Detection and Analysis of Long-Term Threats using Large Dynamic Uncertain Graph Models

by

Paulo Gustavo Quinan
B.Sc., Universidade Federal de Santa Catarina, Brazil, 2010
M.Sc., Universidade do Estado de Santa Catarina, Brazil, 2013

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Detection and Analysis of Long-Term Threats using Large Dynamic Uncertain Graph Models

by

Paulo Gustavo Quinan
B.Sc., Universidade Federal de Santa Catarina, Brazil, 2010
M.Sc., Universidade do Estado de Santa Catarina, Brazil, 2013

Supervisory Committee

Dr. Issa Traore, Supervisor
Department of Electrical and Computer Engineering

Dr. Isaac Woungang, Co-Supervisor
Department of Computer Science — Toronto Metropolitan University

Dr. Xiaodai Dong, Departmental Member
Department of Electrical and Computer Engineering

Dr. Kui Wu, Outside Member
Department of Computer Science

Abstract

In the past decade, new types of long-term threats, such as the advanced persistent threat (APT), have emerged. Their complexity brings many challenges for detection, prevention and posterior forensic analysis of intrusions. In contrast, the intrusion detection systems (IDSs) employed in these tasks work independently of one another, and integrating their alerts with security information and event management systems is mostly an *ad hoc* process. Forensic analysis is also hampered and is made exponentially more complex in this scenario.

To address these challenges, this dissertation proposes a new knowledge graph model, called the Activity and Event Network (AEN), that leverages data from both the traditional security ecosystem and beyond the organization perimeter to capture the activities and relationships of network agents as well as their inherent dynamicity and uncertainty, and through that, increase situational awareness of the threat environment and allow detecting, responding and investigating sophisticated and stealth attacks. In practice, the AEN serves as a basis upon which different detection mechanisms, threat analyses and forensic investigations of both novel and known attack patterns, can be performed.

To validate those capabilities, three unsupervised intrusion detection mechanisms are proposed as follows. A signature-based scheme that employs an isomorphic subgraph matching algorithm to search for graphical attack patterns in the graph. An anomaly detection mechanism that involves calculating anomaly scores based on the bits of meta-rarity metric for statistical features and underlying distributions extracted from the graph. And a belief propagation mechanism that leverages the alerts from different IDSs that have been inputted into the graph as indicators of compromise with the goal of obtaining better detection performance in comparison to the IDSs by themselves, and works by deriving graphs akin to Markov random field from the main AEN graph and performing a probabilistic inference on the derived graphs. Also part of this detection mechanism is a "human-in-the-loop" online parameter adaptation mechanism based on the stochastic gradient descent algorithm, which was conceived with the aim of reducing the initial burden of selecting the system's parameters and allowing for frequent adaptation of parameters in dynamic environments.

The three detection mechanisms were evaluated individually and in combination using two different datasets, namely the ISOT-CID Phase 1 dataset and the CIC-IDS2017 dataset. The results obtained were promising and demonstrated the effectiveness of all three mechanisms according to their expected characteristics. When combined, the three mechanism obtained an average reduction of over 80% in the number of errors when compared to traditional IDSs such as Snort and Kitsune.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

AEN Activity and Event Network. iii, viii–x, 3, 6, 26, 27, 35, 36, 43, 44, 48–52, 54, 66, 67, 70, 74–79, 81–83, 85, 86, 88, 89, 100, 101, 109–111, 113, 115, 116, 120–122

AGD algorithmically generated domain. 29, 31

APT advanced persistent threat. iii, 1, 2, 4, 6–11, 40, 48

ASN autonomous system number. 29

ATA advanced targeted attack. 7

AUC area under the ROC curve. 98

AV anti-virus. 27, 29

BM bookmaker informedness. 86, 87, 96, 98, 100, 104, 105, 108, 109, 112–114, 121

BN Bayesian network. 23, 77

C&C command and control. 8, 10, 41

CIC Canadian Institute for Cybersecurity. xii, 6

CIC-IDS2017 2017 CIC Intrusion Detection Evaluation dataset. iii, vii–x, 6, 85, 86, 89–91, 94, 99, 100, 102, 112–116, 118–121, 123

CIDR classless inter-domain routing. 32

CPT conditional probability table. 25, 26

DAG directed acyclic graph. 23, 77

DBN deep belief network. 23

DDoS distributed DoS. 59

DNS domain name system. 3, 27, 29, 31–33, 39, 40, 49, 117

DoS denial of service. xi, xii, 3, 51, 56, 57, 59, 60, 64–66, 76, 89, 93, 95, 120

DPI deep-packet inspection. 28, 66

FPR false positive rate. 87, 96

IDS intrusion detection system. iii, 1–4, 9, 10, 17, 27–29, 38, 39, 41, 42, 49, 52, 61, 83, 85, 100, 105, 113, 121

IOC indicator of compromise. iii, 3, 29, 80, 81, 85, 121

IQR interquartile range. 107

ISOT Information Security and Object Technology. xiii, 6

ISOT-CID ISOT Cloud Intrusion Detection dataset. iii, vii–x, 6, 49, 50, 85–89, 92, 93, 97, 98, 100–102, 104–111, 118–121, 123

ISP internet service provider. 29, 40

MCC Matthews correlation coefficient. 86, 87, 96, 98, 100, 104, 105, 108, 109, 112–114, 121

MRF Markov random field. iii, 3, 23, 24, 77, 78, 82, 121

MTU maximum transmission unit. 61

NAT network address translation. 29, 32

OS operating system. 30

PGQL Property Graph Query Language. xi, 53–55

PPV positive predicted value or precision. 44, 87, 91, 95, 98, 104

RAT remote access trojan. 8, 41

RMSE root mean square error. 107

ROC receiver operating characteristic. ix, xii, 87, 96–99

SIEM security information and event management. iii, 2, 3, 27, 28

SVM support vector machine. 23

TPR true positive rate or sensitivity. 87, 96

VM virtual machine. 87

Chapter 1

Introduction

1.1 Problem Statement

Large computer networks are continuously facing a diverse range of attacks. Traditionally, these attacks were mostly attacks of opportunity, quickly moving to weaker targets wherever they may be found. However, in the past decade new types of long-term threats have emerged. One notable example is the *advanced persistent threat (APT)*, which is a type of targeted and long-term campaign perpetrated by highly skilled and determined attackers with a very specific objective, namely sabotage or data exfiltration [96]. It gained notoriety with the Stuxnet [31] in 2010. Since then many other APTs have been identified in the wild, like the Flame [8], the RSA hack [85] and the breaches by the Deep Panda group [54].

Each of those attacks had unique objectives, modes of operation and techniques employed. However, there is an important commonality in how adaptive multi-modal (many times using zero-day vulnerabilities) approaches were employed to penetrate, escalate privileges, gain control and maintain presence in high-security networks while evading detection for months or even years all while exfiltrating large amounts of information.

The increased attack complexity brings many extra challenges for detection, prevention and posterior forensic analysis of intrusions. The tools most commonly used for these tasks, the rule-based intrusion detection systems (IDSs), are stateless and usually deployed around the network perimeter and analyze primarily packet headers. They are very capable of identifying and blocking obvious threats and known attacks in real-time, and logging their findings for future human analysis but they have limited capabilities to identify novel attacks, internal threats and compromised hosts, and cannot meaningfully link seemingly innocuous and unrelated events to expose higher level attack patterns. Consequently, these tools are bound to fail when faced with highly customized attack vectors that APTs comprise.

Other, less used, tools like anomaly-based IDSs, try to detect novel attack types but

suffer from high levels of false positives and are still incapable of effectively connecting the dots among intrusions.

Moreover, while the different types of IDSs are valuable tools and their specific strengths can be complementary, they are employed independently of one another, and the integration of their alerts into security information and event management (SIEM) systems is mostly an *ad hoc* process. This, combined with the high number of false positives generated by some of those tools, causes what Sommer and Paxson [94] defined as the semantic gap that security analysts face when dealing with these tools and trying to (1) understand their alerts and the alerts' contexts and (2) clarify how they should act on it. The end result is that proper prevention is very challenging even dealing with real intrusions.

Finally, the forensic analysis of intrusions is also hampered by the high complexity and long-term nature of APTs. Both understanding how the intrusion was accomplished and performing a correct attribution, that is, the identification of the root causes of an attack, are made exponentially more complex in this scenario. Better tools capable of aggregating data, supporting real-time analyses and dealing with large volumes of data and dynamic networks are considered important areas of focus which can lead to improvements in defence capabilities [120, 49, 23]. Moreover, the use of artificial intelligence is considered fundamental to counteract advances in offense capabilities of attackers [122]. If distributed attacks are not correlated, the separate attack instances are disjoint and their sources are individually considered to be the root of each attack. It is only when those seemingly disjoint attacks are identified as part of a complex intrusion campaign that the complete attack can be understood and its individual sources can be identified as simply stepping stones to a larger attack.

Based on the above, the goal of this research is to answer the following research questions:

Research Question 1 *How can network entities be modelled in order to expose higher level attack patterns associated with long-term attack campaigns such as advanced persistent threats?*

Research Question 2 *How can the uncertainty inherent to each piece of data be quantified and integrated in a general network model?*

Research Question 3 *What type of patterns or statistical features can be extracted from a model of network entities that can aid the detection of attacks?*

Research Question 4 *How can alerts from different intrusion detection systems be integrated into a combined detection model in order to leverage their strengths and mitigate their weaknesses and thus improve detection performance?*

1.2 Proposed Approach

To address the aforementioned challenges, a security knowledge graph model, called the Activity and Event Network (AEN), that models the diversity of the actors and objects of a computer network as well as the uncertainty and dynamic nature of those elements as they change through time is proposed.

The AEN graph model is built with data from heterogeneous sources, including those available within the network's security perimeter, such as network traffic logs, flow data, system and application logs and IDS alerts, as well as external data sources obtained through third-party services like domain name system (DNS) queries, WHOIS and geolocation services. Of special note are the data sources that report security events or suspicious activity, such as IDSs and SIEMs. Generically, these are called *detectors*, and their reports or logs are generically called *alerts*. The inclusion of the alerts is done in an integrated fashion which allows the AEN to leverage the strengths and mitigate the weaknesses of the independent detection tools.

In reality, the purpose of the AEN is to serve as a base upon which different detection mechanisms, threat analyses and forensic investigations of both novel and known attack patterns, as well as long-term and stealth attack methods, can be performed.

To validate those capabilities, three unsupervised intrusion detection mechanisms are proposed for the AEN. The first one is a signature-based scheme that employs an isomorphic subgraph matching algorithm to search for graphical attack patterns, called attack fingerprints, in the graph. As a proof of concept, fingerprints for scanning, denial of service (DoS) and password guessing attacks are provided.

The second scheme is an anomaly detection mechanism that involves calculating anomaly scores based on the bits of meta-rarity metric introduced by Ferragut et al. [32] for statistical features and underlying distributions extracted from the AEN graph. In total, 15 features are proposed.

The final mechanism works by deriving graphs akin to Markov random fields (MRFs) from the main AEN graph and performing a probabilistic inference on the derived graphs using a modified belief propagation method. For this reason, the AEN defines a probability model that takes into consideration the uncertainty regarding each piece of data and assigns a probability of correctness value to each element in the graph.

This scheme leverages the alerts from different IDSs that have been inputted into the graph as indicators of compromise (IOCs) with the goal of obtaining better detection performance in comparison to the IDSs by themselves. The alert generated by the other two detectors are also fed back to the model and can be used by the belief propagation mechanism

just like alerts from more traditional IDSs.

Also part of this detection mechanism is a "human-in-the-loop" online parameter adaptation mechanism based on the stochastic gradient descent algorithm, which was conceived with the aim of reducing the initial burden of selecting the system's parameters and allowing for frequent adaptation of parameters in dynamic environments.

In summary, the proposed model is a proactive network threat identification and analysis tool that is capable of identifying common and novel network cyber-attacks either purely through its own detection mechanisms or by integrating and improving upon alerts from other IDSs, and providing the ability to gain insight into malicious actors and related patterns of behaviours, and help expose hidden relationships between the different network elements.

1.3 Research Contributions

The contributions from the research are as follows:

Contribution 1 *A network model capable of identifying long-term behaviour of network entities, exposing higher level attack patterns associated with long-term attack campaigns such as advanced persistent threats.*

Contribution 2 *A probability model that models the inherent uncertainty regarding each piece of data added to model.*

Contribution 3 *A set of graphical attack patterns, called attack fingerprint, and statistical features that are used by two distinct unsupervised intrusion detection mechanisms, one signature-based, which employs isomorphic subgraph matching algorithms to search for known attacks, and one anomaly-based, which uses the bits of meta-rarity metric to identify anomalous behaviour.*

Contribution 4 *An unsupervised intrusion detection mechanism built upon the graph's probabilistic model using a modified belief propagation algorithm which leverages alerts from different intrusion detection systems so that it is able to amplify their strengths and mitigate their weaknesses and thus improve detection performance, and includes a "human-in-the-loop" online parameter adaptation mechanism based on the stochastic gradient descent algorithm which reduces the initial burden of parameter selection and allows for frequent adaptation of parameters in dynamic environments.*

1.4 List of Publications

The following are the publications derived from this research:

[1] Paulo Quinan. Tutorial: Attack graphs in cybersecurity — evolution and practice. 2nd International Conference, ISDDC 2018, 28–30 November 2018.

[2] Nitika Gupta, Issa Traore, and Paulo Magella de Faria Quinan. Automated event prioritization for security operation center using deep learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 5864–5872, February 2020.

[3] Onyekachi Nwamuo, Paulo Magella de Faria Quinan, Issa Traore, Isaac Woungang, and Abdulaziz Aldribi. Arguments against using the 1998 DARPA dataset for cloud IDS design and evaluation and some alternative. In Selma Boumerdassi, Éric Renault, and Paul Mühlethaler, editors, *2nd Machine Learning for Networking (MLN 2019), Proceedings*, pages 315–332. Springer, April 2020.

[4] Paulo Gustavo Quinan, Issa Traoré, Ujwal Reddy Gondhi, and Isaac Woungang. Unsupervised anomaly detection using a new knowledge graph model for network activity and events. In Éric Renault, Selma Boumerdassi, and Paul Mühlethaler, editors, *4th Machine Learning for Networking (MLN 2021), Proceedings*, pages 117–130. Springer, March 2022.

[5] Chenyang Nie, Paulo Gustavo Quinan, Issa Traoré, and Isaac Woungang. Intrusion detection using a graphical fingerprint model. In *22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2022)*, pages 806–813, July 2022.

[6] Paulo Gustavo Quinan, Issa Traoré, and Isaac Woungang. Activity and event network graph and application to cyberphysical security. In Issa Traoré, Isaac Woungang, and Sherif Saad, editors, *Artificial Intelligence for Cyber-Physical Systems Hardening*, chapter 10, pages 217–233. Springer Cham, November 2022.

[7] Paulo Gustavo Quinan, Issa Traoré, Isaac Woungang, Ujwal Reddy Gondhi, and Chenyang Nie. Hybrid intrusion detection using the AEN graph model. *Advances in Science, Technology and Engineering Systems Journal*, 8(2):44–63, March 2023.

[8] Paulo Gustavo Quinan, Issa Traoré, and Isaac Woungang. Improved threat detection through the activity and event network graph belief propagation model. Submitted to *Computers & Security*, 2023. Under review.

[9] Ornella Lucresse Soh, Issa Traoré, Paulo Gustavo Quinan, Isaac Woungang, and Wei Lu. Systematic APT stage prediction using score calibration and multi-class classification. Submitted to *Expert Systems with Applications, Elsevier*, 2023. Under review.

1.5 Outline

The rest of this dissertation is structured as follows.

Chapter 2 summarizes and discusses the literature on APT, on the graph models employed in this research, on intrusion detection using graphical belief models and on belief propagation.

Chapter 3 describes the AEN model, including its data sources, elements, probability model, threat analysis capabilities and metrics, as well as the architecture of the implemented system.

Chapter 4 introduces the two proposed unsupervised detection mechanisms, namely the attack fingerprint matching and the anomaly detection.

Chapter 5 describes the belief propagation detection scheme, including its belief model, propagation rules and the proposed online parameter adaptation method.

Chapter 6 presents the experimental evaluation of the proposed detection mechanisms using two different datasets, the Information Security and Object Technology (ISOT) Cloud Intrusion Detection (ISOT-CID) Phase 1 dataset and the 2017 Canadian Institute for Cybersecurity (CIC) Intrusion Detection Evaluation (CIC-IDS2017) dataset, and discusses the obtained results.

Finally, Chapter 7 contains the concluding remarks as well as discusses the future work.

Chapter 2

Background and Related Work

This chapter provides an overview of key background knowledge underlying the work presented in this dissertation and also summarizes and discusses related work.

2.1 Background on Advanced Persistent Threats

The term advanced persistent threat (APT) was coined by the United States Air Force around 2006 as a general, unclassified, moniker used to describe state-sponsored organizations capable of perpetrating attacks against military, governmental or otherwise sensitive and strategic assets [7]. In [7], Bejtlich describes the words comprising the term APT as follows:

- **Advanced** means the adversary is highly skilled and can adapt to counter its target defences crafting custom exploits and utilizing zero-day vulnerabilities if required.
- **Persistent** means the adversary is not an opportunistic intruder, but instead has a specific objective and, therefore, is willing to engage in a long-term campaign and will work to maintain presence as long as required to accomplish its objective.
- **Threat** means the adversary is organized, well-funded and motivated, and, therefore, consists of real threat even to well defended assets.

Through time the term has evolved from being a proper noun used for specific, known, actors to describe the class of attacks perpetrated by actors with the same characteristics as them. It also acquired a broader scope as it began to be used in the private domain, most commonly for intellectual property theft [19].

Some authors distinguish between the broader "targeted attacks" and the nation-state specific APTs [95], while others use terms like advanced targeted attack (ATA) [58] but only as an equivalent of APT. In practice, the APT moniker is applied regardless of the

target and of the actors responsible for the attack as long as the attackers, per NIST's [96] definition: "(i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives".

Despite being highly variable and adaptive, APTs follow a similar life cycle [41, 35, 61, 95, 19, 114]. Different authors propose different numbers of phases, but they mostly converge in the general life cycle including the following phases:

- *Reconnaissance*: This is the starting point of any attack. In this stage the attackers gather the maximum amount of intelligence possible about the target, its resources, personnel and relationships so they can identify possible points of weakness that can be leveraged;
- *Weaponization*: After identifying possible attack vectors, the attackers start to prepare and test the tools for the attack. Depending on the case, the attackers can use publicly available malware, purchase zero-day vulnerabilities from the dark web, or even build their own tools in the most advanced cases. With the tools in hand, the attacker test them for detection against replicas of the target environment, and assess their risk of detection and retaliation;
- *Delivery*: In this phase the attackers deliver their exploits to the target. The two most common delivery methods are spear-phishing and watering-hole attacks [36]. For web-based attacks to be successful, they need to exploit a vulnerability in an application, usually a PDF reader, browser or Microsoft Office suite programs. For air-gaped networks, USB sticks and other forms of physical media may be infected and used as delivery. If possible, bribed or disgruntled employees, or double-agents may be used as well;
- *Foothold Establishment*: Once the malware is delivered and installed it tries to take control of the infected system and initiate command and control (C&C) communication using remote access trojans (RATs), rootkits and other means. In [41], Hutchins et al. split this phase into Exploitation and Installation;
- *Progression*: Once inside, the attackers start working towards expanding their control over the network. Having access to a single computer is a precarious position since this machine might be moved, shut-down or wiped clean, and besides, the infected machine might not be be the final target of the attack. With that in mind, the intruders try to move laterally to other machines so they can open extra doors inside and gain access to more important resources. This phase works in a cycle of internal

reconnaissance, lateral movement and access escalation, with some authors, like [61, 19], actually splitting this phase likewise;

- *Mission Realization*: Once the attackers reach their target they can start fulfilling their mission. Usually that involves data exfiltration through their open control channels, but might also involve sabotage as in the case of Stuxnet.
- *Cover tracks*: Finally, to avoid detection and attribution, the attackers try to cover all their tracks by deleting logs and any unnecessary file and malware installed. In practice, this phase is continuously happening throughout the whole campaign so the attackers can remain undetected for as long as necessary.

Based on the APT life stages, Hutchins et al. [41] proposed the *intrusion kill chain* model to guide defenders in developing mitigations against APTs. It works by creating a "courses of action" matrix with incrementally stronger mitigations designed to detect, deny, disrupt, degrade and deceive the intruders at each chain stage.

The premise behind the model is that the adversaries must progress through each stage of the chain (life-cycle) before they can reach their objectives. Therefore, blocking a single link will thwart the attack in course and force the attackers to evolve. Then, with the knowledge of the thwarted attacks, the defenders can learn about the attackers, how they operated and then evolve as well.

The intrusion kill chain model strength is demonstrated by several authors, such as Ioannou et al. [43] and Bhatt et al. [9], who used it as a conceptual framework for APT modelling and detection.

Another modelling framework based on the same life-cycle principle is the *attack pyramid* proposed by Giura and Wang [35], which extends the attack tree model [5, 88] by adding a third dimension, thus allowing the modelling of a multi-stage attack towards a common goal at the top of the pyramid.

2.2 Related Work on Advanced Persistent Threat Detection

The characteristics of APTs make their identification very complex. Many counter-measures exist but are not always enforced or effective [109]. Traditional rule-based IDSs are very capable of identifying and blocking obvious threats and known attacks but have very little chance of identifying the multi-stage, custom crafted, attack vectors that characterize APTs.

This has led to increased interest in anomaly-based IDSs, which are capable of detecting novel attack types but suffer from high number of false positives and are still incapable of

effectively connecting the dots among intrusions. Sommer and Paxson [94] argue that a mismatch between the capabilities of machine-learning and the type of analysis required for intrusion detection has limited the improvement of those systems and that limitations in the descriptive capabilities of these tools have curtailed the understanding of alerts by security analysts creating a "semantic gap" which further limits their adoption. The authors also proposed a set of guidelines for applying machine learning to network intrusion detection.

Even so, more and more research are being performed in anomaly, misuse or behavioural based IDSs given the limitation of rule-based IDSs in identifying novel threats. This trend was confirmed by Luh et al. [58]'s extensive review of works in APT detection.

The review provided several other interesting findings. For instance, it showed an almost even split between malware detection, host intrusions and network intrusions. That is explained by the multi-modal nature of APT attacks, which require a varied set of equally-important tools, each covering different intrusion stages. Consider for instance the Delivery stage, which requires host-based tools to prevent or at least detect infection, and the Operations stage, which requires tools capable of identifying unauthorized network movement and C&C communication.

The paper also showed that the majority of the works were focused on a single area of detection, be it host-based or network-based, with only a few utilizing multiple sources of data. That demonstrates a gap in the literature given what was discussed in the previous paragraph and the importance of linking data from multiple sources to identify hidden patterns of compromise among them.

Nevertheless, many novel APT detection and prevention techniques have been proposed or developed in the past few years.

Giura and Wang [35] proposed a new anomaly-based detection framework, conceptualized on the attack pyramid model created by the same authors. The system uses data from several different network and application logs to categorize intrusion events into the pyramid's planes (life cycle phases) and correlate events across the planes to detect APTs.

A similar proposition, but conceptually based on the intrusion kill chain model, was demonstrated in Ioannou et al. [43] and Bhatt et al. [9]. However, Ioannou et al. [43] used Markov belief models to manage conflict within its data and tried to predict the intruder's next steps, while Bhatt et al. [9]'s model used a rule-based approach and had no predictive capabilities.

Focused on detecting the initial intrusion, Moon et al. [67] used decision tree based behavioural analysis to detect the host infections. Their model employed mostly host based features, but included some network flow features as well.

Finally, Levchuk et al. [56] introduced a model that utilized netflow data to generate

graphs of host network behaviour, and then employed graphical pattern matching techniques to match them against attack fingerprints and identify deviations of known behaviour.

An important focus besides detection is attribution. Although not focused on APTs, Shamsi et al. [90] defined three levels of attribution that are increasingly difficult to achieve: identification of the cyberweapon, identification of the origin (country or city) of the perpetrator, and identification of the actual person or organization that perpetrated the attack. The authors also argued that attribution is possible but is mainly probabilistic, particularly for the latter two levels for which the required evidence is harder to obtain because of the efforts taken by attackers to hide their tracks and of the limited collaboration between stakeholders in many cases.

On that topic, Clark and Landau [22] demonstrated that attribution is particularly problematic for multi-stage attacks. As a consequence, attribution is a highly manual process with limited levels of automation, particularly for multi-stage attacks such as APTs.

2.3 Uncertain and Dynamic Graphs

2.3.1 Uncertain Graphs

Uncertain graphs, also known as probabilistic graphs, are graphs whose structures are not deterministic, that is, where the elements have a *probability of existence*. Their utility arise from the fact that real-world data are very commonly uncertain for two main reasons: (1) Some domains are intrinsically uncertain or probabilistic, like a telecommunication network where connections are unreliable, and therefore, might be unavailable at any given moment; (2) The data from which the graph is constructed is known, or likely, to be imperfect given its collection procedure. In some cases both reasons apply. For instance, [89] modelled protein interaction data which not only are probabilistic by nature but may also be noisy or incorrect given these are obtained from error-prone lab experiments.

The most basic form of uncertain graph is $G = (N, E, p_E)$ where N is a set of nodes, E is a set of edges, and $p_E : E \to (0, 1]$ is a function which assigns existence probability values to edges. That is the traditional use of uncertainty, with it applying only to edges on an *edge-uncertain graph*. However it is possible to define any graph element as uncertain. Zou et al. [126], for instance, defines an uncertain graph where both its nodes and edges are uncertain, or formally $G = (N, E, p_N, p_E)$. In this case p_N assigns the existence probability values to nodes while p_E assigns *conditional* existence probability values to an edge given that both its vertices exist. A similar construction would apply to labels.

Semantically, an uncertain graph G can be understood as a world generator where a

number of "possible worlds" G' can be obtained (some authors use derive, imply or implicate) from it based on all permutations of existence of its uncertain elements. Formally, for an edge-uncertain graph, $G' = (N, E') \sqsubseteq G = (N, E)$. In this case there are $2^{|E|}$ possible worlds where their probabilities are defined by:

$$P(G'|G) = \prod_{e \in E'} p_E(e) \prod_{e \in E \setminus E'} (1 - p_E(e)) \tag{2.1}$$

For other types of uncertain graphs, the only adaptation required is to operate over the other uncertain elements according to their conditional existence, that is, to only include conditional elements whose dependencies exist in that world.

When uncertainty is added, many graph problems gain an extra layer of complexity. For instance, finding the shortest path between two nodes is technically impossible when it is uncertain whether any given path exists or not. That means many problems need to be reframed to take into consideration element probabilities and the multiple "worlds" that these entail, and, consequently, new algorithms need to be developed to satisfy them.

Going back to the shortest path, some techniques might select the shortest path above a certain probability threshold while others utilize distance functions which calculate the median distance weighed according to probability distribution of each possible world [81].

Considering the exponential nature of the cardinality of the set of multiple worlds of any given uncertain graph, most of those problems are not trivial and may, in some cases, be intractable. To reduce algorithm complexity, even by giving up some accuracy, many authors [81, 126, 127, 47, 75, 12] have proposed heuristic approaches where subgraphs are sampled according to specific parameters or structures. In these cases, sampling works to prune undesired search paths, reducing the search space, and so on.

