbani and S. Nari, "Automated malware classification based on network behavior", in *ICNC*, IEEE, 2013, pp. 642–647.
- [35] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey", *Info Sec Journal*, vol. 5, no. 02, p. 56, 2014.
- [36] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An automated classification system based on the strings of trojan and virus families", in *MALWARE*, IEEE, 2009.
- [37] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, and K. Bobrovnikova, "A technique for the botnet detection based on DNS-traffic analysis", in *ICCNCT*, Springer, 2015, pp. 127–138.
- [38] J. Lee and H. Lee, "GMAD: Graph-based Malware Activity Detection by DNS traffic analysis", *Computer Communications*, vol. 49, 2014.
- [39] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)", *CVHT journal*, vol. 14, no. 3, 2017.
- [40] A. Azab, M. Alazab, and M. Aiash, "Machine learning based botnet identification traffic", in *IEEE Trustcom/BigDataSE/ISPA*, IEEE, 2016, pp. 1788–1794.
- [41] C. Hammerschmidt, S. Marchal, R. State, and S. Verwer, "Behavioral clustering of non-stationary IP flow record data", in *CNSM*, IEEE, 2016, pp. 297–301.
- [42] M. B. Ntlangu and A. Baghai-Wadji, "Modelling Network Traffic Using Time Series Analysis: A Review", in *IoTBDS*, 2017, pp. 209–215.
- [43] K.-H. Kim and M.-J. Choi, "Android malware detection using multivariate time-series technique", in *APNOMS*, Aug. 2015, pp. 198–202.
- [44] P. Vinod, V. Laxmi, M. Gaur, and G. Chauhan, "MOMENTUM: metamorphic malware exploration techniques using MSA signatures", in *IIT*, IEEE, 2012, pp. 232–237.
- [45] D. Baysa, R. M. Low, and M. Stamp, "Structural entropy and metamorphic malware", *CVHT journal*, vol. 9, no. 4, pp. 179–192, 2013.
- [46] G. Volis, C. Makris, and A. Kanavos, "Two Novel Techniques for Space Compaction on Biological Sequences", *WEBIST*, 2016.
- [47] W.-J. Li, K. Wang, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis", in *IEEE SMC Information Assurance Workshop*, IEEE, 2005, pp. 64–71.
- [48] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family android malware", in *International Conference on Availability, Reliability and Security*, IEEE, 2015, pp. 333–340.
- [49] S. Verwer, R. Eyraud, and C. De La Higuera, "PAutomaC: A probabilistic automata and hidden Markov models learning competition", *Machine learning*, vol. 96, no. 1-2, pp. 129–154, 2014.

- [50] Z. M. Kovacs-Vajna, “A fingerprint verification system based on triangular matching and dynamic time warping”, *TPAMI*, vol. 22, no. 11, pp. 1266–1276, 2000.
- [51] A. Wang, A. Mohaisen, W. Chang, and S. Chen, “Capturing DDoS attack dynamics behind the scenes”, in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Springer, 2015, pp. 205–215.
- [52] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, “Inferring protocol state machine from network traces: A probabilistic approach”, in *ACNS*, Springer, 2011, pp. 1–18.
- [53] R. J. Campello, D. Moulavi, and J. Sander, “Density-based clustering based on hierarchical density estimates”, in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2013, pp. 160–172.
- [54] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series”, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 10, pp. 359–370, 1994.
- [55] L. Zahrotun, “Comparison Jaccard similarity, cosine similarity and combined both of the data clustering with shared nearest neighbor method”, *CE&AJ*, vol. 5, no. 1, pp. 11–18, 2016.
- [56] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”, *CAM journal*, vol. 20, 1987.
- [57] D. L. Davies and D. W. Bouldin, “A cluster separation measure”, in *TPAMI 1979*, 1979.
- [58] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, “Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail”, in *IEEE Symposium on Security and Privacy*, IEEE, 2012, pp. 332–346.
- [59] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, “WARP: time warping for periodicity detection”, in *Data Mining*, IEEE, 2005, 8–pp.
- [60] A. Adadi and M. Berrada, “Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)”, *IEEE Access*, 2018.
- [61] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, “Explainable machine learning for scientific insights and discoveries”, *IEEE Access*, vol. 8, pp. 42 200–42 216, 2020.
- [62] V. Kalgutkar, N. Stakhanova, P. Cook, and A. Matyukhina, “Android authorship attribution through string analysis”, in *International Conference on Availability, Reliability and Security*, 2018, pp. 1–10.
- [63] I. Ghafir and V. Prenosil, “Blacklist-based malicious ip traffic detection”, in *GCCT*, IEEE, 2015, pp. 229–233.
- [64] T. Yadav and A. M. Rao, “Technical aspects of cyber kill chain”, in *SSCC*, 2015.
- [65] I. Oregi, A. Pérez, J. Del Ser, and J. A. Lozano, “On-Line Dynamic Time Warping for Streaming Time Series”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2017, pp. 591–605.

6

REAL-TIME SEQUENCE CLUSTERING FOR NETWORK ATTACK SUMMARIZATION

For constantly evolving malware, automated behavior discovery methods necessitate real-time sequence clustering. In this chapter, we first propose SECLEDS, the first interpretable algorithm for clustering sequences in an evolving data stream. We also develop a post-hoc explanation method to explain the clusters in real-time. We empirically demonstrate that SECLEDS produces high-quality clusters regardless of drift, stream size, data dimensionality, and number of clusters. We compare against three popular stream and batch clustering algorithms, and show that SECLEDS achieves a comparable F1 score to state-of-the-art while reducing the number of required distance computations by 83.7%. Importantly, SECLEDS outperforms all baselines by 138.7% when the stream contains drift.

We utilize SECLEDS to summarize network traffic while preserving temporal patterns. By clustering network traffic generated by real botnets, we show that SECLEDS can be used to create real-time behavioral profiles of malware-infected devices, while supporting network bandwidths of up to 1.08 GB/s using the (expensive) dynamic time warping distance.

This chapter is based on the paper “SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting” by **Nadeem, A.**, & Verwer, S. in European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), 2022 (pp. 157-173) [1]. Sections 6.3.4 and 6.7 were added afterward.

6.1. INTRODUCTION

Stream clustering is the problem of clustering a potentially unbounded stream of items in a single pass, where the items arrive sequentially without any particular order, *e.g.*, network traffic, financial transactions, and sensor data. Stream clustering algorithms are required to have low memory overhead, high computational efficiency, and concept drift robustness, *i.e.*, due to evolving data distributions [2]. Maintaining high cluster quality in a fully online setting is extremely difficult. Therefore, hybrid online-offline algorithms are popular among existing approaches, *e.g.*, CluStream [3], StreamKM++ [4], DenStream [5], and BIRCH [6]. These algorithms have an online component that summarizes the data stream, and an offline component that periodically uses that information to create the final clusters. There also exist algorithms that store part of the stream for handling outliers, *e.g.*, BOCEDs [7] and MDSC [8]. Existing stream clustering algorithms handle concept drift by having variable number of clusters: They add new clusters for newly observed behavior and discard clusters that contain too many old data items. This leads to higher memory requirements for managing buffers and intermediate solutions. Batch clustering algorithms can also be used in a streaming setting by considering a batch size of one, *e.g.*, Minibatch k-means [9]. However, they start to under-perform when the stream contains drift.

In recent years, sequential data has increasingly become popular because of the powerful insights that it provides regarding behavioral analytics [10], *e.g.*, for attacker strategy profiling [11], malware behavior characterization [12], fraud detection [13], and human activity recognition [14]. Clustering sequences in an offline setting is challenging in itself because sequences are often out-of-sync, requiring expensive alignment-based distance measures, which are often not supported by many clustering algorithms [15]. K-medoids or Partitioning Around Medoids (PAM) has often been used to cluster sequences because the k-centers are represented by actual data items, called medoids or prototypes [16], [17]. This has multiple benefits: i) it makes the cluster interpretation simpler; ii) it enables the use of non-metric distances such as dynamic time warping (DTW), and iii) it allows to estimate exact storage requirements based on the k-fixed clusters. Although the state-of-the-art offline k-medoids algorithms, *i.e.*, FastPAM1 [18] and BanditPAM [19] have reduced the runtime complexity to $\mathcal{O}(n \log n)$, they are still not efficient enough to be used in streaming settings, and the cluster quality will degrade over time as the stream evolves. To the best of our knowledge, there exists no streaming version of the k-medoids algorithm that can efficiently cluster sequential data with concept drift.

Contributions. In this chapter, we propose SECLEDS, a lightweight streaming version of the k-medoids algorithm with *constant memory footprint*. SECLEDS has two unique properties: Firstly, it uses p-medoids per cluster to maintain stable high-quality clusters. Note the difference from IMMFC [20], which uses the information of multiple medoids in independent sub-solutions to select the final medoids. We initialize the p-medoids using a non-uniform sampling strategy similar to k-means++. Secondly, a medoid voting scheme is used to estimate a cluster's center of mass. The offline k-medoids has a SWAP step that tests each point in a cluster to determine the next medoid. SECLEDS cannot do this because it does not store any part of the stream. Instead, it maintains votes for

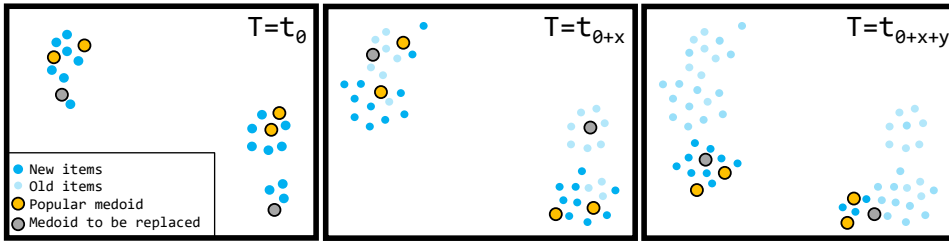


Figure 6.1: An illustration of SECLEDS' clusters following an evolving data stream. The medoids close to recent data gain more votes, while the medoids with the least votes are replaced with new data items from the stream. The k -clusters handle concept drift by capturing different concepts in the stream at different time steps.

each medoid that estimate how representative (valuable) it is given the data seen so far. A user-supplied decay factor enables SECLEDS to slowly forget the votes regarding past data. The least representative medoids are then replaced with new data items. This way, rather than creating new clusters for new concepts, the k -clusters themselves evolve with the data stream. Figure 6.1 shows how the clusters follow a data stream as it evolves. Thus, the k -clusters represent different concepts in the stream at different time steps. We release SECLEDS as open-source¹. It addresses the following real-world constraints:

- I. A runtime efficient medoid-based clustering algorithm with a fixed memory footprint that can handle high-bandwidth data streams;
- II. An algorithm that produces high-quality clusters in a streaming environment while being able to deal with concept drift;
- III. Accurate sequence clusters using alignment-based distances;
- IV. Cluster explanations for understanding the captured concepts, and
- V. Minimal parameter settings to support ease-of-use.

Empirical results. We experiment on several real and synthetic data streams that contain 2D points, univariate, and multivariate sequences. We empirically demonstrate that SECLEDS produces high-quality clusters regardless of drift, stream size, and number of clusters. We use the following state-of-the-art and popular clustering algorithms as baselines: a) Streaming: CluStream, StreamKM++; b) Batch: Minibatch k -means, and c) Offline: BanditPAM. Particularly, BanditPAM is used as a benchmark for the best achievable clustering on a static dataset. The results show that i) SECLEDS achieves comparable F1 score to BanditPAM, while reducing the required number of distance computations by 83.7%; ii) SECLEDS outperforms all baselines by 138.7% when the stream contains drift, and iii) SECLEDS is faster than BanditPAM and CluStream on most clustering tasks.

¹SECLEDS: <https://github.com/tudelft-cda-lab/SECLEDS>

Use cases. We also discuss two use cases from the cybersecurity domain where network traffic is typically randomly sampled to keep the storage requirements within a predefined budget. Consequently, temporal patterns in the network traffic are lost that could have been useful for downstream tasks, *e.g.*, behavioral analytics. We propose a smarter sampling technique that uses medoid-based stream clustering (SECLEDS) to summarize the network traffic and create behavioral profiles of malware: (i) SECLEDS clusters sequences of network traffic and periodically stores the medoids of each cluster, thus reducing the storage needs while preserving temporal patterns in the data. By clustering real-world network traffic coming from botnets, we provide evidence that SECLEDS (and SECLEDS-dtw) can support network bandwidths of up to 2.79 GB/s (and 1.08 GB/s), respectively. (ii) SECLEDS is used together with MalPaCA [12] to generate behavioral profiles of bot-infected hosts for real-time capability assessment. We show that the profiles generated from real-time sequence clustering are more comprehensive compared to the profiles created from random sampling.

6.2. PRELIMINARIES

Stream. Given a sensor that receives an unbounded stream of multivariate data points $X = \{x_1, x_2, \dots\}$ with dimensionality d , arriving at time steps $T = \{t_1, t_2, \dots\}$, a sequential data stream is defined as $S = \{s_1, \dots, s_n, \dots\}$, where s_i is a time window w over X such that $s_i = \{x_i, x_{i+1}, \dots, x_{i+w}\}$, and y_i is its associated class label. Traditional point clustering considers $w = 1$, while for sequence clustering, we consider $w > 1$. We use two configurations, *i.e.*, $d = 2, w = 1$ (2D point clustering) and $d = 1, w = 100$ (univariate sequence clustering). A case of bivariate variable length sequences is given in appendix 6.9.4.

Concept drift. Real-world data streams often change unexpectedly over time. This shift alters the statistical properties of their underlying distribution. In machine learning, this is called concept drift [21], [22]. Concept drift is typically categorized into four types [23]: (i) *Sudden drift* where a new concept arises abruptly; (ii) *Gradual drift* where an old concept is slowly replaced by a new one; (iii) *Incremental drift* where a concept incrementally turns into another one, and (iv) *Recurring concepts* are old concepts that reappear from time to time. Years of research has gone into developing *concept drift detectors* that either monitor the underlying data distribution, error rate, or perform hypothesis testing to trigger model retraining [23], [24]. Typical stream clustering algorithms handle concept drift by introducing new clusters for new concepts, and discarding old irrelevant clusters [25]. Although intuitively appealing, this requires user-supplied parameters that define what ‘new’ means.

6.3. SEQUENCE CLUSTERING IN EVOLVING DATA STREAMS – SECLEDS

SECLEDS (Sequence Clustering in Evolving Data Streams) is a lightweight streaming variant of the classical k-medoids (PAM) algorithm. To support high bandwidth data streams, SECLEDS does not store any part of the stream in memory – it receives an item, assigns it to one of the k-clusters, and then discards it. This way, SECLEDS has a guar-

anteed constant memory footprint [I]. However, this requirement cannot be achieved using the offline BUILD and SWAP steps of PAM. Instead, SECLEDS performs a non-uniform sampling (similar to k-means++) on an initial batch of the stream to initialize the medoids. It also makes use of *multiple medoids per cluster* to provide a stable cluster definition in a streaming setting where noise and concept drift are common properties [II]. These medoids are utilized to craft local and global cluster explanations [IV].

The efficiency of a chosen distance measure is usually the primary performance bottleneck in sequence clustering. Thus, minimizing the number of distance computations is key to scaling SECLEDS to large data streams. This is achieved by introducing a *medoid voting scheme* whose purpose is twofold: (i) It determines the center of mass of a cluster for estimating how representative a medoid is given the data seen so far. (ii) By using this information, old irrelevant medoids are replaced by new ones that are located near recent data. Hence, better medoids can be found without having to perform additional distance computations [I]. This also allows SECLEDS to support robust but computationally expensive distance measures specifically meant for sequential data, e.g., dynamic time warping [III]. Finally, SECLEDS handles concept drift by regularly *forgetting* past data and occupying newer regions/concepts in the data stream. This is achieved by applying exponential decay λ to the medoid votes at each time step [II].

SECLEDS has a modular implementation in Python. The k-clusters, p-medoids per cluster, and decay rate λ are the only three user-supplied parameters needed for the algorithm, making it useful for exploratory data analysis [V]. We believe these parameters are easier to tune compared to many radius- or density-based hyperparameters in existing clustering algorithms, which require a deeper understanding of the data distribution in advance.

6.3.1. STABLE CLUSTER DEFINITION VIA MULTIPLE MEDOIDS

A new data item s is assigned to a cluster cid with the least average distance to its medoids. With multiple medoids per cluster, this provides a robust cluster assignment. Additionally, the medoid voting scheme encourages the medoids to represent different sections of a class so they can gain votes: If they are too close together, some of them do not receive votes and get replaced eventually. It also ensures that outliers are quickly replaced because of fewer votes.

Concept drift and cluster initialization determine how the medoids behave. Figure 6.2 illustrates three scenarios with varying medoid behavior for a single cluster as a function of votes gained over time: (a) Assuming no concept drift, when the stream is roughly evenly shuffled, all the medoids receive uniform votes on average. This is because all medoids are close to parts of the stream at different time steps. At a specific time step, the medoid closest to the most amount of recent data becomes popular. Figure 6.2-A shows that each medoid becomes popular at some point in the stream, indicating that the medoids represent different sections of the underlying class. (b) When the stream incrementally drifts, the cluster follows the evolving stream by replacing the least popular medoids with recent data items from the stream. Since the new medoids are now closer to new data, they gain more votes and become popular. This has roughly the same effect as the first case. (c) When the data arrives one class at a time, all clusters are initialized in a single class. As data from a new class appears, one medoid from the closest

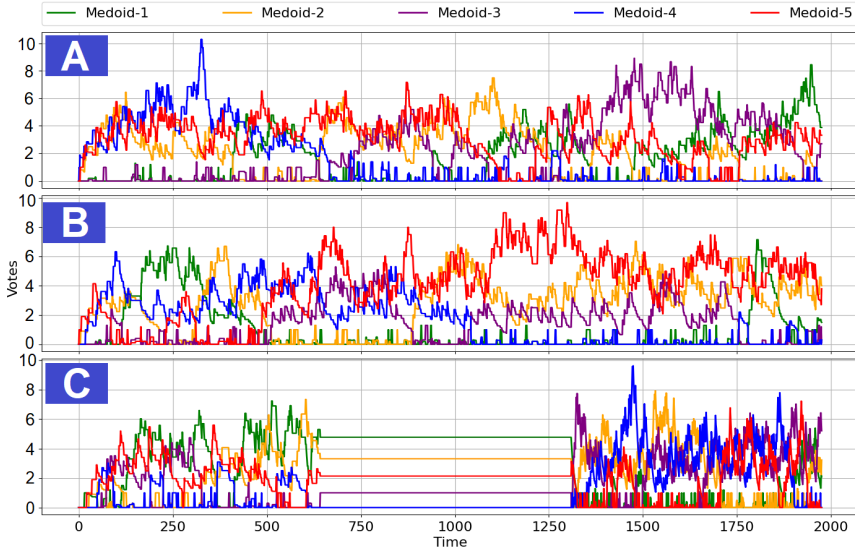


Figure 6.2: The effect of cluster initialization and concept drift on medoid selection, where $p = 5$. (A): Given a uniformly distributed stream, every medoid becomes popular at some point. (B): For an incrementally drifted stream, the medoid close to drifted data becomes popular. (C): For a class-ordered stream, all clusters start from one class, until one medoid migrates to the correct class, followed by other medoids. Here, the correct class is observed from t_{1300} . Medoid-3 migrates first and becomes popular, while medoid-1 migrates last.

cluster migrates to it and starts gaining votes. Over time, the older popular medoids lose their votes because of exponential decay, and eventually migrate to the new class. This is shown from t_{1300} onward in Figure 6.2-C, highlighting the importance of multiple medoids in noisy streams.

6.3.2. CENTER OF MASS ESTIMATION

The voting scheme provides an estimate of a cluster's center of mass by assigning more votes to recently observed data in the stream S , while exponential decay helps to forget votes regarding older data. Without decay, older clusters with popular medoids never evolve. Thus, these properties help to replace irrelevant medoids, *e.g.*, those that are located a) close to the least amount of recent data, or b) in a region where new data no longer arrives. Note that we only apply exponential decay to the *most recently updated cluster*, so that we do not forget valuable information about other clusters while the data from this class arrives.

6.3.3. THE SECLEDS ALGORITHM

SECLEDS has three modules: an initialization module (INIT), an assignment module (ASSIGN), and an update module (UPDATE). The task is to assign each item in S to one of the k -clusters. SECLEDS maintains and updates a model of the stream seen so far in the form of k -clusters, $\mathcal{C} = \{C_1, \dots, C_k\}$. For clarity, we use t to denote the clusters at time t (these superscripts are removed in Algorithm 2). Each cluster is represented by a set

of p -medoids and their votes, *i.e.*, for $1 \leq i \leq k$, $C_i^t = \{(m_{i,1}^t, v_{i,1}^t) \dots (m_{i,p}^t, v_{i,p}^t)\}$, where for $1 \leq j \leq p$: $m_{i,j}^t \in S$ is the j^{th} medoid of the i^{th} cluster at time t having $v_{i,j}^t \in \mathbb{R}$ votes.

INIT. A batch \mathbb{B} from the start of S is used to initialize the clusters. The batch can be small but enough to select $k \cdot p$ medoids. In the experiments, we used a batch size of $(1.5 \cdot k \cdot p)$. We use a non-uniform sampling strategy (similar to k -means++ [26]) to select the primary medoid of each cluster: SECLEDS selects the first medoid of the first cluster $(m_{1,1}^t)$ arbitrarily from the batch. Another $k-1$ medoids are sampled with a probability proportional to the squared distance between $m_{1,1}^t$ and other items in \mathbb{B} . This initializes the primary medoid of each cluster. The other $p-1$ medoids for each cluster C_i^t are the items in \mathbb{B} that are closest to its primary medoid $m_{i,1}^t$. This way, the medoids maintain cluster separation by reducing the risk of medoids from multiple clusters overlapping each other. All medoids start with 0 votes.

ASSIGN and UPDATE. With the clusters initialized, the stream processing begins. ASSIGN and UPDATE are called for each item in S . ASSIGN has 3 steps: (i) An incoming item s at time t is assigned to the cluster C_{cid}^t for which its previous medoids C_{cid}^{t-1} have the least average distance to s , formally defined in Eq. 6.1 for any given distance function $d(\cdot, \cdot)$. (ii) The closest medoid to s receives a vote, while exponential decay λ is applied to the other medoids *i.e.*, for all $1 \leq j \leq p$ and $j' \neq j$: $v_{cid,j}^t = (v_{cid,j}^{t-1} + 1)$ if $j = \arg \min_j d(s, m_{cid,j}^{t-1})$, otherwise $v_{cid,j}^t = v_{cid,j}^{t-1} \cdot (1 - \lambda)$. This way, the medoids maintain an estimate of their centers of mass without storing any part of the stream. (iii) The votes of all other clusters remain the same, *i.e.*, $v_{i,j}^t = v_{i,j}^{t-1}$ for all $i \neq cid$ and $1 \leq j \leq p$.

$$C_{cid}^t = \arg \min_{1 \leq cid \leq k} \frac{\sum_{j=1}^p d(s, m_{cid,j}^{t-1})}{p} \quad (6.1)$$

At every time step t , the new data item s is promoted to be a medoid of C_{cid}^t : the medoid having the least votes which is not the newest medoid is replaced by s , *i.e.*, $\{m_{cid,j}^t = s, v_{cid,j}^t = 0\}$ where $j = \arg \min_j v_{cid,j}^{t-1}$ and $m_{cid,j}^{t-1} \neq \eta_{cid}^t$, where η_{cid}^t keeps track of the newest medoid of cluster cid at time t . Inspired by Tabu search [27], including η_{cid}^t ensures that the most-recently updated medoid is not selected to be replaced each time. Tabu search is a local search meta-heuristic that selects which values to change except for the last δ ones ($\delta = 1$ in this case).

Time complexity. Given k clusters, p medoids, b batch size, and n items in the stream, SECLEDS has a time complexity of $\mathcal{O}(n)$: SECLEDS selects the first medoid at random from the initial batch, and then performs b distance computations to find the other $k-1$ primary medoids. The rest of the $k(p-1)$ medoids are also selected using the same distance information. In total, this requires $\mathcal{O}(kb)$ distance computations. For every $s \in S$, SECLEDS computes the average distance to each cluster, which requires kp distance computations. Over an entire run, this gives nkp distance computations. In the UPDATE module, SECLEDS reallocates medoid votes without any distance computations, making the runtime negligible. Since k and p are small user-supplied parameters, the overall runtime complexity is $\mathcal{O}(n)$.

Algorithm 2: SECLEDS for clustering sequences in evolving streams

Input: Data stream, nclusters, nprototypes: S, k, p

```

1 function SECLEDS( $S, k, p$ )
2    $b \leftarrow 1.5 \cdot k \cdot p$ 
3    $\mathbb{B} \leftarrow$  Collect  $b$  items from  $S$ 
4    $\mathcal{C} \leftarrow$  INIT( $\mathbb{B}, k, p$ ) // INIT
5   forall  $s$  in  $S[b:]$  do
6      $cid \leftarrow \operatorname{argmin}_{1 \leq cid \leq k} \frac{1}{p} \cdot \sum_{j=1}^p d(s, m_{cid,j})$  // ASSIGN
7      $j \leftarrow \operatorname{argmin}_j d(s, m_{cid,j})$  for all  $1 \leq j \leq p$ 
8      $v_{cid,j} \leftarrow (v_{cid,j} + 1), v_{cid,j'} \leftarrow v_{cid,j'} \cdot (1 - \lambda)$  for  $j' \neq j$ 
9      $j \leftarrow \operatorname{argmin}_j v_{cid,j}$  where  $m_{cid,j} \neq \eta_{cid}$  for all  $1 \leq j \leq p$  // UPDATE
10     $m_{cid,j} \leftarrow s, v_{cid,j} \leftarrow 0$ 
11    yield  $cid$ 
12 function INIT( $\mathbb{B}, k, p$ )
13   Choose  $m_{1,1} \in \mathbb{B}$  arbitrarily. Let  $C_1 \leftarrow \{(m_{1,1}, 0)\}$ 
14   for  $i \leftarrow 2 \dots k$  do
15     Choose  $m_{i,1} \in \mathbb{B}$  with probability  $d(m_{i,1}, m_{1,1})^2, m_{i,1} \neq m_{1,1}$ 
16     Let  $C_i \leftarrow \{(m_{i,1}, 0)\}$ 
17   for  $i \leftarrow 1 \dots k$  do
18      $dist \leftarrow d(b, m_{i,1})$  for all  $b \in \mathbb{B}$  and  $b \neq m_{i,1}$ 
19     Choose  $\{m_{i,2} \dots m_{i,p}\}$  having smallest values in  $dist$ 
20     Update  $C_i \leftarrow \{(m_{i,1}, 0) \dots (m_{i,p}, 0)\}$ 
21   return  $\{C_1, \dots, C_k\}$ 

```

Space complexity. After initialization, SECLEDS only stores the p medoids and their votes for the k clusters. Since these are (small) user-defined parameters, the space complexity of SECLEDS is $\mathcal{O}(1)$.

6.3.4. REAL-TIME CLUSTER EXPLANATIONS

SECLEDS is an interpretable clustering algorithm because the medoids serve as natural explanations for their respective clusters. However, the medoids are not always easy to interpret, especially for high-dimensional datasets, such as (multivariate) sequences. Ideally, we want to visualize the medoids in their original high-dimensional space (e.g., sine curves, character strokes), while still being able to interpret their meaning in the context of the whole dataset.

ExClus [28] is an explanation method for clustering algorithms that creates feature-wise statistical explanations based on the difference between the data distributions of the cluster and the whole dataset. We borrow initial ideas from ExClus and create an explanation method for sequential clustering in a streaming setting. Note that existing work is severely limited in terms of explanation methods for unsupervised learning, especially in a streaming setting. For instance, although the temporal heatmaps proposed

by Nadeem *et al.* in [12] provide cluster explanations, they require the cluster sequences to be stored in memory, which is not possible for SECLEDS.

We explain each cluster locally and provide a global explanation for all the clusters. These explanations use the concept of feature-wise data distribution. For every feature f in a d dimensional (multivariate) item, the feature-wise data distribution for the cluster cid is the probability density function $Y_{cid,f}$ estimated by the feature-wise mean $\mu_{cid,f}$ and standard deviation $\sigma_{cid,f}$ of the p medoids in that cluster (see Eqs. 6.2-6.4). If the medoids are sequences, we first aggregate each medoid by computing its mean, *i.e.*, $|m_{cid,j,f}|$. Since the medoids are always stored in memory, the cluster distribution is straightforward to compute. In contrast, we must compute the data distribution of the stream seen so far $Y_{new,f}$ in an incremental fashion, *i.e.*, we update the distribution as new data items arrive (see Eqs. 6.5-6.7). Given n items seen so far and s being the newest item, $\mu_{old,f}$ and $\sigma_{old,f}$ represent the mean and standard deviation of the previous $n-1$ items, and $\mu_{new,f}$ and $\sigma_{new,f}$ represent the mean and standard deviation of the whole stream seen so far.

$$\mu_{cid,f} = \sum_{j=1}^p |m_{cid,j,f}| \quad (6.2)$$

$$\sigma_{cid,f} = \sqrt{\frac{\sum_{j=1}^p (|m_{cid,j,f}| - \mu_{cid,f})^2}{p}} \quad (6.3)$$

$$Y_{cid,f} \sim N(\mu_{cid,f}, \sigma_{cid,f}^2) \quad (6.4)$$

$$\mu_{new,f} = \frac{(n-1) \cdot \mu_{old,f} + s}{n} \quad (6.5)$$

$$\sigma_{new,f} = \sqrt{\frac{(n-2) \cdot \sigma_{old,f}^2 + (n-1) \cdot (\mu_{new,f} - \mu_{old,f})^2 + (s - \mu_{new,f})^2}{n-1}} \quad (6.6)$$

$$Y_{new,f} \sim N(\mu_{new,f}, \sigma_{new,f}^2) \quad (6.7)$$

The *local explanation* utilizes the p medoids in each cluster – it computes their feature-wise data distribution, and plots them next to the feature-wise data distribution of the entire data stream seen so far. It also shows the most centrally located (most representative) medoid in that cluster. The central medoid is displayed in the most authentic representation possible, *e.g.*, a medoid is displayed as a character stroke for sequences representing hand-written characters. Utilizing both the statistical (*i.e.*, data distribution) and sequential (*i.e.*, medoid sequence) representation helps to faithfully represent a cluster. The local explanation provides a sense of *cluster cohesion* by showing how similar the medoids are within a cluster. The *global explanation* shows the feature-wise data distribution for each cluster and that for the data stream seen so far. This provides a sense of *cluster separation* by showing the distinct (non-overlapping) concepts captured by each cluster. Figure 6.3 shows the local and global cluster explanations for the univariate sine-curve dataset. The explanations for multivariate network traffic are given in the appendix 6.9.5.

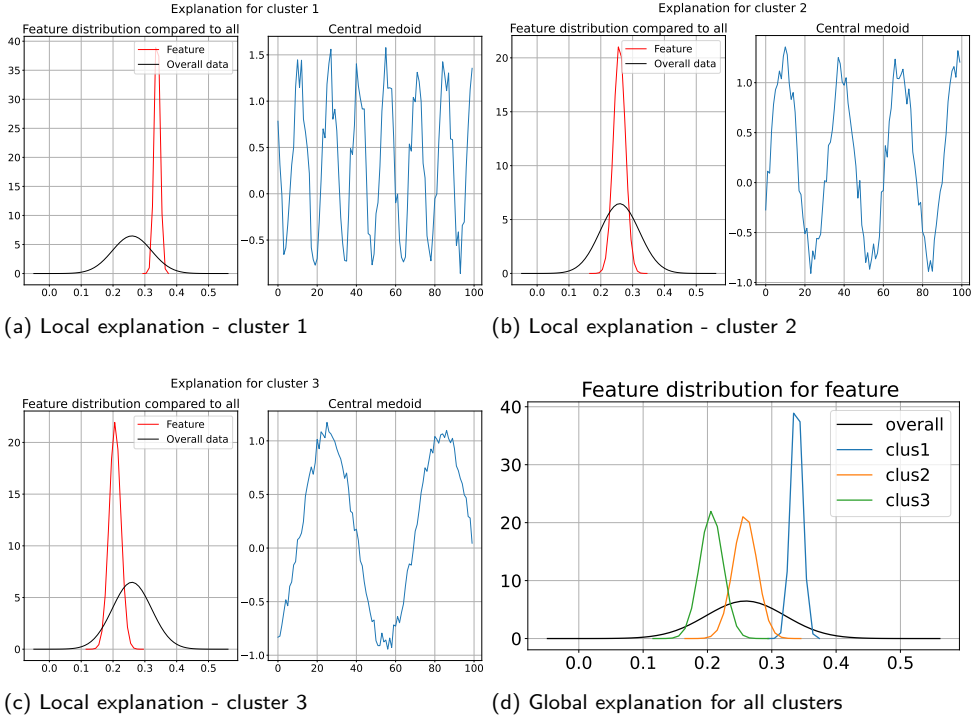


Figure 6.3: The local and global explanations generated by clustering $n = 50,000$ univariate sine curves in $k = 3$ clusters with $p = 5$ medoids each. The local explanation shows the feature distribution of the medoids within a cluster versus that of all the curves, and the single most representative medoid (sine curve) in that cluster (a-c). The global explanation shows the feature distributions of the 3 clusters versus that of all the curves (d).

6.4. DATASET AND EXPERIMENTAL SETUP

Datasets. We use three synthetic and two real datasets containing 2D points and univariate/multivariate sequences, see Table 6.1. The data generation process is given in appendix 6.9.1. The synthetic datasets are released in the SECLEDS code repository.

Blobs. The blob dataset was created using `scikit-learn` [29]. The dataset contains $n = 100,000$ two-dimensional points ($d = 2$), equally distributed in $k = 10$ classes, with varying standard deviations.

Sine-curve. A sine-curve generator was used to create $k = 4$ synthetic univariate sine curves of length 1,250,000 each, using varying *frequency*, *phase*, and *error* (see appendix 6.9.1). Each curve is partitioned using a non-overlapping window of length $w = 100$ to obtain the experimental dataset. In total, $n = 50,000$ curves are obtained, equally divided across $k = 4$ classes.

Sine-curve-drifted. Incremental concept drift is added to the Sine-curve dataset by shifting the phase of each curve by a factor of $(drift \cdot c_id)$, where c_id is the curve index in the stream, and $drift = 0.05$. Note that adding drift to the frequency of the sine curves produces similar results.

Table 6.1: Summary of the experimental datasets.

Dataset	Type	Drift	Stream size (n)	Clusters (k)	Dimensions (d,w)
Blobs	Synthetic	No	100,000	10	(2,1)
Sine-curve	Synthetic	No	50,000	4	(1,100)
Sine-curve-drifted	Synthetic	Yes	50,000	4	(1,100)
CTU13-9(A)	Real-world	Yes	213,386	2	(1,100)
CTU13-9(B)	Real-world	Yes	1418	2	(4,100)

CTU13-9. CTU13 [30] is an open-source dataset composed of network traffic (Netflows) coming from real botnet-infected hosts and normal hosts. We use scenario-9, containing 10 *Neris* botnet infected hosts and 6 benign hosts. A total of 2,087,509 (normal and botnet) Netflows were captured over 5 hours and 37 minutes. We divide the network traffic into 16 sub-streams based on the 16 hosts in the dataset.

We create two variants of CTU13-9 based on the sequence creation strategy. In *CTU13-9(A)*, we obtain $n = 213,386$ univariate sequences of length $w = 100$ (capturing average bytes) using a sliding window model [31] with $step_size = 1$. Figure 6.10 in the appendix 6.9.1 shows a t-SNE plot of the sequences. The unclear class separation emphasizes just how challenging the clustering task is. In *CTU13-9(B)*, we obtain $n = 1418$ multivariate sequences ($d = 4$) of length $w = 100$ (capturing total bytes, inter-arrival times, source ports, destination ports) using a sliding window model with $step_size = w$. Note that there are fewer usable sequences in CTU13-9(B) because of the non-overlapping sliding window, and the multivariate nature of the sequences – we only consider those Netflows for sequence construction that have, *e.g.*, valid port numbers.

We use Euclidean distance for all datasets. For the Sine-curve and network traffic datasets, we additionally use dynamic time warping (DTW). Note that Euclidean distance can only be used with fixed-length sequences and often produces less accurate results compared to DTW [32], [33].

Stream configuration. A data stream S of size n is constructed from a chosen experimental dataset. For each experiment, the clustering task is executed *trials*-times, randomly shuffling the stream each time, to make the results order-invariant. A clustering task invokes SECLEDS and the baselines such that each algorithm observes the exact same order of data arrival. In this chapter, we set *trials* = 10, unless otherwise reported. All experiments are run on Intel Xeon E5620 quad-core processor with 74 GB RAM.

Evaluation. We use two metrics for performance evaluation: i) *runtime* to cluster a stream size of n , and ii) *F1 score* computed from the pairwise co-occurrences of items in the stream using Eq. 6.8, as originally defined in [34].

$$eval(a, b) = \begin{cases} y_a = y_b \wedge C_x = C_y, & \text{true positive} \\ y_a = y_b \wedge C_x \neq C_y, & \text{false negative} \\ y_a \neq y_b \wedge C_x = C_y, & \text{false positive} \\ y_a \neq y_b \wedge C_x \neq C_y, & \text{true negative} \end{cases} \quad (6.8)$$

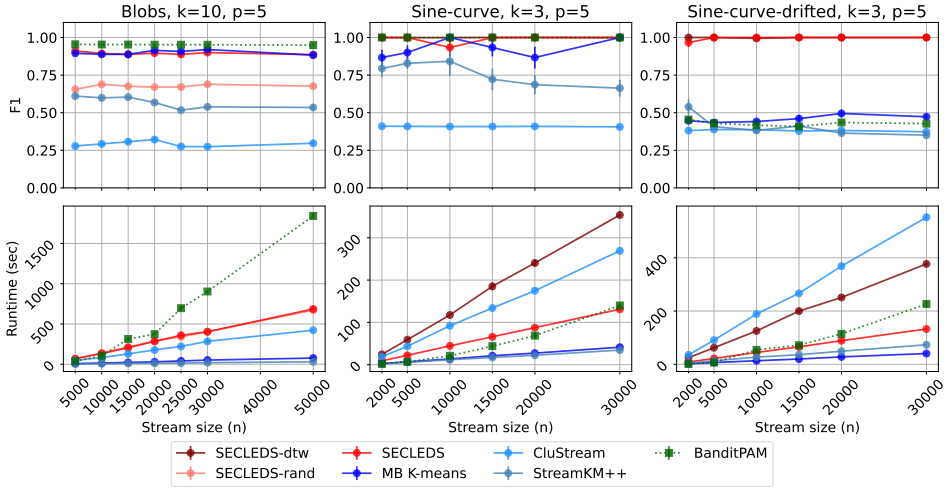


Figure 6.4: Clustering Blobs and Sine-curves: SECLEDS’s runtime grows approximately linearly with stream size, while maintaining competitive F1 score with the best-performing baselines, *i.e.*, BanditPAM and Mini-batch k-means. SECLEDS consistently performs better than all baselines in the presence of concept drift.

6

where y_a and y_b are labels of items a and b that are placed in clusters C_x and C_y . Since clusters do not have pre-defined labels, data from one class may be assigned to arbitrary clusters in different runs. Thus, instead of looking at the predicted label, we measure F1 using the pairwise co-occurrences of true labels.

Baselines. We compare SECLEDS with state-of-the-art open-source partition-based clustering algorithms with k -fixed clusters: a) Streaming: CluStream, StreamKM++; b) Batch: MiniBatch k-means, and c) Offline: BanditPAM. MiniBatch k-means and StreamKM++ are online versions of the k-means algorithm, while CluStream is an adaptive, online-offline algorithm. BanditPAM (v1.0.5) is used as a benchmark for the best achievable clustering on a static dataset. We set $time_window = 1$, $max_micro_clusters = k \cdot p$, $half_life = 0.5$ for CluStream; $chunk_size = 1$, $half_life = 0.5$ for StreamKM++, and $batch_size = 1$, $max_iter = 1$ for Minibatch k-means.

6.5. EMPIRICAL RESULTS

Key findings. In this section, we empirically demonstrate the following results:

1. SECLEDS produces high-quality clusters, regardless of concept drift, stream size n , data dimensionality (d, w), and number of clusters k . SECLEDS shows competitive F1 compared to the best performing baseline (BanditPAM), while reducing the number of required distance computations by 83.7%.
2. SECLEDS outperforms *all baselines* by 138.7% when the stream contains concept drift. SECLEDS outperforms the *best-performing streaming baseline* by 58.2% on Blobs, 33.3% on Sine-curve, and 143.7% on Sine-curve-drifted.