Of special interest for this work, [18, 126, 127, 75] proposed different heuristics to discover frequent subgraphs by identifying isomorphic subgraphs through encoding of nodes, edges and paths which allowed them to be sorted into more efficient structures and more easily pruned.

Finally, on the topic of clustering and sparsification of uncertain graphs, [102] filtered edges based on the most probable path. Following the same technique, [12] proposed a clustering technique that identifies the most probable clusters of an uncertain graph. Following a different approach, [76] proposed a sparsification heuristic capable of redistributing the probabilities of the pruned nodes, thus maintaining an approximate equivalent graph structure.

2.3.2 Dynamic Graphs

Dynamic graphs, also known as temporal, time-varying or evolving, are graphs that change with time. They can be used to model any kind of dynamic system, which exists in almost all fields of science, but relatively little is known about them when compared to static graphs.

Given the dynamic nature of computer science programs, dynamic graphs are intrinsically part of the field and are extensively used in it, albeit usually without consideration or knowledge of the underlining theoretical framework.

Dynamic graphs were first formalized by Harary and Gupta [40]. The authors provided a taxonomy of dynamic graphs according to the dynamicity of its elements, so a *node-dynamic graph* is one where the nodes are dynamic and an *edge-dynamic graph* is one where edges are dynamic.

As with uncertain graphs, the extra temporal dimension radically transforms many graph properties and problems. Casteigts et al. [17] and Michail [63] provide overviews of those differences. These two papers compiled much of the general concepts and formalisms of dynamic graphs into concise theoretical frameworks.

Formally, dynamic graphs can be defined in two different ways. The first one is to define the dynamic graph $G = (V, E)$ as a series of static subgraphs $(G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k))$ where each subgraph represents a slice of time. This concept allows for the modelling of many time properties like change and for the application of any static graph analysis on each time slice, like done in [18, 87]. However, it can only work with discrete events and consequently cannot be used to completely study certain chronological relationships [17].

That gave rise to the second, more general, definition, per Casteigts et al. [17], where each dynamic element of the graph exists over a time span $\mathcal{T} \subseteq \mathbb{T}$ through the presence function $\rho : E \times \mathcal{T} \to \{0, 1\}$. This definition can model both discrete-time systems, by defining time domain $\mathbb{T}$ as $\mathbb{N}$, and continuous-time systems by defining it as $\mathbb{R}^+$ instead. The paper also defined a latency function $\zeta : E \times \mathcal{T} \to \mathbb{T}$ which describes the latency of the element. It follows that an edge-dynamic graph can be defined as $G = (V, E, \mathcal{T}, \rho_E, \zeta_E)$.

A temporal concept of special interest is the *journey*, formalized by Bui-Xuan et al. [13], which can be defined as a time-respecting walk through a path over time [63]. Formally, given graph G, a journey can be defined as $\mathcal{J}(u, v, \sigma) = \{R(u, v), \sigma\}$ [13], where:

- $u, v \in V$.
- $R(u, v) = \{e_1, e_2, \ldots, e_k\},\ e_i \in E,\ s(e_1) = u,\ t(e_k) = v,\ t(e_i) = s(e_{i+1})$, that is R is a route, or path, in G starting at u and ending at v.

- $\sigma = \{t_1, t_2, \ldots, t_k\}, t_i \in \mathcal{T}, t_i \leq t_{i+1}$ is the time schedule indicating when each edge in R is to be traversed such that e_i is traversed at time t_i.

From the above definition, it is clear that journeys are constrained by the chronology of edges and nodes, and can only exist when each subsequent edge exist after the previous edge was traversed. In other words, they define how, and if, an entity can navigate a graph through time while taking into consideration the existence of nodes and edges at any given moment as well as their latencies. Journeys, therefore, can be used to model the flow of information across the system being modelled and whether a path is both spatially and chronologically valid.

This chronological constraint allows journeys to be categorized into *direct* and *indirect* [16]. Journeys are direct if and only if each subsequent edge already exists as soon as one edge is traversed. Otherwise, journeys are indirect. This means that indirect journeys require at least one waiting period in a node along the way.

There are two dimensions in which journeys can be measured: Topographical and chronological or temporal. This entails two length measures [13, 17]. The first is the *topological length*, or hop-count, which is defined as $|\mathcal{J}| = k$, that is, the number of hops from the starting node to the end node. The second one is the *temporal length*, or *duration*, which is defined as $\sigma(e_k) - \sigma(e_1)$, that is, the elapsed time between the traversal of the first edge to the traversal of the last edge.

There are several other intrinsic concepts to journeys such as eccentricity at a given point in time [13], reachability, connectivity over time, round connectivity, recurrent connectivity [17], availability [63], among others. All of these concepts are variable through time and can be calculated for arbitrary times and nodes as long as the the whole temporal history is maintained. That presents a problem when dealing with large, or long-lived, graphs upon which those calculations might become untractable. To alleviate this problem Kostakos [52] proposed some techniques to calculate proximity and availability averages that would be useful to provide a general understanding of the graph structure without maintaining the whole history. Those techniques provide insight into how other aggregates can be calculated through time.

Among those concepts, the *reachability*, formalized by [17], is of special interest for this work. Given two nodes u and v of graph G, v is reachable from u if there exists a journey between them. Formally, $u \rightsquigarrow v \iff \exists \mathcal{J}_{(u,v)} \in \mathcal{J}_G^*$, where $\mathcal{J}_G^*$ is the set of all possible journeys in G. Further, the reachability can be constrained in time by restricting $\mathcal{J}_G^*$ to a time slice $[t_1, t_2)$.

A final concept of interest is the *horizon* of a node, which is defined as the set of all nodes reachable from it [17]. Formally, given a node u, its horizon is defined as the set

$\{v \in V : u \rightsquigarrow v\}$.

Another important aspect of dynamic graphs is how it can be used to model real-time systems that continuously receive data. That, of course, presents many problems when online data mining is desired. On this topic, Chen and Wang [18] proposed some interesting techniques on how dynamic graphs (called graph streams by the authors) can be encoded and structured to reduce the computation required to perform continuous subgraph pattern searches as the graph changes.

Chen and Wang [18] also demonstrated how dynamic and uncertain graphs can be combined so that at any point in time the probability of existence of an element is not binary, but is, instead, probabilistic. By building upon Casteigts et al. [17]'s dynamic graph definition above, it is possible to redefine the time presence function ρ so it is combined with the uncertain existence function p so $\rho : E \times \mathcal{T} \rightarrow (0, 1]$. The general *edge-uncertain dynamic graph* definition would follow the regular dynamic graph as $G = (V, E, \mathcal{T}, \rho_E, \zeta_E)$.

2.4 Attack Graphs

Attack graphs are graphical representation of the security state of a system and the vulnerabilities which an attacker can exploit, or have exploited, in order to compromise it. They draw their strength from the capability of correlating multiple independent threats in order to expose interconnections that generate vulnerability chains which can threaten the whole network. This allows attack graphs to be used in many areas of network security, from threat modelling to forensic analysis.

Traditionally, attack graphs were generated manually by seasoned security analysts which would gather information about the network configuration and topology and the known vulnerabilities affecting the network, and by utilizing their knowledge of common attacks and patterns of interaction between vulnerabilities would laboriously compose the attack graph for later analysis [57].

Typically, vulnerability scan reports would be used to identify exploitable vulnerabilities although in some cases vulnerability databases would be used directly [4, 46]. In any case, this information only identifies known vulnerabilities and does not consider the possible interaction between them. To address that, the analyst would also need to be capable of identifying possible threats to the network components and, most importantly, the interactions between those vulnerabilities, which requires a high degree of expertise.

Manual generation can be suitable for small networks, however, when dealing with even medium sized networks the process becomes tedious, error-prone and generally impractical [69]. Consider for instance a 10-machine network where each machine contains 5 vulnerabil-

ities. To generate the graph, an analyst would have to keep track of all the combinatorial relationships between those 50 vulnerabilities. Another limitation is that analyses are more difficult given the semi-structured nature of the generated graph.

As an example, Figure 2.1, gives a visual perception of the underlying complexity of an attack graph of a very small 5-machine network with only 5 vulnerabilities per machine. In it, the nodes represent the possible states of the network, with the source node at the top being the initial state and the sink node at the bottom being the goal of the attack, while the edges represent the transitions between two states stemming from the exploitation of a vulnerability. Each new machine or vulnerability would make the graph exponentially bigger and denser.

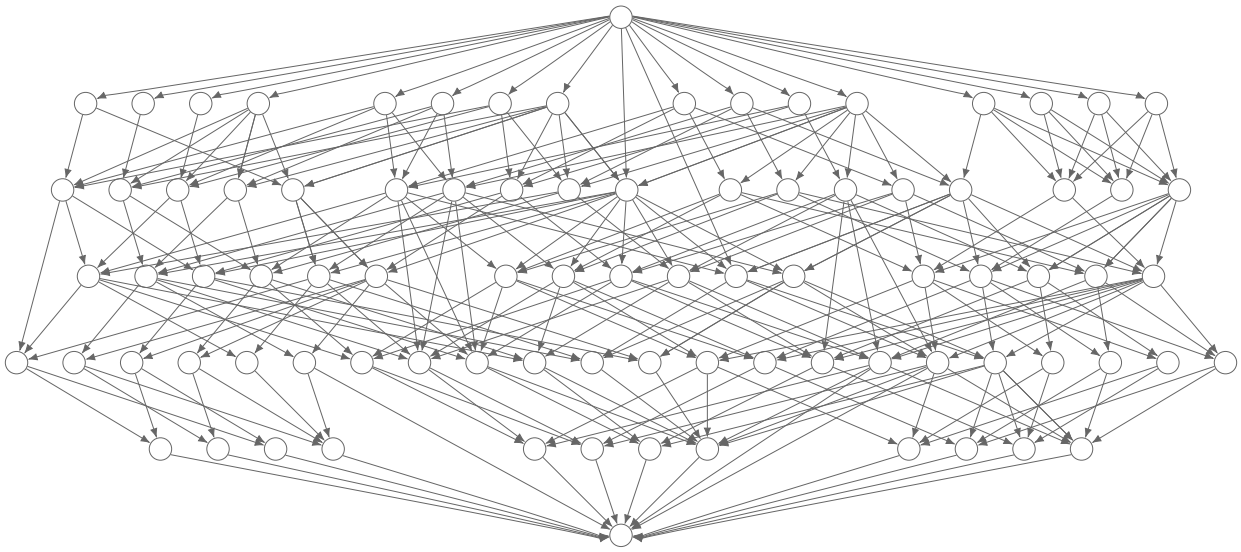


Figure 2.1: Attack graph example of a 5-machine network and 5 vulnerabilities per machine

To overcome those limitations many automatic generation algorithms and tools such as the Attack Graph Toolkit [93], MulVAL [72], NetSPA [6] and TVA [44] have been developed over the years. Those tools work with similar data as before, however, the data now needs to be formatted according to what the tool can understand. Network configuration and topology information are encoded into well structured files, vulnerability scan reports are used to identify real vulnerabilities while common attack patterns are encoded into reusable templates.

The automated generation is easier, faster and less error-prone depending only on the quality of the data collected. Because of the use of formal model checkers it is theoretically possible to generate attack graphs that are exhaustive (one which contains all possible attack paths) and succinct (one which does not contain attack paths which cannot be successful)

given a complete input data [46].

There are many analyses that can be performed on attack graphs. As part of the threat modelling of a network system, it is possible to perform risk analysis of a specific asset, that is, identify what kind of impact a service has on the overall security of the network [46, 70]. That, in turn, allows for a general network hardening analysis and effort guidance where each element and desired change to the network can be assessed according to its impact on the general security of the network so a high effort, low impact issue can be marked as less important even when by itself the issue is considered more critical than others.

Several risk analysis metrics have been proposed based on probabilities or level-of-effort estimations of vulnerability exploitation and attacker capabilities:

- Most likely attack path [53]
- Attack success likelihood [70]
- Most damaging vulnerabilities [46]
- Weakest adversary required to compromise a network [74]
- Resilience to zero-day attacks [112]

Another possible use of attack graphs is the correlation of real time IDSs alerts with the state of the network identifying how far into the attack chain is an attack. This allows for the prioritization of the analysis of the alerts, so alerts correlated to states more likely to result in a compromise of the network can be analyzed before others [124, 27].

A third analysis use case is forensics. After an attack has happened, the security logs can be analyzed to identify the vulnerability chain exploited by the attacker so the attack can be retraced. This provides the analyst with a better understanding of the attack and allows for faster defence measures to be taken [4].

There are many different types of attack graphs, each one aims to either elucidate different aspects of the security issues faced by the system or at least to fix limitations of the previous one.

The first type of what can now be understood as an attack graph was the **privilege graph**, first proposed by Dacier [26]. This type of graph focuses on privileges obtained by an attacker in a host and on the vulnerabilities that allow the attacker to gain those privileges. In it, each node represents a the attacker's set of privileges and the edges the vulnerabilities.

Privilege graphs served as a base for the first real attack graph, the **state attack graph**, which was first proposed by Phillips and Swiler [80] and later formalized by Sheyner et al. [92].

State attack graphs represent different ways in which an intruder can reach a certain goal, such as gaining root access on a host. Its nodes identify a security state of the system, for example the machines the attacker has accessed, reachability conditions, the user privilege level obtained among others, with the leafs being the attack goals. Its edges represent attacker actions that lead from one state to the next. Figure 2.2 illustrates a simple attack graph with 3 machines (represented by numbers) and 4 vulnerabilities (represented by letters).

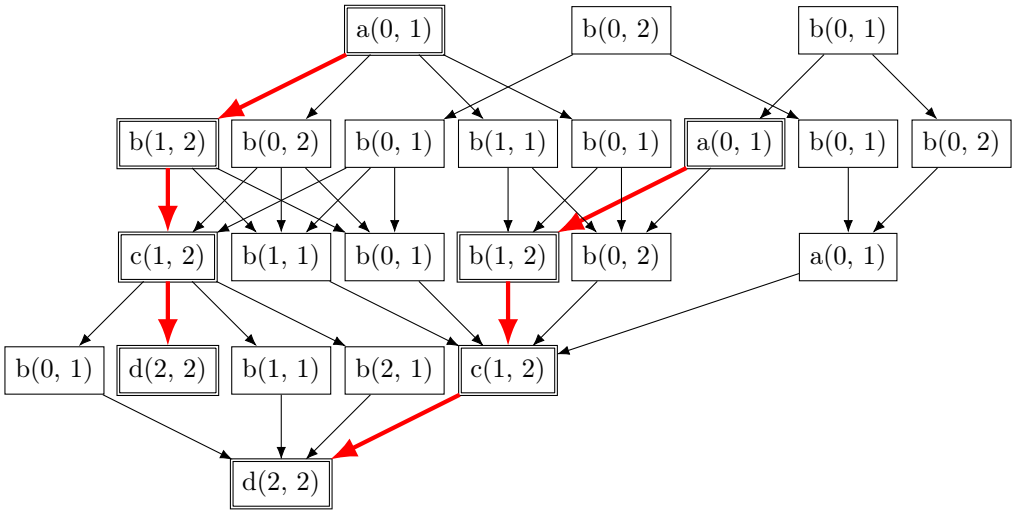


Figure 2.2: State attack graph using the state enumeration representation with 3 machines (represented by numbers) and 4 vulnerabilities (represented by letters). The two thick red paths represent the same attack path.

Formally, they are defined by Sheyner et al. [92] as $G = (S, \tau, S_0, S_S, L)$, where:

- S is a set of security states;
- $\tau \subseteq S \times S$ is a transition relation;
- $S_0 \subseteq S$ is a set of initial states;
- $S_S \subseteq S$ is a set of success states, *i.e.*, the intruder's goals;
- $L : S \to 2^{AP}$ is a labelling of states with a set of propositions true in that state.

As source, it uses the same data mentioned above like network configuration and topology information, database of common attacks patterns and an attacker profile.

One good characteristic of the state attack graph is that attack paths correspond to paths in the graph, which greatly simplifies the analysis. However, the easiness of analysis of this representation has a down side, which is the exponential growth of the graph according to its vulnerabilities and network size, making generation intractable even for a few dozen of

hosts. Figure 2.2 shows this well, where the two thick red paths show the same attack path, which is represented twice in the graph.

Because of that, this original representation, called the state enumeration, was superseded by representations with smaller generation complexity. These other representations are based on the premise that the attacker's control over a network increases monotonically [4, 71], meaning, that the attacker will not relinquish any access or privilege obtained in order to advance the attack. It is this premise that allows states, and consequently chains, to only be represented once in an attack graph, thus greatly simplifying the graph itself and its generation complexity.

One example of other representation is the exploit dependency, where each state is dependent on one or more exploits, meaning they only need to be represented once. Figure 2.3 shows the exploit dependency representation of the same scenario as Figure 2.2. Note how in this case, the same attack path as the previous figure is only represented once.

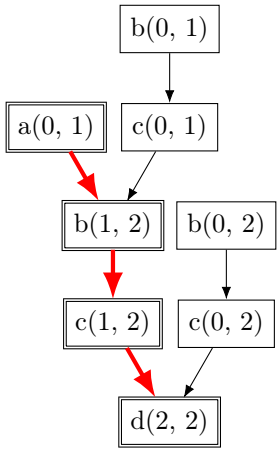


Figure 2.3: State attack graph using the exploit dependency representation with 3 machines (represented by numbers) and 4 vulnerabilities (represented by letters). Thick red lines represent the same attack path as the previous figure.

The complexity of the graph generation algorithm is now quadratic instead of exponential. As a consequence the order of an exploit dependency graph is much smaller than the respective state enumeration graph. Considering again a network with 10 machines and 5 vulnerabilities, the state enumeration graph can have close to a hundred thousand nodes, while the exploit dependency graph reduces that to a few thousand.

Wishing to further reduce graph complexity, Ou, Boyer and McQueen [71] proposed the **logical attack graph**. More than a different representation of the state attack graph, the logical attack graph is a different type of attack graph which represents the logical dependencies among attack goals and network configuration instead of network states. It aims to elucidate the reasons why an attack can happen instead of how it can happen.

Logical attack graph scales quadratically as the network increases and, as such, are much more efficient than the state enumeration representation [71].

Its nodes are logical statements that only encode some aspects of the network. There are two types of nodes:

- Fact nodes, which represent the conditions that must exist for an action to happen. There are two types of fact nodes:
  - Primitive facts, which represent basic facts of the network and its components.
  - Derived facts, which represent facts that are derived from other facts based on specific derivation rules.
- Derivation nodes, which represent the derivation rules between facts, that is, the actions required by the attacker to change one aspect of the network. Those would be the edges of the state attack graph. In practice, they define the reasons why a derived fact would become true.

Contrary to the state attack graph, in the logical attack graph the attacker's goal is the root of the graph while the primitive facts are the leaves.

The logical attack graph edges specify the dependency, or causality, relations between nodes, however there is a distinction between the type of dependencies of the different node types. On one hand, a derived fact node can be caused by any of its children derivation nodes independently, while, on the other hand, a derivation node depends on all of its children nodes to be true so its rule can be satisfied.

Figure 2.4 shows an example of a logical attack graph in which primitive fact nodes are represented by small solid circles, derived fact nodes are represented by large circles and derivation nodes are represented by rectangles.

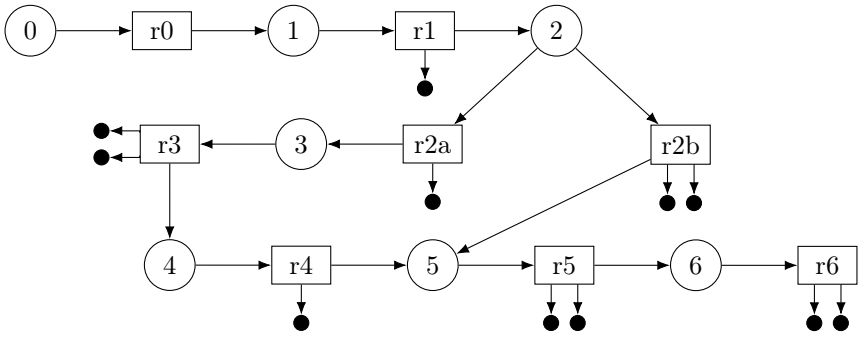


Figure 2.4: Logical attack graph example. Primitive fact nodes are represented by small solid circles, derived fact nodes are represented by large circles and derivation nodes are represented by rectangles.

Another type of attack graph is the **multiple-prerequisite attack graph**, first proposed by Ingols, Lippmann and Piwowarski [42] with the goal of reducing graph complexity even further. It explicitly represents the prerequisites for an attack along with the vulnerabilities and possible states of the host in the network.

The multiple-prerequisite graph has the following three node types:

- State nodes, which represent an attacker's level of access on a particular host.
- Prerequisite nodes, which represent required capabilities of an attacker, either a reachability group (can reach) or a credential (can access), so a vulnerability can be exploited.
- Vulnerability instances, which represent a particular vulnerability on a specific service of a specific host.

Edges represent the causality between nodes, specifically: edges from state nodes always lead to the prerequisites an attacker is able to obtain from it; edges from prerequisite nodes point to the specific vulnerability instances that require that prerequisite to be successfully exploited; edges from vulnerability instance nodes lead to the state that the attacker can attain by exploiting that vulnerability.

Contrary to the previous attack graph types, there is only one node per each state, prerequisite or vulnerability instance, that means the multiple-prerequisite attack graph scales linearly as the number of network elements increase. That means the generation process is much more efficient than before. On the other hand analysis is harder given the more complex graph structure with loops and more node types.

Figure 2.5 shows an example of a multiple-prerequisite attack graph in which state nodes are represented by circles, prerequisite nodes are represented by rectangles and vulnerability instances are represented by triangles.

As it can be seen from the graph types discussed above, there are many attack graph variations. The same is true for the tools that generate those graphs. The most famous are:

- **Attack Graph Toolkit/Scenario Graph Toolkit** [93]: Open source tool developed at Carnegie Mellon University. It generates state attack graphs and was last updated in 2005;
- **MulVAL** [72]: Open source tool developed at Kansas State University. It generates logical attack graphs and was last updated in 2014;
- **NetSPA/FireMon** [6]: Initially developed at MIT as NetSPA, it was then folded into FireMon. It generates multiple-prerequisite graphs. Attack graph generation is only part of the product;

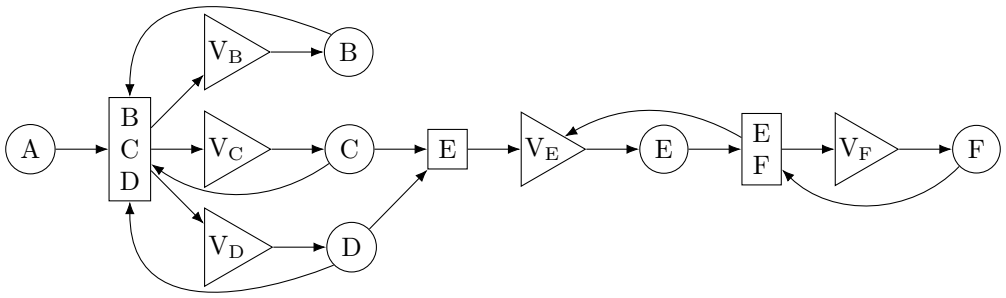


Figure 2.5: Multiple-prerequisite attack graph example. State nodes are represented by circles, prerequisite nodes are represented by rectangles and vulnerability instances are represented by triangles. All prerequisite nodes are reachability (can reach) nodes.

- **TVA/Cauldron** [44]: Initially developed at George Mason University as TVA, it was then turned into Cauldron. It generates penetration dependency graphs and some aggregation attack graphs. Attack graph generation is only part of the product.

As can be noted from the list above, there is a lack of open-source and modern tools, with only some commercial tools supporting attack graph generation.

In general there are many open challenges when working with attack graphs. The first one is the collection of data. Manual generation of attack templates, threat databases and attacker profiles makes generation time-consuming and thus only done sporadically [46, 69]. The commercial tools mentioned above try to improve on that by collecting data automatically.

The second issue is the amount of different graph types, which make it more challenging for security analysts to start learning the details of each variation and utilizing them. Tangentially, the limited option of tools constrain usage of the attack graph concept.

Finally, scalability and complexity of analysis are still very important issues. Even the multiple-prerequisite graph, which only grows linearly, can become very big when dealing with larger networks. In those cases, graphical representation is difficult and consequently manual analysis is hard. That means better graph aggregations and automated analyses are necessary in order to improve defences capability. Specifically, real-time analysis, dealing with large volumes of data and dynamic networks are considered important areas of focus of research [120, 49, 23].

2.5 Intrusion Detection using Graphical Belief Models

Graphical belief-based models have many applications. Particularly, several such models have been proposed for intrusion detection employing a variety of mechanisms to deal with uncertainty and identify intrusions.

Bayesian networks (BNs) and MRFs are commonly used for signature-based intrusion detection [37, 45, 117, 115, 60], where some predefined features are modelled as the network, and a training phase is required to extract the network's parameters, meaning the conditional probabilities, making them fully supervised. Being signature-based, these systems provide good results for the modelled attacks, but they are unable to identify novel attacks. The usage of probabilistic graphical models for anomaly-based detection is less common. As an example, Tylman [105, 106] proposed a model that uses different BNs for both misuse and anomaly detection and is integrated with Snort; however, no performance results were provided.

Neural network models are also commonly used, with a recent focus on deep learning models. Several studies have reported good performance with deep belief networks (DBNs) [34, 125, 113], but these models require large amounts of data and time for training. Zhang *et al.* [123] tried to alleviate this problem by building smaller support vector machine (SVM) models by slicing the data using sliding window for near real-time identification of intrusions and by using DBN for classification of the attacks. Two common issues with these works is their limited capabilities of identifying novel attacks and the effort required for the optimal selection of the network parameters and the fine-tuning of the DBN weights with several different algorithms employed in these tasks.

Another common limitation of all the works mentioned compared to the one in this thesis, is that their modelling of statistical features from network data or alerts, instead of the network itself, makes them unable to expose associations between hosts and other actors.

2.6 Belief Propagation

Belief propagation is a message-passing algorithm for performing statistical inferences on graphical models, where nodes represent the discrete random variables and edges represent the conditional dependencies between the nodes [78, 79]. The degrees of beliefs of the values (or states) of each variable, which correspond to marginal probabilities, are computed based on the neighbours' own beliefs and their conditional probabilities, which maintain the locality of the calculations.

The algorithm was conceived for directed acyclic graphs (DAGs), specifically BNs, on

which it computes exact marginals. Since then, variations of the algorithm, most famously the loopy belief propagation algorithm, have been used successfully in many other probabilistic graphical models, including those with cycles, like MRFs and factor graphs [119], in which case it only approximates marginals [68]. There are also similar algorithms that independently have been proposed in different fields, with different objectives. Those include the forward-backward algorithm, the Viterbi algorithm [110] and the Kalman filter [48].

One limitation of the original algorithm is that computing exact marginals is computationally expensive and has been shown to be an NP-hard problem [24], making it intractable for large graphs. Moreover, some cycles may cause the algorithm to fail to converge or produce wrong results. To alleviate these problems, certain techniques can be employed, such as clustering, conditioning the nodes to remove loops or performing stochastic simulations [79]. Examples of such techniques are the junction tree algorithm [55] and the Kukichi's cluster variation method [50].