3. SECLEDS-dtw clusters ~5.5h of network traffic in just 8% of the time. Thus, it can handle networks with bandwidths of up to 1.08 GB/s, which is significantly higher than the requirements of a typical enterprise network.
4. SECLEDS-dtw is used to perform intelligent temporal pattern-preserving traffic sampling, which improves the quality of the behavioral analytics that can be performed on the sampled network traffic, as opposed to standard random sampling.

Point vs. sequence clustering. We use Blobs with $k = 10$ on stream sizes $n = (5000, \dots, 50,000)$, and Sine-curve with $k = 3$ on stream sizes $n = (2000, \dots, 30,000)$. For both, we set $p = 5$, $\lambda = 0.1$, $trials = 10$. The mean and standard deviation of the F1 scores and runtimes are given in Figure 6.4. The benchmark (BanditPAM) achieves a mean F1 of 0.95 and 1.0 for Blobs and Sine-curve, respectively.

SECLEDS outperforms both CluStream and StreamKM++ on the Blobs dataset, and additionally outperforms Minibatch k-means on the Sine-curve dataset. Minibatch k-means performs exceptionally well on point clustering, but loses its edge on sequence clustering. This is because the centroids are computed by collapsing temporally-linked dimensions into single values that do not adequately represent the sequences. An improvement in F1 score is observed for CluStream and StreamKM++ on the higher dimensional Sine-curve dataset because of fewer clusters ($k = 10$ vs. $k = 3$).

We also compare the effect of Euclidean and dynamic time warping distance on the Sine-curve dataset. Although, they both produce equivalent results, it must be noted that Euclidean distance only works with fixed-length sequences. An example of SECLEDS-dtw on clustering bivariate sequences $d = 2$, $w = (\text{min}:15, \text{max}:121)$ from *UJI Pen Characters* [35] is given in the appendix 6.9.4.

Initialization quality. Stream clustering algorithms are greatly impacted by the quality of cluster initialization. To test this, we compare SECLEDS against SECLEDS-rand (initialized with randomly selected medoids from the initial batch \mathbb{B}). Evidently, the clusters take a long time to converge, regardless of the stream size. The cumulative F1 score over time for these configurations is given in the appendix 6.9.2, showing that although the impact of poor initialization is reduced over time, SECLEDS-rand does not completely recover from it. Thus, the distance-based non-uniform sampling strategy proves to be extremely helpful in initializing good clusters.

Clustering with Concept drift. We use Sine-curve-drifted with $k = 3$, $p = 5$, $\lambda = 0.1$, $trials = 10$ on stream sizes $n = (2000, \dots, 30,000)$. SECLEDS outperforms *all baselines* by 138.7%, and outperforms the *best-performing streaming baseline* by 143.7%, on average. BanditPAM no longer serves as a benchmark because it only has a static view of the data, *i.e.*, it does not distinguish between class distributions at $T = t_x$ and $T = t_{x+1}$. Both SECLEDS and CluStream maintain their F1 scores with concept drift, but SECLEDS is 161.8% better than CluStream. StreamKM++ and Minibatch k-means observe a significant reduction in their performance. We hypothesize that it might be due to the lack of exponential decay in k-means, which limits the movement of the centroids towards

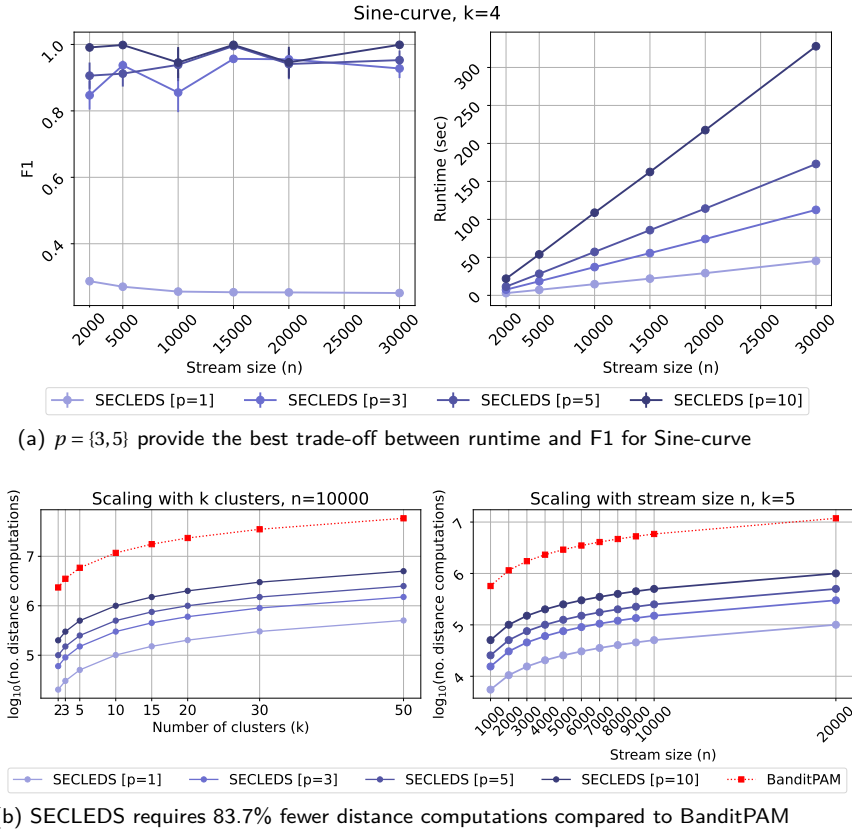


Figure 6.5: Scaling with n , k , and p : (a) Empirical results. (b) Theoretical estimates.

newer data. This experiment provides strong evidence for SECLEDS' ability to handle concept drift with only k -fixed clusters.

Runtime analysis. StreamKM++ and Minibatch k -means are among the fastest clustering algorithms on all datasets, which is expected since they are based on k -means. CluStream does not scale well for high-dimensional datasets, and is much slower than SECLEDS on sequence clustering. As the stream size n grows, SECLEDS also becomes faster than the high-performance implementation of BanditPAM on both point and sequence clustering. Interestingly, the runtimes of BanditPAM, CluStream, and StreamKM++ seem to be affected by concept drift: Given the same dataset and constant parameters, their runtimes increase approximately twofold when there is drift in the data. We hypothesize that this is a side effect of the sampling strategy used to speed up these algorithms.

Scaling with n , k , and p . We use Sine-curve with $k = 4$, $p = \{1, 3, 5, 10\}$, $\lambda = 0.1$, $trials = 10$ on stream sizes $n = (2,000, \dots, 30,000)$. The mean and standard deviation of the F1 and runtime of SECLEDS are reported in Figure 6.5a. A single medoid per cluster, which is

Table 6.2: Clustering real network traffic: Compared to BanditPAM, SECLEDS requires fewer distance computations, is faster, and has a better cluster quality. SECLEDS-dtw is slower but produces better clusters than SECLEDS. Overall, ~5.5h of network traffic is clustered in under 27 minutes (Bold = best scores).

Algorithm	Stream config.	# Distances ($k = 2$)	Run time ($k = 2$)	F1 ($k = 2$)	F1 ($k = 5$)
BanditPAM	Time-ordered	10.3×10^6	978.03s	0.64	0.38
	Cross-validated		984.8s	0.64	0.38
SECLEDS	Time-ordered	2.1×10^6	629.39s	0.85	0.82
	Cross-validated		631.84s	0.79	0.76
SECLEDS-dtw	Time-ordered	2.1×10^6	1623.05s	0.85	0.88
	Cross-validated		1626.89s	0.81	0.80

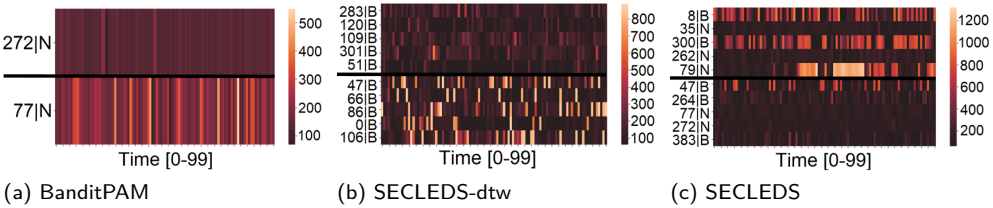


Figure 6.6: Visualizing the medoids of BanditPAM, SECLEDS & SECLEDS-dtw on $k = 2$, $p = 5$. Each row is a medoid. The label denotes curve identifier y_i .

standard for PAM-based algorithms, does poorly in a streaming setting. Intuitively, more medoids help to improve the stability of the clusters, but the relationship is not linear. If p is set too low, the medoids keep jumping to various regions in the dataset, and if it is set too high, the medoids slow down the evolution of the clusters, having an equally detrimental effect on the performance. The optimal value of p with respect to performance and runtime is dataset-dependent. For Sine-curve, $p = \{3, 5\}$ are good alternatives. Additionally, although SECLEDS has multiple medoids per cluster, it performs significantly fewer distance computations compared to the (almost linear) BanditPAM. Figure 6.5b shows this for increasing stream size n , number of clusters k , and number of medoids p , with BanditPAM as reference.

6.6. USE CASE A: INTELLIGENT NETWORK ATTACK SUMMARIZATION VIA SECLEDS

The first use case shows how to use SECLEDS for network traffic summarization while also preserving temporal patterns.

A typical enterprise network has a bandwidth of 25 Mbps², which produces about 17,000 packets *per second*, consuming 2 terabytes of storage space *each day!* To conserve space, the packets are aggregated into network flows (Netflows) at the router level, and only a fraction of them are stored for analysis *i.e.*, 1 in N Netflows are stored. Naturally,

²<https://mosaicnetworkx.com/it-challenges/bits-bytes-understanding-enterprise-network-speeds/>

randomly sampled network traffic does not preserve the temporal patterns in the data, thus limiting the efficacy of traffic profiling and behavioral analytics.

We propose to cluster sequences of Netflows using SECLEDS, and to periodically store a medoid snapshot of each cluster, since they are representative of the network traffic seen so far. This way, each snapshot stores an overview of temporally-linked Netflows. The number of clusters k can be chosen depending on the required granularity of behaviors captured by the clusters. It can also be approximated from an initial batch using, *e.g.*, [36]. The number of medoids p can be configured according to the available storage space, network bandwidth, and the intervals at which to store the medoids.

EXPERIMENTAL SETUP

We demonstrate this use case by generating a stream from the CTU13-9(A) Netflows. The construction and feature engineering processes are given in the appendix 6.9.1. In short, the ground truth provides two classes, *i.e.*, $y_i \in \{\text{botnet}, \text{normal}\}$. Univariate sequences of average bytes per Netflow are used to separate the two classes. We use two configurations for the stream: i) *Time-ordered*: the sequences arrive in order of their timestamps, and ii) *Cross-validated*: we shuffle the stream. We run SECLEDS and SECLEDS-dtw with $k = 2$, $p = 5$, $\text{trials} = 5$, and compare the performance against BanditPAM. The results are given in Table 6.2.

6

6.6.1. CLUSTERING PERFORMANCE

SECLEDS is faster and produces better medoids compared to BanditPAM. Figure 6.6 visualizes the final medoids produced by all three algorithms in the form of temporal heatmaps. Temporal heatmaps have previously been used to visualize temporal similarities in [12]. Each row shows a sequence (medoid), and the colors indicate the magnitude of the curve at each time step. Both medoids of BanditPAM are from the normal class. Although the medoids of SECLEDS-dtw are all from the botnet class, it is evident that they capture distinct behaviors of the malicious hosts. SECLEDS finds medoids from both classes, but the clusters are impure, *i.e.*, the cluster contains medoids from different classes. As such, the cluster quality of SECLEDS-dtw is significantly better than that of SECLEDS.

The results indicate that there are many smaller classes in the network stream, reflecting the various behaviors of benign and infected hosts. This implies that the optimal value of k must be higher than the predefined number of classes. Thus, we allow SECLEDS to disrupt the standard assumption used in clustering literature (*i.e.*, $k = \text{no. of classes}$), and move towards a more realistic setting where there can be more clusters than the number of predefined classes. When k is set to a larger number, the clustering algorithms find smaller, purer data regions, *e.g.*, for $k = 5$, SECLEDS-dtw produces 4 pure clusters (2 normal and 2 botnet), while SECLEDS only produces 1 pure (normal) cluster, see appendix 6.9.3 for their temporal heatmaps. Table 6.2 shows the F1 scores for $k = 5$. Note that although the clustering results for $k = 5$ are better than $k = 2$, the former obtains a lower F1 score as a side-effect of the metric: It penalizes higher number of clusters when less class labels are available by lowering the recall. Regardless, we recommend to over-estimate k in order to sample many regions from the network traffic.

6.6.2. NETWORK BANDWIDTH SUPPORT

The experiments in Table 6.2 show that SECLEDS is faster than SECLEDS-dtw, as expected. SECLEDS clusters the entire stream in 3.1% of the traffic collection time, and SECLEDS-dtw in 8% of the collection time. This experiment provides evidence that SECLEDS can handle much larger network bandwidths.

In summary, SECLEDS-dtw summarizes traffic from networks with a bandwidth of up to 1.08 GB/s in real-time, which is more than sufficient for small to medium enterprises. Specifically, SECLEDS-dtw spends $\frac{1626.89}{213.386} = 0.0076$ seconds on average to cluster a single sequence of length $w = 100$. Thus, SECLEDS-dtw can cluster 131.58 sequences per second. Assuming that the sequence windows w are non-overlapping over the traffic stream, SECLEDS-dtw can process 13,158 individual Netflows per second. The CTU13-9 dataset is composed of 115,415,321 packets aggregated into 2,087,509 Netflows. Assuming uniform distribution, each Netflow contains ~ 55.2 packets. SECLEDS-dtw can process 726,315.79 packets per second. Given that each network packet is about 1500 bytes, this makes a total of 1.089 Gigabytes per second. Similarly, SECLEDS summarizes network bandwidths of up to 2.79 GB/s in real-time.

6.7. USE CASE B: MALWARE BEHAVIORAL PROFILING VIA SECLEDS

The second use case shows how to use SECLEDS to build behavioral profiles of malware-infected hosts in (semi-) real-time.

We evaluate the quality of temporal pattern-preserving traffic sampling (via SECLEDS) for a downstream behavioral analytics task – automated capability assessment of malware infected hosts. Specifically, we utilize MalPaCA (described in Chapter 5) to construct behavioral profiles of bot-infected hosts through their sampled network traffic. To this end, we show that the quality of the behavioral profiles varies dramatically depending on the sampling method, *i.e.*, temporal pattern-preserving sampling vs. standard random sampling.

BACKGROUND AND PROPOSAL

MalPaCA [12] automatically discovers behaviors within malware’s network traffic and utilizes this information to create their behavioral profiles. These behavioral profiles provide insights into the observed operational capabilities of malware, which when used in conjunction with malware family labels, provide more insights into the consequences of a malware infection to both consumers and malware analysts. MalPaCA utilizes DTW and ngrams to measure distances between (sequential) *network connections*³, and clusters them using HDBScan, which is an offline, density-based agglomerative clustering algorithm. Each cluster is hand-labeled with the concept (behavior) it captures (either via data analysis or OSINT). As such, the cluster labels collectively serve as an inventory of the operational capabilities observed in the dataset. Each malware sample is then assigned a behavioral profile based on the cluster membership of its network connections.

The strength of MalPaCA lies in its use of HDBScan to create arbitrary number of clusters for the unique concepts in the dataset. However, it is limited to offline settings

³A network connection is defined as packets/Netflows transmitted between a (source,destination) IP pair.

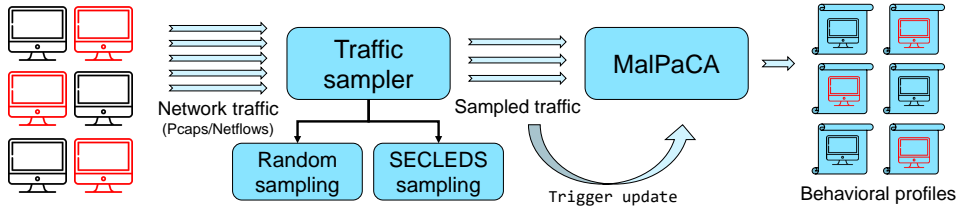


Figure 6.7: Embedding MalPaCA [12] in a traffic sampling framework (random/SECLEDS) to generate real-time behavioral profiles of bot-infected hosts.

– MalPaCA loads the entire dataset in memory to produce behavioral profiles, which is infeasible for larger data streams. Meanwhile, malware behavior is known to evolve over time, creating the need to continuously monitor the network traffic for any behavioral updates. One idea is to replace HDBScan with SECLEDS to enable real-time network connection clustering. However, MalPaCA would be limited to discovering only a predefined k number of concepts, which is unreasonable for a behavior discovery application.

We propose to combine the powers of SECLEDS and MalPaCA by embedding MalPaCA in a traffic sampling framework. The idea is that network connections are sampled in real-time, which are incrementally clustered by MalPaCA to perform (semi-) real-time behavioral profiling. We compare the impact of SECLEDS sampling vs. random sampling on the generated behavioral profiles. Figure 6.7 shows the experimental setting. As new traffic is sampled, MalPaCA re-clusters the network connections using a cached model, and updates the behavioral profiles.

6

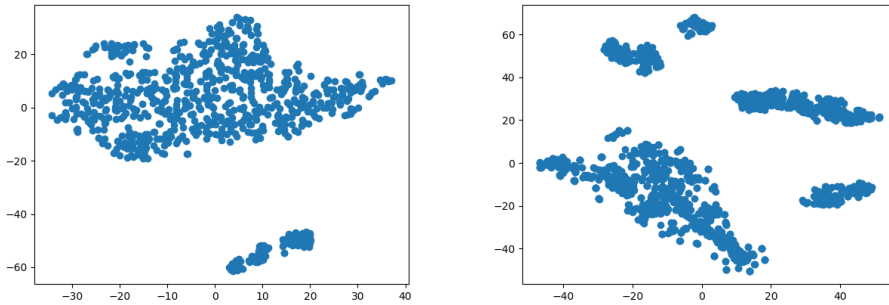
EXPERIMENTAL SETUP

We utilize the stream generated from the CTU13-9(B) Netflows, and create behavioral profiles for the hosts in CTU13-9. We follow the feature selection of MalPaCA – the multivariate Netflow sequences represent $\langle total\ bytes, inter\ arrival\ time, source\ port, destination\ port \rangle$.

In the SECLEDS sampling method, we set $k = 5$ and $p = 5$ to cluster the multivariate sequences since these parameters provided the best results in Section 6.6. As the stream is being clustered, we store the unique medoids every 5 seconds. Overall, we sample 123 medoids, which translates to 12,300 Netflows. For a fair comparison, we also randomly sample 12,300 Netflows from the data stream to demonstrate the effect of random sampling on the behavioral profiles. This reflects a sampling rate of 1 in 12 Netflows.

The sampled Netflows are incrementally given to MalPaCA. MalPaCA keeps track of the network connections represented by the Netflows. Each connection is broken down into lengths of 10 Netflows⁴, which are used to identify the numerous behaviors exhibited by a host. For random sampling, MalPaCA tracks 3215 network connections, which result in 791 usable sequences of length 10. For SECLEDS sampling, MalPaCA tracks 39 network connections, which result in 1191 usable sequences of length 10. Note how SECLEDS sampling creates more usable sequences from fewer connections, while the ran-

⁴In contrast to MalPaCA's default connection length of 20, we select 10 due to data scarcity.



(a) t-SNE plot for randomly sampled Netflows (b) t-SNE plot for SECLEDS sampled Netflows

Figure 6.8: The distribution of the sampled Netflows (projected in two dimensions via t-SNE). The clusters in randomly sampled Netflows are easier to separate compared to those in SECLEDS sampling.

Table 6.3: Statistical summary of the Netflow sequences that are sampled via random/SECLEDS sampling, and clustered using MalPaCA.

Sampling configuration	Stream size (w=100)	Sampled Netflows	MalPaCA connections (w=10)	Clusters	Silhouette index	Average cluster size
Random	1418	12,300	791	2	0.4	385.5
SECLEDS	1418	12,300	1191	6	0.15	155.3

domly sampled Netflows are often insufficient to create adequately long connections. This is a direct side-effect of the sampling method – SECLEDS sampling stores temporally related Netflows, while random sampling stores randomly selected Netflows from several different hosts.

The connections (with $w = 10$) created from the sampled Netflows are clustered by MalPaCA to generate the behavioral profiles for 10 malicious and 3 benign hosts⁵. Table 6.3 shows the number of sequences clustered by MalPaCA for both configurations.

We perform a grid search on a validation dataset (roughly 5% of the input dataset) to estimate the optimal HDBSCAN parameters for both random and SECLEDS sampling. We select $min_cluster_size = 25$ and $min_samples = 5$ as these parameters provide the best possible Silhouette index of 0.4 and 0.15 for random and SECLEDS sampling, respectively. The Silhouette index for randomly sampled Netflows is higher because the sampling creates a distribution that is easier to separate compared to SECLEDS sampling (see Figure 6.8 for the data distribution of both configurations). However, as we show later, the Silhouette index is not an accurate representation of cluster quality for sequential datasets.

⁵Instead of 16 hosts, we cluster only 13. The sampling process loses connections belonging to 3 benign hosts.

Table 6.4: Behavior inventory derived from the MalPaCA clusters for random/SECLEDS sampled Netflows.

Cluster	Behavior	Cluster purity
Random Sampling		
c1	Potential DNS C&C	Mixed
c2	HTTP(s) Virut, Google requests	Mixed
SECLEDS Sampling		
c1	Internal C&C	Botnet
c2	Spam, Click Fraud, Google requests	Mixed
c3	HTTP(s) Virut, Google requests	Mixed
c4	DNS C&C - periodic requests	Mixed (Mostly botnet)
c5	DNS C&C - increasingly fast requests	Botnet
c6	DNS C&C - fast and light requests	Mixed (Mostly normal)

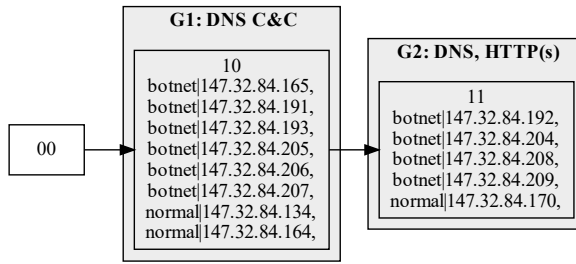
6.7.1. BEHAVIORAL PROFILING VIA RANDOM SAMPLING

MalPaCA discovers 2 clusters from randomly sampled Netflows. Both of the clusters contain a mix of botnet and normal connections (see Table 6.4 for the behavior inventory). By investigating the temporal heatmaps and collecting OSINT regarding the involved IP addresses, we observe that the first cluster captures DNS traffic to server 147.32.80.9. Lagraa *et al.* [37] classify it as botnet traffic. However, it is unclear why the benign hosts also contact this server. The second cluster captures HTTP(s) traffic. The infected hosts utilize HTTPs, while the benign ones utilize HTTP. The IP addresses suggest that the malicious traffic is associated to a potential Virut malware infection and several web attacks (port scan and click fraud). The normal traffic captures requests to benign services, such as Google servers.

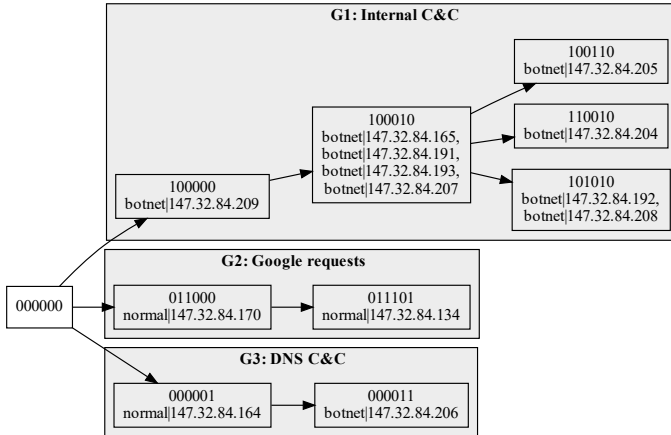
Figure 6.9a shows the directed acyclic graph (DAG) that represents the relationship between the behavioral profiles of the 13 hosts. While we observe some behavioral differences between infected and benign hosts, the profiles are too coarse to meaningfully separate the two host types. We observe two meta-behavioral groups. The first group contains 6 malicious hosts and 2 benign hosts. The second group contains 4 malicious hosts and 1 benign host. The difference between these two groups is as follows: The hosts in the first group contact the suspected (botnet) DNS server. The hosts in the second group, in addition to contacting the DNS sever, also utilize HTTP(s) to conduct web attacks and generate requests to benign web services.

6.7.2. BEHAVIORAL PROFILING VIA SECLEDS SAMPLING

MalPaCA discovers 6 clusters using the temporal pattern-preserving sampling. 2 clusters contain only botnet traffic, while the other 4 contain a mix of botnet and normal traffic (see Table 6.4 for the behavior inventory). By investigating the temporal heatmaps and collecting OSINT regarding the involved IP addresses, we were able to assign labels to all 6 clusters. Cluster 1 contains only botnet traffic, capturing requests to a C&C server controlled by the CTU13 data collectors, *i.e.*, 212.117.171.138. There are requests to 4 other IP addresses that have not been involved in any major attacks, to the best of our knowledge. However, because of the behavioral similarity with the known C&C server, we assume they also correspond to C&C traffic. Similarly, cluster 5 also contains botnet



(a) Behavioral profiles generated from random sampling



(b) Behavioral profiles generated from SECLEDS sampling

Figure 6.9: The DAGs show the hierarchical relationships between the normal and infected hosts based on their behavioral profiles emerging from the two sampling configurations.

traffic. The IP address 147.32.80.9 has previously been speculated to be linked to a botnet. The traffic captured in this cluster makes several DNS requests to this server. While there is a uniform mix of botnet and normal traffic in cluster 2, the botnet traffic connects to 74.117.116.126 and 92.43.56.151, which are linked to browser hijacking and click fraud services. The normal traffic, although sending requests to Google servers, seem behaviorally very similar to the botnet traffic. Cluster 3, in contrast, contains HTTP(s) requests to an IP that has been involved in a Virut infection. Clusters 4 and 6 contain a mix of botnet and normal traffic to the suspected (botnet) DNS server – cluster 4 predominantly contains botnet traffic with periodic requests, and cluster 6 predominantly contains benign traffic with fast requests (and small payloads).

Figure 6.9b shows the DAG that represents the relationship between the behavioral profiles of the 13 hosts. The first observation we make is that although some of the clusters are mixed, we can distinguish infected and normal hosts based on their behavioral profiles with 100% accuracy, *i.e.*, 6 cluster membership strings⁶ are associated to botnet infected hosts, and 3 are associated to normal hosts. This implies that the collective clus-

⁶A binary string encoding the collective cluster membership of a host's connections, see Chapter 5 for details.

ter membership of a host can reliably indicate if it is infected. The second observation is regarding the hierarchical relationship between the behavioral profiles. There is an intrinsic relationship among malicious (resp. benign) profiles – there is an overlap in the behavior of the various malicious (resp. benign) hosts. For instance, the malicious hosts almost always contact the internal C&C server in addition to exhibiting other behaviors. Similarly, the benign hosts almost always send benign requests to similar services in addition to exhibiting other behaviors. The only overlap between benign and malicious profiles happens at the bottom of the graph, where both hosts (*i.e.*, 147 . 32 . 84 . 164 and 147 . 32 . 84 . 206) send DNS requests to the suspected (botnet) server.

Overall, we observe three meta-behavioral groups. The first group contains 9 infected hosts that contact the internal C&C server and the suspected DNS server, and are involved in click fraud, spam, and a *Virut* infection. The second group contains 2 normal hosts that send benign requests to services like Google and Alexa. The third group contains one infected and one normal host that contact the suspected DNS server with varying time-delays between requests.

6.7.3. DISCUSSION

Keeping everything constant, the only difference between SECLEDS sampling and random sampling is the preservation of the temporal order among Netflows. We observe that the DAG produced by SECLEDS sampling is significantly more insightful than the DAG produced by random sampling. This use case provides concrete evidence supporting the use of temporal pattern-preserving sampling for real-time behavior profiling. We show that SECLEDS sampling leads to more detailed behavioral profiles compared to random sampling, while incurring minimal computational overhead. Even when processing Netflows without access to individual packet headers (as was the case in Chapter 5), SECLEDS sampling enables MalPaCA to produce more granular and specific behavioral profiles, which can be used to profile benign and infected hosts. This is an important finding since it is generally believed that aggregating packet captures into Netflows loses too much behavioral information. We show that preserving the temporal order of Netflows may still be sufficient to create relatively detailed behavioral profiles. In contrast, when already aggregated packet captures, *i.e.*, Netflows, are further randomly sampled, the behavioral information becomes so abstract that MalPaCA is unable to differentiate between malicious and benign behaviors. Furthermore, we show MalPaCA's ability to discover interesting behaviors using only 10 Netflows, which further demonstrates that useful insights can be derived from fewer data if they are temporally related.

6.8. CONCLUSIONS

We propose an interpretable sequence clustering algorithm, SECLEDS, which is a streaming version of *k*-medoids with constant memory footprint. SECLEDS uses a combination of multiple medoids per cluster and a medoid voting scheme to create *k*-clusters that evolve with evolving data streams. We also propose an explanation method for explaining the clusters in streaming settings. Testing on several real and synthetic datasets and comparing against state-of-the-art baselines, we demonstrate that i) SECLEDS achieves competitive F1 score compared to the benchmark (BanditPAM) on streams without con-

cept drift; ii) SECLEDS outperforms all baselines by 138.7% on streams with concept drift, and iii) SECLEDS reduces the number of required distance computations by 83.7% compared to the benchmark, making it faster than BanditPAM and CluStream for several clustering tasks.

We also show how SECLEDS can be used as an intelligent temporal pattern-preserving sampling technique capable of supporting high-bandwidth network streams of up to 1.08 GB/s using the expensive dynamic time warping distance. We utilize SECLEDS to perform real-time capability assessment of bot-infected hosts, and show that the traffic sampled by SECLEDS creates more comprehensive behavioral profiles compared to the standard (random sampling) approach. These results reinforce the importance of designing lightweight medoid-based stream clustering algorithms.

Acknowledgments. We thank Dr. Christian A. Hammerschmidt for providing an initial implementation of the k-medoids algorithm, and Ruben te Wierik, Silviu Fucarev, and Rami Al-Obaidi for their contributions to the SECLEDS algorithm.

6.9. SUPPLEMENTARY MATERIAL

6.9.1. DATASET GENERATION PROCESS

Sine-curve. The parameters used to create the 4 sine curves are: $frequency = \{(0.1, 0.12), (0.2, 0.22), (0.4, 0.42), (0.6, 0.62)\}$, $error = \{0.2, 0.4, 0.7, 0.1\}$, and $phase = \{5, 12, -10, -20\}$. For each curve $i \in \{1, \dots, k\}$: (i) The phase value is $phase_i$; (ii) a random frequency is selected from $frequency_i$ range, and (iii) a random error of factor $error_i$ is added at every time step of the curve. Figure 6.10 shows the 2 dimensional representation of the dataset via a t-SNE plot and temporal heatmap. 5 randomly sampled sequences from the 4 classes are shown in the heatmap. The x-axis represents time, thus each row is a sequence. The colors show the magnitude of the curve at each time step. Temporal heatmaps have previously been used to visualize temporal similarities in [12].

CTU13-9(A). The network traffic is divided into several sub-streams based on the source IP address. There are 16 sub-streams, each collecting Netflows for one of the 16 hosts in the dataset (10 infected with Neris botnet, 6 benign). We assume that the Netflows arrive in sequences of length $w = 100$, which are constructed by sliding a window w over each of the sub-streams with $step_size = 1$. This results in $n = 213,386$ Netflow sequences. Using the two natural classes in the dataset, we follow [30] and label the sequences coming from infected sub-streams as *botnet*, while the others as *normal*.

A preliminary feature engineering is conducted to find the Netflow feature(s) that differentiates the two classes. There is extensive research studying the impact of various features in traffic classification. For simplicity, we consider a single feature at a time. Searching for better feature combinations is left as future work. We visualize the class separation obtained by i) Netflow duration, ii) total bytes, iii) total packets, iv) inter-arrival time between Netflows, v) average bytes per Netflow, and vi) average packets per Netflow. We select average bytes to separate the two classes. Figure 6.10 shows a t-SNE plot of the sequences. The unclear class separation emphasizes just how challenging the clustering task is.

6.9.2. CLUSTER INITIALIZATION QUALITY

Figure 6.11 shows the F1 scores over time for SECLEDS and SECLEDS-rand (randomly initialized medoids) for two stream sizes. It is evident that poorly initialized clusters are unable to perform optimally. Since the medoids in a single cluster are blind to each other, once poorly initialized, the clusters never converge. Further investigation is required to make the medoids aware of each other since the additional distance computations will increase the complexity of SECLEDS.

6.9.3. NETWORK TRAFFIC SAMPLING WITH SECLEDS

The CTU13-9(A) dataset contains two big classes corresponding to the normal and botnet Netflows. However, the experiments with $k = 2$ indicate that there are several smaller regions in the data distribution, reflecting to the various behaviors of the normal and bot-infected hosts. We demonstrate this by setting $k = 5$. Similar to the $k = 2$ case, BanditPAM finds all clusters in the normal class. Figure 6.12 shows the final medoids obtained by SECLEDS-dtw and SECLEDS. It shows that SECLEDS-dtw finds four fully pure clusters, *i.e.*, two in the botnet class and two in the normal class. SECLEDS (with

6

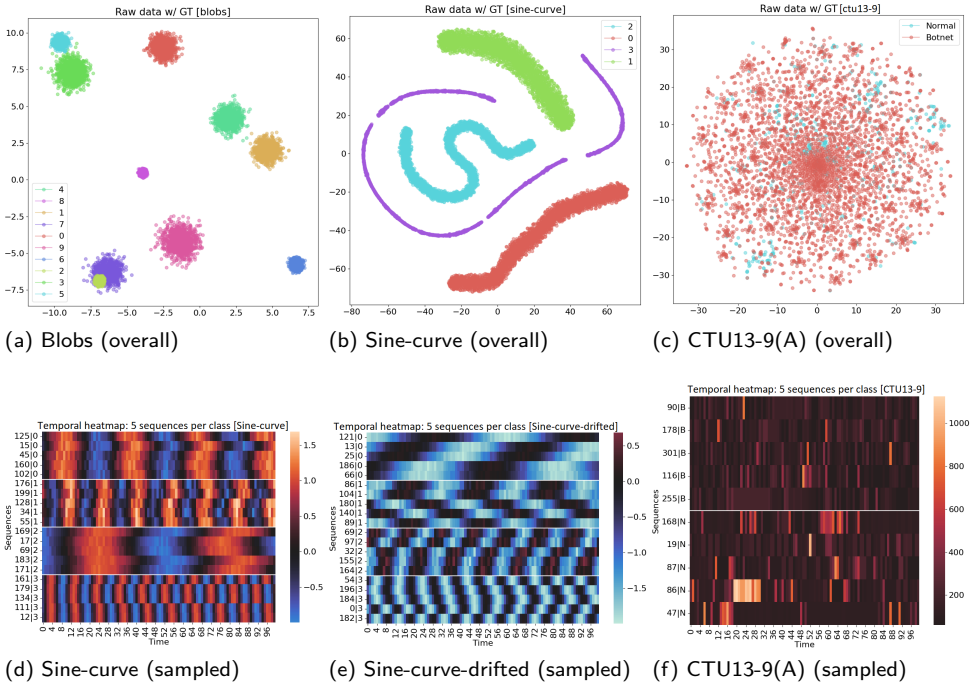


Figure 6.10: Experimental datasets: (a) Scatter plot showing the Blobs dataset ($k = 10$). (b) t-SNE plot showing a 2D representation of the Sine-curve dataset ($k = 4$). (c) t-SNE plot showing a 2D representation of the CTU13-9(A) dataset ($k = 2$). (d) Temporal heatmap showing 20 randomly sampled sequences from Sine-curve. (e) Temporal heatmap showing 20 randomly sampled sequences from Sine-curve-drifted. (f) Temporal heatmap showing 10 randomly sampled sequences from CTU13-9(A).

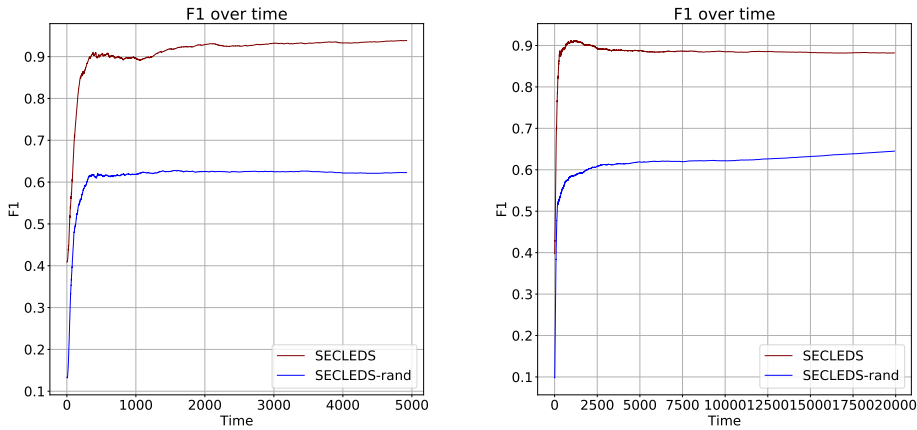
(a) Blobs, $k = 10; p = 5; n = 5000$ (b) Blobs, $k = 10; p = 5; n = 20,000$

Figure 6.11: F1 score over time for SECLEDS and SECLEDS-rand. Randomly initialized clusters perform sub-optimally regardless of stream size.

Euclidean distance) struggles to find more than one pure cluster. This experiment provides evidence for alignment-based distance measures, such as dynamic time warping, to have superior performance than Euclidean distance for sequence clustering tasks.

6.9.4. CLUSTERING HANDWRITTEN CHARACTER STROKES

Despite the growing interest in sequential features over the years, one of the biggest bottlenecks in clustering sequences is the choice of an apt distance measure. Euclidean distance is by far the most popular metric used for clustering tasks. However, Euclidean distance does not perform accurately for out-of-sync sequential data, which is a common property of sequences. Although expensive, alignment-based distances, such as dynamic time warping, have instead been shown to be more accurate. Another benefit of dynamic time warping is that it can handle sequences of arbitrary lengths. We demonstrate this by clustering a dataset of handwritten characters from the UCI machine learning repository (Dua, 2017). The *UJI Pen Characters* dataset contains 467 bivariate sequences from 13 classes, *i.e.*, $\{C, O, S, U, V, W, 1, 2, 3, 5, 6, 8, 9\}$. The length of the shortest character stroke is 15, while that of the longest stroke is 121. Figure 6.13 shows the t-SNE plot of the dataset.

SECLEDS-dtw is the only algorithm that can cluster this dataset straight out-of-the-box. The other baselines use Euclidean distance, which do not support arbitrary-length sequences. Figure 6.13 also shows the 5-fold cross-validated F1 score over time for clustering the character strokes. BanditPAM is used as a reference benchmark, which requires the pairwise distance matrix of all sequences to cluster the dataset. Note that such a distance matrix is not available in a streaming setting, since it requires the entire dataset to be stored in memory. Additionally, we compare the performance with a

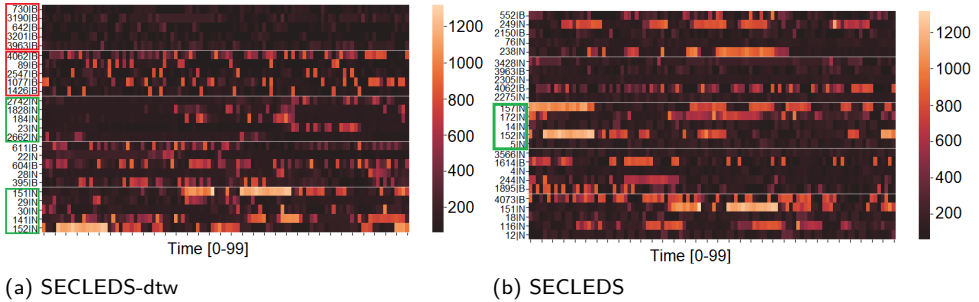


Figure 6.12: Visualizing the medoids of SECLEDS-dtw & SECLEDS on $k = 5$, $p = 5$. Each row is a medoid. The label shows curve identifier $|y_i$. The pure clusters are highlighted: green for normal and red for botnet class.

randomly initialized SECLEDS, referred to as SECLEDS-rand-dtw. BanditPAM obtains the highest achievable F1 score of 0.294 since very few classes are properly separated. SECLEDS-dtw obtains a comparable F1 score of 0.285, while SECLEDS-rand-dtw obtains a score of 0.255. This experiment shows that dynamic time warping is indeed a more suitable distance measure for sequence clustering.