Generally, to perform inference, the network has two types of variables: the observed variables, also known as evidence, which are variables whose probability distribution is known (observed), and the hidden, or latent, variables, which are inferred from the observed variables. The influence one variable exerts over another is based on their conditional probabilities, while the initial beliefs of hidden variables are based on their a priori probabilities (referred to as priors), which help guide and regulate the inference calculation. In some variations of the algorithm, those two types of probability are generalized as potential and compatibility functions [119].

Both the priors and conditional probabilities are predefined or otherwise learned externally. For instance, they can come from expert knowledge, locally derived from assumptions about the input data, calculated from other datasets or a combination of methods.

To define the message passing equations, uppercase letters, such as X, are used to refer to a variable, and the corresponding lowercase letters, in this case x, are used to refer to a state of that variable. It follows that the probability that variable X is of value x given the available evidence $\mathbf{e}$ is defined as $P(X = x \mid \mathbf{e})$, or using a shorter notation, $P(x \mid \mathbf{e})$.

The evidence $\mathbf{e}$ can be decomposed into $\{\mathbf{e}^+, \mathbf{e}^-\}$ where $\mathbf{e}^+$ is the evidence of x contained in the descendants of the node and $\mathbf{e}^-$ is the evidence of x contained in the ancestors of the node. Based on that, and applying Bayes' theorem, the belief function is thus defined [79]:

$$\begin{aligned} BEL(x) &\triangleq P(x \mid \mathbf{e}) \\ &= P(x)P(\mathbf{e} \mid x) \\ &= \alpha P(x \mid \mathbf{e}^+)P(\mathbf{e}^- \mid x) \end{aligned} \tag{2.2}$$

where α is the normalization constant rendering $\sum_x BEL(x) = 1$.

Basically, the belief of a state x combines the support, or influence, of its parents, $P(x \mid \mathbf{e}^+)$, which are modulated by the conditional probability tables (CPTs) of the neighbouring variables given x, and the influence of its children, $P(\mathbf{e}^- \mid x)$, which are modulated by the CPTs of x. It follows that these influences are what must be propagated between nodes. The messages a node passes to its neighbours are derived from those components which, in turn, are calculated from the messages received in the previous iteration.

There are two types of messages, one sent from parents to children and one sent from children to parents. Moreover, one message consisting of a vector of support values from each of the children states is sent for each state of the parent variable. Given three nodes X, Y and U such that X is a child of Y and a parent of U, the message sent by X to Y is notated as $m_{Y \leftarrow X}(y)$ while the message X sends to U is notated as $m_{X \rightarrow U}(x)$. Note the arrow, which points to the direction of the message, with the parent always on the left side. To help understanding, Figure 2.6 shows the messages sent between neighbours in a small graph.

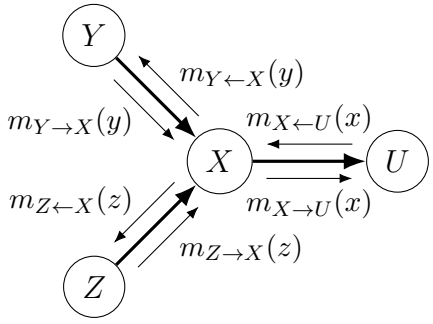


Figure 2.6: Example of the messages passed between neighbours by the belief propagation algorithm. Each variable computes the belief of each of its states based on the messages received for each state.

The message X sends to its child U for state x is defined as

$$m_{X \rightarrow U}(x) = \alpha \prod_{Z \in Chl(X) \setminus U} m_{X \leftarrow Z}(x) \sum_{\mathbf{c}} P(x \mid \mathbf{c}) \prod_{Y \in Par(X)} m_{Y \rightarrow X}(y) \tag{2.3}$$

where $Chl(X)$ is the set of children of X, $Par(X)$ is the set of parents of X and $\mathbf{c}$ is the set of all possible combinations of the values of the parents of X.

The message X sends to its parent Y for state y is defined as

$$m_{Y \leftarrow X}(y) = \alpha \sum_{x} \prod_{Z \in Chl(X)} m_{X \leftarrow Z}(x) \sum_{\mathbf{c} \setminus Y} P(x \mid \mathbf{c}) \prod_{Z \in Par(X) \setminus Y} m_{Z \rightarrow X}(z) \tag{2.4}$$

Note the cyclic and iterative nature of the process in which the messages X sends to Y are calculated from the messages received by X in the previous iteration from all its neighbours excluding Y, which is done to prevent Y from "influencing itself".

Also note the $P(x \mid \mathbf{c})$ which represents values of the CPT of x. In practice, the number of possible combinations of states grows exponentially according to the number of parents of X. That makes the problem of crafting and maintaining such a table intractable for elements with even a small number of parents. Moreover, there are cases, such as when the set of parents are not predefined, in which it is unfeasible or outright impossible to create all CPTs for all variables of the graph. For those reasons, many approximations for those values have been proposed, such as applying predefined combination rules or assuming the independence of variables.

It is also possible to redefine $BEL(x)$ in terms of the messages received from the neighbours of X as

$$BEL(x) \triangleq \alpha \prod_{Z \in Chl(X)} m_{X \leftarrow Z}(x) \sum_{\mathbf{c}} P(x \mid \mathbf{c}) \prod_{Y \in Par(X)} m_{Y \rightarrow X}(y) \tag{2.5}$$

Finally, the loopy belief propagation algorithm has the following steps which are applied to each variable:

1. Update the current belief of the variable based on previous messages from its neighbours
2. Propagate the messages to its children
3. Propagate the messages to its parents

For graphs without loops, the algorithm can be applied breath-first from root to leaves and back. If applied in another order or in parallel, the maximum number of passes required for convergence is equal to the diameter of the graph. For graphs with loops, the order of execution is not important. In this case, the number of iterations required for convergence is undetermined, and may in some cases not converge at all or produce wrong marginal values.

2.7 Summary

This chapter provided an overview of key concepts and foundational models employed in this research. It also revisited related work and identified gaps in the literature that motivate the research proposed in this dissertation.

The next chapter will introduce the AEN model which provides a foundation to capture and analyze long-term threats.

Chapter 3

AEN Model

This chapter describes in detail the AEN model, which is a large graph model that enables describing and analyzing continuously in real-time key security relevant information about the operations of networked systems and data centres. The model allows identifying long-term and stealthy attack patterns, which may be difficult to capture using traditional approaches.

The first two sections of the chapter present the data sources of the model, the features extracted from them, as well as the graph elements derived from those features. Afterwards, some motivating examples are provided to highlight the usefulness of the model. The next two sections describe the AEN's probability model and how it can be leveraged for analyzing threats. Finally, the architecture of the implemented system is presented.

3.1 Data Sources

To construct the graph, heterogeneous data sources are used to extract data features capable of identifying nodes, relationships and their attributes. These involve both data sources available within the security perimeter as well as external data sources available through third-party services. Examples of internal data sources include network traffic logs, flow data, system logs, firewall logs, IDS alerts, anti-virus (AV) logs and email security logs, while examples of external data sources include DNS queries and WHOIS.

Of special note are the data sources which report security events or suspicious activity, like IDSs or SIEM systems. Generically, they are called *detectors* and their reports or logs are generically called *alerts*.

The features extracted from the data sources can be divided into two categories related to how they are collected:

1. **First-order** features, which are collected directly from the sources;
2. **Second-order** features which are derived from the available data either by actively using different tools and services or by mining them into aggregates.

First-order features can be further divided into two groups according to where they are sourced from:

- **Network**, which are the more traditional data used by IDSs and are derived from analysis of the network traffic. Its features are:
  - *IP address*: Layer 3 source and destination addresses derived from shallow packet inspection. This allows the identification of hosts and the direction of communication.
  - *Protocol*: Layer 4 protocol derived from shallow packet inspection. This helps with the identification of connections.
  - *Port*: Layer 4 source and destination ports derived from shallow packet inspection, if part of protocol. This helps the model to distinguish between connection of the same protocol and with the identification of applications.
  - *Application*: In some cases it is possible to identify the Layer 7 application related to the packet by performing a deep-packet inspection (DPI) if the packet body is not encrypted or by using a service registry that associates the destination port of the host to an application. It is helpful in understanding traffic characteristics and creating usage patterns, but in practice, it is only available when privacy considerations allow it.
  - *Content-Type*: Same as above.
- **Application**, which are the data collected from log analysis of known applications in the network and from service calls either to or from said applications in case those applications are programmed to provide such functionality. This would include data sources like hypervisor logs, syslogs and IDS or SIEM alerts. Its features are:
  - *Application*: By analyzing logs of known applications and linking it back to a request, the relevant relationship can be marked as belonging to a specific application.
  - *User*: Application users derived from specific application logs such as SSH, FTP and web servers. This allows the model to link usage of user identifiers per host.

  - *Authentication State*: User state in the application derived from specific application logs. In other words, whether the user is trying to authenticate, is authenticated, has disconnected etc. This helps with the understanding of traffic patterns;
  - *Attack Type*: Derived from detector alerts, this information serves as an IOC, which can be used as a base for further threat analysis. Any type of detector, including network, host or hypervisor-based IDSs, AVs, firewall and spam detectors, and the two detectors described in Chapter 4, can provide this type of data.
  - *Alert Confidence*: Same as above.
  - *Device fingerprint*: Can be calculated by client-side code and sent alongside a request to an application, usually a web server (with the client being the browser). This allows the model to identify devices even when using different IP addresses.

Second-order features can also be further divided into two groups:

- **Third-Party**, which are collected by actively contacting external services or performing scans in order to collect more data about a certain entity. Its features are:
  - *Domain name*: Derived by a reverse DNS lookup of an IP address. When an IP address is derived a DNS $\rightarrow$ Reverse DNS lookup cycle is formed until the complete Domain/IP relationship is found. This is useful to identify attacks using Fast-Flux DNS and algorithmically generated domains (AGDs).
  - *IP Address*: Following the item above, an IP address can also be a second-order feature when it is derived from DNS lookups or network address translation (NAT) table conversions.
  - *Name server*: The authoritative Name Server of a domain. It is derived from DNS lookups and useful for the same purposes as the domains themselves.
  - *Autonomous system number (ASN)*: Once again, derived from DNS lookups, the ASN enables the correlation between an IP address and its network provider, usually an internet service provider (ISP).
  - *Location*: Derived by using a geolocation service. Useful to correlate hosts from similar locations and attribute attacks to physical locations.
  - *Name*: A name associated with a domain can be derived by *whois* queries. Allows correlations when different domains belong to the same person or organization.
  - *Email*: Same as above.

  - *Operating system (OS)*: The OS can be derived with a certain accuracy by performing an active scan of a machine.

- **Aggregates**, which are mined from the aforementioned features and from the model itself. They are also divided into two groups:

  - *Temporal*: The result of the accumulation of data through time and subsequent aggregation into statistics related to historical usage of first-order features. These features can be used to identify network's and hosts' behavioural patterns. Being time dependent, these features require decay conditions so they can be marked as outdated and in some cases given smaller weights as time goes by. Many features can be calculated this way, for instance:

    * Frequency of connections between nodes.
    * Number of recent (failed or successful) authentications to an application of a host.
    * Whether or not an IP address has been previously, or recently, used in attack.

  - *Spatial*: Obtained from spatial graph analysis like group behaviour clustering and entity chaining, these features allow for the identification of hidden relationships between nodes and the extraction of subgraphs of interest into evidence or attack graphs.

3.2 Graph Elements

By analyzing the first and second-order features described in the preceding section, it is possible to identify some distinguishable characteristics in them that give clues into how the graph can be constructed. The first one is how each feature can be better used to model the network. Some of them, like IP addresses and domain names, can form relationships among themselves. These features are the nodes of the graph and their relationships are the edges. On the other hand, features like protocol and port are better employed as descriptors of said relationships and, therefore, are used to define attributes of either nodes or edges.

More generally, the nodes of the model are defined by any feature for which useful relationships could be formed, and the edges of the graph are defined by those relationships and their direction. The remaining features are used to define attributes of either nodes or edges.

3.2.1 Nodes

There are different types of nodes with each type having different features, as follows:

- **Account**: Represents an account in a system or application. It is derived from application and system logs and has the following properties:
  - Identifier
  - Application
- **Alert Group**: Aggregation of related detector alerts, called raw alerts internally, that might be generated in a short time frame thus associated to a single event or attack. It has the following properties:
  - Protocol
  - Source IP
  - Source port
  - Destination IP
  - Destination port
  - Service
  - Classification
  - Start time
  - Stop time
  - Severity
  - Confidence
  - Alert count
- **Domain**: Represents the domain name of an IP address. It is derived from a reverse DNS lookup of an IP address. When an IP address is identified, a DNS $\rightarrow$ Reverse DNS lookup cycle is formed until the complete Domain/IP relationship is found. This is useful to identify attacks using Fast-Flux DNS and AGDs. It has the following property:
  - Name
- **Host**: Represents a network host that communicates with other hosts in the network. It is an active node type and it is the most important node type of the model, from which almost all other elements originate. It is derived from different data sources like

network data (packets or flow data), logs and alerts. It may be labelled with host-specific aggregates and historical information like if it was ever part of an attack, etc. It has the following property:

  - Identifier: Piece of information that can be used to consistently and uniquely identify a host through time, like IP address, MAC address, fingerprint or just a generic identifier value. Note that the IP address is not the ideal identifier for this case but given its ubiquity, it is used when other identifiers are not available.

- **IP Address**: Represents an individual IP address. It can be a first-order feature, derived from network data, logs or alerts, or a second-order feature when it is derived from DNS lookups or NAT table conversions. It has the following property:

  - IP address

- **IP Range**: Represents a range of IP addresses. It is derived from WHOIS queries and has the following property:

  - Classless inter-domain routing (CIDR) block

- **Location**: Represents a geographical location. It is derived from WHOIS queries or from IP geolocation services. It is useful to correlate hosts from similar locations and attribute attacks. It has the following property:

  - Tag: a combination of city, state/province, region and country according to what is available on the result of the query

- **Organization**: Represents an organization which controls a range or IP addresses or owns domain names. It is derived from WHOIS queries and has the following property:

  - Name

- **Person**: Represents a person who is listed as being an administrator or owner of a range of IP addresses, organization or domain names. It is derived from WHOIS queries and has the following properties:

  - Name
  - Email

Nodes can be classified into two distinct groups:

- **Active nodes**: Also called actors, active nodes are nodes that can be a source, a target, or a stepping stone (*i.e.*, intermediary) in an attack. Examples of actors are hosts, users and accounts. Active nodes can be further divided into:
  - *Internal actors*, which belong to or are under the control of the organization;
  - *External actors*, which are located outside the perimeter of the organization.
- **Passive nodes**: Nodes that carry some information or granular attributes for actors like DNS derived domain names and location nodes.

3.2.2 Edges

Each node type has different types of edges originating from and terminating in them. The different types of edges and their features (when applicable) are as follows:

- **Authentication Attempt** $Account \rightarrow Host$: Represents an authentication attempt of an account into a host. It has the following properties:
  - Timestamp
  - Source IP
  - Successful: Whether the authentication was successful or not
- **Triggered By** $AlertGroup \rightarrow Host$: Describes the source of an alert group
- **Used** $Host \rightarrow IPAddress$: Expresses that a host used an IP address to send data
- **IP Located At** $IPAddress \rightarrow Location$: Links an IP address to its location as defined by the WHOIS or geolocation query performed when the IP address was first added to the graph or when the relationship was considered stale
- **Part Of** $IPAddress \rightarrow IPRange$: Links an IP address with its IP range as defined by the WHOIS query performed when the IP address was first added to the graph or when the relationship was considered stale
- **Resolved To** $IPAddress \rightarrow Domain$: Links an IP with its domain name as returned by the reverse DNS query performed when the IP address was first added to the graph or when the relationship was considered stale
- **Controls** $Organization \rightarrow IPRange$: Expresses that an organization controls an IP Range as described by the WHOIS queries performed when the IP Range was first added to the graph or when the relationship was considered stale

- **Organization Located At** *Organization* → *Location*: Links an IP address to its location as defined by the WHOIS query performed when the first IP belonging to the organization was first added to the graph or when the relationship was considered stale
- **Owns** *Person* → *Domain*: Represents an ownership relationship between a person and a domain as described by the WHOIS query performed when an IP that resolved to this Domain was first added to the graph or when the relationship was considered stale
- **Session** *Host* → *Host*: Represents a communication session between two hosts on the same protocol, ports and within a certain activity time window. Its direction is based on who initiated the connection. It is primarily an aggregation of network data, but data from other sources like logs and alerts that can be used to identify such communication is also used in order to fill up possible gaps in the network data available. It has the following properties:
  - Start time
  - Stop time
  - Protocol
  - Source port
  - Destination port
  - Source size: Sum of the length of all packets from source to destination
  - Destination size: Sum of the length of all packets from destination to source
  - Service: Describe the service or service type related to the session, such as SSH, FTP or HTTP. This is useful for some types of threat detection mechanisms such as attack fingerprint matching schemes. It is derived from the Application feature.
  - TCP state: Only applicable for TCP packets, it describes the state of the TCP connection after the initial SYN packet is sent and a session is created. There are eight possible states, with the first seven mirroring the TCP states related to connection establishment and termination [82], only with a slight change of semantics because the client and server states are combined:
    - `SYN_SENT`: The initial session state when the session is created from a SYN packet sent by the source host. This means the SYN packet was sent and the source host is now waiting for the SYN-ACK packet.
    - `SYN_RECEIVED`: With the session at the `SYN_SENT` state, the destination host has sent the SYN-ACK packet meaning it received the original SYN packet and is now waiting for the ACK packet that will conclude the handshake.

    - `ESTABLISHED`: The source host has sent the ACK packet while the session was at the `SYN_RECEIVED` state, which concluded the three-way handshake, establishing the connection. Once established, only FIN and RST packets can change the state of the session.
    - `FIN_WAIT_1`: With the session at the `ESTABLISHED` state, either host has sent the FIN packet to initiate the connection termination.
    - `CLOSE_WAIT`: The other host has sent the ACK packet while the session was at the `FIN_WAIT_1` state.
    - `FIN_WAIT_2`: The other host has sent the FIN packet or the ACK-FIN packet to terminate the connection while the session was in either `FIN_WAIT_1` state or `CLOSE_WAIT`.
    - `CLOSED`: The host that initiated the termination has sent the ACK packet while the session was at the `FIN_WAIT_2` state, which concluded the four-way handshake, terminating the connection.
    - `OTHER`: A catch-all state that indicates any other scenario, such as when the first packet of the session is not a SYN packet.
  - TCP first packet flags: Only applicable for TCP packets, it holds the TCP flags set on the first packet of the session. It is useful when the TCP state is `OTHER`.
  - Packet count: The number of packets exchanged between the hosts as part of the session
  - Fragmented packet count: The number of fragmented packets exchanged between the hosts as part of the session
  - SYN flag count: The number of packets exchanged between the hosts as part of the session that had the SYN flag set
  - ACK flag count: The number of packets exchanged between the hosts as part of the session that had the ACK flag set
  - Alert count: The number of alerts generated as part of the session

3.3 Model Definition

To start describing the AEN model, it is important to consider the most basic constituent elements of a network of interconnected devices like hosts, how they communicate with one another and that those elements have their own characteristics. The model, therefore, needs to incorporate those pieces of information. However, the data used to extract them might be incomplete, noisy or simply incorrect. That means the model must not only be able to

describe the different elements and their properties, but must also be able to describe the uncertainty about them. The probability model of the AEN is described in Section 3.5.

Also of importance is the network's dynamism: At any moment devices or hosts can be added or removed from the network; they can start or stop exchanging data; IP addresses can be recycled; devices can be infected and later be fixed, patched or updated etc. It follows that to be able to track the past existence of elements and their relationships through time, the model needs to maintain information on the time spans in which each element has existed. The dynamic graph model defined by Casteigts *et al.* [17] is used to formalize this control, with two changes: the first is that the model is expanded to combine probabilistic and temporal existence, while the second is the exclusion of latencies since they are negligible in the scope of this work. This is used as the base for the identification of important chronological relationships, and combined with the probability model, is the base of model's threat analysis capabilities as described in Section 3.6.

To summarize, the graph model has the following characteristics:

- Nodes have their own attributes. Thus nodes are labelled.
- Nodes can have multiple relationships at the same time. Thus nodes can have multiple edges between them.
- Relationships have a source and a destination. Thus edges are directed.
- Relationships have their own properties. Thus edges are labelled.
- Both relationships and nodes can be uncertain. Thus nodes and edges are weighed by probabilities of correctness, meaning, that the information they represent is correct.
- Nodes and edges change through time and have an existence time span.
- Changes occur continuously.
- Processing times and latencies are negligible.

These characteristics define the graph as a *dynamic uncertain directed multi-graph*. Formally, it is defined as the 11-tuple

$$\mathcal{G} = (\mathcal{N}, \mathcal{E}, s, t, \Sigma_{\mathcal{N}}, \Sigma_{\mathcal{E}}, \ell_{\mathcal{N}}, \ell_{\mathcal{E}}, \mathcal{T}, \pi_{\mathcal{N}}, \pi_{\mathcal{E}}) \tag{3.1}$$

where:

- $\mathcal{N}$ is the set of all nodes of the graph as mentioned above.
- $\mathcal{E}$ is the set of edges representing relationships between nodes.

- $s : \mathcal{E} \to \mathcal{N}$, which assigns edges to their source nodes.
- $t : \mathcal{E} \to \mathcal{N}$, which assigns edges to their target nodes.
- $\Sigma_{\mathcal{N}}$ is a finite alphabet of available node labels.
- $\Sigma_{\mathcal{E}}$ is a finite alphabet of available edge labels.
- $\ell_{\mathcal{N}} : \mathcal{N} \to \Sigma_{\mathcal{N}}$ is a map describing the labels of nodes.
- $\ell_{\mathcal{E}} : \mathcal{E} \to \Sigma_{\mathcal{E}}$ is a map describing the labels of edges.
- $\mathcal{T} \subseteq \mathbb{R}^+$, meaning that the time domain of the graph is in the positive real numbers.
- $\pi_{\mathcal{N}} : \mathcal{N} \times \mathcal{T} \to (0, 1]$, which assigns correctness probability values to nodes over time.
- $\pi_{\mathcal{E}} : \mathcal{E} \times \mathcal{T} \to (0, 1]$, which assigns conditional correctness probability values to edges over time given their endpoints.

3.4 Motivating Examples

The most basic nodes are the hosts (IP addresses) and the most basic relationships are the connections between them. Network features like protocol and port serve as attributes of edges while detector alerts allow the model to mark nodes and their connections as either suspicious or malicious.

With these features the most simple graph permitted by the model can be constructed. An example running from figures 3.1 through 3.4 help understanding of this graph structure and of the model's capabilities in identifying relationships between connections and consequently building attack scenarios. The example does not delve into implementation details but provides a general overview of the model's behaviour during an attack.

The scenario is a small network containing 3 machines: *Host A* is a server running a web application. It has a firewall blocking all ports from outside the internal network, but is open internally to ssh and ftp; *Host B* is a support server used for tasks such as testing. To support remote workers it is open to outside the network; *Host C* is used to host internal services and, as such, is completely blocked to the outside world. Not shown are other network appliances like routers and firewalls.

The starting scenario at Figure 3.1 shows *Host A* receiving some connections at port 80 and *Host B* connected to *Host C* via ssh without having any incoming connection. Since those are normal operations, those connections are considered benign and are marked as such.

Then, in Figure 3.2, *Host G*, a machine controlled by an attacker, starts to scan *Host A* in search of open ports and vulnerabilities. Since only port 80 is open, this scan does not

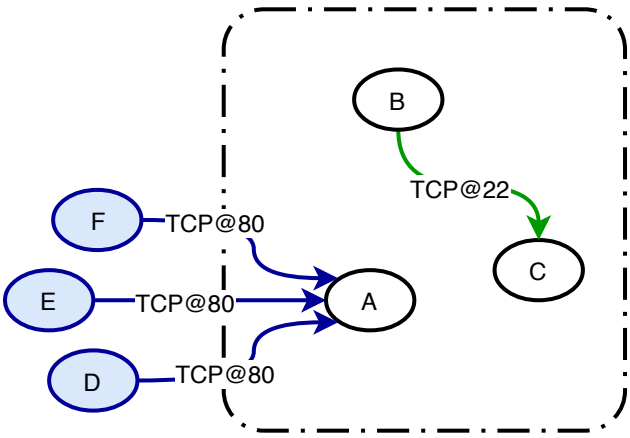


Figure 3.1: Initial example scenario showing benign traffic

provide any valuable information to the attacker, but it allows the model to flag the host as suspicious due to its behaviour. By this time, *Host F* had finished communicating with *Host A* and *Host B* had closed the ssh session with *Host C*. When that happens the model will mark the connection as finished allowing future pruning, denoted here as slightly transparent elements.

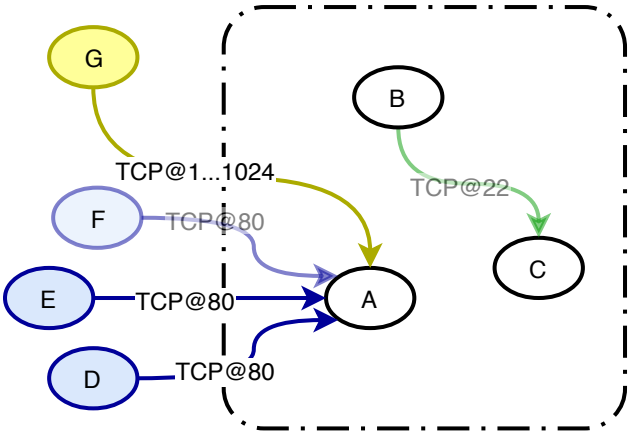


Figure 3.2: Second step in example showing first suspicious traffic

Figure 3.3 shows the next steps in the attack. Being unable to penetrate *Host A*, the attacker then tries, and succeeds, to penetrate *Host B* through a reused leaked credential of one of the employees. After gaining access to *Host B*, the attacker uses the unencrypted identity key stored in *Host B* to connect to *Host C* and then open a reverse proxy to another machine under his control (*Host H*).

Since *Host B* is expected to receive external ssh connections, *Host C* is expected to receive internal ssh connections and to open outbound connections, neither the IDS nor the firewall will block either of those connections nor generate any alarms because of them. On

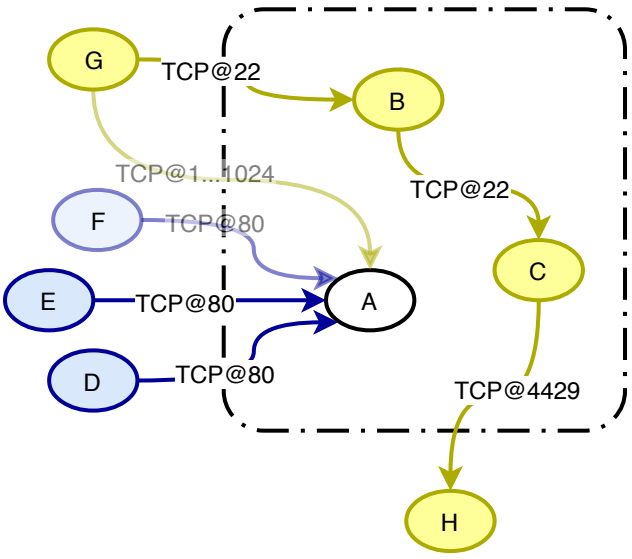


Figure 3.3: Third step in example showing still unidentified intrusion

the other hand, since *Host G* had recently been marked as suspicious, these connections are marked the same way by the graph model.

Finally, in Figure 3.4, the attacker uses the reverse proxy to try to break into *Host A*. At this moment the IDS detects that an exploit is being executed from *Host C* against the machine, blocks the connection and generates an alarm. Normally attack attribution in this scenario is limited to *Host C* unless a human forensics investigator pours through the logs of both *Host C* and *Host B* in order to identify how the intrusion happened. Still, the fact leaked valid credentials were used, and that the connections between *Host G* and *Host C* were already closed would further hinder a broader attribution process.

However, by modelling the relationships between the different nodes and their connections, the proposed system is able to mine the connection chain between *Host G* to *Host H* and back to *Host A*. Seemingly unrelated attacks are linked together and intrusions that initially evaded detection are automatically put together to generate evidence and attack graphs.

As discussed above, this example shows only a very simplified version of what a complete graph would look like. To expand the graph, the remaining features need to be included. These features will not be always available, requiring the model to work around them. However, their availability will help improve the model's performance.

New node types can be derived from features like domain name, location, name, email and user. Those nodes offer valuable knowledge when mining for connections between hosts. The relationships between these nodes vary. For instance: a DNS lookup in either direction will result in a relationship where the domain name "resolves" to the IP address; when a user

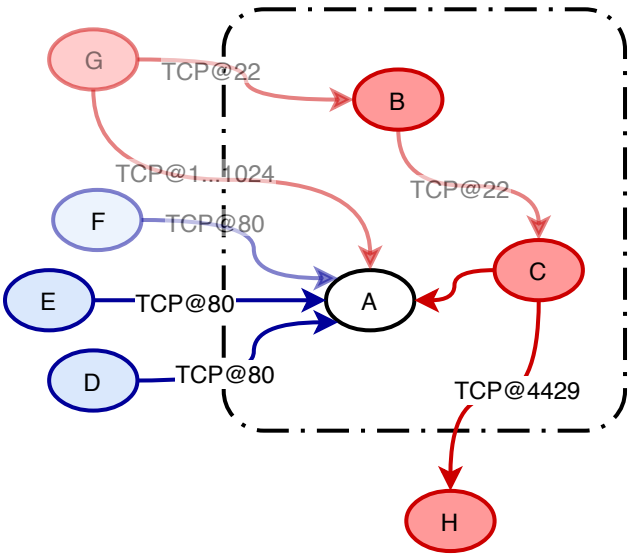


Figure 3.4: Fourth step in example showing intrusion detected

name connected to an application is linked to a connection, the source node "has connected to target node using" user name.

Temporal aggregates can result in relationships in cases such as source node "has recently connected to" target node, but also in node attributes (labels) when they describe historical characteristics of a node such as whether it has been previously used in an attack.

To help illustrate how powerful some of these can be and how would the model behave when they are available, the example above can be continued and expanded. This second part of the example is based on the real attribution of 3 APT attacks to the Deep Panda group [54]; it describes how the proposed system could have mitigated those real-life attacks and facilitated their understanding and attribution.

To start, Figure 3.5 shows the same scenario as Figure 3.4 but expanded to include pertinent DNS and *WHOIS* lookups. In it, the two external hosts considered malicious by the model (*Host G* and *Host H*) have their domain names and owner information derived from the aforementioned queries. However, even though both hosts are considered malicious, their respective domain names are not both considered malicious themselves. The reason is that *Host G* resolved to a general ISP domain, therefore the model cannot consider this domain as malicious and its owner information is not relevant. On the other hand, *Host H*'s domain is obscure enough to be considered under control of an attacker, making it important to obtain the owner's information through a *WHOIS* query.

Moving forward, Figure 3.6 shows the same network some time in the future. *Host B* and *Host C* were cleaned and, therefore, are not considered suspicious anymore. One employee brought a laptop (*Host J*) and is using it to connect to *Host A*. Since this is a normal

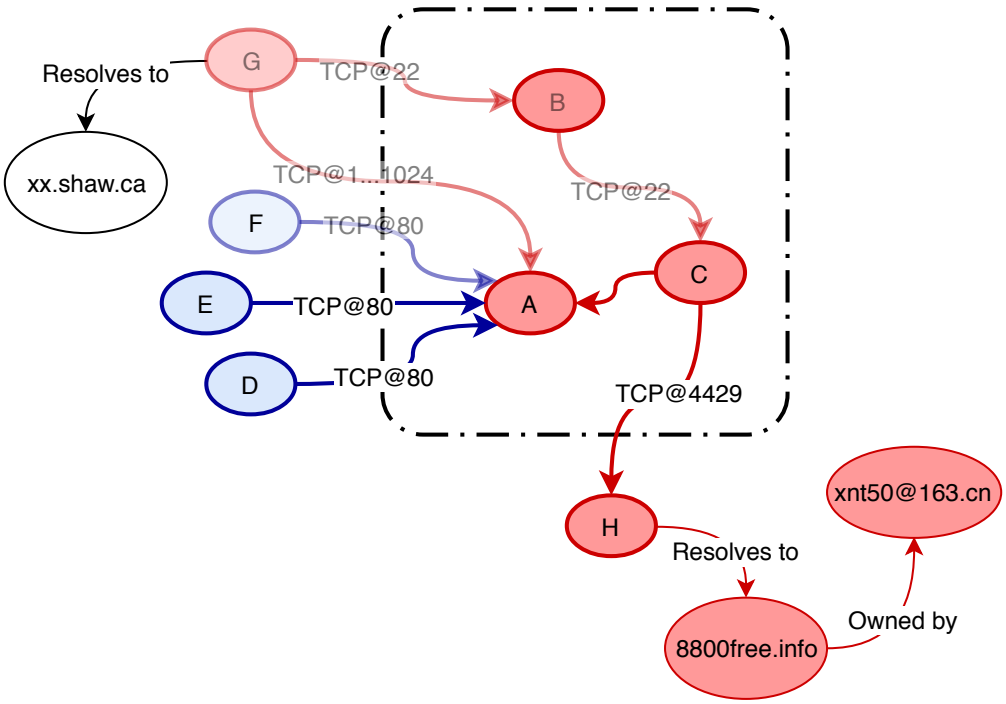


Figure 3.5: Fourth step of example extended to show domain and owner features

connection, it is considered benign.

However, unbeknownst to the employee, the laptop was compromised while outside the network by an infected excel file delivered through a spear phishing email. Its payload was a RAT, giving full control powers to the attacker. To evade detection, the RAT lies dormant until a connection to the target host (*Host A*) is established.

Once that happens, a connection to the attacker's C&C server (*Host K*) is opened. Generally outbound TCP connections to port 443 of external hosts are not blocked by firewalls or IDSs except in the most hardened networks (those would not allow external laptops anyway), therefore this connection would normally be permitted in most networks. However, in this scenario, as soon as this connection is opened, the system performs a *whois* query for the domain being connected to. It returns the same owner of the previously considered malicious domain. That allows the model to automatically flag this connection, and its whole connection chain, as malicious, preventing any exfiltration of data or any kind of lateral movement by the attacker. Figure 3.7 illustrates this scenario.

Once again seemingly unrelated attacks are linked together and an intrusion that would otherwise escape detection is now quickly detected. Forensic analysis would also be facilitated since the complete chain of connections could be extracted into evidence and attack graphs.

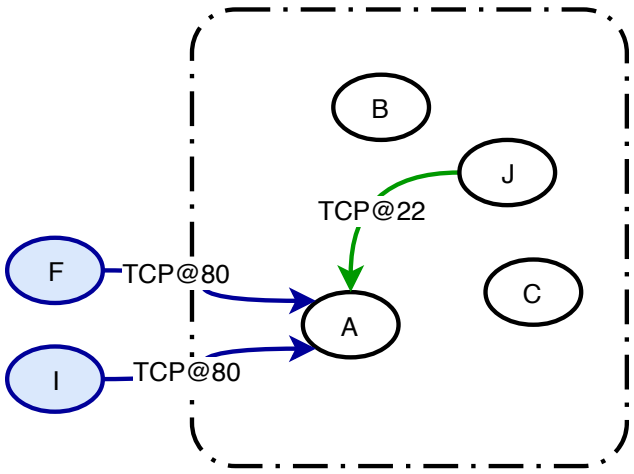


Figure 3.6: First step of second part of example showing benign traffic of laptop recently connected to network

3.5 Probability Model

The uncertainty of the graph's elements are modelled via the probability of correctness, π. At inception, each graph element is assigned a correctness probability which stems from its originating data.

Some graph elements are derived from data that in some way can be categorized as deterministic. For example, when packets related to a TCP handshake between two hosts are received, the model can be certain that those two hosts are communicating and in which direction, or when an application reports that an authentication attempt was made for a certain account, there is no doubt regarding the application or the account being used. In these cases, π is trivial and set to 1, or more technically to $1 - \varepsilon$, where ε represents the inherent probability the data injected into the system were faked, tampered with or corrupted.

In contrast, a few data sources are inherently imperfect. Examples of these data sources include IDSs or other detectors, user/device fingerprinting schemes, geolocation services, and similar sources. Moreover, in some cases, the data have a higher probability of being faked or spoofed, such as a stray IP packet not part of an established communication between two hosts, which is more likely to have a spoofed source address. In these cases, the correctness probability will correspond to the expected accuracy of the data.

As an example, consider the hypothetical scenario where the geolocation service assigns for a given IP address 90% probability of it being from France and 10% probability of it being from Spain at a given point in time $t \in \mathcal{T}$. Based on that, one *IP Address* node and two *Location* nodes will be added to the graph. The two countries exist beyond any doubt so

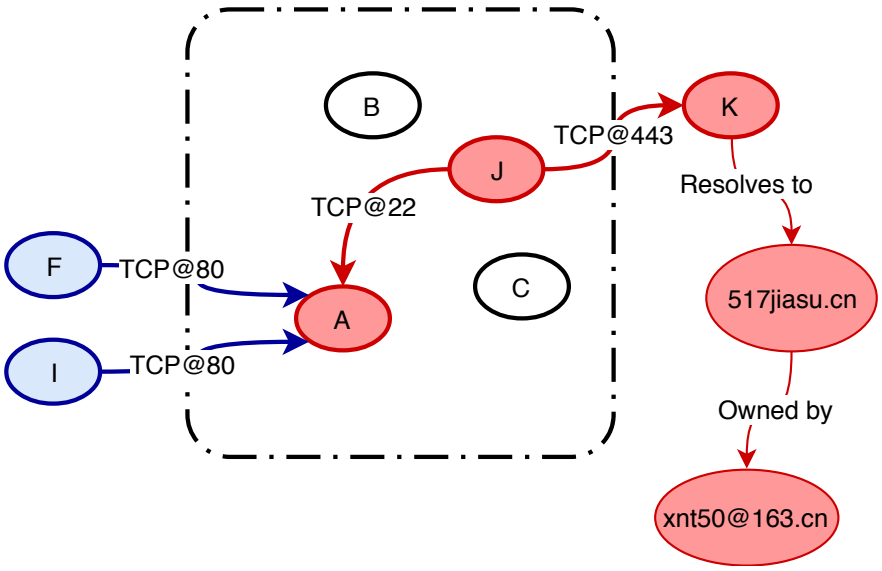
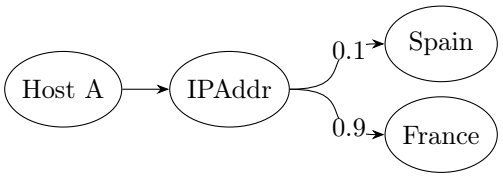


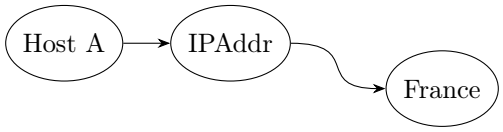
Figure 3.7: Final step of second part of example showing intrusion detected

their π are set to 1. Likewise, the IP address, at least as a member of the set of all possible IP addresses, also exists, and is therefore also assigned $\pi = 1$. From there, one edge between the *IP Address* and each of the two *Location* nodes constrained to t are also added with their probability of correctness being set to the respective probability returned by the geolocation service. In such a manner, both possible, but conflicting, relationships can be modelled and, at a later point, used as part of different inference processes.

Figure 3.8 depicts a sample graph based on the above scenario with Figure 3.8a showing the full graph representing the 2 conflicting relationships and Figure 3.8b showing the resulting "most probable graph" derivation. Future data might result in updated values and thus different derivation results.



(a) Graph representing two possible locations for an IP address node with different probabilities



(b) Most probable graph

Figure 3.8: Example of probability modelling in the AEN graph and the resulting most probable subgraph derivation.

In the above scenario, all probabilities are equivalent to probabilities of existence, however, that is not always the case. Consider now an alert added to the graph. Short of an

attacker being able to inject a false alert into the system, the model can be sure that the alert exists and was generated by the detector. Therefore, its probability of existence is equal to $1 - \epsilon$. However, that does not properly represent the uncertainty regarding the alert. Instead, what is important in this case is describing the probability of the alert being correct, that is, not a false alarm.

Generally, a detector can be considered as a binary classifier that categorizes events as either malicious (or at least suspicious) or benign and generates alerts when an event is classified as malicious. In those circumstances, the alert describes a suspicious activity from a host or account. However, as a classifier, the detector is not infallible, and its alerts can be correct or false alarms.

The AEN therefore models the probability that the alert is a true positive. In other words, its expected precision (positive predicted value – PPV), as its probability of correctness. The PPV is used in this case because it describes the ratio of true positives among all predicted positive elements, meaning it is congruent with the probability of a predicted positive observation (an alert in this case) being a true positive.

The granularity of the precision calculation can also be adjusted. A finer grained approach would be to group alerts based on a common factor and calculate the PPV for each group separately. The grouping can be done in several different ways like by severity, by family of attacks, by individual attack type inside a family or even by individual rules or anomaly metrics. In one hand, these would provide more specific probability values but on the other hand would require much more data in order to be calculated meaningfully. If there are no examples or not enough examples of a given alert in the datasets, then the respective values cannot be calculated using this approach or at least not calculated meaningfully. In these cases, the coarser grain general detector precision would have to be used.

Some detectors will also provide a confidence score for the alert, which can be used to further refine the alert's probability. In such cases, the alert probability is computed by combining the detector's precision and the confidence score into a probability value through a score calibration process, which is described in detail in [121].

Finally, recall that alerts are not added directly to the graph; instead, similar alerts of the same type, origin and target that occur in a short time frame are combined into alert groups. In most cases, all alerts of the same type will have the same π. Hence, the probability of any alert can be used as the probability of the group. An exception to this is any case where different alerts have different confidence scores. In those cases, to obtain the value of π for an alert group A from its constituent alerts A_i, the probability of the alert with the highest

level of confidence is picked. Formally, $\pi(A)$ is defined as:

$$\pi(A) = \max_{i=1}^{n} \ \pi(A_i) \tag{3.2}$$

The probability of correctness serves as a starting point to inferences, derivations or other types of analyses in the graph, as seen in the following sections and chapters.

3.6 Threat Analysis

Before continuing with the applications of the probability model, it is necessary to recall the concepts of journey and horizon described in Subsection 2.3.2 so they can be adapted for the scope of this work.

First, the journey as defined in [13] can be adapted to the scope of this work by including a special constraint such that edges must have existed at the time of the traversal. Formally, given $\mathcal{G}$ as defined in Equation (3.1), a journey can be defined as $\mathcal{J}(u, v, \sigma, \epsilon) = \{R(u, v), \sigma, \epsilon\}$, where:

- $u, v \in \mathcal{N}$;
- $R(u, v) = \{e_1, e_2, \ldots, e_k\},\ e_i \in \mathcal{E},\ s(e_1) = u,\ t(e_k) = v,\ t(e_i) = s(e_{i+1})$, that is R is a route, or path, in $\mathcal{G}$ starting at u and ending at v;
- $\sigma = \{t_1, t_2, \ldots, t_k\}, t_i \in \mathcal{T}, t_i \leq t_{i+1}$ is the time schedule indicating when each edge in R is to be traversed such that e_i is traversed at time t_i;
- $\epsilon(e_i, t_i) > 0$, as a special constraint, means that the edges must have existed at the time of traversal.

Furthermore, the horizon of a node $u \in \mathcal{N}$ can be understood as the set of all possible nodes that u could have affected or exchanged data with. In the case of an attack, all the nodes for which a direct journey exists are nodes that the attacker could have compromised directly, even if only temporarily. In contrast, nodes for which only indirect journey exists can only have been compromised if, somewhere along the journey, the attacker was able to permanently compromise the system through some kind of outbound connection to its servers or through the installation of automated malware.

This demonstrates the importance and capacity of the horizon in regards to the forensic analysis of the network. It provides a complete view of the network from the point of view of the originating node and defines what could possibly be within its reach, should it be a threat to the network. In other words, the horizon can be viewed as the subset of nodes that can be threatened by another node.

Under this optic, the horizon of a node u is here defined as its ***threat horizon***, denoted as $\overrightarrow{TH_u}$ (*i.e.* the subset of $\mathcal{N}$ in which all elements are reachable from u). Formally:

$$\overrightarrow{TH_u} \subseteq \mathcal{N}, \quad \overrightarrow{TH_u} = \{w \in \mathcal{N} : u \rightsquigarrow w\} \tag{3.3}$$

Inversely, the ***reverse threat horizon*** of a node v, denoted as $\overleftarrow{TH_v}$, identifies, from the point of view of a target node, which network nodes could have attacked and compromised it. It is defined as the subset of $\mathcal{N}$ in which all elements can reach v. Formally:

$$\overleftarrow{TH_v} \subseteq \mathcal{N}, \quad \overleftarrow{TH_v} = \{w \in \mathcal{N} : w \rightsquigarrow v\} \tag{3.4}$$

The threat horizon can be used as a starting point for any node-specific analysis. Generically, the source node of a threat horizon and the target node of a reverse threat horizon are called the **focal points** of analysis.

Normally, such analyses would be restricted to hosts such that the threat horizon of host u will represent all the hosts for which u is a threat. However, by expanding the selection to include other node types that are directly or indirectly associated with multiple hosts, like accounts or domains, and using them as focal points, the threat horizon will represent the reachability of all hosts associated with that focal point, thus exposing those otherwise separate threats as a single one.

It is possible to start the threat analysis by calculating some intra-journey metrics like distances, diameters and probability, and some inter-journey metrics like minimum and maximum journey times between two nodes, average waiting period per node, recurrent connectivity etc. Although every metric applies to both Threat Horizon directions, there is usually a small distinction in how each metric is defined for each direction. Some metrics, however, are defined the same way regardless of the direction, in those cases the arrow is omitted.

The diameter of a Threat Horizon is defined as the maximum distance, be it topological or temporal, between the focal point and another node, aka, the antipode node. Specifically, the temporal-antipode node is of special analytical interest since it represents the latest node to reach, or to be reached from, the focal point. On a forward Threat Horizon this can demonstrate that the attacker has not yet reached its desired target and therefore moved to compromise another machine, while on a reverse threat horizon this can help identify the oldest possible source of compromise.

A limitation of the use of the diameter by itself is that it can characterize dead ends, decoys or otherwise unimportant nodes as nodes of interest. To avoid that, a few metrics can be used to filter one-off connections like the connection recurrency of the focal point and the antipode (that is, how frequently they are connected) and the sum of all connection

durations between the two nodes.

Another important metric is the probability, which is the product of the probabilities of all elements present in the threat horizon and the inverse probability of the elements not present in the threat horizon conditioned to the existence of their dependencies. Since there are two types of uncertain elements, the nodes and the edges, the threat horizon probability equation can be split into two components, each based on the generic uncertain graph probability formula, that is, Equation (2.1).

The first component, PN, describes the probability of the graph nodes. Because the nodes are assumed to be independent, their probability is calculated by the product of the correctness probabilities of the nodes that are part of the threat horizon multiplied by the product of the inverse existence probabilities of the nodes that are not part of the threat horizon without any other restriction. Formally, PN is defined as:

$$PN = \prod_{n \in TH_u} \pi_{\mathcal{N}}(n) \prod_{n \in \mathcal{N} \setminus TH_u} (1 - \pi_{\mathcal{N}}(n)) \tag{3.5}$$

The second component, PE, describes the probability of the threat horizon's edges. This probability is only related to the edges between the nodes that are part of the threat horizon. That is so because the edges are dependent on the correctness of both their source and target nodes. Therefore, if one of the two nodes is not present, it is as if the edge does not exist, and consequently it has no bearing on the total probability of the threat horizon. Similarly to PN, PE is equal to the product of two terms. The first term is the product of the existence probabilities of the edges that respect the time schedule of the threat horizon, while the second term is the product of the inverse existence probabilities of the edges that do not respect the time schedule of the threat horizon. Formally, PE is defined as:

$$PE = \prod_{e \in (s(TH_u) \cup t(TH_u)),\ e \in \sigma} \pi_{\mathcal{E}}(e) \prod_{e \in (s(TH_u) \cup t(TH_u)),\ e \notin \sigma} (1 - \pi_{\mathcal{E}}(e)) \tag{3.6}$$

Finally, the total threat horizon probability is defined as the product of the individual probability of its two components, Equation (3.5) and Equation (3.6):

$$P(TH_u) = PN \times PE \tag{3.7}$$

Based on the metrics above it is possible to define threat analytical metrics. The first one is the *threat credibility*, which is defined as the probability that an attacker was able to compromise an internal node of the network. This metric is based upon the probability of traversal towards any internal node regardless of the source.

The second metric is the *threat persistence*, which is defined as the possible duration of an attack based on combined maximum possible journey times as well as waiting periods weighed by their probabilities so journeys that are not credible have a discounted effect on the overall metric.

It follows that a threat horizon is classified as an APT if its credibility and persistence are greater than some predefined thresholds.

3.7 System Architecture

The architecture of the AEN framework implementation is depicted in Figure 3.9.

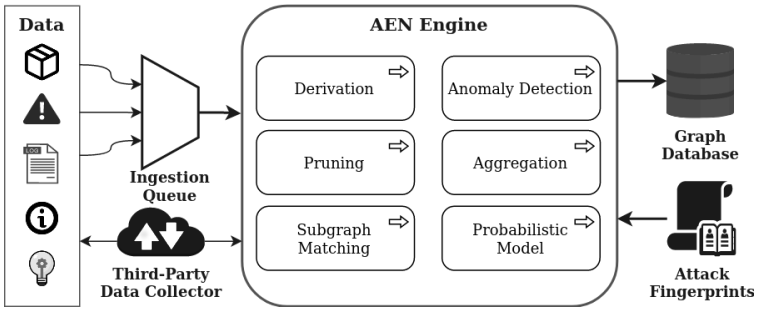


Figure 3.9: System Architecture

The central component of the system is the AEN engine, which is responsible for maintaining the AEN graph. It provides key functionalities like processing and aggregating incoming data through data receivers, attack fingerprint matching, proactive third-party data collection to supplement other incoming data, anomaly detection and the probabilistic model underlying the AEN graph.

The engine stores the graph in a custom-made, in-memory, graph database with capabilities to add, update, remove and search for graph elements. Persistence is obtained via frequent storage of snapshots which can be reloaded from the disk in case of a failure or to review a previous graph state. These snapshots can also be shared with other tools and systems that provide other auxiliary operations or functionalities like visualization of the graph or scalability via read replicas.

Both the engine and the graph database are implemented in Java and are executed in a single process which allows for direct memory access of the graph elements, thus avoiding any extra serialization overhead.

The engine provides two operation modes. The first one is the online mode, in which real-time data is continuously collected from external machines and devices by a client application which then pre-processes and sends the live data to the engine. The second one is the offline

mode, in which previously collected data is added to the graph directly. In both cases, the AEN Engine uses a data ingestion queue which sorts the data chronologically and controls the flow of data.

The current implementation supports network traffic data, either raw or flow data, some syslogs, IDS alert logs from Snort, Zeek and custom alerts derived from Kitsune's anomaly detection output and pre-collected IP addresses information derived from DNS and WHOIS queries. The engine can also actively query IP addresses information when the data is not available. Moreover, it can generate and ingests alerts from its two detectors described in Chapter 4, the fingerprint matching mechanism and the anomaly detection mechanism, as well as run the belief propagation mechanism described in Chapter 5 periodically and classify hosts as malicious based on the results returned by the model.

To handle such variety of data sources, particularly in online mode where data may come faster than it can be processed, the engine ingest data through a data queue which uses adapters capable of transforming data from each source into a pattern that can be injected into the model. Take the network packet data for instance. Its adapter needs to parse *pcap* data and extract features from which the graph is constructed. On the other hand, IDSs alert data might be generated in different formats by different IDS requiring different adapter for each. In this case, the adapter would extract data from those logs and feed the system so it is capable of correlating alerts to connection chains and extracting attack graphs. The adapter is also responsible for assigning probability values to features which are later turned into the internal probabilities of the model.

Attached to the engine is the *third-party connector*, which, as the name implies, is responsible for contacting external services in order to obtain third-party features. The results of those calls follow the same process as the first-order features of going through the ingestion queue before reaching the engine.

The AEN platform also contains an administration console which can be used to visualize the graph, either fully or partially depending on its size, manually add data to the graph and perform some forensic analyses. Figure 3.10 shows an example of the visualization provided by the tool for a graph derived from the ISOT-CID Phase 1 dataset.

3.8 Summary

This chapter described the AEN model, including its data sources, elements, probability model, threat analysis capabilities and metrics, as well as the architecture of the implemented system.

In the next chapter, the two proposed unsupervised detection mechanisms, namely the

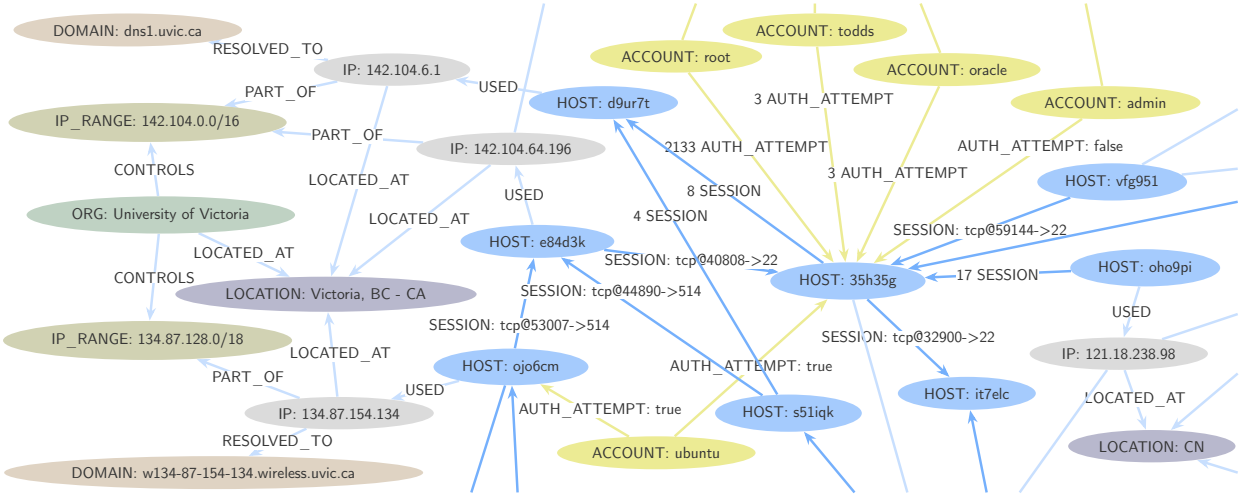


Figure 3.10: AEN graph sample derived from ISOT-CID Phase 1 dataset.

attack fingerprint matching and the anomaly detection, will be introduced.