6

6.9.5. CLUSTER EXPLANATIONS FOR MULTIVARIATE NETWORK TRAFFIC

We borrow the features from MalPaCA [12], and cluster multivariate network traffic from CTU13-9(B) using SECLEDS-dtw. The features are sequences of average bytes per Netflow (bytes), inter-arrival times of the Netflows (iat), source ports (srcport), and destination ports (dstport). Each feature is assigned equal weight for clustering. We set $k = 2$ (for the default botnet and normal labels), and $p = 5$ medoids per cluster. Figure 6.14 shows the local and global explanations for the clusters obtained from SECLEDS-dtw. Overall, the explanations show that the two clusters capture sufficiently different concepts. The global explanations show that two clusters clearly represent different distributions of the source port feature, while there is partial overlap in the distributions of average bytes, inter-arrival time, and destination port numbers. The local explanation for each cluster further breaks down the data distribution for each feature. We also plot the feature-wise time-series of the most centrally located medoid. We observe clear differences between the two central medoids in terms of the average bytes, source and destination ports. The inter-arrival time has slight behavioral differences, though it is unclear whether these differences translate to a meaningful concept.

REFERENCES

- [1] A. Nadeem and S. Verwer, “SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, 2022.
- [2] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. d. Carvalho, and J. Gama, “Data stream clustering: A survey”, *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, pp. 1–31, 2013.

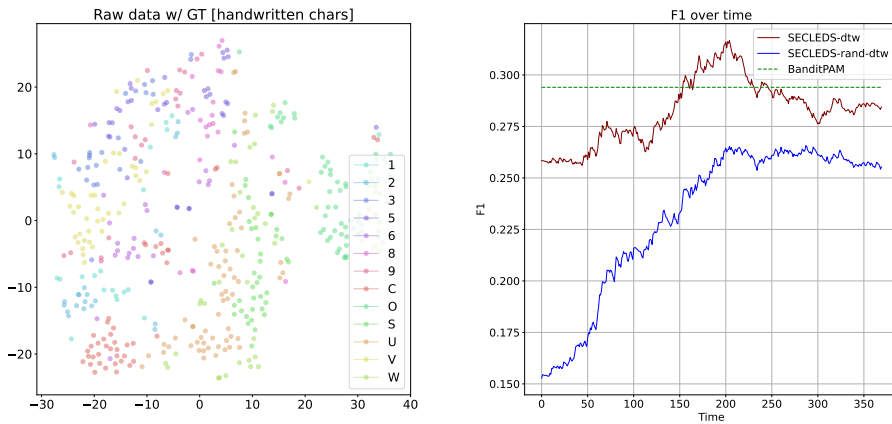
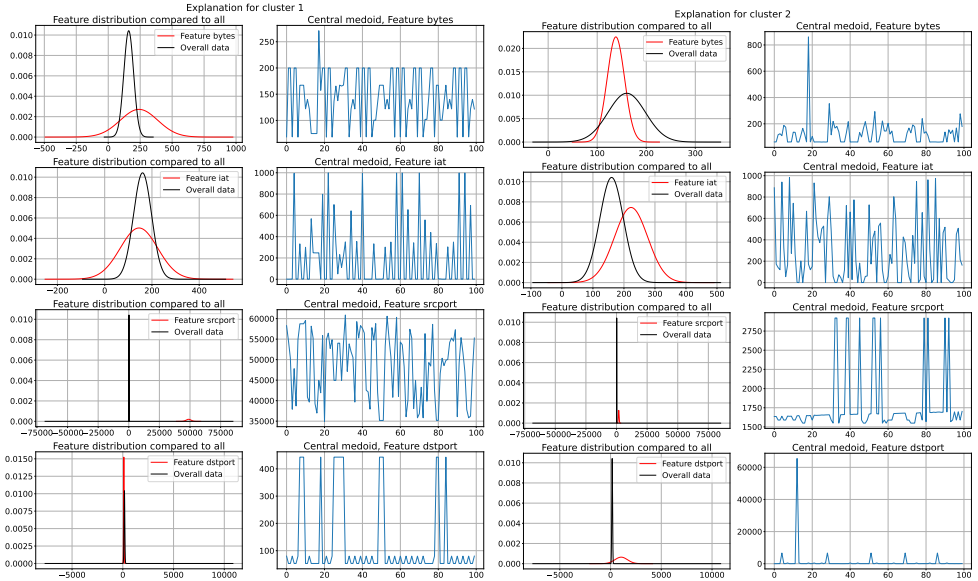


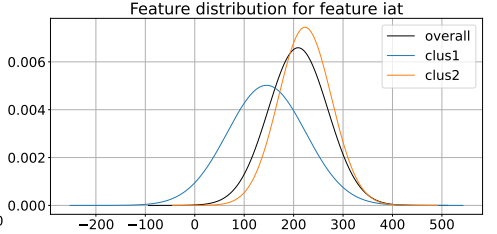
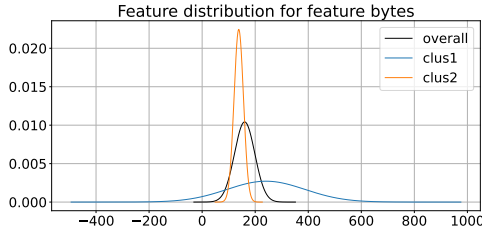
Figure 6.13: Clustering the *UJI Pen Characters* dataset: (Left) t-SNE plot showing the 2D representation of the character strokes. (Right) F1 score over time for BanditPAM, SECLEDS-dtw and SECLEDS-rand-dtw. SECLEDS-dtw has comparable F1 score to BanditPAM, demonstrating that dynamic time warping is a suitable distance measure for sequence clustering in a streaming setting.

- [3] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, “A framework for clustering evolving data streams”, in *VLDB*, Elsevier, 2003, pp. 81–92.
- [4] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, “StreamKM++: A clustering algorithm for data streams”, *JEA*, vol. 17, pp. 2–1, 2012.
- [5] F. Cao, M. Estert, W. Qian, and A. Zhou, “Density-based clustering over an evolving data stream with noise”, in *SDM*, SIAM, 2006, pp. 328–339.
- [6] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: A new data clustering algorithm and its applications”, *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.
- [7] M. K. Islam, M. M. Ahmed, and K. Z. Zamli, “A buffer-based online clustering for evolving data stream”, *Information Sciences*, vol. 489, pp. 113–135, 2019.
- [8] C. Fahy and S. Yang, “Finding and tracking multi-density clusters in online dynamic data streams”, *IEEE Transactions on Big Data*, 2019.
- [9] D. Sculley, “Web-scale k-means clustering”, in *WWW*, 2010, pp. 1177–1178.
- [10] V. Boeva and C. Nordahl, “Modeling Evolving User Behavior via Sequential Clustering”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2019, pp. 12–20.
- [11] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, “Alert-driven Attack Graph Generation using S-PDFA”, *IEEE Transactions on Dependable and Secure Computing*, 2021.



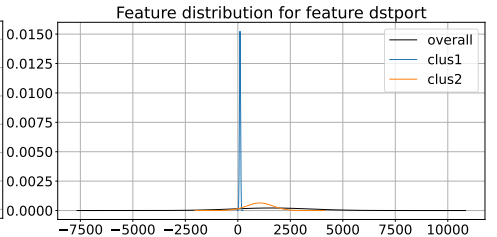
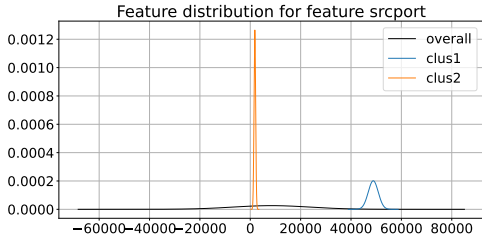
(a) Local explanation - cluster 1

(b) Local explanation - cluster 2



(c) Global explanation - feature: average bytes

(d) Global explanation - feature: inter arrival time



(e) Global explanation - feature: source port

(f) Global explanation - feature: destination port

Figure 6.14: The local and global explanations generated by clustering multivariate network traffic in $k = 2$ clusters with $p = 5$ medoids each. The local cluster explanation (a-b) shows, for each feature, the feature distribution of the medoids within the cluster compared to the distribution of all the curves, and the single most representative mediod (centrally located Netflow sequence) in that cluster. The global explanations (c-f) show, for each feature, the feature distributions of the 2 clusters compared to the distribution of all the curves.

- [12] A. Nadeem, C. Hammerschmidt, C. H. Gañán, and S. Verwer, “Beyond labeling: Using clustering to build network behavioral profiles of malware families”, in *Malware Analysis Using Artificial Intelligence and Deep Learning*, Springer, 2021, pp. 381–409.
- [13] J. Guo, G. Liu, Y. Zuo, and J. Wu, “Learning sequential behavior representations for fraud detection”, in *ICDM, IEEE*, 2018, pp. 127–136.
- [14] D. J. Cook, N. C. Krishnan, and P. Rashidi, “Activity discovery and activity recognition: A new partnership”, *IEEE transactions on cybernetics*, vol. 43, no. 3, pp. 820–828, 2013.
- [15] A. Khaleghi, D. Ryabko, J. Mary, and P. Preux, “Online Clustering of Processes”, in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, vol. 22, PMLR, Apr. 2012, pp. 601–609.
- [16] T. Wang, Q. Li, D. J. Bucci, Y. Liang, B. Chen, and P. K. Varshney, “K-medoids clustering of data sequences with composite distributions”, *IEEE Transactions on Signal Processing*, vol. 67, no. 8, pp. 2093–2106, 2019.
- [17] A. V. Ushakov and I. Vasilyev, “Near-optimal large-scale k-medoids clustering”, *Information Sciences*, vol. 545, pp. 344–362, 2021.
- [18] E. Schubert and P. J. Rousseeuw, “Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms”, in *SISAP*, Springer, 2019, pp. 171–187.
- [19] M. Tiwari, M. J. Zhang, J. Mayclin, S. Thrun, C. Piech, and I. Shomorony, “Bandit-PAM: Almost linear time k-medoids clustering via multi-armed bandits”, *NeurIPS*, vol. 33, pp. 10 211–10 222, 2020.
- [20] Y. Wang, L. Chen, and J.-P. Mei, “Incremental fuzzy clustering with multiple medoids for large data”, *IEEE transactions on fuzzy systems*, vol. 22, no. 6, pp. 1557–1568, 2014.
- [21] N. Lu, G. Zhang, and J. Lu, “Concept drift detection via competence models”, *Artificial Intelligence*, vol. 209, pp. 11–28, 2014.
- [22] I. Žliobaitė, M. Pechenizkiy, and J. Gama, “An overview of concept drift applications”, *Big data analysis: new algorithms for a new society*, pp. 91–114, 2016.
- [23] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under concept drift: A review”, *TKDE*, vol. 31, no. 12, pp. 2346–2363, 2018.
- [24] R. S. M. Barros and S. G. T. C. Santos, “A large-scale comparison of concept drift detectors”, *Information Sciences*, vol. 451, pp. 348–370, 2018.
- [25] R. Hyde, P. Angelov, and A. R. MacKenzie, “Fully online clustering of evolving data streams into arbitrarily shaped clusters”, *Information Sciences*, vol. 382, pp. 96–114, 2017.
- [26] S. Lloyd, “Least squares quantization in PCM”, *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.
- [27] F. Glover, “Future paths for integer programming and links to artificial intelligence”, *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

- [28] X. Vankwikelberge, B. Kang, E. Heiter, and J. Lijffijt, “ExClus: Explainable Clustering on Low-dimensional Data Representations”, *Proceedings of BNAIC/Bene-Learn*, 2021.
- [29] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python”, *Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [30] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods”, *Computers & Security*, vol. 45, pp. 100–123, 2014.
- [31] A. Zubaroglu and V. Atalay, “Data stream clustering: A review”, *Artificial Intelligence Review*, vol. 54, no. 2, pp. 1201–1236, 2021.
- [32] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, “Experimental comparison of representation methods and distance measures for time series data”, *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, 2013.
- [33] D. Guijo-Rubio, A. M. Durán-Rosal, P. A. Gutiérrez, A. Troncoso, and C. Hervás-Martínez, “Time-series clustering based on the characterization of segment typologies”, *IEEE transactions on cybernetics*, vol. 51, no. 11, pp. 5409–5422, 2020.
- [34] C. Manning, P. Raghavan, and H. Schütze, “Introduction to information retrieval”, *Natural Language Engineering*, vol. 16, no. 1, pp. 100–103, 2010.
- [35] D. Dua and C. Graff, *UCI Machine Learning Repository*, 2017. [Online]. Available: <https://archive.ics.uci.edu/datasets>.
- [36] J. de Andrade Silva and E. R. Hruschka, “Extending k-means-based algorithms for evolving data streams with variable number of clusters”, in *ICMLA*, IEEE, vol. 2, 2011, pp. 14–19.
- [37] S. Lagraa, J. François, A. Lahmadi, M. Miner, C. Hammerschmidt, and R. State, “BotGM: Unsupervised graph mining to detect botnets in traffic flows”, in *2017 1st Cyber Security in Networking Conference (CSNet)*, IEEE, 2017, pp. 1–8.

7

CONCLUSIONS, OUTLOOK, AND SOCIETAL RELEVANCE

Understanding the tactics, techniques, and procedures (TTPs) employed by cyber adversaries is crucial for generating effective threat intelligence and detection mechanisms. While the state-of-the-art relies on expert-driven measurement studies to characterize the attacker TTPs, these characterizations of attacker behavior quickly become outdated with the rapidly evolving cyber threat landscape. Data-driven approaches are becoming increasingly important to automate behavior characterization and to reduce the strain on the cyber workforce. However, understanding attacker behavior via cyber data is extremely challenging – cyber data is typically unlabeled, noisy, infrequent, and contains intricate patterns that evolve over time. Effective feature representation addresses these challenges, *i.e.*, sequential features are widely acknowledged for their efficacy in behavior modeling [1]. Although there is a growing interest in the use of sequential features for various learning problems, they are expensive to process, and have limited algorithmic support. Their high dimensionality also makes it difficult to visualize and understand them. The interpretability of sequence learning models requires special consideration – because the goal of these models is to gain insights into attacker behavior, it is counter-productive if the resulting models are black-box (uninterpretable).

This thesis tackles the challenges of the cyber domain by following a two-pronged approach: 1) We develop algorithms that learn from sequential features, infrequent events, and evolving data. 2) We develop interpretable tool-chains that utilize these learning algorithms for modeling adversary behavior. We demonstrate *interpretable unsupervised sequence learning* as a means to extract threat intelligence regarding attacker behavior from cyber data (which is often of suboptimal quality). We present the effectiveness of interpretable sequence learning by discovering attacker strategies from intrusion alerts, and by constructing behavioral profiles of malware.

In contrast to the dominant research paradigm that utilizes out-of-the-box machine learning models for classification, we show in Chapters 3 - 6 that it is possible to utilize machine learning models for extracting insights regarding attacker behavior from

network-induced cyber data. We also show that these models need not be black-box: We develop interpretable models and explanation methods for complex time-series data. We even develop an interpretable real-time sequence clustering algorithm. These models use minimal assumptions on the input data to aid generalizability: Instead of solving toy problems, we use (pseudo-)realistic data to train the machine learning models. In addition to validating unsupervised models using different metrics, we conduct thorough qualitative analyses that often show the limitations of commonly used metrics in capturing interesting behavioral insights.

In summary, we make the following contributions in this dissertation:

- We propose the first comprehensive taxonomy that defines the notion of explainability within cybersecurity. By framing existing literature according to the stakeholders and application objectives, we identify the lack of interpretable models by design, and the exclusion of stakeholders from the ML pipeline. We provide a tutorial that shows the utility of common XAI methods for exposing weaknesses in models, and also identifies potential pitfalls that may steer practitioners towards misleading conclusions.
- We develop powerful interpretable models and explanation techniques that help security practitioners understand sequential data for unsupervised analysis tasks, *i.e.*, we summarize attacker strategies in intrusion alerts, explain the behavioral relationships among malware samples, and explain multivariate sequential clusters in real-time.
- We demonstrate the effectiveness of sequential data for behavior modeling, *i.e.*, we develop sequential clustering models for attacker strategy and malware capability discovery. We also show that sequential features obtain higher quality behavioral clusters compared to commonly used statistical aggregates.
- We develop an interpretable suffix-based probabilistic deterministic finite automaton (S-PDFA) to accentuate infrequent intrusion alerts and to model the semantics of alerts. We show that the S-PDFA is effective at learning sequential patterns, and is learned in under 0.5 seconds.
- We develop SAGE as a pioneering system that learns attack graphs from intrusion alerts. We also develop a web-based dashboard with querying/prioritization capabilities to consolidate the alert-driven attack graphs, and highlight alerts that might require the urgent attention of security practitioners. We show that SAGE compresses over 1.4 million alerts in 401 attack graphs in under 5 minutes. The S-PDFA identifies an average of 1.2 – 1.4 attack variations per attack graph.
- We show for the first time that the alert-driven attack graphs provide threat intelligence regarding attacker strategies. They are useful for identifying strategic differences, scripted attacks, and fingerprintable attempts. They capture the increasing expertise of the attackers, and are used to rank them based on this expertise.
- We develop MalPaCA as a novel tool-chain to automatically create behavioral profiles of malware samples, a task previously dominated by manual approaches. Using an industry-acquired dataset containing 1196 malware samples and 3.6 million

packets, we discover 18 capabilities such as port scans, device broadcasting, and the reuse of C&C servers. A global dendrogram is utilized to identify malware samples that do not adhere to their family labels, showing their noisy and unreliable nature as ground truth.

- We develop SECLEDS, the first real-time interpretable sequence clustering algorithm with support for concept drift. It reduces the required distance computations by 83.7% in order to utilize expensive alignment-based distance measures. A novel medoid voting scheme is introduced to handle concept drift. SECLEDS is evaluated on 3 synthetic and 2 real datasets. It outperforms 4 state-of-the-art clustering algorithms by 138.7% when the stream contains concept drift.
- SECLEDS is utilized as a novel network traffic sampling technique that preserves temporal relationships. We show that SECLEDS supports bandwidths of over 1 GB/s for network traffic summarization. We combine SECLEDS with MalPaCA for (semi-) real-time malware capability assessment. By utilizing SECLEDS-sampled network traffic, MalPaCA creates behavioral profiles from 6 discovered behaviors that distinguish benign and malicious hosts with 100% accuracy.

7.1. ADDRESSING THE CHALLENGES OF CYBER DATA

In this section, we describe our contributions in the machine learning domain. We build sequential learning algorithms that can handle infrequent events (S-PDFA introduced in Chapter 3), and evolving data (SECLEDS introduced in Chapter 6).

7.1.1. S-PDFA FOR INFREQUENT DATA

Software logs, *e.g.*, intrusion alerts, are usually highly imbalanced. The majority of the data typically relates to low-severity events, while only a minority pertains to attacks or failures. Other than conducting anomaly detection, it is very difficult to learn from this data without the risk of discarding infrequent severe events. The task becomes even more challenging when these infrequent events are in the form of sequences.

Learning from infrequent sequences is an open problem. Sequential models, such as Markov chains [2], finite state automata [3], and process mining models [4] are promising solutions. While Markov chains are easier to learn, they do not model the semantic meaning between events, *i.e.*, two identical events leading to a different future will be modeled as the same state in a Markov chain. This is not desirable if the goal is to differentiate between sequences based on their contextual meaning. Process mining models suffer from the same shortcoming. Deterministic automata do not have such limitations.

In the experimental datasets in Chapter 3, we observe that the infrequent events are always at the end of sequences, so by learning a suffix-based probabilistic deterministic finite automaton (S-PDFA), we amplify their importance without disturbing the data distribution. The Alergia merge criteria ensures that states with similar pasts are merged. The Markovian property further ensures that states have unique input transition symbols, *i.e.*, the immediate futures are identical. Furthermore, we discard “sink states” (states that are too infrequent to learn anything from) to avoid arbitrary merges that lead to a lower-quality model. Using the Perplexity metric, we show that the S-PDFA

works better for modeling infrequent events compared to alternative models, such as Markov chains and suffix trees. Because the Markov chain does not model context, it makes vast over-generalizations. The suffix tree and the S-PDFA lie at two opposite ends of the learning spectrum – the suffix tree is a larger model that represents the data as is, while the S-PDFA is a smaller model obtained by making generalizations to merge different states. As such, the S-PDFA is both small and interpretable (see Figure 3.6). It is also effective at modeling the training data and at generalizing to unseen test data. The S-PDFA is learned using Flexfringe in under 0.5 seconds. These characteristics make the S-PDFA a powerful model for mining infrequent patterns in sequential datasets.