Chapter 4

Intrusion Detection using the AEN Model

This chapter shows how the AEN model can be used for threat identification by introducing two unsupervised detection mechanisms, one signature-based and one anomaly-based. The signature-based mechanism employs an isomorphic subgraph matching algorithm to search for generic attack patterns, called attack fingerprints, in the AEN graph. As a proof of concept, fingerprints for three main attack categories are provided: scanning, denial of service, and password guessing. The anomaly-based mechanism, in turn, works by extracting statistical features from the graph upon which anomaly scores, based on the bits of meta-rarity metric first proposed by Ferragut et al. [32], are calculated. In total, 15 features are proposed.

4.1 Attack Fingerprint Matching

4.1.1 Fingerprint Framework

4.1.1.1 Attack Fingerprint Visualization

As a visual example of how attack fingerprints can be mined from the AEN graph, Figure 4.1 shows how certain network activity can create evident patterns in the graph. Specifically, the figure shows a visualization of a subset of a graph generated from an example dataset containing a distributed password guessing attack. The hosts are represented by blue nodes at the center, accounts that were used in authentication attempts are represented by orange nodes and the attempts themselves (edges) are either blue when successful or orange when unsuccessful.

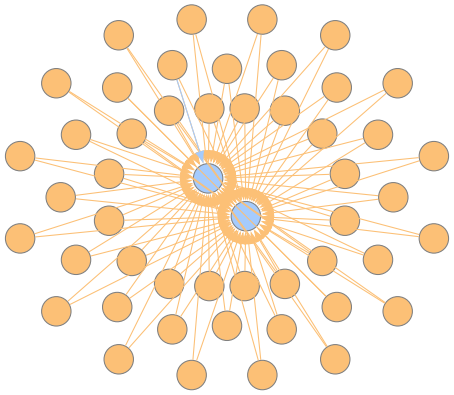


Figure 4.1: Visualization of an AEN graph containing a password guessing attack.

The figure shows a "cloud" of failed authentication attempts against the two central hosts using the same set of accounts. Furthermore, the majority of the edges are orange, indicating failed attempts, but there is a single blue (successful) edge. This pattern makes evident a successful combined credential stuffing and spraying attack where, after several failed attempts, one login was successful.

Likewise, other types of attacks insert their own distinct attack patterns in the graph. It follows that those patterns serve as fingerprints of these attacks and can, therefore, be mined using subgraph isomorphism matching algorithms to identify instances of an attack.

The statefulness of the AEN plays an important role here because it permits the formation of long term patterns. That is in contrast with traditional IDSs which can only identify short-term patterns. In the given example, the attack could have been carried out over several weeks, which would have created a challenge for traditional detection mechanisms. In contrast, because the AEN maintains the relationships over a longer term, those patterns can emerge and be identified.

4.1.1.2 Problem Definition

Given a graph $G = (N_G, E_G)$ where N_G is the set of nodes and $E_G : N_G \times N_G$ is the set of edges, the graph $F = (N_F, E_F)$ is isomorphic to a subgraph $G\prime$ of G if all nodes and edges of F can be mapped to nodes and edges of G. More formally, $F \cong G\prime \sqsubseteq G$ if there is a bijective function $f : N_F \mapsto N_G$ such that $\forall u \in N_F,\ f(u) \in N_G$ and $\forall (u_i, u_j) \in E_F,\ (f(u_i), f(u_j)) \in E_G$.

The definition above can easily be extended to apply to more complex graphs that contain labels and properties, such as the AEN, by applying f to those labels and properties as well.

Finally, the problem of matching a fingerprint is defined as the following: Given an AEN graph G and a fingerprint F, find all distinct subgraphs of G that are isomorphic to F.

4.1.1.3 Describing Attack Fingerprints

In this chapter, the attack fingerprints are described using the Property Graph Query Language (PGQL) query language [108] because it provides a standardized language for describing queries, or patterns, to be searched for in a property graph. This was done with the belief that comprehension is easier when using PGQL than when the fingerprints are described algorithmically. Another reason is that PGQL's syntax follows SQL where possible, except that instead of querying tables, it aims to find matches in the nodes and edges of a graph. Doing so requires specific symbols and constructs for that purpose, but is still easily understandable by those who already know SQL.

Other graph query languages, such as Cypher [33], are also descriptive for this purpose but are less like SQL and have a distinct set of supported features. Still, in most cases, PGQL queries can easily be adapted to other graph query languages.

A simple example of PGQL query is as follows:

```
SELECT s, d
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.duration > 30
```

Algorithm 4.1: PGQL query example

The `SELECT` clause specifies what values are to be returned, while the `MATCH` clause specifies the pattern to match. The parentheses are used to describe nodes, while the square brackets describe edges, with the arrow specifying the direction, if any. Inside the brackets, the colon separates the variable name to the left and the optional label, or type, to the right. The above example matches any pattern in the graph that involves two nodes, `s` and `d`, of type `HOST` connected by a directed edge, `e`, of type `SESSION`, from `s` to `d`, whose duration is greater than 30, and then returns the two nodes for each match.

In general, PGQL allows for a rich description of graph patterns; however, it has limitations which make it impossible to fully express certain attack patterns and, in particular, the information that is to be retrieved from the graph. For the specific use case of this work, PGQL has the following key limitations:

1. Subquery is not supported in the FROM clause.
2. There is limited array aggregation support: In some cases, it is desirable to group matches by destination (the victim) and get an array of sources. However, the current PGQL specification supports only array aggregation of primitive types in paths (using the `ARRAY_AGG` function). Therefore, only properties like IDs can be aggregated in this

way. Note that there are some cases where only the `LISTAGG` function is supported. In those cases, the `ARRAY_AGG` function is used to substitute for `LISTAGG` as if the former had similar support as the latter for simpler pattern description.

To overcome these limitations, the fingerprints contain a post-processing phase during which the query results are processed to reach a final match result.

A final aspect in the descriptions of the fingerprints are the node and edge property names used in this section, which are camel-cased and shortened variations of the names used in Section 3.2.

4.1.1.4 Attack Fingerprint Matching

Finding matches to fingerprints in the AEN graph requires the application of an isomorphic matching algorithm. How this is accomplished depends on the graph engine used to store the AEN graph.

In engines that natively support PGQL, such as PGX and Oracle's RDBMS with the OPG extension, the fingerprints can be used directly to query the database, with only the post-processing phase requiring further implementation. In this case, the matching algorithm is implemented by the graph engine itself.

Similarly, in engines that support other graph query languages, such as neo4j, the fingerprints need first to be converted to the supported query language, but after that, only the post-processing phase requires implementation.

In contrast, when using any graph engine that does not support a graph query language, the whole fingerprint matching algorithm must be implemented. There are several general isomorphic matching algorithms, including Ullmann's algorithm [107] and its derivations (*e.g.* VF2 [25]), the Turboiso [39] algorithm and the DAF [38] algorithm. However, the specific characteristics of the AEN graph and the proposed fingerprints means that a simpler searching algorithm can be employed.

Specifically, the small diameter of the fingerprints means that recursion or any type of partial matching is unnecessary, while the types and properties of the nodes and edges allow for large swathes of the search space to be quickly pruned. In practice, the custom graph engine implemented for the AEN speeds up searching at the cost of extra memory by maintaining separate sets of nodes per type, as well as separate sets of edges per type and per source and destination pair. This can be considered analogous to the indexes employed by the general algorithms mentioned previously, such as the Turboiso and the DAF.

Consequently, finding pairs of nodes per type and edges between them is a constant-time operation. Conversely, iterating the set of edges or groups is done linearly because

no index per property is maintained. However, this operation can be trivially parallelized. Aggregating values, such as summing up properties of elements in groups, must also be done linearly.

A generic fingerprint matching algorithm equivalent is presented in Algorithm 4.2. The algorithm starts by pairing nodes of the desired types (note that each pair is directed). Then, for each pair, it finds all edges between the source and the destination nodes. For each of those edges, the `matches` function is used to test whether the edge matches all of the `WHERE` clauses of the fingerprint and then accumulates the matching edges.

Afterwards, edges are grouped into sets according to the `GROUP BY` expression specified in the fingerprint. Subsequently, each set (group) is tested for matches to all of the `HAVING` clauses of the fingerprint. If true, the results are extracted from the elements in the set based on the `SELECT` expression. These results map to what is returned by the PGQL queries.

```
pairs ← pairNodes(nodes, srcType, destType)

matched ← {}

foreach pair in pairs do
  s, d ← pair
  edges ← getEdges(s, d, edgeType)

  foreach e in edges do
    if matches(e, whereClauses) then
      matched ← matched ∪ {e}

groups ← group(matched, groupingExpr)

preResults ← {}

foreach g in groups do
  if matches(g, havingClauses) then
    gr ← extractPreResult(g, selectExpr)
    preResults ← preResults ∪ {gr}

return preResults
```

Algorithm 4.2: Generic fingerprint matching algorithm

As mentioned previously, many fingerprints also require a post-processing phase, for

which Algorithm 4.3 is the generic algorithm used. It returns a set of matches of the fingerprint as described in the respective subsection of each fingerprint.

```
postProcGroups ← group(preResults, postProcGroupingExpr)

results ← {}

foreach ppg in postProcGroups do
  if matches(ppg, postProcClauses) then
    r ← extractFinalResult(ppg, postProcSelectExpr)
    results ← results ∪ {r}

return results
```

Algorithm 4.3: Generic fingerprint post-processing phase algorithm

Finally, as explained in the following sections, some fingerprints employ the sliding window algorithm to define time windows. In those cases, Algorithms 4.2 and 4.3 are applied for each time window separately, although it would be straightforward to apply the post-processing phase to the combined results of all time windows and group them accordingly. Because the consecutive time windows share some elements, it is possible that the same results will appear in different windows. Therefore, an optional final step when the sliding window is used is the deduplication of results in different time windows, which can be applied to the results returned by Algorithm 4.3.

Also, to speed up searching in those cases, the sets of edges between nodes are sorted by the desired sliding property so that the start and end indexes of each window can be quickly identified through the application of a binary search. Moreover, the algorithm maintains a cursor to the initial position of the previous window so that older elements do not need to be searched.

4.1.1.5 Implemented Attack Fingerprint

As a proof of concept, a total 10 attack fingerprints are implemented and discussed in the following sections covering three families of attacks: Scanning, DoS and password guessing. Table 4.1 summarizes the attacks.

Family	Attack
Scanning	Vertical Port Scanning
Denial of Service	ICMP Ping Flood IP Fragmentation Attack TCP SYN Flood TCP “Out-of-State” Flood UDP Flood HTTP Flood
Password Guessing	Basic Password Guessing Spraying Password Guessing Credential Stuffing

Table 4.1: Implemented attack fingerprints

4.1.2 Scanning Attacks

Scanning attacks consist of probing machines for openings that can be further explored for vulnerabilities and then exploited. They are part of the initial information gathering phase of an attack.

These attacks can target a variety of protocols and applications but are most commonly employed for scanning TCP and UDP ports [10]. They can be deployed by a single source or be distributed among several attackers. In addition, there are many different techniques used for scanning, with each focusing on different layers and using different methods to avoid detection [99, 10].

Scanning attacks can be classified based on several different properties. With regard to their footprints, they can be classified into three major types [99]:

- Vertical scan, which scans multiple ports on a single host
- Horizontal scan, which scans the same port across multiple hosts
- Block scan, which is a combination of both vertical and horizontal scans, whereby multiple ports are scanned across multiple target hosts

With regard to their timing, they can be classified as a slow scan or a fast scan, with the latter being easier to spot than the former, given its speed and short duration [29].

In the following, a fingerprint for single-source fast vertical scans is proposed. Fingerprints for other types of scans can be derived by slightly modifying the fingerprint parameters as demonstrated in Subsection 4.1.3 for the different DoS attacks.

As already mentioned, vertical scans target a specific host by sweeping across the port space, looking for open ports and running services. Unique characteristics can be summarized as follows [10]:

- The packets are sent from one source host to one destination host.
- The packets have several different destination ports.
- The amount of data/bytes exchanged is never large. For TCP scans, for instance, connections are almost never even established.
- The time frame of each single session is very short.

Taking into consideration these characteristics, a typical attack can be defined as one with a short duration and a small amount of data exchanged, particularly from the victim. Otherwise the attack would be too loud and easier to spot, but with a large number of ports involved. This definition can be described by the following fingerprint:

```
SELECT s, d
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.destSize < sizeThr
    AND e.duration < durThr
  GROUP BY s,d
  HAVING count(DISTINCT e.destPort) > portThr
```

Algorithm 4.4: Fingerprint for scanning attack

where:

- `destSize` corresponds to the cumulative size of the packets of the session sent by the destination of the session, which in this case is the target host;
- `sizeThr` defines a threshold for a maximum expected `destSize`;
- `duration` corresponds to the total time duration of a session;
- `durThr` defines a threshold for a maximum expected `duration`;
- `destPort` corresponds to the target ports; and
- `portThr` defines a threshold for the minimum number of distinct destination ports.

When applied, the fingerprint returns all pairs of source and destination hosts where the sources and destinations correspond to the attackers and the victims, respectively, according to the aforementioned characteristics.

4.1.3 Denial of Service

DoS is a family of attacks that aim to disrupt the service of a target server or network resource and make it completely or partially unavailable to users. They are broadly divided into two categories [64, 101]:

- Volumetric attacks, where the target is inundated with huge amounts of traffic that overwhelm its capabilities. These include most flood attacks and amplification attacks.
- Semantic attacks, also known as resource depletion attacks, where weaknesses in applications or protocols are exploited in order to render a resource inoperable without requiring the same large volume of traffic as pure volumetric attacks. These include attacks like TCP SYN flood and slow-rate attacks like Slowloris [14].

Based on the source of attack, DoS attacks can be single-source or distributed, in which case they are commonly referred to as distributed DoS (DDoS). In this subsection, the term DoS is used to refer to both types.

Another common characteristic of many DoS attacks is that the source IP address can be spoofed in order to hide the true source of the attack and to deflect replies away from the attacker. This introduces asymmetry into the traffic load between the attacker and the victim [11, 118]. In other words, the IP addresses identified by the fingerprints as sources of attacks might be spoofed IP addresses in many cases.

In this subsection, focus is given to selected flood attacks covering both categories of DoS attacks under different layers of the OSI model, specifically layers 3 (network layer), 4 (transport layer) and 7 (application layer).

The fingerprints follow a basic pattern of counting the number of matching sessions of a specific attack type within a short time frame. For this reason, a sliding window mechanism is employed with large overlaps between each window with the fingerprints being applied separately for each window. Sliding windows were used instead of simply slicing the timeline so that any short duration attack that would otherwise be divided between two windows could be fully inside at least one window. This had no effect on long duration attacks, as they would fully cover at least one window regardless. In the fingerprints, the start and end times of a time window are represented by `twStart` and `twEnd` parameters, respectively.

4.1.3.1 ICMP Ping Flood

ICMP ping flood is an attack where a high volume of ICMP echo/ping requests are sent to a target IP address in the expectation of flooding the victim with more traffic than it is capable of handling [118].

Based on this, the primary typical characteristics of an ICMP ping flood are defined as follows:

- The attacker host sends a large number of ping requests (*i.e.*ICMP packets) to the target host.
- The packets correspond to echo requests and replies and thus are small.
- The time frame for any single session is very short.

These characteristics can be expressed by the following fingerprint:

```
SELECT s, d, count(e)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.protocol = 'icmp'
    AND e.destSize < sizeThr
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.stopTime < TIMESTAMP 'twEnd'
  GROUP BY s,d
  HAVING count(e) > sessionThr
```

Algorithm 4.5: Fingerprint for ICMP Flood DoS attack

where `sessionThr` defines a threshold for the minimum number of distinct sessions to trigger the fingerprint.

The query returns the number of sessions between each pair of hosts matching the defined conditions, where the source is the attacker, or the spoofed host, and the destination is the victim. These results might be enough for single-source attacks, but to obtain a final result for distributed attacks, they need to be further processed.

This post-processing step is completed by aggregating the results for each destination host in each time window and applying a further threshold, `cntThr`, on the aggregated count (sum) of matching session per destination. The final result is then a set of attack instances, each one containing the victim host, the cumulative sum of matching sessions and a set of attacker hosts.

Since each time window is considered separately, longer attack instances can end up being reported repeatedly in multiple adjacent windows. To improve on that, the results can be deduplicated by aggregating the results of a same target that fall in contiguous windows.

4.1.3.2 IP Fragmentation Attack

IP packet fragmentation is a normal event whereby packets larger than the maximum transmission unit (MTU) of the route (normally 1500 bytes) are fragmented into smaller packets that are reassembled by the receiver. A problem arises when systems have trouble reassembling the packets or will expend too many resources doing so. Attackers take advantage of the situation by crafting special fragmented packets that are impossible to reassemble, causing targets to either crash due to related bugs or to expend more and more resources trying to handle the reassembly of these degenerate packets [101, 83].

Different protocols can be used for fragmented attacks, including UDP, ICMP and TCP. Moreover, fragmented packets can be used to deceive IDSs by crafting fragmented packets that are rejected by the IDS but not by the end system, or vice versa, such that the extra or missing packets prevent the IDS from identifying an attack it otherwise would [83].

From that, the general characteristics of an IP fragmentation attack were identified as follows:

- A medium to high absolute number of fragmented packets can be observed.
- The ratio of fragmented packets to all packets is high.
- The time frame of a single session is very short.

To be able to capture the ratio of fragmented packets, two properties were introduced to the session edge: one that tracks the number of packets, `pktCnt`, comprising the session and another that tracks the number of fragmented packets among those, `fragPktCnt`.

The fingerprint can be expressed as the following query:

```
SELECT s, d, count(e), sum(e.fragPktCnt)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.fragPktCnt / e.pktCnt > fragRatioThr
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.stopTime < TIMESTAMP 'twEnd'
  GROUP BY s, d
  HAVING count(e) > sessionThr
```

Algorithm 4.6: Fingerprint for IP Fragmentation attack

where `fragRatioThr` defines a threshold for the minimum ratio of fragmented packets to all packets of each session that is considered to be matching the fingerprint.

As before, this query returns the number of sessions matching the defined conditions between each pair of hosts, where the source is the attacker and the destination is the victim. In addition, it also returns the sum of the fragmented packet counts from all grouped sessions.

A post-processing phase is included where the results are aggregated by destination host in each time window so that distributed attacks can be identified. A further threshold, `fragPckCntThr`, was applied to the aggregated sum of fragmented packet counts to guarantee that normal absolute amounts of fragmented packets exchanged between hosts are filtered out.

The final result is then a set of attack instances, each containing the victim host, the cumulative sum of matching sessions and fragmented packet counts, and a set of attacker hosts.

Finally, a deduplication step can also be executed to combine instances from adjacent time windows.

4.1.3.3 TCP SYN Flood

TCP SYN flood attacks exploit the three-way TCP handshake process by sending a large volume of SYN requests to a target host without ever completing the handshake process with the expected ACK requests. This causes the target server to hold multiple partially initiated connections, eventually filling its connection buffer and thus preventing subsequent real connections from being established. In some cases, this will result in crashes due to unhandled resource starvation [11].

Therefore, for this attack, the `tcpState` property of the session is used to identify the state of the TCP connection. Moreover, two other related properties, `synFlagCount` and `ackFlagCount`, are used to track the number of packets added to the session that had the SYN flag and the ACK flag set, respectively.

A limitation of the use of these properties is that it requires the packet information in the graph to be correct, which is not guaranteed in all cases. Examples include cases where the network data injected into the model is not complete, whether due to sampling or an unexpected data loss, and also cases where the system has just gone live and thus only started receiving the network data after the connections were established. Another is the case where the system is fed with NetFlow data instead of raw network data and the NetFlow application did not properly track the TCP state or the number of packets containing each flag in any given flow. In these cases, the model is not able to properly track the correct state of the connections. This makes fingerprints that rely on that information ineffective in identifying attacks.

To mitigate those issues, some correction heuristics were employed to change a session's

attributes, such as the TCP state, in cases where inconsistencies between the data and the attributes were encountered. An example is when it is observed that large amounts of data are being exchanged between two hosts on a TCP session, indicating a fully established connection, but the state of the connection indicates otherwise.

In short, the characteristics of a TCP SYN flood attack can be summarized as follows:

- The attacker keeps sending SYN packets to the victim and never replies to the SYN-ACK packet, resulting in a large number of sessions in the `SYN_RECEIVED` state.
- The time frame of any single session is very short.

The above pattern can be expressed in the following query:

```
SELECT s, d, count(e)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.tcpState = SYN\_RECEIVED
    AND e.synFlagCount / e.ackFlagCount > synAckRatio
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.stopTime < TIMESTAMP 'twEnd'
  GROUP BY s, d
  HAVING count(e) > sessionThr
```

Algorithm 4.7: Fingerprint for TCP SYN flood attack

where:

- `destSize` corresponds to the cumulative size of the packets in the session sent by the destination of the session, in this case, the target host.
- `sizeThr` defines a threshold for the maximum expected `destSize`.
- `duration` corresponds to the total length of a session.
- `durThr` defines a threshold for the maximum expected `duration`.
- `destPort` corresponds to the target ports.
- `portThr` defines a threshold for the minimum number of distinct destination ports.

This query mostly follows the same pattern as the preceding ones, with the distinction that it has a condition to only select a session if its TCP state is `SYN_RECEIVED`.

The post-processing phase follows the same pattern as the preceding ones as well, so distributed attacks can be identified and the `cntThr` applied.

4.1.3.4 Other TCP "Out-of-State" Flood Attacks

Aside from the aforementioned SYN flood attack, there are many other less common TCP-based layer 4 flood attacks variants that exploit illegal or unexpected combinations of TCP packet flags sent without first establishing a TCP connection (thus the "out-of-state" term) with the objective of causing a DoS [62]. The lack of a prior connection causes some systems to return RST packets, which can exacerbate bandwidth consumption problems related to the attack. Finally, bugs stemming from unexpected conditions can also cause issues. Examples of flag combinations used in these attacks include:

- ACK-PSH
- PSH-RST-SYN-FIN
- ACK-RST
- URG-ACK-PSH-FIN
- URG

Because these attacks involve out-of-state packets that form the initial packets of the sessions, it is possible to refine the session's TCP state property to track these cases. Specifically, the `tcpFirstPktFlags` property of the session edge was used.

With that, it is possible to define a generic query following the same pattern as the SYN flood attack query, but parameterized by the first packet flags corresponding to the sought after attacks:

```
SELECT s, d, count(e)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.protocol = 'TCP'
    AND e.tcpState = OTHER
    AND e.tcpFirstPktFlags = attackFlags
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.stopTime < TIMESTAMP 'twEnd'
  GROUP BY s, d
  HAVING count(e) > sessionThr
```

Algorithm 4.8: Fingerprint for TCP "Out-of-State" flood attacks

The query requires the same post-processing as the SYN flood attack. It is also possible to modify the fingerprint so that it matches any of the possible out-of-state attack flag combinations instead of matching each one individually by allowing `tcpFirstPktFlags` to be equal to any of the known invalid flag combinations.

4.1.3.5 UDP Flood

UDP flood attacks are flood attack aimed at UDP datagrams. It is considered a volumetric attack because it does not exploit any specific characteristic of the UDP protocol. Instead, it works by sending a large volumes of UDP packets to random or fixed ports on a target host, depleting its available bandwidth, which makes it unreachable by other clients. The attack can also consume a lot of the target's processing power as it tries to determine how to handle the UDP packets [116].

In summary, the key characteristics of a UDP flood attack are as follows:

- The attacker sends UDP packets to the victim at a high rate of frequency.
- The amount of data exchanged per session is relatively fixed and mostly the same.
- The time frame for any single session is very short.

These characteristics can be expressed by the following fingerprint:

```
SELECT s, d, count(e)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.protocol = 'UDP'
    AND e.destSize < sizeThr
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.stopTime < TIMESTAMP 'twEnd'
  GROUP BY s,d
  HAVING count(e) > sessionThr
```

Algorithm 4.9: Fingerprint for UDP flood attack

Once again, this query mostly follows the same pattern as the preceding ones, with the distinction being the condition to only select UDP sessions.

The post-processing phase follows the same pattern as the preceding ones as well, so distributed attacks can be identified and the `cntThr` applied.

4.1.3.6 HTTP Flood

HTTP flood is a layer 7 DoS attack in which a target server is saturated with a high volume of HTTP requests. This can slow the server as it tries to handle the high volume and eventually makes the servers unable to handle legitimate traffic [98].

Because a TCP connection must be established for these attacks to be performed, the spoofing of IP addresses is not possible [101], which makes the identification of source IP addresses more reliable.

An HTTP flood attack can use different types of requests and methods (*e.g.* GET, POST), with the most damaging ones being the heaviest requests for a server to handle, such as those involving heavy processing of input or pushing large amounts of data into a database [98, 101]. Consequently, less bandwidth is required to bring down a web server using an HTTP flood attack than is required for another type of DoS attack.

Several different techniques are employed in HTTP flood attacks. Some send a large number of requests, while others send fewer, but very large or very focused, requests. In either case, the attack involves sending large amounts of IP packets to the target. Therefore, the key characteristics of an HTTP flood attack as follows:

- The attacker sends HTTP packets to the victim at a high rate of frequency.
- The amount of data exchanged per session is high.

Naturally, the fingerprint for these attacks needs to be able to identify HTTP sessions. For this reason, the `service` property of session edge was used to identify sessions that are marked as HTTP-related.

The challenge in this case is populating the field, given that HTTP is a layer 7 protocol and can, in many cases, be encrypted. Two techniques were employed for this purpose. The first was performing DPI to search for identifiers of HTTP messages, such as the version, in packet content. Once a session is identified as being "HTTP-related", it is marked as such and no DPI is required thereafter. A limitation of this technique is that it requires clear text traffic, which in most scenarios today would require the AEN to be deployed after a TLS termination proxy. DPI is also computationally expensive.

For those reasons, a second technique was employed using a service registry comprised of IP addresses and ports of services of interest, such as web services and SSH. This allows for a quick discovery of services but also for some false positives if invalid packets are sent to those servers, such as when non-HTTP packets are sent to an HTTP service.

With the capacity to identify HTTP sessions, a query can be defined as follows:

```
SELECT s, d, count(e), sum(e.pktCount)
  MATCH (s:HOST)-[e:SESSION]->(d:HOST)
  WHERE e.protocol = 'TCP'
    AND e.service = 'HTTP'
    AND e.srcSize < sizeThr
    AND e.startTime > TIMESTAMP 'twStart'
    AND e.startTime < TIMESTAMP 'twEnd'
  GROUP BY s, d
```

```
HAVING sum(e.pktCount) > pktCntThr
```

Algorithm 4.10: Fingerprint for HTTP flood attack

This query is somewhat distinct from the preceding ones because it is based on the number of packets exchanged in a given time window rather than the number of sessions and also because it does not consider short-term sessions. Both differences are a consequence of the fact that HTTP flood attacks require fully established connections. The seemingly redundant clause to select only TCP sessions when there is already a clause to select only HTTP sessions is included to filter out part of the invalid packets, such as UDP packets sent to that specific service, in case the service registry was used.

Finally, the post-processing phase follows the same pattern as the preceding queries as well, with a further aggregation per destination host so that distributed attacks can be identified and the `cntThr` applied.

4.1.4 Password Guessing

Password guessing is when the attacker tries to gain access to a system by persistently attempting to guess user passwords [73, 111]. The passwords attempted are normally derived from either leaked password associated with a particular user or dictionaries of common passwords, in which case the attack is also known as a brute-force attack.

There are a few different types of password guessing attacks, but they all share the main characteristic of generating a high volume of failed login attempts, which are normally logged by the applications into which the authentications are attempted. For this reason, the AEN ingests application and system logs, like those from SSH, to extract authentication information and insert that it into the graph through nodes of type `ACCOUNT` and edges of type `AUTH_ATT` ("authentication attempt") that link an account with the target host of the authentication. To track whether the authentication attempt was successful, the edge has a Boolean property called `succ`.

Another important characteristic of password guessing attacks is that they do not necessarily happen in a short time frame. Sometimes the whole process can last for days, or even longer. That means there is no need to consider the time frame of the attempts. Incidentally, that means the AEN must keep authentication-related elements for longer than it would for many other types of elements.

Three types of password guessing attacks were investigated:

- Basic: One account on one host is targeted with a brute-force attack.
- Spraying: Multiple accounts on one host are each attacked one or a few times.

- Stuffing: The same account is targeted on multiple hosts one or a few times per host.

4.1.4.1 Basic password guessing

A basic password guessing attack is one where a single account on a single host is targeted with a brute-force attack. It is the most common type of password guessing attack [73, 111]. Because it only focuses on one-to-one relations, the graph patterns should be `(:ACCOUNT)-[:AUTH_ATT]->(:HOST)`. For the sake of brevity the whole fingerprint will not be described because it is a generalization of the *spraying password guessing* fingerprint that follows.

4.1.4.2 Spraying password guessing

In a spraying password guessing attack, instead of multiple passwords being tried with a single account, the attacker tries to breach multiple accounts with a single password or a few passwords [66]. In this manner, the attacker can circumvent the most common authentication protection measures, such as account lockouts.

These attacks can be performed either from a single source, in which case tracking attempts per IP address might be a useful detection method, or from distributed sources, which makes detection harder. This is one of the strengths of the proposed fingerprint, as it focuses exclusively on authentication attempts per victim host, which makes it, by design, effective regardless of the number of source hosts involved.

In summary, spraying password guessing attacks have the following characteristics:

- One host is targeted with a high number of authentication attempts.
- The attempts are spread over several accounts such that no account has more than a few attempts.
- One or more hosts can participate in the attack.
- The time frame of an attack can be very long.

Based on the above characteristics, the query is defined as follows:

```
SELECT h, count(DISTINCT a), ARRAY_AGG(e.id) as attempts
  MATCH (a:ACCOUNT)-[e:AUTH_ATT]->(h:HOST)
  WHERE count(!e.succ) > attemptThr
  GROUP BY h
  HAVING count(!e.succ) / count(e) > authFailRatioThr
    AND count(DISTINCT a) > accountThr
```

Algorithm 4.11: Fingerprint for spraying password guessing attack

where:

- `attemptThr` defines a threshold for the minimum number of failed attempts per account to exclude regular login attempt failures from real users.
- `authFailRatioThr` defines a threshold for the minimum ratio of failed attempts in relation to the total number of authentication attempts.
- `accountThr` defines a threshold for the minimum number of distinct accounts that will trigger the fingerprint.

Each match returned by the query contains the victim host, the associated number of distinct target accounts on the host and, as a special output, the array of identifiers of the matching edges, called `attempts`, which is used in the post-processing step to retrieve the list of targeted accounts by fetching from the graph the source nodes of the edges in the array.

Furthermore, if retrieving the hosts responsible for the attack is desired, another query can be performed to find the source hosts of the authentication attempts in the `attempts` array based on its source properties.

4.1.4.3 Credential Stuffing

The credential stuffing attack, also known as a targeted password guessing attack, consists of trying the same credentials (*e.g.* user name and password combination) on multiple hosts [111]. The credentials used in these attack are traditionally obtained from leaks of previous attacks or are default passwords in systems that have them. The latter type is particularly common in IoT devices [77]. More advanced attacks use slight variations of the passwords for cases where the user has the same base password with small modifications per host. Ultimately, this type of attack targets password reuse by the same user on different websites and hosts or poorly designed systems. This consequently means the attacker will not try more than a few different combinations per account but will try the same combinations against multiple targets.

As with any password guessing attack, credential stuffing can be performed from either a single source or distributed sources. However, because this attack targets multiple hosts that normally do not coordinate their detection and prevention efforts, the attacker is more likely to be able to carry out the attack using a single source than with other types of password guessing attacks.

In practice, a single attack campaign can perform credential stuffing on several accounts at once. This type of attack would be detected by the fingerprint as multiple credential stuffing attacks happening together, or possibly as multiple spraying attacks, depending on the number of accounts targeted. Also note that the AEN does not have access to the password, so it cannot determine if the same passwords are being used in multiple hosts, only that multiple failed authentication attempts are being performed for a given account on multiple hosts.

In summary, credential stuffing attacks have the following characteristics:

- Multiple hosts are targeted with a high number of authentication attempts across them.
- Only one account is targeted.
- Only a few attempts are made per host.
- One or more hosts can participate in the attack.
- The time frame of an attack can be very long.

Based on the above characteristics, the query is defined as follows:

```
SELECT a, count(DISTINCT h), ARRAY_AGG(e.id) as attempts
  MATCH (a:ACCOUNT)-[e:AUTH_ATT]->(h:HOST)
  WHERE count(!e.succ) > attemptThr
  GROUP BY a
  HAVING count(!e.succ) / count(e) > authFailRatioThr
    AND count(DISTINCT h) > hostThr
```

Algorithm 4.12: Fingerprint for credential stuffing attack

where `hostThr` defines a threshold for the minimum number of distinct hosts that will trigger the fingerprint.

In contrast to the spraying query, this query groups by account rather than host and counts the number of matching hosts instead of the number of matching accounts. As a result, each match returned by the query contains the targeted account, the associated number of distinct hosts where authentications were attempted and the `attempts` array, which in this case is used to retrieve the list of victim hosts by fetching from the graph the destination nodes of the edges in the array in the post-processing phase.

As with the previous query, retrieving the hosts responsible for the attack is possible by performing another query to find the source hosts of the authentication attempts in the `attempts` array, based on its source properties.

4.2 Anomaly Detection

4.2.1 Measure of Anomalousness

Anomaly detection approaches use statistical methods to help identify outliers or rare events, which are flagged as anomalous. Anomaly is defined as something that deviates from what is standard, normal or expected [3]. When applied to intrusion detection, this involves assuming that anomalous events are more likely to be malicious.

Tandon and Chan [100] defined the anomaly score of an event as the negative log likelihood of that event, which was later adapted by Ferragut et al. [32] as the bits of rarity metric. Formally, given a random variable X with probability density or mass function f, the rarity of an event x is defined as:

$$R(x) = -\log_2 P_f(x) \tag{4.1}$$

The negative means that rarer events have a higher rarity value. Moreover, using the log helps with numerical stability, while the base 2 causes the rarity of the event to be measured in bits. Finally, note that the negative log of zero is defined by convention to be positive infinity.

Ferragut et al. [32] also demonstrated that the bits of rarity metric has some important limitations when used for anomaly detection because it is not *regulatable* or *comparable* between two different types of data. As a supporting example, the authors defined two uniform discrete distributions, one with 100 values and one with 2000 values. If a threshold of 10 is chosen to define what is anomalous or not, then no event of the first distribution will be considered anomalous while all events of the second distribution will be even though they are all equally likely.

Note how, in this example, it is not possible to select a threshold that can be used to regulate the number of anomalous events identified in a sample of any distribution. Furthermore, note how it is not possible to compare the rarity of the events of two distinct distributions because the rarity metric of an event is an absolute value that does not describe the rarity of that event relative to its distribution.

For these reasons, Ferragut et al. [32] proposed a regulatable and comparable anomalousness metric called bits of meta-rarity based on the "probability of the probability" of the event rather than just the probability of an event. More formally, given a random variable X with probability density or mass function f defined on the domain $\mathcal{D}$, the bits of meta-rarity

anomaly score $A : \mathcal{D} \to \mathbb{R}_{\geq 0}$ of en event x is defined as:

$$A(x) = -\log_2 P_f(f(X) \leq f(x)) \tag{4.2}$$

Going back to the previous example, note how for any value x of either distribution, $P_f(f(X) \leq f(x)) = 1$ and consequently $A(x) = -\log_2 1 = 0$. This implies that neither distribution has any anomalous event, regardless of the threshold used (as long as it is greater than 0), and also that the anomaly scores of the different distributions can be compared.

Moreover, for a given threshold θ, the probability that $A(x)$ exceeds that value is bounded by $2^{-\theta}$ such that the ratio of events flagged as anomalous in a sample is never more than $2^{-\theta}$ as long as f fits the sample well. This condition applies for any f, which makes the anomaly regulatable through θ. In practice, θ can be defined based on the available capacity to process anomalous events regardless of the underlying distribution.

Note here the importance of a high goodness of fit of f, without which the above condition will not hold.

For a continuous variable, $P_f(f(X) \leq f(x))$ is defined as the area under f restricted to those t such that $f(t) \leq f(x)$, that is

$$P_f(f(X) \leq f(x)) = \int_{\{t|f(t)\leq f(x)\}} f(t)\mathrm{d}t \tag{4.3}$$

For a discrete variable, $P_f(f(X) \leq f(x))$ is defined as the sum of all probabilities less than or equal to $P_f(x)$, that is

$$P_f(f(X) \leq f(x)) = \sum_{\{t|f(t)\leq f(x)\}} f(t) \tag{4.4}$$

As a further example, consider the discrete variable $X = \{x_1,\ x_2,\ x_3\}$, such that $f(x_1) < f(x_2) < f(x_3)$. Thus, the anomaly scores of these events are given by

$$\begin{aligned} A(x_1) &= -\log_2(f(x_1)) \\ A(x_2) &= -\log_2(f(x_1) + f(x_2)) \\ A(x_3) &= -\log_2(f(x_1) + f(x_2) + f(x_3)) \end{aligned}$$

In this case, it is clear that $A(x_1) > A(x_2) > A(x_3)$; in other words, as the events become more common, they become less anomalous.

Bringing this to the scope of this work, the variables are the features extracted from

the graph as described in Subsection 4.2.2. Each feature is a multinomial variable with k categories, each defined by an n-tuple. For instance, the categories of feature `totalsessions` are defined by the 2-tuple $(SourceHost,\ DestinationHost)$, whose values are defined by the total number of sessions between those hosts.

Now, recall the importance of a suitable distribution for each variable. This is normally obtained through a training phase. However, because the anomaly detection model is fully unsupervised, it does not contain a training phase. Instead, the probability of each observed value is estimated online based on the frequency of that observation in the sample extracted from the graph, which collectively describes the probability mass function of the feature.

This implies that, although the values of each category of a variable might be continuous in theory, they are discrete in practice, which may cause some issues. Consider, for example, the following sample of a feature: $\{x_1 = 22,\ x_2 = 11,\ x_3 = 22,\ x_4 = 555,\ x_5 = 10,\ x_6 = 9\}$.

Intuitively, x_4 should have the higher anomaly score as its value is farther from the values of the others, but that is not the case. Instead, when considering the frequency of each value, x_1 and x_3 have the same values and thus the same higher probability (*i.e.* 2/6) and the same anomaly score. The other values, including the value for x_4, each appear only once and thus result in the same lower probability (*i.e.* 1/6) and the same anomaly score.

To overcome this issue, the values are discretized into bins such that neighbouring values will be mapped to the same bins and thus have a higher probability. In practice, given a bin width h, the binned value of x is defined as

$$b(x) = h \left\lfloor \frac{x}{h} \right\rfloor \tag{4.5}$$

Note that multiplying by h is only done to keep the binned values near the original values but is not required in practice.

To help with understanding, Table 4.2 shows the binned values of a sample using different bin widths. The values with lower probabilities, meaning the most anomalous, for each bin width are bolded.

It can be seen that as the bin width increases, more similar values are mapped into the same bins, which increases their probabilities. With the bin width of 25, all values except that of x_4 are mapped to the same bin, resulting in a probability of 5/6, while x_4 continues to have a probability of 1/6, making it the most anomalous event in the sample for this bin width.

To compute the features, the first step is to split the time range of the data into time windows, with the windows treated independently from each other. For that, the previously discussed sliding window mechanism is employed. Then, for each time window, the features

Category	Bin Width		
	1	5	25
x_1	22	20	0
x_2	**11**	10	0
x_3	22	20	0
x_4	**555**	**555**	**550**
x_5	**10**	10	0
x_6	**9**	**5**	0

Table 4.2: Anomaly detection value binning examples

are extracted from the AEN graph.

Afterwards, for each feature $X = \{x_1, \ \ldots, \ x_k\}$ where k is the size or number of categories of X, its values are binned according to Equation (4.5). After this process, there will be n bins, with each bin representing a value range. The probability of each bin, $p(b_j)$, where $j = \{1, \ \ldots, \ n\}$, is defined as the ratio of the number of elements in the bin over the total number of categories of the feature:

$$p(b_j) = \frac{|b_j|}{k} \tag{4.6}$$

Clearly, the distribution of p approximates the distribution of f such that $P_f(f(X) \leq f(x_i))$ can be approximated through $p(b(x_i))$. Therefore, it is useful to define the anomaly score of a bin b_j following Equations (4.2) and (4.4):

$$A(b_j) = -\log_2 \sum_{\{b_m | p(b_m) \leq p(b_j)\}} p(b_m) \tag{4.7}$$

From that, the anomaly score of x_i is defined as equal to the anomaly score of its binned value:

$$\begin{aligned} A(x_i) &= A(b(x_i)) \\ &= -\log_2 \sum_{\{b_m | p(b_m) \leq p(b(x_i))\}} p(b_m) \end{aligned} \tag{4.8}$$

The last step of the anomaly detection is to compare the anomaly scores with a predefined threshold such that if the anomaly score of an element is greater than the threshold, that element is considered anomalous. Specifically, the source element of the anomalous feature tuples are the ones actually considered anomalous. For instance, for feature `totalsessions`, it is the source host, not the destination host, that is reported as anomalous.

4.2.2 Feature Model

In this subsection, the proposed feature model of the anomaly detection mechanism is described. The features are extracted from the AEN graph and are categorized into session features, which are extracted from session data, and authentication features, which are extracted from authentication data.

Note that all features are contained within a time window, meaning that each operation described below is performed only on the edges that were created in that time window. Also note that the features are directed, so any feature extracted for a pair of hosts h_1 and h_2 is different from that same feature between h_2 and h_1.

4.2.2.1 Session Features

The main type of edge in the AEN graph model is the session edge, which represents a communication session between two hosts. The anomaly detection model leverages session edges to extract useful features that can support threat identification.

There are a total of nine session features:

- Total sessions: The total number of sessions between a pair of hosts.
- Unique destination ports: The number of unique ports of a destination host for which there are sessions from a source host.
- Unique destination hosts with same destination port: The number of unique destination hosts to which a source host connected with the same destination port.
- Unique destination ports for a source host: The number of unique destination ports for which a host has sessions.
- Mean time between sessions: The mean time between the start of the subsequent sessions between a pair of hosts.
- Mean session duration: The mean duration of the sessions between a pair of hosts.
- Mean session size ratio: The mean ratio of the destination size (bytes sent from the destination host of the session) over the source size (bytes sent from the source host of the session) for a pair of hosts.
- Mean session velocity: The mean velocity of the sessions between a pair of hosts. The session velocity is defined as the ratio of the total number of packets of a session to the total duration of the session, which is expressed in packets/sec.
- Mean session source size: The mean source size of the sessions sent from a host.

4.2.2.2 Authentication Features

The authentication data contained in the AEN graph are potential sources of useful information for the detection of anomalous authentication behaviour. There are a total of six different authentication features:

- Total authentication failures: The total number of failed authentication attempts between a pair of hosts.
- Total authentication failures per account per host: The total number of failed authentication attempts by a host using a specific account to all other hosts.
- Total authentication failures per account: The total number of failed authentication attempts between a pair of hosts using a specific account.
- Unique accounts: The number of unique accounts that were used in failed authentication attempts between a pair of hosts.
- Unique accounts per host: The number of unique accounts that were used in failed authentication attempts by a host to all other hosts.
- Unique target hosts per host per account: The total number of hosts that a host attempted, but failed, to authenticated using a specific account.

4.3 Summary

This chapter described two unsupervised detection mechanisms that use the AEN graph as a base. The first detection mechanism was a signature-based scheme that employs an isomorphic subgraph matching algorithm to search for graphical attack patterns, called attack fingerprints, in the graph. In total, ten fingerprints were provided, one for scanning, six for DoS and three for password guessing attacks.

The second scheme was an anomaly detection mechanism that works by extracting statistical features from the AEN graph and calculating anomaly scores based on the bits of meta-rarity metric introduced by Ferragut et al. [32]. In total, 15 features were proposed, nine for session data and six for authentication data.

The next chapter will present the belief propagation detection scheme, including its belief model, propagation rules and its online parameter adaptation method.

Chapter 5

Improved Threat Detection Using Belief Propagation

By building upon the basic probability model defined in Section 3.5, the AEN graph can be thought of as a BN on which probabilistic inferences can be performed. Specifically, the belief propagation algorithm [78, 79] can be applied to it with the goal of identifying malicious elements of the network.

Originally, the belief propagation algorithm was designed to work on DAGs, however, while the AEN model is directed, it can contain cycles. Moreover, there is no directed causal relationship between hosts with regards to their maliciousness. On the contrary, each pair of hosts that communicate exert a symmetrical influence on one another since regardless of whoever initiated the communication, either host can hypothetically infect the other. Therefore, as explained in Section 5.2, undirected graphs called threat graphs are derived from the main AEN graph for the stated inference purposes, which are treated akin to pairwise MRFs [30, 97] whose potential, ϕ, and compatibility functions, ψ, are the priors and conditional probabilities, respectively.

This chapter will explain how the beliefs are defined and propagated in the AEN, and how the AEN can autonomously learn better parameters through the employment of reinforcement learning techniques as long as labels of a subset of the related elements of the graph or their source data are provided.

5.1 Belief Model

Like with the general belief propagation model, in the AEN, each node is considered as a discrete random variable, notated by uppercase letters, such as X, that has a set of mutually exclusive values, or states, notated by the corresponding lowercase letters, in this case x.

Each of these values is assigned a belief that corresponds to the probability of the variable being of that value given the available evidence $\mathbf{e}$, that is, $P(x \mid \mathbf{e})$.

However, contrary to Equation (2.2), there is no decomposition of the evidence of x. Therefore, the belief function is simply defined as [79]:

$$\begin{aligned} BEL(x) &\triangleq P(x \mid \mathbf{e}) \\ &= \alpha P(x) P(\mathbf{e} \mid x) \end{aligned} \tag{5.1}$$

where α is the normalization constant rendering $\sum_x BEL(x) = 1$.

Note that, as stated earlier, $P(x)$ is defined as the potential of x, notated as $\phi(x)$, following MRF notation. Also observe that $P(\mathbf{e} \mid x)$ is equivalent to the likelihood of x given a fixed $\mathbf{e}$, or $\mathcal{L}(x \mid \mathbf{e})$. Therefore, the belief of a state x can be viewed as the combination of the prior probability of x and its likelihood given the observed evidence. Moreover, if the local Markov property is assumed to hold, that is, if each variable is assumed to be conditionally independent of all other variables given its neighbours, then the likelihood of x can be redefined as the support or influence that each neighbour has on x, something that is modulated by the conditional probabilities *between* x and its neighbour's states. Note that the word "between" is used because given two states, x and y, the conditional probability between them follows the direction of the original edge in the AEN graph, such that an edge from Y to X will have a defined $P(x \mid y)$ but not a $P(y \mid x)$. Consequently, the conditional probability between two states is symmetrical regardless of the direction of the influence, be it from X to Y or from Y to X. For this reason, given an edge from Y to X, $\psi(x, y) = \psi(y, x) = P(x \mid y)$ is defined as the compatibility function between x and y. In practice, $\psi(x, y)$ defines the strength of the influence between the two states.

Like with the original algorithm, the messages passed between neighbours describe the support that the node which sent the message provides to a state of the receiving node. However, there is only a single message instead of two. Formally, given two neighbours X and Y, the messages that X sends to Y for a state y is defined as

$$m_{XY}(y) = \sum_x \phi(x)\psi(x, y) \prod_{Z \in N(X) \setminus Y} m_{ZX}(x) \tag{5.2}$$

where $N(X)$ is the set of all neighbours of X.

The cyclic and iterative nature of the process is maintained. Likewise, the filtering of the previous messages of the node who is receiving the message is also maintained in order to prevent it from "influencing itself". Figure 5.1 shows the messages sent between neighbours in a small graph. Note how, compared to Figure 2.6, this graph has undirected edges and

that the same messages are passed between all nodes.

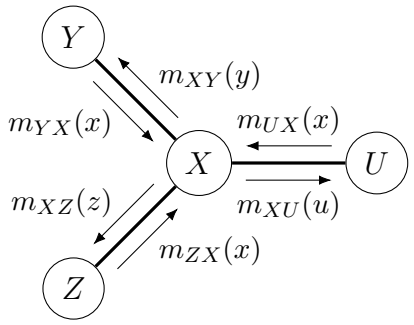


Figure 5.1: Example of the messages passed between neighbours by the belief propagation algorithm in the AEN. Each node computes the belief of each of its states based on the messages received for each state.

With that, it is possible to redefine the belief of a state X based on the messages received by a variable X from all its neighbours $N(X)$ as

$$BEL(x) \stackrel{\triangle}{=} \alpha\phi(x) \prod_{Y \in N(X)} m_{YX}(x) \tag{5.3}$$

Finally, the belief propagation algorithm which is performed iteratively on each node until all nodes converge, or up to a limit if there is no convergence, is described as:

1. Update the node's belief based on previous messages from its neighbours.
2. Propagate a support message, $m_{XY}(y)$, to each of its neighbours.

The order of execution is not important, as long as the above algorithm is performed on each node in every iteration. One interesting aspect of this algorithm is that both the priors and the conditional probabilities, meaning the parameters of the algorithm, can be updated at any moment. The new probabilities will instantaneously affect related beliefs that will spread through the graph over time.

As a starting point for understanding how this can be applied to the AEN, consider a simplified network with only hosts as nodes and the edges representing the communications between them. In this model, the hosts have 2 states: malicious (M), and benign ($\neg M$). Therefore, the probability of compromise of a host can be defined as the marginal probability of the malicious state of that host.

To initialize the nodes with informative values, the prior probability of each state, $P(M)$ and $P(\neg M)$, can be derived from the probability that any random host on the internet or on the internal network under investigation, is malicious or benign at any given moment.

Now consider two hosts, S (the source) and D (the destination), and one edge between them, and let M_S and M_D denote their respective malicious states. It follows that the conditional probabilities of D are defined as the probabilities that D is of a certain state given the state of S. More formally, the conditional probabilities are defined as $P(M_D|M_S)$, $P(M_D|\neg M_S)$, $P(\neg M_D|M_S)$ and $P(\neg M_D|\neg M_S)$.

Once the elements are initialized, the propagation algorithm can be executed in order to update the beliefs of the hosts based on those of their neighbours. However, with only hidden variables, the results are not expected to be representative. Therefore, evidence that the hosts are compromised needs to be added to the network. This is where alerts generated by detectors are useful because they serve as IOCs, or evidence of malicious behaviour.

As described in Section 3.5, these alerts are combined into alert groups, with each group being incorporated into the graph as a node with an edge between itself and a host node to indicate that the group of alerts was triggered by that host. Consequently, alert groups influence the hosts they are linked to, and through that, the propagation algorithm is able to assign better marginals to the hosts.

The states of a host describe whether or not it is malicious. In contrast, the states of an alert group describe whether its alerts are collectively correct or not. That means that alert groups can fall into one of two states: true positive (T) or false positive ($\neg T$).

The priors, that is $P(T)$ and $P(\neg T)$, of an alert group A, are therefore, equivalent to $\pi(A)$, as described in Section 3.5.

Finally, the conditional probabilities $P(M|T)$, $P(M|\neg T)$, $P(\neg M|T)$ and $P(\neg M|\neg T)$ are defined as the probabilities that the host triggering the alerts in that group is of a certain state, given the state of the alert group. These values can be derived from the previously described accuracy calculation. As such, generic values that apply to every type of alert can be used, but ideally, the severity and type of the alert are taken into consideration for more informative values. To understand why that is the case, consider, for instance, a low severity alert, such as a "ping" alert, that might be added to the network. This alert might have $P(T)$ close to 100%, but generating this type of alert does not, by itself, imply that the host is compromised.

After hosts and alerts, it is possible to add other types of elements, such as domains, IP ranges and accounts, to the graph to open up new inference possibilities. Like hosts, these elements work similarly as hidden variables because they do not represent IOCs. Instead, they serve as "indicators of association", meaning they can be used to represent higher-level, but usually weak, associations between elements even when there are no directly observed interactions. Ultimately, they can expose the associations between hosts, either directly, as when two hosts have used the same IP address, or indirectly, as when two hosts have used

different IP addresses that are part of the same IP range. Consequently, they can influence the state of the hosts.

These elements have the same states as the hosts (malicious and benign), but they are defined differently because the elements cannot be considered malicious or compromised, as with hosts. Instead, their states describe the probability that an associated element is of the same respective state. Calculating the prior and conditional probabilities works similarly to calculations for the hosts.

One interesting capability allowed by these element types is that of "clamping" their states, which turns them into observed variables. For instance, take the organization named *University of Victoria*. Normally, as a hidden variable, it influences and is influenced by the elements directly linked to it, such as its IP ranges. It also transitively influences and is influenced by the hosts, in such a way that if many hosts from the said organization are believed to be malicious, its *malicious* state will have a higher probability, which in turn will affect the probabilities of other associated elements, and so on. However, by clamping the *University of Victoria* node to a fixed state, say benign, it will not be affected by its neighbours. At the same time, it will be able to strongly influence or even force neighbouring elements into the same benign state.

With that, the final network model is created. The belief that hosts are malicious or benign is influenced by the alerts generated by the detectors that serve as IOCs, or evidence. It will also be influenced by their relationships to other elements in the network, such as the other hosts they communicate with and their direct or indirect associations via other elements like IP addresses, organizations and locations according to predefined probability distributions and propagation rules.

5.2 Propagation Rules

As stated previously, the belief propagation algorithm is not applied directly to the AEN graph. This is so because AEN graphs are not pure causal graph since the influence that hosts exert over one another with regards to their maliciousness is symmetrical. Additionally, they are not even snapshots of the network at a point in time, but rather, contain a certain history of the network, with edges from a wide range of time. This confers the AEN its analytical powers, but also means that, in some cases, independent nodes, meaning nodes which could not have influenced one another given their communication history, are only a few hops away. In other words, there can exist a pair of nodes u and v in $\mathcal{G}$ where $u \not\rightsquigarrow v$ and $v \not\rightsquigarrow u$ but where there is a route $R(u, v)$. Therefore, to filter out these elements, the timing properties of the AEN's element must be taken into consideration.