LIMITATIONS AND FUTURE WORK

The suffix-based model does not entirely solve the infrequent event modeling problem. For sparse datasets, such as the ones used in Chapter 3, we are still left with several infrequent events (some of which are interesting for further analysis) that form sink states. These states essentially form sub-trees representing infrequent (sub-)sequences that are not used during the learning process. It remains unclear what to do with sink states. In Chapter 3, we still utilize their state identifiers in the attack paths. However, it creates the problem that the state identifiers of the attack paths that originate from sink states do not represent contextually different paths. This makes it difficult to identify the true number of contextually different actions in an attack graph. We are currently exploring the impact of merging sink states with the core S-PDFA model.

The comparison between the attack graphs resulting from the suffix tree and the S-PDFA opens up a fundamental question: When is learning/making generalizations effective, and when is it sufficient to simply show raw data instead? There is probably a threshold of data sparsity after which learning becomes more effective. However, identifying this threshold for different datasets and problems remains an open challenge.

7.1.2. SECLEDS FOR EVOLVING DATA

Sequence clustering is an important task for various domains, such as bioinformatics and cybersecurity. The biggest challenge in clustering sequences is defining a notion of similarity. Numerous distance measures exist to this end, *e.g.*, dynamic time warping (DTW), Fréchet distance, and sequence alignment algorithms. Because sequences can be out of sync, these distance measures are typically computationally expensive, which makes the whole task of sequence clustering time-intensive. This is why sequence clustering is typically never done in a streaming setting, which requires fast-paced distance computations. However, many applications require real-time sequence clustering, *e.g.*, for classifying similar emerging attack campaigns, or categorizing collective anomalies.

In Chapter 6, we present SECLEDS, which is a lightweight (almost linear) algorithm for clustering sequences in streaming settings. SECLEDS is a streaming variant of the popular *k*-medoids algorithm that has commonly been used to cluster complex data types. This is because the clusters are represented by medoids, allowing customized distances to be computed between two arbitrary data instances. This trait of *k*-medoids enables it to operate on non-metric distance measures, *e.g.*, DTW. In contrast, many clustering algorithms, *e.g.*, *k*-means, are required to use metric distances.

SECLEDS can use expensive distance measures in a streaming setting because of a

neat property: In contrast to existing algorithms that use pairwise distances to identify important medoids, SECLEDS estimates a cluster's center of mass using a medoid voting scheme where the medoids close to newly observed data become important (by receiving votes) and outdated medoids are eventually forgotten (using exponential decay). This way, SECLEDS reduces the required number of distance computations by 83.7%, which allows us to use expensive distance measures, like DTW, in a streaming setting.

The medoid voting scheme also enables SECLEDS to handle concept drift, *i.e.*, by allowing the clusters to evolve with an evolving data stream. This is a novel property since existing algorithms typically maintain fixed cluster definitions with arbitrary number of clusters to capture different concepts. In contrast, we show that it is possible to capture different concepts with a fixed number of evolving clusters. We demonstrate that SECLEDS outperforms 4 state-of-the-art distance-based clustering algorithms by 138.7% when the stream contains drift. As the clusters do not have a static definition, we need a real-time explanation method to explain the evolving clusters at any given time. We construct local and global cluster explanations based on the medoids, providing incremental statistical insights into the multivariate sequence clusters as new data arrives (see Figure 6.3).

LIMITATIONS AND FUTURE DIRECTIONS

SECLEDS' clusters are characterized by multiple medoids. This is done to minimize the probability of making mistakes, given a noisy stream. An open question is to estimate the optimal number of medoids for clusters with varying densities. Another challenge is to estimate the cluster density without storing the stream in memory.

The cluster medoids are updated at each iteration to keep up with an evolving data stream. This means the traditional notion of convergence (*i.e.*, no more cluster updates) does not apply to SECLEDS. Although we empirically show that SECLEDS works well regardless of concept drift, stream size, data dimensionality, and the number of clusters, we have not formalized an objective function optimized by the algorithm. This function is different from the traditional k-medoids, which minimizes the intra-cluster distances. Because we update medoids based on the votes, we hypothesize that SECLEDS maximizes the collective votes of each cluster. However, since the votes do not monotonically increase because of exponential decay, the objective function is non-linear. We have left the formalization of the objective function as future work.

Finally, it is unclear how SECLEDS fares with extremely imbalanced datasets, such as those seen in Chapter 3. It may be possible to combine the S-PDFA from Chapter 3 and SECLEDS from Chapter 6 for clustering infrequent patterns in an evolving data stream. As such, the merge criteria of the S-PDFA can be used as a distance measure in SECLEDS. However, this combination is not straightforward since the S-PDFA assumes the knowledge of the future, which is unavailable for streams evolving in real-time. Further research is warranted to combine the two approaches.

7.1.3. SOCIETAL RELEVANCE

A Python implementation¹ of the SECLEDS algorithm is released as open-source. It can be used out-of-the-box for clustering sequences in large data streams. Moreover, the S-

¹SECLEDS: <https://github.com/tudelft-cda-lab/SECLEDS>

PDFFA can be learned using the open-source tool, Flexfringe, for different alert datasets. The S-PDFA model and the SECLEDS algorithm are not limited to the cybersecurity domain, thus making their impact much broader. For SECLEDS, we particularly experiment with sequential datasets outside of the security domain, *e.g.*, sine curves and handwritten character strokes (see Figures 6.10 and 6.13). Moreover, the interpretable nature of the S-PDFA and SECLEDS provides evidence that security solutions do not have to rely on black-box models. As such, these algorithms can be used for behavioral analytics in different domains: SECLEDS can be used to cluster sequences in real-time for, *e.g.*, grouping similar trajectories in human activity recognition [5], and clustering similar user preferences in recommender systems [6]. The S-PDFA can be used to cluster sequences that contain infrequent events for, *e.g.*, modeling failures in automotive safety [7], and modeling anomalous seismic waves in earthquake analysis [8].

7.2. EXPLAINABILITY IN CYBERSECURITY

In this section, we present our findings pertaining to Q1, and reflect on the scientific implications of our systematization of XAI research within cybersecurity:

Q1: *What are the challenges and opportunities for explainable artificial intelligence (XAI) within cybersecurity?*

Chapter 2 investigates the state-of-the-art XAI approaches for cybersecurity problems. We identify 3 stakeholders and 4 objectives of explanations, *i.e.*, *XAI-enabled user assistance* provides decision support to *model users*, *XAI-enabled model verification and explanation verification/robustness* provide decision support to *model designers*, and *XAI-enabled exploitation* provides decision support to *adversaries*.

While learning interpretable models, and explaining complex models using feature importance by themselves are not a new concept, the field of XAI that consolidates them and creates additional explanation methods is a nascent field with numerous open problems, especially within cybersecurity. For each stakeholder, we present opportunities for future research by posing fundamental questions:

- *Model users:* While over half of the reviewed studies develop explanations for model users, they are typically not included in the evaluation process. How can we incorporate model users in explanation evaluation without incurring excessive costs?
- *Model designers:* While model designers have a critical role in ensuring the security of the machine learning pipeline, they are typically ignored in the security literature. To what extent can robust explanation methods help model designers validate machine learning pipelines?
- *Adversaries:* Although adversaries can exploit explanations to further bolster their attacks, there are limited works that investigate the robustness of explanations against confidentiality, integrity, and availability attacks. Is it possible to develop explanations that are useful to model users and designers, but not to adversaries?

Additionally, considering the substantial role of model designers in the security of a machine learning pipeline, we provide a tutorial that shows the utility of commonplace XAI tools (*e.g.*, LIME, SHAP, LEMNA) for discovering weaknesses in their models. With the help of a network attack detection use case, we demonstrate how to discover spurious features and causes of misclassifications. We also highlight the pitfalls of post-hoc explainability in potentially steering practitioners towards misleading conclusions.

7.2.1. LIMITATIONS

It was difficult to discover literature that solves similar problems but does not use standard terminologies. Although we try to limit this by including synonyms of known keywords, we cannot ensure that we do not miss certain topics. There may also be bias in the literature selection introduced by our own familiarity with certain topics. For instance, AI-enabled reverse engineering literature was included primarily because we had been exposed to this area for several years. In addition, we did not add some related works since they did not solve security problems, *e.g.*, model safety literature published in machine learning venues focusing on image and text data.

7.2.2. SOCIETAL RELEVANCE AND FUTURE DIRECTIONS

Overall, we find that there is substantial room for improvement in this field of research. Black-box models are frequently used without understanding their risks, and applications typically apply off-the-shelf XAI methods without considering the stakeholders. It is possible that related areas of research may not be getting attention just because popular research overlooks them. For instance, reverse engineering applications attempting to explain the input data from the model are an overlooked aspect of XAI. We present methods for reverse engineering attacker strategies and malware capabilities from network-induced cyber data in Chapters 3, 5, and 6, respectively.

An important takeaway from the reviewed literature is that most of the research (approximately 74%) utilizes black-box models that are supplemented with post-hoc explanations. It makes sense since trying to obtain high accuracy inevitably makes the pipeline black-box. The problem with black-box models is that they cannot be understood on their own, and there are no guarantees that the post-hoc explanations are faithful to the parent black-box model, or are robust to adversarial attacks. Research on certifiably equivalent robust explanations is severely limited [9]. In contrast, interpretable models can directly be investigated, and do not require additional components to explain their behavior. This is why we recommend interpretable models for building trustworthy and deployable security solutions. Chapters 3 and 6 of this thesis present solutions for adversary behavior analysis by developing interpretable models that explain (complex) high-dimensional sequential data. Seemingly intuitive, it remains challenging to build interpretable models for sequential data. Nonetheless, post-hoc explanations of performant black-box models may guide the search for better interpretable models, *e.g.*, by learning interpretable models from features that are suggested by post-hoc explanations as being the main contributors to high performance.

The XAI literature specifies that effective explanations are tailored to a specific stakeholder [10]. There is a general lack of understanding within the security literature of what it means to create explanations for different stakeholders, *i.e.*, the security liter-

ature under-specifies who the explanations are for, and what is being explained. This is in addition to the existing problem of interpreting the explanations correctly. Even for simple methods, such as feature importance using LIME and local-SHAP, it is not safe to assume that the explainee necessarily knows how to interpret the explanations correctly. There are even cases where post-hoc methods still produce so-called explanations for random models (that are not even trained) [11]. Meanwhile, XAI research is rarely evaluated with human analysts even though it is clear that application-grounded evaluation is the gold standard [12]. User studies are expensive and usually suffer from low response rates, especially in the cybersecurity domain. This is because it is difficult to engage security practitioners in such evaluations when they are already overworked and fatigued with safety-critical tasks. While we also struggle with this challenge in this thesis, we address it by evaluating our tool-chains with carefully selected metrics and thorough qualitative analyses in Chapters 3 – 6. We are currently organizing follow-up user studies to evaluate these tool-chains with security practitioners, see Section 7.3.3.

In terms of the research itself, there is a big overlap in security and machine learning literature when it comes to model robustness. This is an important distinction because the security community is not always exposed to the machine learning literature. Even more importantly, realistic cyber adversaries and offensive security scenarios are rarely discussed in the machine learning literature, most of which is applied on image and text data. Security applications, in contrast, often deal with graphs and sequence data [1]. This means that many of the findings from the machine learning literature, albeit solving a similar problem, do not apply directly to a security context. Finally, it would be naive to assume that XAI research is not interdisciplinary. Yet, very few works collaborate with psychologists and criminologists to create effective explanation methods. This is a missed opportunity since computer scientists typically do not have adequate training in cognitive sciences to foresee and correct potential issues in explanations before deploying them. At the very least, drawing insights from psychology literature may provide clues for quantifying understandability (and developing a metric for interpretability [13]), which remains a very important open problem in XAI.

7

7.3. ATTACKER STRATEGY DISCOVERY

In this section, we present our findings pertaining to attacker strategy discovery, and reflect on the scientific implications of the proposed solutions for society at large.

7.3.1. ALERT-DRIVEN ATTACK GRAPHS

In this section, we answer Q2:

Q2: *How can we learn attack graphs directly from intrusion alerts using interpretable unsupervised sequence learning?*

Traditional attack graphs are created using expert knowledge of network topology and vulnerability reports, which do not provide sufficient incident response capabilities. Learning attack graphs from intrusion alerts is a long-standing problem in the security domain since it is challenging to discover sequential constraints that do not result in a state-space explosion. We solve this problem in Chapter 3 by developing SAGE,

which is an unsupervised, interpretable sequence learning tool-chain that takes intrusion alerts as input, discovers attacker strategies in the data, and outputs them as attack graphs. SAGE utilizes the S-PDFA (Section 7.1.1) to discover attacker strategies in the alert datasets. The S-PDFA forms an integral part of the SAGE pipeline since it is responsible for accentuating infrequent (severe) alerts, and modeling alert context. In addition, the S-PDFA is interpretable, enabling analysts to graphically summarize the entire alert dataset. Essentially, the S-PDFA is a clustering approach that merges similar attacker strategies. Attack graphs are then extracted from the S-PDFA for each objective exploited on the victim hosts in the network. The attack graphs provide actionable information to analysts. For instance, the fraction of attack paths exploiting specific services indicates the level of vulnerability of the target network (see Table 3.4). Moreover, the S-PDFA identifies contextually different ways of exploiting an objective, which provides an estimate for the number of required countermeasures (see Table 3.7).

We evaluate SAGE on 3 different security testing competition datasets (penetrating testing and blue team exercises), and show that the selected S-PDFA accurately models the underlying alerts, and also generalizes to unseen alerts. We also show that SAGE compresses over 1.4 million intrusion alerts in exactly 401 attack graphs in under 5 minutes (without a priori expert knowledge). Instead of investigating thousands of tabular intrusion alerts, security analysts can triage alerts by analyzing attack graphs that show how specific attacks transpired. This addresses the ‘alert fatigue’ problem commonly faced by practitioners [14].

7.3.2. BEHAVIORAL ANALYTICS BY ALERT-DRIVEN ATTACK GRAPHS

In this section, we answer Q3:

Q3: *What kind of threat intelligence can be extracted from alert-driven attack graphs for discovering similarities and differences between attacker strategies?*

Chapter 3 already shows SAGE as an alert-compression tool. In Chapter 4, we take a deep-dive into the alert-driven attack graphs generated by SAGE in order to identify the threat intelligence that they provide. We also build a web-based dashboard to consolidate the alert-driven attack graphs, and provide querying and prioritization capabilities to analysts (see Figure 4.2).

We demonstrate that the attack graphs facilitate the visualization and comparison of attacker strategies pertaining to a specific objective exploited on a victim. Having the strategies of all the attackers that exploit an objective in a single attack graph helps to find common attack paths and unique strategies (see Figure 4.7). Comparing attacker strategies across different attack graphs provides clues regarding the tools used by the attackers. For instance, if the attackers employ identical (or highly similar) strategies to target multiple victim hosts, it likely points to a scripted attack (see Figure 4.9). In contrast, if the attack paths are sufficiently different, or if there are long gaps between attacker actions, then it is likely a manual attack (see Figure 4.8). Additionally, we rank attacker expertise based on the fraction of severe alerts triggered by them (see Table 4.3). We also fingerprint attackers based on the uniqueness of the attack paths they employ (see Table 4.2).

From a defense perspective, the S-PDFA inadvertently captures the increasing expertise of the attackers, *e.g.*, by identifying objectives exploited using the least number of actions (see Figure 4.7). We recommend to flag such paths as critical, requiring immediate remediation. From a training perspective, we use the attack graphs to enumerate all unique paths employed by red teams (see Figure 4.11). These paths help to investigate whether a shortest path exists to exploit an objective, or whether multiple team members follow overlapping paths due to communication problems (see Figure 4.12).

7.3.3. FOLLOW-UP EFFORTS

Modeling assumptions. Since evaluating an unsupervised approach for modeling attacker strategies is difficult, we relearn attack graphs with alternative modeling assumptions and compare against the output of SAGE. In [15], we study the impact of replacing the S-PDFA with a PDFA on the attack graphs. As expected, the PDFA discards many of the infrequent severe events as sink states, which results in larger attack graphs (*i.e.*, due to sink state identifiers). The unexpected benefit is that the attack paths are more spread-out, which has a positive impact on the readability of the graphs. The PDFA is worse than the S-PDFA in terms of interpretability since it no longer learns from contextually different infrequent events (*i.e.*, sink states).

In [16], [17], we investigate the impact of merging sink states with the core S-PDFA model on the attack graphs. When the sinks are merged with the model, there is a significant decrease in the total number of states (and thus, state identifiers). The resulting attack graphs are slightly smaller than the baseline attack graphs, with a negative impact on the readability and interpretability, and a negligible impact on the complexity of the graphs. Some of the merges cannot be logically explained, *i.e.*, paths that were contextually different are now merged (due to insufficient evidence to prevent merges). The degree centrality of a vertex appears to be inversely proportional to the readability of an attack graph. We find that the number of edges in the attack graphs effectively remains the same while the number of vertices decreases, negatively impacting the readability of the attack graphs, especially for longer attack paths.

There appears to be a trade-off between compactness and readability – paths that are well-separated are more readable, but some merges are appreciated based on highly similar futures and pasts. We apply constraints on the merging of sink states, *i.e.*, instead of merging arbitrary sinks, only the sinks that have the same distance to a specific state are merged. Preliminary results show a small improvement, however the readability and interpretability of the attack graphs remains worse than the baseline attack graphs.

Concurrent events. An important open question pertains to modeling concurrent events. In the experimental datasets, we observe several attacker actions (with different attack stages) that happen simultaneously. Because the S-PDFA is a sequential model, it artificially creates a sequential order out of such concurrent actions, which leads security analysts to misleading conclusions when reviewing the attack graphs. An informal approach to address this is to show the time difference between the end of one action and the start of the next action. Edges with negative gaps refer to overlapping or concurrent actions. However, the underlying problem still remains. One possible solution is to utilize process mining to model concurrent events. The problem with process mod-

els, however, is that they cannot model context. In a follow-up work [18], we combine S-PDFA (for modeling context) and process mining (for modeling concurrent events) to get the best of both worlds. The idea was to extract state sequences from the S-PDFA that are already enriched with context information, and give them as input to a process mining model. Although promising, the results of the follow-up study were inconclusive because of the sparsity of the alert dataset.

Adversarial robustness. The S-PDFA cannot find valid paths for perturbed traces. This is a major limitation of the S-PDFA since it will not be able to handle a noisy trace which is otherwise very similar to other traces in the dataset. This happens because the traces are replayed through the model using exact matching. In a follow-up work [19], we replace the exact matching with sequence alignment to enable the S-PDFA to still find a valid path for traces with missing events or additional noisy events. The idea is to align the perturbed traces with the S-PDFA, and to iteratively relearn the S-PDFA using the perturbed traces until the model converges. This process maps infrequent traces (that contain perturbations) with the frequent part of the S-PDFA model, and often results in a more compact model capable of handling perturbations. Intuitively, the alignment-based matching enables us to learn the core (frequent) behavior present in the traces. For CPTC-2018, this sequence-to-model alignment results in a 49% smaller model than the original model proposed in Chapter 3. This way, we robustify the S-PDFA model against adversarial perturbations.

Dashboard evaluation. Application-grounded evaluation in the form of user studies is required to understand the efficacy of the attack graphs in reaching correct conclusions. For instance, concurrent actions are currently shown in a sequential order, which causes analysts to reach misleading conclusions. Moreover, it is important for analysts to recognize that the attack graphs do not show causality – they show the sequential and probabilistic relationships between alerts. In a follow-up work [20], we conducted an empirical study with a small set of security practitioners in the form of a virtual questionnaire. The survey asked the participants to solve various security tasks with the help of the alert-driven attack graph dashboard. Although we could not collect statistically significant number of responses, the preliminary analysis suggests that the dashboard is more useful for analyzing attacker strategies compared to the currently employed alert management tools. The participants also reported many usability issues in the user interface, which points to a significant room for improvement in the dashboard design.