With this objective, the AEN leverages the concept of host's threat horizon to derive graphs, called threat graphs, from the main graph for each host constrained by its threat horizon and reverse threat horizon. Formally, the threat graph of a host u, is defined as $TG_u(\mathcal{N}', \mathcal{E}') \sqsubseteq \mathcal{G}$ where $\mathcal{N}' = \overrightarrow{TH_u} \cup \overleftarrow{TH_u}$ and $\mathcal{E}'$ is the set of all edges that respect the time schedule of either the threat horizon or the reverse threat horizon transformed to be undirected. Note that the threat graph is not an induced graph of the main graph as not every edge between two nodes will respect the time schedule of either the threat horizon or the reverse threat horizon. In practice, TG_u can be defined as the union of all journeys to and from u, or more formally:

$$TG_u(\mathcal{N}', \mathcal{E}') \sqsubseteq \mathcal{G}, \quad TG_u(\mathcal{N}', \mathcal{E}') = \bigcup_{v \in \mathcal{N}} \mathcal{J}^*_{(u,v)} \cup \mathcal{J}^*_{(v,u)} \tag{5.4}$$

where $\mathcal{J}^*_{(u,v)}$ is the set of all journeys between u and v.

The threat graph is then treated as a MRF upon which the host's probability of compromise is calculated based on its threat graph. Note that the resulting graph is not the moralized[1] form of the graph. Actually, the graph maintains the same structure except that the edges are transformed to be undirected.

Following the characteristics of the threat horizon, the threat graph only contains hosts, session, alert groups and other elements that could have influenced or have been influenced by its focal host, so it is more suited for propagating the beliefs in the model. On the other hand, the algorithm is more expensive to execute since it needs to propagate beliefs individually for each host.

The algorithm is divided in two parts. The first part is performed on each host (called the focal host) and consists of the following steps:

1. Derive the threat graph.
2. Apply the basic loopy belief propagation algorithm in the threat graph until it converges.
3. Update the focal host's beliefs in the main graph with its threat graph beliefs.

After all hosts are updated, the second part of the algorithm consists of a regular propagation, which is performed on the main graph in order to update the beliefs of the non-host elements. In this part, the beliefs of the hosts are propagated to the other elements but are not updated.

[1]Moralization is the process whereby a directed graph is converted into an undirected graph and every pair of non-adjacent nodes that share a child are "married" by the introduction of an edge between them [55]

The propagation can occur after every change to the graph or on certain schedule for performance reasons, and after completed, the beliefs of all the elements of the AEN will be updated according to the current available information.

5.3 Online Parameter Adaptation

Good parameters, meaning priors and conditional probabilities, are fundamental for the adequate performance of the inference calculations. However, manually defining appropriate values through expert knowledge, analysis of the data or other means can in some cases result in sub-optimal values for the given scenario. Moreover, conditions can change with time and thus the probabilities need to be updated occasionally.

To assuage this problem, an online parameter adaptation mechanism is proposed that borrows from reinforcement learning techniques so the model can adapt its parameters over time with reduced human guidance. As such, the adaptation process is thought of as an optimization problem where the goal is to iteratively approximate the optimal parameters for the given model based on limited information.

The process occurs under a weak supervision model where the labels of a subset of the hosts in the graph are provided as rewards from which relevant probabilities are derived to serve as target values from the adaptation algorithm. In practice, the process of building the rewards would follow a human-in-the-loop model where security analysts evaluate a random sample of hosts in the network based on available data and assign a classification for those hosts.

Given the nature of the data and of the process itself, the rewards are not expected to yield optimal values, but their utility increases the more closely they approximate the population's distribution in comparison to the AEN's classification that is completed before each adaptation. The size of the subset is also not specified, depending primarily on the available manpower, but it should be greater than a minimum threshold in order to maintain an acceptable margin of error, particularly for unbalanced distributions [20] such as those expected in relation to maliciousness of hosts in a network. There is also a point where growing sample sizes stops providing benefits [104].

From a reward and the current graph, the AEN must derive one target value, or objective, for each parameter being adapted, notated as $\mathcal{O}(y)$, where y is the parameter. In this dissertation, the focus will be on adaptation of the two priors, $P(M)$ and $P(T)$, since they are more dependent on the network or dataset under investigation and on the respective IDSs, making them less stable, compared to conditional probabilities. However, the same process proposed in this chapter can be applied to those probabilities as well.

The derivation process is straightforward for $\mathcal{O}(P(M))$, defined as the probability of a host being labelled as malicious in the sample, or $\hat{P}(M)$. Conversely, $\mathcal{O}(P(T))$ cannot be derived directly from the reward, so it must be extracted from the graph. This involves extracting all the alert groups associated with the hosts of the sample and assigning a derived label to each of those alert groups based on the label of the associated host according to pre-defined rules. From that, $\mathcal{O}(P(T))$ can then be defined as the probability of true positive elements in that sample of alert groups, or $\hat{P}(T)$. Note also that in case different alert groups have different classification rules and different priors, these can be further split into subsets and the objective of each subset calculated and updated separately.

The vagaries of the sampling process, as well as eventual classification errors, mean the distributions of different rewards are not expected to be the same even if extracted from the same population. This means that the targets are noisy and vary for each adaptation event, which explains why these values cannot simply be used to replace the original parameters but actually serve as targets towards which the parameters should move.

Ultimately, the goal of the adaptation process is to minimize the error, or loss, between the current parameters and the target parameters, which are surrogates to the unknown optimal parameters. To achieve this goal, the stochastic gradient descent algorithm with momentum [86] is employed such that one pass of the algorithm is performed in each adaptation step.

The formula for the adaptation of a parameter y is as follows:

$$y_{t+1} = y_t + \Delta y_t \tag{5.5}$$

where Δy_t is defined as

$$\Delta y_t = -\gamma \nabla L(\mathcal{O}(y_t), y_t) + p\Delta y_{t-1} \tag{5.6}$$

where γ is the learning rate, also known as step size or gain, p is the momentum, and L is a loss function.

The learning rate is normally low (usually between 0.1 and 0.2) in order to reduce the effect of fluctuations in the target values. The momentum is normally higher (usually between 0.5 and 0.9) and serves to speed up the rate of change when the targets are continuously pointing to the same direction and to dampen the adaptation when subsequent target values diverge. Both these properties are desirable in this scenario given the aforementioned noisy rewards expected by the system.

Given the nature of the formula, values can move out of range. To guard against that, y is always constrained to range $[\varepsilon, 1 - \varepsilon]$.

Many variations of the stochastic gradient descent algorithm exist. Decaying parameters, either time-based or iteration-based, are not adequate for this use case considering the ex-

pected dynamics of the elements being modelled. Conversely, the use of averaging methods where k previous Δy are used instead of a single one, can be adequate and would smooth out oscillations even further. Combining the k previous rewards into a single reward would be equivalent to averaging.

Once the new values are obtained, the probabilities of the graph elements are updated, which instantaneously affects the related beliefs. Finally, the new beliefs are propagated, finishing the process.

In summary, the parameter adaptation process adheres to the following algorithm:

1. Accept a reward consisting of labels of a subset of the hosts in the graph to the system.
2. Derive target probabilities from the reward and in some cases from related graph elements.
3. Perform a single iteration of the stochastic gradient descent algorithm with momentum for each probability to be adapted.
4. Set the updated values in the model.
5. Propagate the new beliefs.

5.4 Summary

This chapter described an unsupervised detection mechanism based on performing a probabilistic inference using a modified belief propagation method on graphs derived from the main AEN graph that leverages the alerts from different IDSs that have been inputted into the graph as IOCs. The mechanism's belief model and propagation rules were presented, as well as a "human-in-the-loop" online parameter adaptation mechanism based on the stochastic gradient descent algorithm, which reduces the initial burden of selecting the system's parameters and allows for frequent adaptation of parameters in dynamic environments.

The next chapter will present the experimental evaluation of the three proposed detection mechanisms using two different datasets, the ISOT-CID Phase 1 dataset and the CIC-IDS2017 dataset, and discuss the obtained results.

Chapter 6

Experiments

6.1 Setup and Procedures

The proposed threat detection mechanisms were evaluated in two separate sets of experiments, one using the ISOT-CID Phase 1 dataset [2] and one using the CIC-IDS2017 dataset [91]. Specifically, for each dataset, different sets of experiments were performed to evaluate the individual performance of the fingerprint matching mechanism, the anomaly detection mechanism and of the belief propagation mechanism when inputted with alerts from different detectors and under two different scenarios, with and without parameter adaptation.

The goal of the experiments was to evaluate the detection mechanisms' capability in correctly classifying hosts as malicious or benign. Additionally, for the belief propagation experiments, it is also a goal to compare the model's performance with the observed stand-alone performance of the detectors whose alerts were inputted into the model for each experiment.

Separate experiments were performed for each day of each of the datasets and for each of the detection mechanisms according to the following procedure:

1. Generate an AEN graph based on the available data of the specific day and in the case of the belief propagation experiments, with the specific alerts for the given experiments.
2. Apply the detection mechanism against the generated graph.
3. Classify each host according to the classification rules of given mechanism. The rules employed are described ahead.
4. Calculate the overall performance of the detection mechanism.

To calculate the performance of each detection mechanism, three main metrics were chosen: F1 score, bookmaker informedness (BM), and the Matthews correlation coefficient (MCC).

As discussed next, both datasets are unbalanced; therefore, the accuracy was not suited for them and was not used for the evaluation. Conversely, the three chosen metrics, particularly the latter two, are generally considered to be better suited for this scenario [59, 21]. The PPV was also included in the performance calculations of the detectors because it was used to guide the $P(T)$ parameter of the belief propagation mechanism, and in the case of Snort and the fingerprint matching mechanism, because it is suitable for evaluating the performance of signature-based schemes. Furthermore, the sensitivity (true positive rate – TPR) and the false positive rate (FPR) were also provided for the anomaly detection mechanism so those values can be correlated to the receiver operating characteristic (ROC) curves presented.

The aforementioned metrics are defined by the following formulas, where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives:

- TPR: $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$
- FPR: $\frac{\mathrm{FP}}{\mathrm{FP+TN}}$
- PPV: $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$
- F1 score: $\frac{2\mathrm{TP}}{2\mathrm{TP+FP+FN}}$
- BM: $\frac{\mathrm{TP}}{\mathrm{TP+FN}} - \frac{\mathrm{FP}}{\mathrm{FP+TN}}$
- MCC: $\frac{\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN}}{\sqrt{(\mathrm{TP+FP})(\mathrm{TP+FN})(\mathrm{TN+FP})(\mathrm{TN+FN})}}$

The details for each of the three detection mechanisms are described in their own sections ahead.

6.2 Datasets

6.2.1 ISOT-CID Phase 1

The ISOT-CID Phase 1 dataset [2] contains system calls, system and event logs, memory dumps and network traffic (TCPdump) data extracted from Windows and Linux virtual machines (VMs) and OpenStack Hypervisors collected from a production cloud computing environment, more specifically, Compute Canada's WestGrid. It includes both benign and malicious traces of several human-generated attacks and of unsolicited Internet traffic, and is split into four days of data.

The dataset includes the time stamps and IP addresses related to each attack, as well as the IP addresses that generated benign traffic. Also included is a label file that labels

each packet in the dataset's network traffic data as benign or malicious, with the malicious packets also labelled by the type of attack. Unsolicited traffic is labelled as malicious but does not have an attack type label.

Only parts of the available data were used for the generation of the graph. Those were the network traffic data from which communication patterns between hosts were extracted, and the system logs from which authentication information was extracted. The total number of hosts and edges, as well as the number of host nodes and session edges of the generated AEN graphs for each day are shown in Table 6.1. Note that these numbers are related to the graphs without any alerts, which are only added for the belief propagation experiments.

Day	Nodes	Hosts	Edges	Sessions
Day 1	376	78	12432	8313
Day 2	635	134	45334	17544
Day 3	653	86	31405	9355
Day 4	491	94	8258	4637
Combined	2155	392	97429	39849

Table 6.1: Daily and combined graph details for ISOT-CID Phase 1 dataset. Each row shows the total number of nodes and edges of the graphs as well as the number of hosts and sessions.

The graph elements are labelled based on the dataset labels, with *host* nodes labelled as malicious if they are the source of at least one packet labelled as malicious. Table 6.2 shows the distribution of hosts per class for each day. As it can be seen, the dataset contains traffic of a relatively small number of hosts, averaging 98 hosts per day, and is unbalanced towards the malicious class with a prevalence of around 82%.

Day	Benign	Malicious	Total	Prevalence
Day 1	18	60	78	76.9%
Day 2	18	116	134	86.5%
Day 3	16	70	86	81.3%
Day 4	16	78	94	82.9%
Combined	68	324	392	82.6%

Table 6.2: Daily and combined host labels for ISOT-CID Phase 1 dataset. Each row shows the number of hosts labelled as each class, the total number of hosts, and the prevalence of hosts labelled as malicious.

The labels of other elements are derived from the host labels such that elements related to the host inherit its labels. For instance, a *session* edge is labelled as malicious if its source

host is labelled as malicious. The malicious session edges are also labelled with the attack type when available. Note that labels are independent for each day of the data, meaning that they are not maintained from one day to the next. Table 6.3 shows the sessions' attack type labels. Like with hosts, there is a high prevalence of malicious sessions, however, most malicious sessions are not labelled with an attack type. Moreover, each day has at least a few samples of different known types of attacks but there were no samples of any DoS attacks.

Day	Benign	Malicious					Total	Prevalence
		PG	Ping	PS	UL	Unk		
Day 1	1034	21	1	3	13	7241	8313	87.6%
Day 2	3268	38	–	11	4	14223	17544	81.4%
Day 3	614	20	–	12	2	8707	9355	93.4%
Day 4	755	–	–	–	2	3880	4637	83.7%
Combined	5671	79	1	26	21	34178	39849	85.8%

Table 6.3: Daily and combined session attack type labels for ISOT-CID Phase 1 dataset. Each row shows the number of sessions labelled as each class, the total number of sessions, and the prevalence of sessions labelled as malicious. PG stands for password guessing, PS for post scanning, UL for unauthorized login and Unk for unknown.

6.2.2 CIC-IDS2017

The CIC-IDS2017 [91] contains network traffic data (both pcap and NetFlow) of benign traffic, as well as several samples of attack scenarios including SSH and FTP password guessing, DoS, web attacks and instances of host infiltration. It is split into four days of data, with some days being further divided into morning and afternoon. For the experiments, each day was considered separately, with mornings and afternoons combined.

The dataset is labelled on the flow level, with each flow being labelled as either benign or with the attack performed. However, to build the graph, the packet data (pcap files) were used to extract the communication patterns between hosts since packets are more finely detailed than flow data. Because no system or application logs were available, no authentication information could be extracted from the data. As a consequence, no authentication-related elements (*account* nodes and *authentication attempt* edges) were present in the graphs of this dataset which in turn means that no password guessing instances can be found by the fingerprint matching mechanism, and that no authentication feature can be triggered by the anomaly detection mechanism. The total number of hosts and edges, as well as the number of host nodes and session edges of the generated AEN graphs for each day are shown in

Table 6.4.

Day	Nodes	Hosts	Edges	Sessions
Day 1	27653	8498	279271	243264
Day 2	29216	9017	298033	259938
Day 3	27828	8545	323502	287272
Day 4	27035	8331	460893	425649
Combined	111732	34391	1361699	1216123

Table 6.4: Daily and combined graph details for CIC-IDS2017 Phase 1 dataset. Each row shows the total number of nodes and edges of the graphs as well as the number of hosts and sessions.

The graph labelling followed the same rules as the previous dataset, with *host* nodes labelled as malicious if they were the source of at least one flow labelled as malicious and with the labels of other elements being derived from the host labels. Table 6.5 shows the distribution of hosts per class for each day. As can be seen, the dataset contains traffic with a high number of hosts, averaging nearly 8,600 hosts per day. Moreover, it is highly unbalanced towards the benign class, with 2 days having only a single malicious host. This creates a challenge when measuring classification performance, because a single classification error can cause a true positive value of 0.

Day	Benign	Malicious	Total	Prevalence
Day 1	8497	1	8498	0.01%
Day 2	9016	1	9017	0.01%
Day 3	8543	2	8545	0.02%
Day 4	8321	10	8331	0.12%
Combined	34377	14	34391	0.04%

Table 6.5: Daily and combined host labels for CIC-IDS2017 dataset. Each row shows the number of hosts labelled as each class, the total number of hosts, and the prevalence of hosts labelled as malicious.

One distinction compared to the previous dataset is on how the attack type labels were defined. For that, the dataset labels were combined into generic attack type labels. For instance, the "DoS slowloris" and the "DoS GoldenEye" labels were both labelled as "Denial of Service" attacks. Then, each flow was matched with its respective *session* edge that was labelled with the generic attack type. In a few cases, sessions had more than one attack type. This was an artifact of how the sessions are created from packets such that one session can map to more than one flow. Moreover, some malicious sessions have no attack type labels in

cases where none of their mapped flows were labelled as malicious, even though their source hosts were labelled as such.

The details of the resulting sessions' attack type labels are shown in Table 6.6. As shown by the table, the prevalence of malicious sessions varied from low ($\approx$ 3% on day 1) to high ($\approx$ 87% on day 4). As for the attack type labels, there was a high number of attack samples in all days. However, the types of attacks present for each day varied, with most types of attacks only present for one day. Moreover, the number of sessions with unknown attack type was high on days 3 and 4, which stems from the fact that these days had multiple attack sessions that were combined in a single graph.

Day	Benign	Malicious							Total	Prev.
		BN	DoS	Inf	PG	PS	WA	Unk		
Day 1	236310	–	–	–	6953	–	–	1	243264	2.9%
Day 2	243367	–	16537	1	–	–	–	33	259938	6.4%
Day 3	197749	–	–	6	1363	–	643	87511	287272	31.2%
Day 4	55352	1228	45392	–	–	158678	–	165026	425649	87.0%
Combined	732778	1228	61929	7	8316	158678	643	252571	1216123	39.7%

Table 6.6: Daily and combined session attack type labels for CIC-IDS2017 dataset. Each row shows the number of sessions labelled as each class, the total number of sessions, and the prevalence of sessions labelled as malicious. BN stands for botnet, Inf for infiltration, PG for password guessing, PS for port scanning, WA for web attack and Unk for unknown.

6.3 Fingerprint Matching

The fingerprint matching scheme classifies a host as malicious if the host is found to be part of an attack by at least one fingerprint.

Table 6.7 shows the parameters adopted for the experiment. Parameters with the same values used by multiple fingerprints are marked as "Multiple/Default", while parameters that are unique to a single fingerprint or values that are distinct from the default are marked according to the fingerprint.

Finally, the discussion of the performance of the fingerprints is based on the PPV because it describes the detection performance of the scheme without taking into consideration the true and false negatives, which is desirable for evaluating the performance of a signature-based intrusion detection scheme. Specifically, the precision is better suited for the task because it describes the ratio of true positives among all predicted positive elements, which

Fingerprint	Parameter	Value
Multiple/Default	`attemptThr`	50
	`authFailRatioThr`	0.8
	`cntThr`	700
	`sessionThr`	100
	`sizeThr`	600 bytes
	`twSize`	20 seconds
	`twStep`	10 seconds
Basic Pwd Guessing	`attemptThr`	4
Credential Stuffing	`hostThr`	4
HTTP Flood	`pktCntThr`	15000
	`sizeThr`	1200 bytes
	`twSize`	2 minutes
	`twStep`	1 minute
IP Fragmentation Attack	`fragPckCntThr`	600
	`fragRatioThr`	0.8
	`sessionThr`	20
Spraying Pwd Guessing	`accountThr`	4
TCP SYN Flood	`synAckRatio`	100
UDP Flood	`sessionThr`	300
Vertical Port Scanning	`durThr`	1 second
	`portThr`	50

Table 6.7: Attack fingerprint matching experiment parameters

is expected to be high for a signature-based intrusion detection scheme. In contrast, other metrics that take into consideration the true or the false negatives are not necessarily expected to result in high values given that the provided fingerprints only cover a few specific types of attacks and not all attack types that exist in the dataset.

6.3.1 ISOT-CID Phase 1

The results obtained after running the fingerprints on the generated graph for each of the four days of the ISOT-CID Phase 1 dataset are shown in Table 6.8 and Table 6.9, with the former showing the classification performance of the proposed fingerprints combined for each day and the latter showing the individual performance of each fingerprint. Fingerprints for which no matches were found are omitted.

As shown in Table 6.8, the fingerprints had very high precision for all days of the dataset

Day	TP	TN	FP	FN	PPV	F1	BM	MCC
Day 1	28	17	1	32	0.97	0.63	0.41	0.36
Day 2	24	18	0	92	1.00	0.34	0.21	0.18
Day 3	38	16	0	32	1.00	0.70	0.54	0.43
Day 4	23	16	0	55	1.00	0.46	0.29	0.26
Combined	113	67	1	211	0.99	0.52	0.33	0.28

Table 6.8: Daily performance of the fingerprint matching mechanism for the ISOT-CID Phase 1 dataset.

Fingerprint	Day 1		Day 2		Day 3		Day 4		Combined	
	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
Basic Pwd Guessing	24	0	19	0	35	0	22	0	100	0
Spraying Pwd Guessing	28	1	24	0	38	0	23	0	113	1

Table 6.9: Individual fingerprint performance for ISOT-CID Phase 1 dataset. Fingerprints for which no matches were found are omitted.

with only a single false positive match resulting in a combined precision of over 0.99, which, as discussed previously, was expected given that the scheme is signature-based and given the high prevalence of malicious hosts in the dataset. The other metrics show a more varied performance depending on the day, which was once again expected, given the small number of attack types covered by the fingerprints. On one hand, for day 2, the great majority of the malicious hosts were not identified as such, which resulted in low values for the three metrics. On the other hand, the performance was better for day 3, because most malicious hosts were identified correctly.

Looking at the performance of the individual fingerprints in Table 6.9, it can be seen that there were only two fingerprints for which matches were found, namely the basic password guessing fingerprint and the spraying password guessing fingerprint. To understand the reason for that, refer back to Table 6.3, where two notable pieces of information are shown. The first is that there are no known samples of DoS attacks in the dataset, which means that none of those fingerprints were expected to be matched. The second is that, while there were a few port scanning attacks, they involved a very small number of sessions (the maximum being 26 on day 4), which maps to a small number of ports scanned in total since each session only has one destination port. Moreover, as the dataset documentation states, these attacks were horizontal scans targeting only a few ports across several hosts in the network, while the available port scanning fingerprint is designed for vertical scans. Therefore, it was expected to find no matches for that fingerprint.

Note that it would be possible for samples of those attacks to be unknowingly present in the dataset from the collected unsolicited traffic. However, no instances of those attacks were observed, except for some instances of password guessing attacks.

Also of note is the fact that the network data in the dataset were sampled and thus contain gaps that can skew some of the graph elements. This can also explain some false negatives and even cause false positives.

6.3.2 CIC-IDS2017

The results obtained after running the fingerprints on the generated graph for each of the four days of the CIC-IDS2017 dataset are shown in Table 6.10 and Table 6.11, with the former table showing the classification performance of the proposed fingerprints combined for each day and the latter showing the individual performance of each fingerprint. Fingerprints for which no matches were found are omitted.

Day	TP	TN	FP	FN	PPV	F1	BM	MCC
Day 1	0	8494	3	1	0.00	0.00	0.00	0.00
Day 2	0	9011	5	1	0.00	0.00	0.00	0.00
Day 3	2	8542	1	0	0.67	0.80	0.99	0.82
Day 4	6	8320	1	4	0.86	0.71	0.60	0.72
Combined	8	34367	10	6	0.44	0.50	0.57	0.50

Table 6.10: Daily performance of the fingerprint matching mechanism for the CIC-IDS2017 dataset.

Fingerprint	Day 1		Day 2		Day 3		Day 4		Combined	
	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
HTTP Flood	0	0	0	0	0	0	1	0	1	0
TCP SYN Flood	0	0	0	0	2	0	1	0	3	0
UDP Flood	0	1	0	2	0	0	1	0	1	3
Vertical Port Scanning	0	2	0	4	2	1	5	1	7	8

Table 6.11: Individual fingerprint performance for CIC-IDS2017 dataset. Fingerprints for which no matches were found are omitted.

As shown in Table 6.10, the general performance was not as high as with the previous dataset. There were no true positive matches on days 1 and 2, resulting in a value of 0 for all metrics for those days. In contrast, days 3 and 4 had better performance, particularly day 4, with a precision of 0.86.

Having no true positives was expected for day 1 because this day only had password guessing attacks but the graph had no authentication elements, which are part of the password guessing fingerprints. This was not the case for day 2, which had DoS attacks that were HTTP-based, but no matches were found for the HTTP flood fingerprint. Still, this can be explained by the types of attacks performed, such as Heartbleed and Slowloris, which are not flood attacks, making the HTTP flood fingerprint unsuitable for this case. Tuning the fingerprint parameters might allow for these attacks to be found but might also result in some false positives. Moreover, the very low prevalence of malicious hosts, with only a single one for both days 1 and 2, means that not finding that host will result in a PPV of 0 as observed.

Continuing onto day 3, there were matches for the two malicious hosts for both the TCP SYN flood fingerprint and the vertical port scanning fingerprint. Note that according to the dataset labels, as shown in Table 6.6, neither type of attack was expected to be present. However, the dataset documentation states that both port scans and nmap scans were performed on that day, although not labelled, which explains the positive matches.

As for day 4, there were matches for four different fingerprints, including three DoS fingerprints and the port scanning fingerprint. The day's data are labelled as having both of those types of attacks, as well as a botnet attack. Sessions with all three labels were matched. Note that the dataset's documentation is not clear on exactly which attacks were executed as part of the botnet attack, but in any case, the attack's data were the source of some of the matches, too.

Finally, all days had false positives, which would not be expected from a signature-based scheme; however, the absolute number of false positives was low compared to the total number of hosts in the dataset. Moreover, as shown in Table 6.11, this was mostly from the vertical port scanning fingerprint. When analyzing the benign sessions that were matched by that fingerprint, almost 60% had port UDP/137, which is used by the NetBIOS name service. However, in these cases, it was not the destination port that was fixed at 137, but instead, the source port was 137, while the destination port varied. This behaviour is an artifact of how name queries are broadcast in NetBIOS, but the replies are directed to the host that made the query on what was originally the source port of the broadcast query. The rest of the false positives do not have such a clear explanation. Tuning the fingerprint parameters could reduce them but would also reduce the detection rate.

6.4 Anomaly Detection

The anomaly detection scheme classifies a host as malicious if it is found to be the source of any anomalous behaviour, that is, if any of the features reports a score for the host above the experiment's threshold.

The algorithm has four parameters, bin width, time window size, time window step and threshold. To assess the performance of the scheme under different combinations of parameters, and identify the optimal threshold value, the following set of values were used:

- Bin width: 1, 2, 4, 12 and 64.
- Time window size: 30 minutes, 1 hour, 4 hours and 12 hours.
- Time window step: Half the time window size.
- Threshold: From 0 to 20 bits at 0.5 intervals.

As a consequence, for each day experiment, the anomaly detection was executed 20 times, once for each parameter combination (excluding the threshold). Afterwards, the scores were evaluated against the 40 thresholds, resulting in a total of 800 sets of results.

Finally, the discussion of the performance of the anomaly detection scheme is based on the three metrics previously mentioned as being suitable for unbalanced datasets. Namely, the F1 score, the BM, and the MCC.

Nonetheless, the resulting ROC curve for each of the parameter combinations are provided, representing the classifications of the separate days combined together, to show the general performance behaviour of the scheme under different parameters and different thresholds. This serves as a basis for selecting the best parameter combinations and, accordingly, discussing the performance of the algorithm. For this reason, the TPR and the FPR are also presented when discussing the performance of the best parameter combination so they can be correlated to the ROC curves.