7.3.4. LIMITATIONS AND FUTURE DIRECTIONS

The attack graphs generated by SAGE may not be complete since it cannot model attack paths for which no alerts are raised. Although SAGE shows the alerts generated by an intrusion detection system (IDS) in the attack graphs, it is impossible to show attack paths that the attackers employ after bypassing the IDS. A potential solution can be to merge alert-driven attack graphs with traditional (static) attack graphs. Since the static attack graphs are based on vulnerabilities in the network, they can enrich alert-driven attack graphs with paths that are not yet exploited, or paths that have been exploited but did not raise alerts. This approach can also help identify “unknown unknowns”. Another

idea is to conduct a penetration test on the target network, and investigate the resulting attack graphs for missing attack paths. If a specific attack was conducted for which no attack path is generated, it may indicate missing or faulty IDS signatures. It is also important to note that the attack graphs do not filter false alarms, *i.e.*, the alerts are shown regardless of their status. Characterizing these false alarms in the attack graphs is left as future work. Moreover, the experiments are currently conducted on security testing datasets where we are certain that attacks are occurring. The alerts that SOCs receive on a regular basis likely have a significantly different distribution. We are currently coordinating with an industrial SOC to learn attack graphs on their alerts [21].

The attack graphs have only been used for behavioral analytics. We are currently investigating the use of these graphs for attack prediction. The S-PDFA cannot directly be used for predicting the next likely attacker action since it is a suffix-based model – it predicts the past based on the future, while we need a model that predicts the future based on the past. We are working on an expectation maximization approach that traverses the S-PDFA in reverse (bottom to top) in order to predict the potential severe event that will likely occur next based on a partial path. Note that the S-PDFA becomes non-deterministic when traversing it in reverse. This makes the prediction task significantly more expensive, requiring us to limit the length of the prediction path. Another open problem is to utilize the fingerprintable paths as detection signatures. Whether such signatures hold across different datasets and infrastructures remains to be seen. Another interesting direction is to utilize neuro-symbolic AI, *e.g.*, for enriching the vertices of the attack graphs with a knowledge graph (KG) learned from security advisories. Finally, aligning the terminologies used in the alert-driven attack graphs with the standard attack/fault tree/graph terminology may help other fields to leverage them as well.

With regards to the dashboard, we aim to create recommended action-items for each attack stage highlighted in the recommender matrix, making the insights generated by SAGE even more actionable. Additionally, the optimal way to compute alert urgency is left as future work. Preliminary work [22] suggests that a criterion based on the in-degree of the attack graph vertices can be used to measure alert urgency. This way, we identify vertices that form central junctions in the attack graphs, *i.e.*, common vertices visited by numerous attack paths. Deploying countermeasures to eliminate such junctions may disrupt the cyber kill-chain more efficiently than handling high-severity alerts alone.

7.3.5. SOCIETAL RELEVANCE

We successfully bridge the gap between dynamic alert management and static attack graph generation with SAGE and the alert-driven attack graphs. Consequently, we solve a long-standing problem in the cybersecurity and formal methods community [23]. In contrast to existing research that utilizes machine learning to merely summarize alert datasets, we go beyond analyzing the model and construct alert-driven attack graphs that link the model's insights back to the input alert signatures. This way, the attack graphs reduce the workload of security analysts by correlating alerts, facilitating the discovery of critical alerts that would otherwise be time-consuming to discover, and providing actionable intelligence regarding the strategies of the attackers. They also capture attacker behavior dynamics, which enables them to distinguish between novice and expert attackers based on the kind of strategies they employ. This has direct military applica-

tions, and is interesting for criminologists. We have released SAGE in a docker container for cross-platform support². It can be used out-of-the-box to learn attack graphs with minimal parameter tuning.

Since we released SAGE, it has garnered attention from industry: SAGE obtained the Best Demo Award at ICT.Open, 2023 (the national Dutch Computer Science conference attended by academics and industry members), and has been presented at the One Conference, 2023 (a European security conference attended by academics, industry members, and government employees).

7.4. NETWORK ATTACK ANALYSIS

In this section, we present our findings pertaining to network attack analysis, and reflect on the scientific implications of the proposed solutions for society at large.

7.4.1. MALWARE CAPABILITY ASSESSMENT

In this section, we answer Q4:

Q4: *How can we leverage unsupervised sequence clustering to characterize the network behavior of malware in order to discover similarities and differences between malware capabilities?*

Malware behavior analysis is largely done manually. Chapter 5 answers this question by developing MalPaCA – an unsupervised, explainable sequence clustering tool-chain to automate malware capability assessment. The first question is to identify the unique capabilities (behaviors) present in the malware dataset. MalPaCA clusters sequences of unsampled network packets (called network connections) generated by malware samples using a combination of density-based clustering and DTW/ngrams. Each cluster represents a unique behavior, which is hand-labeled by investigating the network connections it contains. The next question is to characterize the behavioral profile of each malware sample. This is done by recording the cluster membership of a malware sample's network connections. We use the behavioral profiles to explain the capabilities of malware samples, thus drastically reducing the time it takes to analyze them. We also create a global dendrogram using these profiles that shows similarities and differences between the capabilities of different malware samples.

MalPaCA operates on abstract sequential features extracted from malware's network packet headers, *e.g.*, packet sizes, inter-arrival times, and port numbers. These features are available even when traffic is encrypted, and have previously been shown to capture behavioral attributes [24]. We empirically show that MalPaCA characterizes network behaviors using as little as 20 packet headers. To our knowledge, this is the shortest sequence size utilized in the literature. We also show that MalPaCA is able to create accurate behavioral clusters because of the sequential features, *i.e.*, aggregating network connections produces faster but inaccurate clustering.

We show the efficacy of MalPaCA on an industry-acquired financial malware dataset. A total of 1196 malware samples resulting in 3.6 million network packets are grouped

²SAGE: <https://github.com/tudelft-cda-lab/SAGE>

in 18 clusters. With only the abstract sequential features, MalPaCA accurately identifies the transmission direction of the traffic. The clusters capture capabilities, such as port scans, device broadcasting, and the reuse of C&C servers. We observe clear differences between malware families in terms of their behavior diversity: The profile created for Gozi-ISFB contains 16 distinct behaviors, while those for Dridex Loader, Gozi-EQ, Zeus-P2P, and Zeus-v1 contain only a single behavior. Some of this discrepancy can be explained by the (in)activation of the malware samples during data collection. The global dendrogram for the malware samples (see Figure 5.6) identifies a number of malware samples whose behavioral profiles do not adhere to their traditional family names – 2 Blackmoon and 5 Zeus-Panda samples are observed communicating with the same C&C server. It is unclear whether this happens because of incorrect labeling or because of substantial overlap in their behavior. Regardless, it highlights the unreliable nature of malware family labels as ground truth, and shows how behavioral profiles effectively complement them.

7.4.2. NETWORK ATTACK SUMMARIZATION

In this section, we answer Q5:

Q5: *How can we characterize the network behavior of malware in real-time using interpretable unsupervised sequence clustering?*

To the best of our knowledge, there are currently no methods that support real-time sequence clustering. We answer this question in Chapter 6 by first developing SECLEDS – the first real-time interpretable sequence clustering algorithm with support for evolving data streams (Section 7.1.2). We empirically evaluate SECLEDS on 3 synthetic and 2 real datasets, and compare its performance against 4 state-of-the-art distance-based clustering algorithms. We show that SECLEDS creates high-quality clusters regardless of concept drift, stream size, data dimensionality, and the number of clusters.

Second, we use SECLEDS to tackle the network traffic sampling problem, *i.e.*, network traffic is often randomly sampled to reduce storage requirements. Typically, only *1 out of N* Netflows are stored, losing the temporal patterns in the data that could have been useful for downstream behavioral analytics. We use SECLEDS as a novel temporal pattern-preserving traffic sampling technique, *i.e.*, it summarizes the network traffic in a way that preserves their temporal relationships. By clustering network traffic from a real botnet, we show that SECLEDS supports network bandwidths of over 1 GB/s. We also investigate the quality of the clusters obtained by using DTW (a sequence-specific distance measure) and Euclidean distance (the default distance measure used by typical clustering algorithms). We show that the clusters obtained from SECLEDS-dtw are purer than those obtained from SECLEDS-Euclidean. This provides evidence in favor of sequential data-specific distance measures.

Third, SECLEDS is used in combination with MalPaCA to perform (semi-) real-time capability assessment of bot-infected hosts. The idea is to sample network traffic in real-time with SECLEDS, and MalPaCA utilizes the sampled cluster medoids to construct behavioral profiles of the hosts in the network. The intuition is that the hosts infected with the same botnet will have a similar behavioral profile. We compare the quality of

the behavioral profiles obtained using SECLEDS-sampling against those obtained using random-sampling. We show that SECLEDS-sampling is able to discover 6 unique behaviors in the sampled network traffic, while random-sampling is only able to discover 2 broad behaviors. As a result, the behavioral profiles created using SECLEDS-sampling are more detailed, and are able to identify similar bot-infected hosts.

Note that existing literature aims to create clusters that either represent malicious or benign hosts. Since MalPaCA's clusters are defined at the behavior granularity, we often obtain clusters that contain network connections from both benign and malicious hosts since they exhibit the same behavior. This is the intended objective as we split network connections into chunks and cluster them such that the clusters represent behaviors. Interestingly, when we look at the behavioral profiles that show the overall cluster membership, we are often able to distinguish between malicious and benign hosts with 100% accuracy (see Figure 6.9). This is because while malicious and benign hosts exhibit overlapping behaviors, they typically do not coexist in all the same clusters. This establishes the behavioral profiles as a neat technique to fingerprint hosts.

7.4.3. LIMITATIONS AND FUTURE DIRECTIONS

Both MalPaCA and SECLEDS create network behavioral profiles of malware. We choose network traffic since it is practical and non-invasive. It is also easy to collect and has a low overhead on end-hosts compared to system logs. However, prevailing research overwhelmingly utilize system logs for malware analysis [25]. Creating holistic behavioral profiles that capture both the system and network behavior is left as future work. In addition to challenges in feature engineering, additional challenges may arise due to the multi-modal aspect of the feature space.

Further research is required to understand whether the system and network profiles are always consistent, and whether the discrepancies discovered in Chapter 5 are because of this inconsistency. More specifically, why do the behavioral profiles present a different picture than the malware family labels? A related question is about differentiating between malware samples that do not activate their capabilities as an evasion tactic versus those that truly have fewer capabilities.

In Chapter 6, we combine SECLEDS and MalPaCA to create (semi-) real-time behavioral profiles of malware. SECLEDS can cluster network traffic in real-time but only using k clusters. The density-based clustering algorithm used in MalPaCA (*i.e.*, HDBScan) can create arbitrary number of clusters but only in an offline setting. Replacing HDBScan with SECLEDS will limit MalPaCA's ability to discover arbitrary behaviors. We leave this as future work. Furthermore, although we empirically show that this combination can be used to cluster network traffic in a streaming setting in order to create (semi-) real-time behavioral clusters, the impact of concept drift on the behavioral profiles could only be tested for about 5.5 hours. Investigating whether such real-time behavioral profiling can help detect evolving behavioral profiles is left as future work.

7.4.4. SOCIETAL RELEVANCE

SECLEDS is the first interpretable algorithm that clusters sequences in real-time. MalPaCA is the first tool-chain of its kind that automatically characterizes malware behavior. This is important for malware analysts since malware behavior characterization is

mostly done manually, and there is no standardized way to label behaviors. In 2020, MITRE announced its Malware Behavior Catalog (MBC)³ that enumerates malware objectives and behaviors. The catalog is regularly updated as they manually investigate newly discovered malware code. MalPaCA essentially automates this process by discovering unique behavioral groups in malware traces. We opt for a dynamic approach since only the behaviors that are exhibited by malware are relevant for behavior profiling. A practitioner only needs to analyze a fraction of a cluster to label it with a capability. Behavioral profiles are automatically assigned to malware samples based on the cluster membership of their traces. Together with SECLEDS, MalPaCA has the ability to discover new behaviors as new malware is discovered. This substantially reduces the amount of time it takes to characterize malware behavior. We release MalPaCA as open-source⁴. It works out-of-the-box for creating behavioral profiles using network traces. Also, it is not limited to malware behavior profiles – it can essentially be used to create behavioral profiles for any entity that produces network traffic, *e.g.*, network hosts, and IoT devices.

Moreover, we empirically show the power of sequential features in Chapters 5 and 6 for behavior characterization. To this end, we demonstrate that it is not required to have long sequences or sequences with privacy-intrusive features. We even show that sequences of raw packet captures (Pcaps) and aggregated network flows (Netflows) are both sufficient to characterize behavior. This provides a strong recommendation to the community to consider temporal pattern-preserving sampling over random sampling.

7.5. FINAL WORDS ON ADVERSARY BEHAVIOR ANALYSIS

Understanding adversary behavior is the first step towards building better detection tools. This thesis contributes to the growing body of literature on ML-based cyber threat intelligence, and provides several solutions for understanding attacker behavior: We create a novel paradigm of alert-driven attack graphs to investigate how attacks transpire, and create behavioral profiles of malware samples that are much more descriptive than their family labels. To this aim, we develop special algorithms that learn sequential patterns from infrequent events, and evolving data in an unsupervised setting.

In essence, this thesis advocates for the use of interpretable models, and highlights the power of sequential features in behavior modeling. It demonstrates that by considering the appropriate stakeholders and rigorously defining the problem, it is possible to develop competitive interpretable solutions, even for sequential data in an unsupervised setting. All solutions proposed in this thesis are open-source, and can be used out-of-the-box for analyzing attacker behavior in previously unexplored datasets. The proposed solutions facilitate practitioners by automatically discovering intricate patterns that would otherwise be too time-consuming to discover manually, ultimately reducing their workload. Furthermore, in contrast to popular research, this thesis shows that an over-reliance on performance metrics can be detrimental, especially for unsupervised sequence clustering tasks. For instance, we have observed that the Silhouette index prefers fewer clusters, and the clustering that achieves the best scores is qualitatively inferior to other clustering configurations that produce more clusters. This is why

³MITRE MBC: <https://github.com/MBCProject>

⁴MalPaCA: <https://github.com/tudelft-cda-lab/malpacapub>

we advocate for interpretable settings in behavior analysis, and do not recommend to replace human intelligence with performance metrics.

This thesis stands as a testament to the possibilities that arise when metrics are not utilized as mere crutches. It serves as a catalyst for further research in the application of interpretable machine learning for understanding attacker behavior. We encourage the security community to look beyond accuracy scores, and focus on extracting actionable insights from machine learning models in order to create *AI-assisted practitioners*.

REFERENCES

- [1] A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer, “Intelligent Malware Defenses”, in *Security and Artificial Intelligence*, Springer, 2022, pp. 217–253.
- [2] 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.
- [3] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, “TABOR: A graphical model-based approach for anomaly detection in industrial control systems”, in *AsiaCCS*, 2018.
- [4] 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.
- [5] M. Masnad, G. MukitHasan, K. M. Iftekhar, and M. S. Rahman, “Human activity recognition using DTW algorithm”, in *2019 IEEE Region 10 Symposium (TEN-SYMP)*, IEEE, 2019, pp. 39–43.
- [6] M. Quadrana, P. Cremonesi, and D. Jannach, “Sequence-aware recommender systems”, *ACM computing surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.
- [7] C. V. Jordan, F. Hauer, P. Foth, and A. Pretschner, “Time-series-based clustering for failure analysis in hardware-in-the-loop setups: An automotive case study”, in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2020, pp. 67–72.
- [8] M. Orozco-Alzate, P. A. Castro-Cabrera, M. Bicego, and J. M. Londoño-Bonilla, “The DTW-based representation space for seismic pattern classification”, *Computers & Geosciences*, vol. 85, pp. 86–95, 2015.
- [9] G. Weiss, Y. Goldberg, and E. Yahav, “Extracting automata from recurrent neural networks using queries and counterexamples”, in *International Conference on Machine Learning*, PMLR, 2018, pp. 5247–5256.
- [10] M. Blumreiter, J. Greenyer, F. J. C. Garcia, *et al.*, “Towards self-explainable cyber-physical systems”, in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, IEEE, 2019, pp. 543–548.
- [11] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, “Sanity checks for saliency maps”, *Advances in neural information processing systems*, vol. 31, 2018.
- [12] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning”, *arXiv preprint arXiv:1702.08608*, 2017.

- [13] H. Liu, C. Zhong, A. Alnusair, and S. R. Islam, “FAIXID: a framework for enhancing ai explainability of intrusion detection results using data cleaning techniques”, *Journal of Network and Systems Management*, vol. 29, no. 4, pp. 1–30, 2021.
- [14] W. U. Hassan, S. Guo, D. Li, *et al.*, “Nodoze: Combatting threat alert fatigue with automated provenance triage”, in *Network and Distributed System Security*, 2019.
- [15] I. Oprea, “Investigating the impact of PDFA implementation on alert-driven attack graphs”, Delft University of Technology, 2023. DOI: <http://resolver.tudelft.nl/uuid:a51db672-a286-4568-85db-05ecbae3cca5>.
- [16] J. Zelenjak, “Investigating the Impact of Merging Sink States on Alert-Driven Attack Graphs”, Delft University of Technology, 2023. DOI: <http://resolver.tudelft.nl/uuid:559f1001-e436-4fd7-b17f-f9e6203ced93>.
- [17] A. Dumitriu, “Investigating the Impact of Sink State Merging on Alert-Driven Attack Graphs”, Delft University of Technology, 2023. DOI: <http://resolver.tudelft.nl/uuid:d55d58b6-eb03-4246-ac76-67121d4b3753>.
- [18] G. H. Jansen, “Mining Attack Strategy: Using Process Mining to extract attacker strategy from IDS alerts”, Delft University of Technology, 2021. DOI: <http://resolver.tudelft.nl/uuid:226be13e-1f26-4ed4-98e5-441bfd0c2006>.
- [19] D. Mouwen, S. Verwer, and A. Nadeem, “Robust Attack Graph Generation”, *Learning and Automata workshop (LearnAut)*, 2022.
- [20] S. L. Diaz, A. Nadeem, and S. Pastrana, “Critical Path Prioritization Dashboard for Alert-driven Attack Graphs”, *arXiv preprint arXiv:2310.13079*, 2023.
- [21] I. Băbălău, “Real-time attack graph generation using intrusion alerts”, Delft University of Technology, 2023. DOI: <http://resolver.tudelft.nl/uuid:8c398cd2-c75c-4529-bcad-e6c8236ed54c>.
- [22] S. Van den Broeck, “Investigating Episode Prioritisation in Alert-Driven Attack Graphs”, Delft University of Technology, 2023. DOI: <http://resolver.tudelft.nl/uuid:4bd5969b-dd1b-40fd-916f-e24fe299e0aa>.
- [23] K. Sowka, V. Palade, H. Jadidbonab, P. Wooderson, and H. Nguyen, “A Review on Automatic Generation of Attack Trees and Its Application to Automotive Cybersecurity”, *Artificial Intelligence and Cyber Security in Industry 4.0*, pp. 165–193, 2023.
- [24] A. Nadeem, “Clustering Malware’s Network Behavior using Simple Sequential Features”, Delft University of Technology, 2018. DOI: <http://resolver.tudelft.nl/uuid:c8a221b9-9289-4978-a356-af64d8f2c5e0>.
- [25] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, “A quantitative study of accuracy in system call-based malware detection”, in *ISSTA*, ACM, 2012, pp. 122–132.

ACKNOWLEDGEMENTS

This 6-year PhD/lecturer position was... confusing, to say the least, which encouraged me to lean on many different people, both for research and teaching. They say it takes a village to raise a child; this PhD journey felt no less, and I am grateful to everyone who tagged along the way.

I want to start by thanking my promoters, Sicco and Inald. Sicco, I am extremely grateful for your guidance and support (that extended well beyond the PhD). Thank you for believing in me, and giving me the freedom to discover my own path. The friendly environment you cultivated in your lab made it feel like a second home, which kept me motivated through (numerous) rejections and a global pandemic! Your supervision style is truly an inspiration to me; I can only hope to be as supportive as you are in my own practice. Inald, thank you for your advice and insightful discussions. Your feedback really helped put the finishing touches on this thesis.

I want to thank my committee members, Prof. Davide Balzarotti, Prof. Jay Yang, Prof. Mariëlle Stoelinga, Prof. Robert Kooij, Prof. Mathijs de Weerd, and Dr. Gosia Migut, for taking the time to review the thesis, and being a part of my defense ceremony. Also, thanks to Clinton for helping me translate the thesis summary into Dutch, and Raima for designing the thesis cover.