6.4.1 ISOT-CID Phase 1

After running the experiments as previously described, the results from different days were combined, and an ROC curve was plotted for each of the parameter combinations. The curves are shown in Figure 6.1. Marked in each plot is the point where the threshold is equal to 0.5.

All curves show a similar pattern in which the sensitivity is poor while the threshold is high, until a point where it sharply rises until reaching its peak performance close to where the threshold is equal to 0.5. After that, it just goes straight to the top-right endpoint. Both

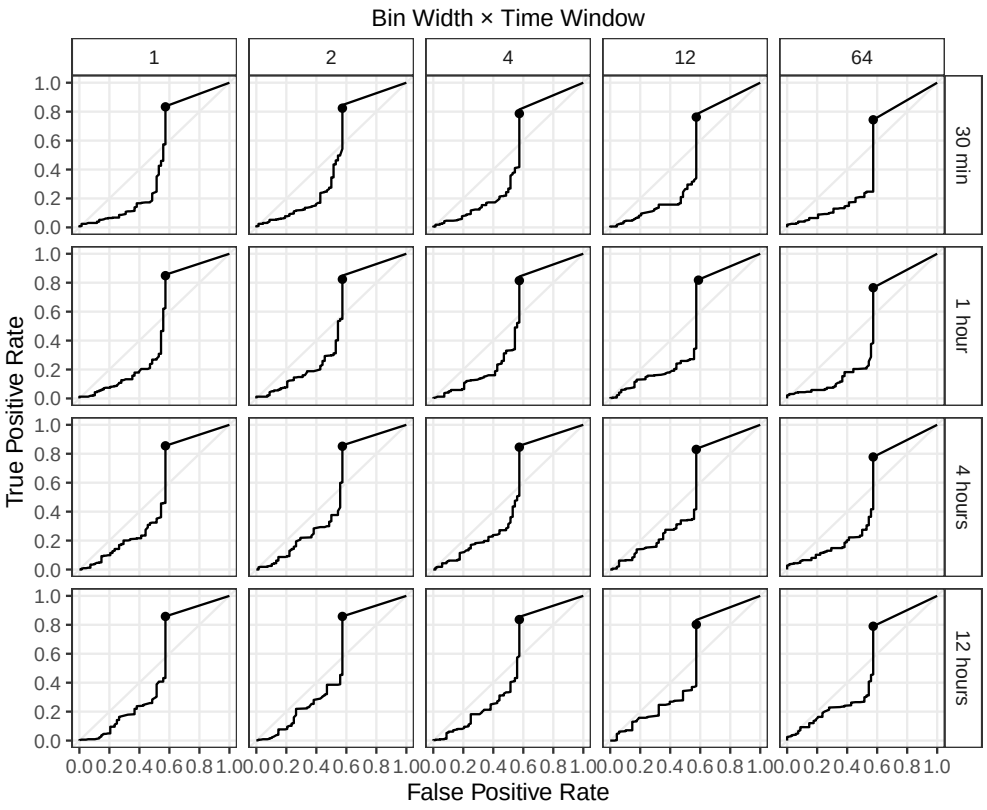


Figure 6.1: ROC curves of the anomaly detection mechanism for the ISOT-CID Phase 1 dataset under different parameter combinations. The point in each plot marks where the threshold is equal to 0.5.

of these characteristics can be explained by the exponential nature of the score, which means that a linear increase in the threshold will cause an exponential decrease in the true positives identified by the model. For this reason, it is common for the maximum score reported by any feature to be between 0 and 0.5, but the probability is exponentially smaller for higher scores.

When comparing the different ROC curves with regard to the other two parameters, a slightly better performance can be observed with longer time windows of 4 and 12 hours and with smaller bin widths of 1 and 2. That demonstrates that the extra information available with longer time windows allows the model to better distinguish anomalous behaviour. However, there is a limit to this, considering that too-long time windows could possibly be hiding shorter duration attacks. Also, the smaller bin widths allow for a greater variability of behaviours to be modelled. In practice, bin widths that are too large result in low resolutions of the distributions of variables that is caused by very diverse values being binned together.

These findings become even more clear when analyzing the other performance metrics, which are more suitable for the unbalanced nature of this dataset. Therefore, to discuss the

findings further, a threshold of 0.5, a bin width of 2 and a time window size of 12 hours were selected. Table 6.12 shows the daily and combined results of the model under that specific parameter combination.

Day	TP	TN	FP	FN	TPR	FPR	PPV	F1	BM	MCC
Day 1	52	8	10	8	0.87	0.56	0.85	0.85	0.31	0.32
Day 2	94	7	11	22	0.81	0.61	0.90	0.85	0.20	0.16
Day 3	61	7	9	9	0.87	0.56	0.87	0.87	0.31	0.31
Day 4	71	7	9	7	0.91	0.56	0.89	0.90	0.35	0.37
Combined	278	29	39	46	0.86	0.57	0.88	0.87	0.28	0.27

Table 6.12: Daily performance of the anomaly detection mechanism for the ISOT-CID Phase 1 dataset.

As can be seen, under the selected parameters, the model was able to detect the majority of the malicious hosts but also generated a relatively high number of false positives and negatives. That behaviour is summarized by the calculated metrics, with a combined PPV of 0.88, a combined F1 score of 0.87, a combined BM of 0.28 and a combined MCC of 0.27. Another notable aspect is the mostly consistent performance observed for each individual day, with only the results for day 2 having a significant deviation from the average.

In general, the observed behaviour was expected, given that the model is anomaly-based and thus prone to generating false alarms. Nonetheless, the observed precision was high. Moreover, the high prevalence of hosts in the dataset means that the malicious behaviour is in fact not anomalous in the dataset. On the contrary, most of the traffic and hosts are labelled as malicious, which explains why the scheme generated a high number of false positives and negatives and points to a general limitation of anomaly-based detection, which can produce degraded results when the malicious behaviour is not uncommon.

6.4.2 CIC-IDS2017

After running the experiments as previously described, the results from different days were combined, and an ROC curve was plotted for each of the parameter combinations. The curves are shown in Figure 6.2. Marked in each plot is the point where the threshold is equal to 0.5.

All parameter combinations showed high levels of performance, with an area under the ROC curve (AUC) of over 0.99. However, that metric by itself is deceiving, given the highly unbalanced nature of the dataset. In reality, the model was able to detect all 14 malicious hosts with thresholds between 0.5 to 5.5 under most parameter configurations, but it reported

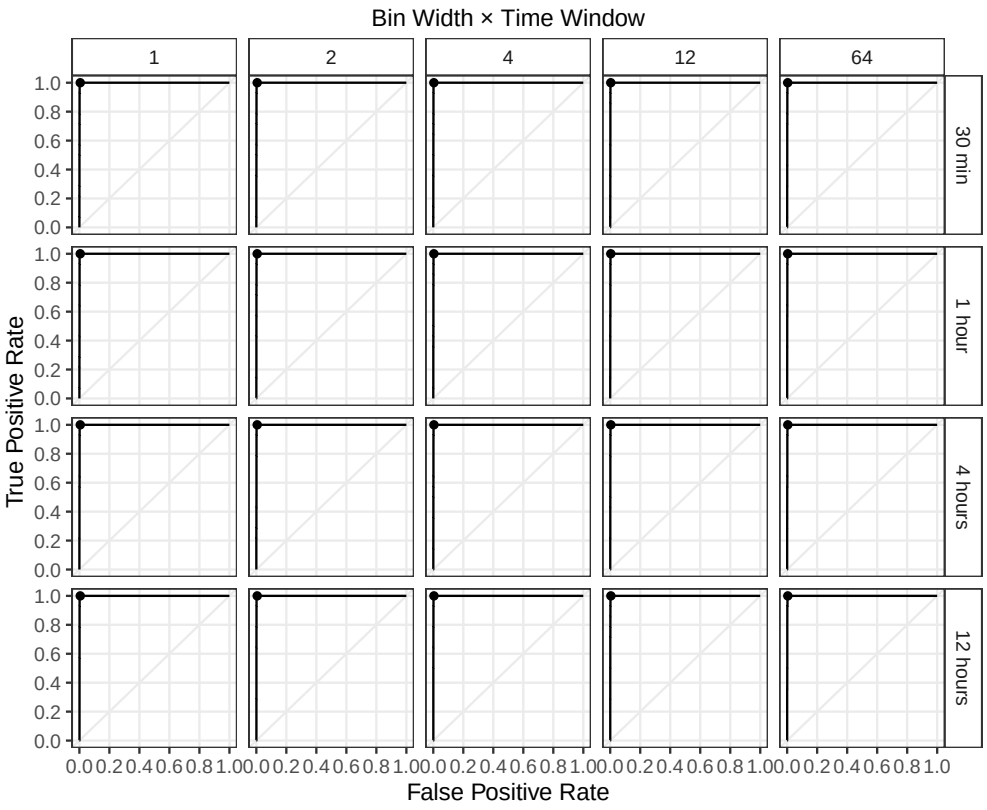


Figure 6.2: ROC curves of the anomaly detection mechanism for the CIC-IDS2017 dataset under different parameter combinations. The point in each plot marks where the threshold is equal to 0.5.

a decreasing number of false positives as the threshold increased. Nonetheless, the relative variation was small in terms of the total number of benign hosts in the dataset.

Although not distinguishable from the ROC curves, the behaviour observed for the previous dataset with regards to the bin width can also be observed for this dataset when analyzing the other metrics, with values of 1 and 2 resulting in a better performance on average than the others. However, the benefit of a longer time window parameter is not as clear for this dataset, which shows a more mixed performance with different values for this parameter.

Based on that, the parameter values selected for further discussion were a threshold of 5, a bin width of 1 and a time window size of 12 hours. Table 6.13 shows the daily and combined results of the model under that specific parameter combination.

As can be seen, under the selected parameters, the model was able to detect all malicious hosts in all days with a relatively low number of false positives. As a point of comparison, using the same parameters as the previous dataset (threshold of 0.5 and bin width of 2) would not affect the number of true positives but would result in an extra 145 false positives

Day	TP	TN	FP	FN	TPR	FPR	PPV	F1	BM	MCC
Day 1	1	8481	16	0	1.00	0.01	0.06	0.11	0.99	0.24
Day 2	1	9001	15	0	1.00	0.01	0.06	0.12	0.99	0.25
Day 3	2	8529	14	0	1.00	0.01	0.13	0.22	0.99	0.35
Day 4	10	8313	8	0	1.00	0.01	0.56	0.71	0.99	0.75
Combined	14	34324	53	0	1.00	0.01	0.21	0.34	0.99	0.46

Table 6.13: Daily performance of the anomaly detection mechanism for the CIC-IDS2017 dataset.

and only slightly worse performance.

Because of the very small prevalence of malicious hosts in the dataset, the observed performance resulted in a low precision and F1 score, and a medium MCC for all days except day 4, which had a higher number of malicious hosts that can compensate for the false positives. Conversely, the BM was above 0.99 for all four days.

In summary, the scheme obtained very good results for this dataset with a varied number of parameter configurations. This shows that the scheme is suitable for detecting malicious behaviour when the prevalence of malicious hosts is low and, therefore anomalous.

6.5 Belief Propagation

6.5.1 Preliminaries

The belief propagation model, was evaluated in experiments using the two aforementioned datasets inputted with alerts from two well-known IDSs independently and with alerts of the two other AEN detection mechanisms combined. The two well-known IDSs used were Snort, which is signature-based, and Kitsune [65], which is anomaly-based.

That resulted in five sets of experiments: three using the ISOT-CID Phase 1 dataset, one with Snort alerts, one with Kitsune alerts, and one with alerts from both the fingerprint matching and the anomaly detection mechanisms, plus two using the CIC-IDS2017, one with Kitsune alerts, and another with the combined alerts from the fingerprint matching and the anomaly detection mechanisms.

Each set of experiments was performed under two different scenarios. The first used pre-defined parameters selected to simulate parameter definition from expert knowledge and no parameter adaptation. The second used best-guess parameters but employed parameter adaptation.

As previously stated, the goal of the experiments was to assess the performance of the

proposed model in classifying hosts as malicious or benign under the two different scenarios when inputting alerts from different detectors, and in comparison with the observed stand-alone performance of the detector.

For Snort and Kitsune, the classification followed the rule where a host is classified as malicious if at least one alert was generated for that host on the given day.

For the experiments with the combined AEN detection mechanisms, the individual performance of each detector are shown in Section 6.3 for the fingerprint matching and in Section 6.4 for the anomaly detection.

For the sake of comparison, two ensemble classifications that fuses the classifications of the individual detectors were also performed. Each of the ensembles used a different classification rule: One used an *and* rule, meaning a host is only considered malicious if both detectors agree that it is malicious, and the other used an *or* rule, meaning that a host is considered malicious if either of the detectors consider it malicious.

Naturally, the *and* rule was expected to generate few false positives and more false negatives. In contrast, the *or* rule was expected to generate more false positives and few false negatives. In practice, the detector with fewer positive predictions sets an upper limit on those numbers when using the *and* rule and a lower limit when the *or* rule is applied.

The belief propagation parameters used in these experiments were the six prior and marginal probabilities related to hosts, sessions and alerts. Other elements of the graph, including domains, locations and authentication attempts, were not given probabilities and thus had no effect on the result of the algorithm.

For the first scenario, the parameters, their selected values and the reasoning for that selection, were as follows:

- Probability of a host being malicious — $P(M)$, defined as a value approximate to the prevalence of malicious hosts in the considered dataset for each experiment because each dataset is not a perfect representation of real network data, and therefore choosing a more generic value would not be adequate for the data used.
- Probability of an alert being correct — $P(T)$, defined according to the average of a sample of precision values found in the literature for the detectors whose alerts are used in the experiments. For Snort, high values above 0.80 are commonly reported [1, 28], including in this work, where a precision of 0.91 was obtained for the ISOT-CID Phase 1. However, in some cases, the reported precision can be as low as 0.31 [103]. For Kitsune, a similar pattern can be observed. In [51], values above 0.90 to values below 0.01 were reported for different datasets while [84] reported a precision around 0.50. The same applies to this work, where the two values obtained (0.64 and 0.002) for the

two datasets used were also distinct. For the fingerprint matching and the anomaly detection mechanisms, both obtained a high precision for the ISOT-CID dataset but a lower precision for CIC-IDS2017 dataset. This shows that the precision of all four detectors can have a lot of variability. Therefore, a single average $P(T)$ value of 0.50 was selected for all of them.

- Probability of a host that caused an alert being malicious given a correct alert — $P(M|T)$: This is fixed at 0.90 because it is expected that a host that generated a true alert is malicious with the exception of edge cases like alerts of low severity.
- Probability of a host that caused an alert being malicious given an incorrect alert — $P(M|\neg T)$ is fixed at 0.10 in contrast to $P(M|T)$.
- Probability of a host that is the destination of a session being malicious given that the source host is malicious — $P(M_D|M_S)$, also defined according to the prevalence of malicious hosts in the considered dataset for each experiment but limited to a lower value between 0.10 and 0.25 given the general expectation of the lateral movement of attackers inside an infected network and on the expected intercommunication patterns of malicious hosts.
- Probability of a host that is the destination of a session being malicious given that the source host is not malicious — $P(M_D|\neg M_S)$ is fixed at 0.25 based on the general expectation that some amount of communication between malicious and benign hosts will take place when attacks are performed, machines are infected and in some occasions even unknowingly initiated by benign hosts.

Table 6.14 shows the summary of the dataset parameters used for this scenario.

Parameter	ISOT-CID Phase 1	CIC-IDS2017
$P(M)$	0.75	0.05
$P(T)$	0.50	0.50
$P(M\|T)$	0.90	0.90
$P(M\|\neg T)$	0.10	0.10
$P(M_D\|M_S)$	0.25	0.10
$P(M_D\|\neg M_S)$	0.25	0.25

Table 6.14: Parameters used in the belief propagation experiments for the scenario without parameter adaptation

These values are not expected to be optimal, and in fact, as shown in the following, are known to be non-optimal but are good candidates for realistic values selected under expert knowledge.

For the second scenario, the parameter adaptations were performed at the end of each day. Therefore, a set of initial values were used to parameterize the experiment of the first day of each dataset. Then, at the end of each day, the parameter were adapted and the resulting values were used as parameters for the following day. In other words, the classification of the first day was parameterized with the initial values, while the classification of the second day was parameterized with values that had been adapted once, and so on.

The rewards consisted of a random subset of the labels of the dataset for the given day, with its size defined by a reward ratio parameter specifying the percentage of labels rewarded to the graph for each day. Note that the reward was only provided after the day's propagation was performed such that it could not influence its own day's performance, but only the subsequent day's.

Of the six parameters, the four conditional probabilities were set to the same values as the first scenario, while the values of $P(M)$ and $P(T)$ were initialized with best-guess values, as follows:

- $P(M)$: This was initialized as 0.20 as it is generally expected that most hosts are not malicious and to be a middle ground between the $P(M)$ values used in the experiments without parameter adaptation.
- $P(T)$: This was initialized as 0.20 so it is distant from the $P(T)$ value used in the experiments without parameter adaptation.

Not adapting the four conditional probabilities have two opposing effects. On one side, it guarantees a faster convergence to an adequate performance, which is desirable given the limited number of iterations. On the other hand, it also prevents the algorithm from exploring a larger area of the parameter space, which precludes the model from reaching more optimal parameters than the fixed ones.

In addition to the belief propagation parameters, also included were the parameters for the adaptation algorithm, *i.e.*, the learning rate γ, the momentum p and the reward ratio r, which were set to relatively aggressive values given the small number of iterations available and were the same for all three experiments. Table 6.15 shows the parameters used for this scenario.

Moreover, a linear loss function $L(\mathcal{O}(y_t), y_t) = y_t - \mathcal{O}(y_t)$ was chosen and the alert classification rule employed was such that an alert group was labelled as true positive if the originating host is labelled as malicious. Otherwise, it was labelled as false positive.

A final parameter was the number of runs performed for each experiment, which was set to 100. This was required due to the random sampling performed for the rewards. Different

Parameter	Value
$P(M)$	0.20
$P(T)$	0.20
γ	0.40
p	0.50
r	0.10

Table 6.15: Parameters used on belief propagation experiments for scenario with parameter adaptation. For the two parameters being adapted, the table shows their initial values. Parameters repeated from previous scenario are omitted.

samples have different performance results, so by running multiples times, the results could be averaged.

In summary, each run of the three experiments of this scenario was executed as follows:

1. Set the initial parameters of the algorithm
2. For each day do the following:
   (a) Build the graph with the available data and alerts.
   (b) Propagate the beliefs throughout the graph.
   (c) Collect the predicted classification of each host.
   (d) Calculate the overall performance metrics of the algorithm.
   (e) Extract a random subset of the labels of the data.
   (f) Run the parameter adaptation mechanism.
   (g) Clear the graph, but maintain the parameters.

After executing all the runs, the average performance is calculated for each day.

With regard to the performance assessment, the three aforementioned main metrics were used. Namely. the F1 score, the BM, and the MCC. The PPV was also included in the performance calculations of Snort and Kitsune because it was used to guide the $P(T)$ parameter. Furthermore, to calculate the average performance of the algorithm under the parameter adaptation scenario, the median was chosen so outliers were filtered out.

6.5.2 ISOT-CID Phase 1

For the ISOT-CID Phase 1 dataset, three sets of experiments were run, one inputting the Snort alerts into the graph, one inputting Kitsune alerts, and one inputting alerts from both the fingerprint matching and the anomaly detection mechanisms. Because of the high

prevalence of malicious hosts in this dataset, as shown in Table 6.14, the $P(M)$ selected for the experiments using this dataset was 0.75 while the $P(M_D|M_S)$ selected was 0.25.

6.5.2.1 Results with Snort Alerts

The Snort alerts were generated using custom-made rules since the built-in rules (community rules) resulted in very low performance levels. Table 6.16 shows the performance of the generated Snort alerts in classifying hosts as benign or malicious.

Day	TP	TN	FP	FN	PPV	F1	BM	MCC
Day 1	26	13	5	34	0.84	0.57	0.16	0.13
Day 2	67	13	5	49	0.93	0.71	0.30	0.21
Day 3	28	12	4	42	0.87	0.55	0.15	0.12
Day 4	56	12	4	22	0.93	0.81	0.47	0.37
Combined	177	50	18	147	0.91	0.68	0.28	0.21

Table 6.16: Daily and combined performance of Snort alerts using custom-made rules for the ISOT-CID Phase 1 dataset.

Table 6.16 shows that Snort had a relatively high number of false negatives, but only a small number of false positives, giving it a combined precision of 0.91, which is consistent with what is expected from signature-based IDSs, particularly given the high prevalence of hosts labelled as malicious. When looking at the other metrics, Snort had low performance on days 1 and 3, misclassifying most of the malicious hosts as benign. Performance on days 2 and 4 was better, specially the latter, which was the only day having what could be considered medium performance. The combined results summarize these findings with a an F1 score of 0.68, a BM of 0.28 and an MCC of 0.21.

After calculating the performance of the IDS, belief propagation was performed for each of the four days of the dataset. Table 6.17 shows the results. In comparison to the base Snort performance, those numbers represent a significant improvement in all metrics, including a large reduction in the number of errors, particularly false negatives.

Also of note are the gains obtained on days 1 and 3 with BM and MCC gains of 0.56 and 0.54, respectively, on day 1 and 0.66 and 0.67, respectively, on day 3. The combined results also demonstrate these gains, F1 score of 0.93, BM of 0.74 and MCCs of 0.66.

Moving to the parameter adaptation scenario, Figure 6.3 summarizes the performance and dispersion of each metric as the days progress for the 100 runs with the solid lines showing the median performance. The median values, as well as the adapted parameters values for each day, are also shown in Table 6.18.

Day	TP	TN	FP	FN	F1	BM	MCC
Day 1	53	15	3	7	0.91	0.72	0.67
Day 2	96	15	3	20	0.89	0.66	0.51
Day 3	66	14	2	4	0.96	0.82	0.78
Day 4	74	14	2	4	0.96	0.82	0.79
Combined	289	58	10	35	0.93	0.74	0.66

Table 6.17: Daily and combined performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Snort alerts.

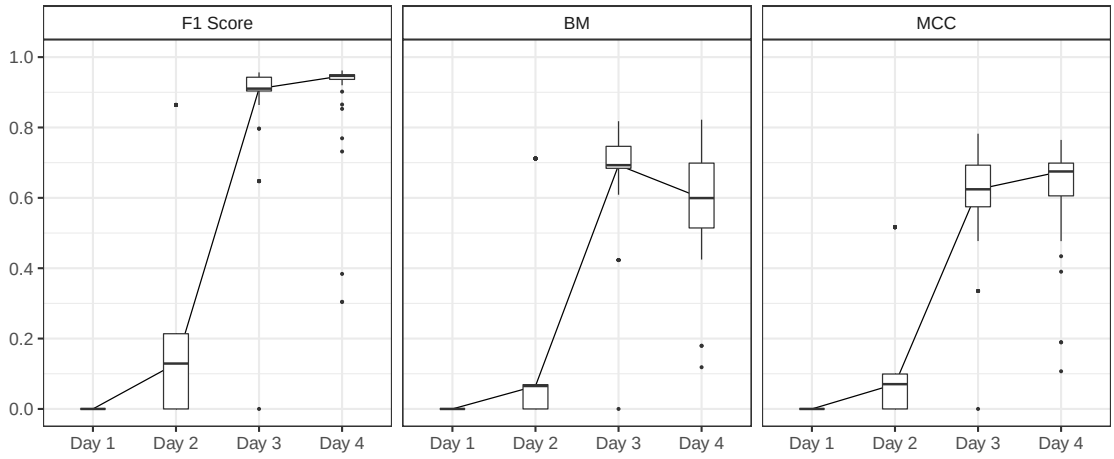


Figure 6.3: Summary of the performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Snort alerts and parameter adaptation. The solid lines plot the median values.

The results obtained for the first day were poor, which was expected given the initial parameters not being adequate for this dataset. However, there is a clear performance improvement trend on subsequent days as the parameters converged close to those of the first scenario, such that by the third day, the median performance reached close to the same level as the first scenario. On day 4, the median number of correctly classified malicious hosts was higher than in the first scenario, but the median number of false positives was higher as well. Although the performance was still good and markedly better than what was obtained by Snort alone, this indicates that the median parameters likely overshoot their optimal values which is an artifact of the aggressive learning rate and momentum used. If more iterations were available and lower values were used, the performance would progress more slowly but the parameters would be less likely to overshoot too much.

The dispersion of the results mostly increased as the days progressed, which was expected given the high learning rate and small number of iterations. Starting on day 2, but mostly on days 3 and 4, there were an increasing number of runs that achieved results close to, and

Day	P(M)	P(T)	TP	TN	FP	FN	F1	BM	MCC
Day 1	0.20	0.20	0	18	0	60	0.00	0.00	0.00
Day 2	0.41	0.52	8	18	0	108	0.13	0.07	0.07
Day 3	0.72	0.79	61	13	3	9	0.91	0.70	0.62
Day 4	0.90	0.90	76	10	6	2	0.95	0.60	0.67

Table 6.18: Median daily performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Snort alerts and parameter adaptation.

for some metrics better than, those of the scenario without parameter adaptation. On days 3 and 4 there were only a few outlier results with poor performances as well.

6.5.2.2 Results with Kitsune Alerts

Kitsune does not generate alerts by itself and requires a training phase, so more preparation was required to obtain alerts for inputting to the graph.

To start, each day's TCPdump data were split into training and test data. The size of the training data was defined based on where the simulated attacks started in the data up to a maximum of 300,000 packets. Moreover, the training data were cleaned of any unsolicited traffic, which in the ISOT-CID dataset are labelled as malicious, so only benign data were used to train the classifier.

Kitsune requires a further split on the training data so that part of the data is used by it to learn the feature mapping (FM) phase while the rest is used to actually train the anomaly detector (AD) phase. Based on those requirements, 20% of the training data, or a minimum of 15,000 packets, were used for the FM phase, and the rest were used for the AD phase.

The final parameter required by Kitsune was the maximum size for any auto-encoder in its ensemble layer, which was set to 1 because this value is generally expected to provide the best performance.

This made it possible to train the classifier, after which Kitsune generated a root mean square error (RMSE) score for each input packet. However, those scores still did not provide a classification for the packet. To obtain classification, a threshold was required. Packets with scores above the threshold were considered malicious. Therefore, a further threshold calculation (TC) training phase was employed, with the threshold defined as the upper bound of the interquartile range (IQR) outlier detection method when applied against the scores obtained in the TC phase. Specifically, the threshold was equal to $Q3+1.5*IQR$. Table 6.19 shows the sizes of the training data for each phase per day.

Following the packet classification, each packet labelled as malicious resulted in an alert being generated that was inputted to the graph. Table 6.20 shows the performance of the

Day	FM	AD	TC	Total
Day 1	60000	240000	250000	550000
Day 2	60000	240000	190000	490000
Day 3	15000	34000	8000	57000
Day 4	48000	192000	350000	590000

Table 6.19: Daily Kitsune training data size (number of packets) for each phase for the ISOT-CID Phase 1 dataset.

generated Kitsune alerts when classifying hosts as benign or malicious.

Day	TP	TN	FP	FN	PPV	F1	BM	MCC
Day 1	18	4	14	42	0.56	0.39	−0.48	−0.41
Day 2	35	3	15	81	0.70	0.42	−0.53	−0.37