I want to extend my heartfelt thanks to the members of the “Cyber Analytics Lab”, as Sicco’s lab was re-branded in September of 2020 (admittedly, during the boredom of Covid lockdown). Thank you, Qin and Nino, for encouraging me to do a PhD with Sicco. Chris, thank you for helping me navigate academia in those early days, and agreeing to be my paronymph at the end. I hope we can stay in touch, no matter where life takes us. Mark, although you were with us very briefly, it was your suggestion that made me a published author. So, thank you for that! Laurens, it was exciting to work with you (and win!) the AdvML challenge. Clinton and Daniël, thank you for all the silliness, whatsapp stickers, birthday parties, conference hangouts (and very little work). Speaking of hang-out, I really enjoyed Omelet Thursdays with the lab, that is, until the dreaded Grenoble eggcident. Simon, I wish you would finally find something to like about the Netherlands. Robert, thanks for shaking up the group with your unique personality. Tom, thank you for driving us back from Scheveningen so quickly after the CYS BBQ. I owe you my life, literally! Daniël, when you moved to Enschede and left the lab, I thought we lost you forever. Little did we know that I will, in fact, see you around!

To all the students I worked with over the years: Sandesh, Geert, Dennis, Ion, Sonia, Silviu, Rami, Ruben, Jegor, Ioan, Vlad, Alexandru, Senne, Jonathan, Mikhail, Johannes, Sung, Hugo, Shivani, and Sarah, thank you for giving me the opportunity to learn from you. A special mention goes to the most rebellious master students we have had in a while, Bram and Wessel. Thanks for bringing a lively atmosphere to the office!

I have generally been fortunate to have office mates who have created such a fun environment that it was difficult to get any real work done. This started in 2018 when

Gamze, Ozzy, Chibuike, and Majid welcomed me into their (fun) PhD office, and continued until 2023 when I had to schedule a dedicated office day for social activities. Ozzy, for all the times you got distracted by my loud laughter through the paper-thin walls of our offices, I bet you regretted upgrading to the other (boring) office! I want to acknowledge members of the CYS group for our occasional chats at the coffee corner: Florine, Huimin, Jing, Tianyu, Lichao, Jelle, Stefanos, Luca, Giovane, and Roland. Sandra and Sophie, thanks for taking care of the administrative side of things. A special thanks also goes to Ruud for helping me set up my home-office during the first Covid lockdown, and all his technical assistance with the vulnerable webserver for my SQL injection lecture. We managed to keep the university network from getting hacked, after all!

During my PhD, I met wonderful colleagues, some of whom had a major impact on my PhD trajectory. I am thankful to Prof. Mauro Conti for organizing the fateful Security & AI summer school in Padova, and to Dr. Stjepan Picek for inviting me to the Security & AI Lorentz workshop in 2019, where I met so many of my future colleagues. I am also grateful to Prof. Lejla Batina, Prof. Mariëlle Stoelinga, Dr. Cristina Cifuentes, and Dr. Vera Rimmer for inviting me to present my research at their institutions, and opening doors for additional collaborations. A special thanks to Prof. Jay Yang for inviting me to the CyberVSR programme at RIT, which turned into a collaboration that lasted over 2 years! I feel fortunate to have worked with you, and honestly believe that CyberVSR served as a gateway for me to enter the cybersecurity community. I also want to thank Prof. Davide Balzarotti and Dr. Leyla Bilge for hosting me at Eurecom towards the end of my PhD, even though it was a shorter visit than expected. Prof. Pieter Hartel, thank you for helping me prepare the application for this 6-year PhD/lecturer position, and for your guidance at the start of my PhD. Dr. Zeki Erkin, I am grateful for the doors you opened for me. Thank you for inviting me to organize CSng, and to teach at Leiden university.

Alongside research, I spent a significant part of my PhD doing teaching, and although they were two separate jobs, I could not have endured the early years of my PhD without the support of the teaching team. Stefan, Amira, Gosia, Frank, Jesse, Thomas, Otto, Sander, Daniël, Christos, Bart, Taico, Tom, Ivo, Mathijs, Chivany, Elena, Danny, and Max, I deeply value the supportive environment you created in the team, which clearly had a far-reaching impact! I also want to thank Prof. Andy Zaidman and Prof. Hans Tonino for leading the teaching team and making it what it is today. It took us some time to navigate the system and make sense of our unconventional (and sometimes, confusing) roles. The weekly meetings played a pivotal role in that! I also want to take this opportunity to thank Dr. Christoph Lofi, Dr. Asterios Katsifodimos, Dr. Maurício Aniche, and Dr. Claudia Hauff, for allowing me to teach in your courses. I know we had different ideologies, but I am glad we made it work in the end for the students. I also want to thank Dr. Kaitai Liang for giving me the opportunity to help organize the introductory cybersecurity course. I think an important reason why teaching kept me motivated through the (occasional) bad research days was the semi-regular feedback I received from the students. So for that, I want to thank all the students who ever reached out!

Looking back, I feel fortunate to have had mentors who saw my potential and encouraged me to dream bigger. Specifically, my teachers from long ago, Tariq Mehmood and Maajid Maqbool, I will never truly understand what made you believe in me, but I am ever grateful for it! Also to the mentors I found later on, especially the team behind

Our Future Leaders (incl. our facilitator, Mark), thank you for empowering me to take several major leaps in my career! They say rejection is also a powerful teacher. Thanks to all the conferences/workshops that rejected my earlier work. Although I could not see it at the time, they really did teach me to become more eloquent (in writing math ;)).

2023 proved to be a particularly turbulent year for me. I wish to extend my special thanks to Norman, Francesca, Marina, Ivan, Miray, and Vera, without whom I would probably not have made it. Norman, thank you for showing up for me in ways no one could have expected. I am so incredibly proud of the person you have become! Marina, Miray, (and Ivan), thanks for all the girls' nights ;)! Marina, I cannot wait to beat you in climbing! Also, sorry about Monaco :p. Miray, you are the embodiment of not judging a book by its cover! Vera, in a world filled with empty promises, thank you for being an ally! I should give a shout-out to Dinesh, Hari, Jaya, and Ishan, for being pro at friendship and meeting up exactly once in 3 years! I also want to thank my friends in Pakistan: Tayyaba, Faiza, and Raza, for introducing me to amazing restaurants with delicious food! Finally, I also spent some time in the French Riviera this year where I (coincidentally) met Elisa, Neli, and their pub quiz team who accompanied me to celebrate my 30th birthday with the prize money, and made it the highlight of the year!

Last but certainly not the least, I want to thank my parents for feeding my ambitions with creative challenges. Mama, you wanted your daughters to become doctors. Papa, you wanted them to become engineers. Now, you've got yourself a daughter who is both: *Dr.ir. Azqa Nadeem!* ;D I want to express my deepest gratitude to you both for all your sacrifices and your unwavering support, especially from over 5000 kilometers away! Your courage for sending me away to a foreign land to pursue my dreams has not gone unnoticed. I am where I am today because of the integrity and perseverance you instilled in me! To my sisters, Raima apy, Ezza, and Hunia, thank you for patiently listening to my endless blabbering, sharing silly memes, and for being my partners in crime, whenever possible! ;) Finally, thank you, Junaid, for pushing me to grow beyond what I thought was possible...

Azqa Nadeem
Delft, Netherlands
January 25, 2024

CURRICULUM VITÆ

Azqa NADEEM

Azqa Nadeem was born in Quetta, Pakistan on October 17, 1993. She earned her bachelor's degree in Computer Science with a gold medal from the National University of Sciences & Technology (NUST) in Pakistan in 2015. She briefly worked as an Application Engineer at LMKR in Pakistan in 2016. Later, she started her master's degree in Computer Science with a specialization in Data Science and Cybersecurity at TU Delft in the Netherlands, and graduated cum laude in 2018. Azqa joined the Cybersecurity group at TU Delft as a PhD student & Junior lecturer in September 2018.

Azqa was awarded a silver medal by the Chief of the Air Staff, Pakistan Air Force in 2011 for her high school grades. From 2011 to 2015, she received a merit-based scholarship for maintaining a perfect 4.00/4.00 CGPA in every semester during her bachelor's. She also received multiple "High Achiever Awards" in 2014 and 2015 from the rector of NUST for her outstanding performance during the Summer@EPFL and CERN Openlab summer internship programmes, respectively. Notably, at CERN, she won the "Best Lightning Talk Award" among 40 CERN Openlab interns, for which she was featured in several national newspapers. In 2015, she was recognized as one of the "25 High Achieving Pakistanis under 25". Azqa's master's education at TU Delft was funded by the Justus & Louise van Effen Excellence Scholarship. In 2019, she was nominated as the "Best Graduate of the faculty of EEMCS" at TU Delft for her master's thesis, which focused on the responsible use of Artificial Intelligence.

During her PhD, she developed Explainable machine learning solutions for Cybersecurity under the supervision of Dr. ir. S.E. Verwer and Prof. dr. ir. R.L. Lagendijk. She also visited Prof. S.J. Yang at Rochester Institute of Technology in the United States of America, and Prof. D. Balzarotti at Eurecom in France in 2020 and 2023, respectively. Azqa dedicated 40% of her time to teaching while earning the University Teaching Qualification (Basis Kwalificatie Onderwijs, BKO) in 2020. She was responsible for integrating Cybersecurity in existing 1st-year bachelor courses at TU Delft. For this, she designed and delivered security lectures to more than 2,200 students across three core Computer Science courses over a span of 5 years. She also supervised 5 master students and 14 bachelor students for their final thesis projects, and served as an examiner for the final projects of 11 bachelor students.

As of January 2024, Azqa joined the Semantics, Cybersecurity, and Services (SCS) group at the University of Twente in the Netherlands as an Assistant Professor.

LIST OF PUBLICATIONS

JOURNAL

1. **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. (2021). Alert-driven Attack Graph Generation using S-PDFA. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 19(2), 731-746.

CONFERENCE AND WORKSHOP

9. **Nadeem, A.** (2024). *Cybersecurity as a Crosscutting Concept Across an Undergrad Computer Science Curriculum: An Experience Report*. ACM Technical Symposium on Computer Science Education (SIGCSE).
8. **Nadeem, A.**, Vos, D., Cao, C., Pajola, L., Dieck, S., Baumgartner, R., & Verwer, S. (2023). *SoK: Explainable Machine Learning for Computer Security Applications*. IEEE European Symposium on Security and Privacy (Euro S&P), pp. 221-240. IEEE.
7. **Nadeem, A.**, & Verwer, S. (2022). *SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting*. Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), pp. 157-173.
6. Mouwen, D., Verwer, S., & **Nadeem, A.** (2022). *Robust Attack Graph Generation*. Learning and Automata workshop (LearnAut).
5. **Nadeem, A.**, Verwer, S., & Yang, S. J. (2021). *SAGE: Intrusion Alert-driven Attack Graph Extractor*. IEEE Symposium on Visualization for Cyber Security (VizSec) pp. 36-41. IEEE.
4. Verwer, S., **Nadeem, A.**, Hammerschmidt, C., Blik, L., Al-Dujaili, A., & O'Reilly, U. M. (2020). *The Robust Malware Detection Challenge and Greedy Random Accelerated Multi-bit Search*. ACM Workshop on Artificial Intelligence and Security (AISec), pp. 61-70.
3. Roeling, M. P., **Nadeem, A.**, & Verwer, S. (2020). *Hybrid Connection and Host Clustering for Community Detection in Spatial-temporal Network Data*. Workshop on Machine Learning for Cybersecurity (MLCS), Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), Springer, pp. 178-204.
2. Azim, T., **Nadeem, A.**, & Ailamaki, A. (2018). *Adaptive Cache Mode Selection for Queries over Raw Data*. International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS), International Conference on Very Large Data Bases (VLDB), pp. 51-65.
1. Pascarella, L., Ram, A., **Nadeem, A.**, Bisesser, D., Knyazev, N., & Bacchelli, A. (2018). *Investigating Type Declaration Mismatches in Python*. IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTesQuE) (pp. 43-48). IEEE.

BOOK CHAPTERS

4. **Nadeem, A.**, Yang, S. J. & Verwer, S. (2023). *Learning About the Adversary*. Autonomous Intelligent Agents for Cyber Defense, Springer, pp. 105-132.
3. **Nadeem, A.**, Rimmer, V., Joosen, W., & Verwer, S. (2022). *Intelligent Malware Defenses*. Security and Artificial Intelligence, Springer, pp. 217-253.
2. Rimmer, V., **Nadeem, A.**, Verwer, S., Preuveneers, D., & Joosen, W. (2022). *Open-world Network Intrusion Detection*. Security and Artificial Intelligence, Springer, pp. 254-283.
1. **Nadeem, A.**, Hammerschmidt, C., Gañán, C. H., & Verwer, S. (2021). *Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families*. Malware Analysis Using Artificial Intelligence and Deep Learning, Springer, pp. 381-409.

EXTENDED ABSTRACTS

4. Zelenjak, J., Oprea, I., Dumitriu, A., Verwer, S., & **Nadeem, A.** (2023). *Using Sink States to Learn from Infrequent Data in an Automaton Model (Thesis abstract)*. Benelux Conference on AI and Belgian-Dutch Conference on Machine Learning (BNAIC/BeNeLearn).
3. **Nadeem, A.**, & Verwer, S. (2022). *SECLEDS: Sequence Clustering in Evolving Data Streams via Multiple Medoids and Medoid Voting (Encore abstract)*. Benelux Conference on AI and Belgian-Dutch Conference on Machine Learning (BNAIC/BeNeLearn).
2. **Nadeem, A.**, Verwer, S., & Yang, S. J. (2022). *Suffix-based Finite Automata for Learning Explainable Attacker Strategies*. International workshop on Heterodox Methods for Interpretable and Efficient AI (HMIEAI).
1. **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. (2021). *SAGE: Intrusion Alert-driven Attack Graph Extractor (Encore abstract)*. Benelux Conference on Artificial Intelligence and Belgian-Dutch Conference on Machine Learning (BNAIC/BeNeLearn), pp. 659-661.

POSTERS

4. **Nadeem, A.**, Diaz, S. L., & Verwer, S. (2022). *Critical Path Exploration Dashboard for Alert-driven Attack Graphs*. IEEE Symposium on Visualization for Cyber Security (VizSec).
3. **Nadeem, A.**, Verwer, S., Moskal, S., & Yang, S. J. (2021). *Enabling Visual Analytics via Alert-driven Attack Graphs*. ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 2420-2422.
2. **Nadeem, A.**, Roeling, M.P., & Verwer, S.E. (2019). *A Hybrid Approach to Allow Unsupervised Clustering of Both Connections and Hosts in Networked Data*. Annual international cybersecurity and AI conference (CyberSec&AI Connected).
1. **Nadeem, A.**, & Verwer, S.E. (2018). *Clustering Malware's Network Behavior using Simple Sequential Features*. Nordic Conference on Secure IT Systems (NordSec).

PRE-PRINTS

1. Diaz, S. L., Pastrana S., & **Nadeem, A.** (2023). *Critical Path Prioritization Dashboard for Alert-driven Attack Graphs*. ArXiv.

ACKNOWLEDGMENTS

1. Wagner, J., Kuznetsov, V., Candea, G., & Kinder, J. (2015). *High System-code Security with Low Overhead*. IEEE Symposium on Security and Privacy (S&P) (pp. 866-879). IEEE.

SIKS DISSERTATION SERIES

-
- 2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Céleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

- 25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
 - 26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
 - 27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
 - 28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
 - 29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
 - 30 Ruud Mattheij (TiU), The Eyes Have It
 - 31 Mohammad Khelghati (UT), Deep web content monitoring
 - 32 Eelco Vriezেকolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
 - 33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
 - 34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
 - 35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
 - 36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
 - 37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
 - 38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
 - 39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
 - 40 Christian Detweiler (TUD), Accounting for Values in Design
 - 41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
 - 42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
 - 43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
 - 44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
 - 45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
 - 46 Jorge Gallego Perez (UT), Robots to Make you Happy
 - 47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
 - 48 Tanja Buttler (TUD), Collecting Lessons Learned
 - 49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
 - 50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
-

- 2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joose (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"

- 30 Wilma Latuny (TiU), The Power of Facial Expressions
- 31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
- 32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
- 33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
- 34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
- 35 Martine de Vos (VUA), Interpreting natural science spreadsheets
- 36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
- 37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
- 38 Alex Kayal (TUD), Normative Social Applications
- 39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
- 40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
- 41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
- 42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
- 43 Maaïke de Boer (RUN), Semantic Mapping in Video Retrieval
- 44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
- 45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
- 46 Jan Schneider (OU), Sensor-based Learning Support
- 47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
- 48 Angel Suarez (OU), Collaborative inquiry-based learning
-
- 2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
- 02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
- 03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
- 04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
- 05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
- 06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
- 07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
- 08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
- 09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
- 10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology

- 11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
 - 12 Xixi Lu (TU/e), Using behavioral context in process mining
 - 13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
 - 14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
 - 15 Naser Davarzani (UM), Biomarker discovery in heart failure
 - 16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
 - 17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
 - 18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
 - 19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
 - 20 Manxia Liu (RUN), Time and Bayesian Networks
 - 21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
 - 22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
 - 23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
 - 24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
 - 25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
 - 26 Roelof Anne Jelle de Vries (UT), Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
 - 27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
 - 28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
 - 29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
 - 30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
-
- 2019 01 Rob van Eijk (UL), Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
 - 02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
 - 03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
 - 04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
 - 05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
 - 06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
 - 07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
 - 08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
 - 09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems

- 10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
- 11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
- 12 Jacqueline Heinerman (VUA), Better Together
- 13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
- 14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
- 15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
- 16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
- 17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts
- 18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
- 19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
- 20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
- 21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
- 22 Martin van den Berg (VUA), Improving IT Decisions with Enterprise Architecture
- 23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
- 24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
- 25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
- 26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
- 27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
- 28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
- 29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
- 30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
- 31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
- 32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
- 33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
- 34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
- 35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming

- 36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
- 37 Jian Fang (TUD), Database Acceleration on FPGAs
- 38 Akos Kadar (OU), Learning visually grounded and multilingual representations
-
- 2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
- 02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
- 03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
- 04 Maarten van Gompel (RUN), Context as Linguistic Bridges
- 05 Yulong Pei (TU/e), On local and global structure mining
- 06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
- 07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
- 08 Ali Mirsoleimani (UL), Structured Parallel Programming for Monte Carlo Tree Search
- 09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
- 10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
- 11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation- Methods for Long-Tail Entity Recognition Models
- 12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
- 13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
- 14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
- 15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
- 16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
- 17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
- 18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
- 19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
- 20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
- 21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
- 22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
- 23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
- 24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
- 25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining

-
- 26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer optimization
 - 27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
 - 28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
 - 29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
 - 30 Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
 - 31 Gongjin Lan (VUA), Learning better – From Baby to Better
 - 32 Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
 - 33 Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
 - 34 Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
 - 35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
-
- 2021 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
 - 02 Rijk Mercur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
 - 03 Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
 - 04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
 - 05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
 - 06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
 - 07 Armel Lefebvre (UU), Research data management for open science
 - 08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
 - 09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
 - 10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
 - 11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
 - 12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
 - 13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
 - 14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
 - 15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
 - 16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
 - 17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks

- 18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
 - 19 Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
 - 20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
 - 21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
 - 22 Sihang Qiu (TUD), Conversational Crowdsourcing
 - 23 Hugo Manuel Proença (UL), Robust rules for prediction and description
 - 24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
 - 25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
 - 26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
 - 27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
 - 28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
-
- 2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
 - 02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
 - 03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
 - 04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
 - 05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
 - 06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
 - 07 Sambit Praharaaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
 - 08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
 - 09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
 - 10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
 - 11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
 - 12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
 - 13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
 - 14 Michiel Overeem (UU), Evolution of Low-Code Platforms

- 15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
 - 16 Pieter Gijsbers (TU/e), Systems for AutoML Research
 - 17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
 - 18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
 - 19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
 - 20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
 - 21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
 - 22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
 - 23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
 - 24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
 - 25 Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
 - 26 Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
 - 27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
 - 28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
 - 29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
 - 30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
 - 31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
 - 32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
 - 33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
 - 34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
 - 35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
-
- 2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
 - 02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
 - 03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
 - 04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval

- 05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
- 06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
- 07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
- 08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
- 09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
- 10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
- 11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
- 12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
- 13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
- 14 Selma Čaušević (TUD), Energy resilience through self-organization
- 15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
- 16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
- 17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
- 18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
- 19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
- 20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
- 21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
- 22 Alireza Shojafar (UU), Volitional Cybersecurity
- 23 Theo Theunissen (UU), Documentation in Continuous Software Development
- 24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
- 25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
- 26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
- 27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions

-
- 28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
 - 29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results
-
- 2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
 - 02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
 - 03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
 - 04 Mike Huisman (UL), Understanding Deep Meta-Learning
 - 05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
 - 06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence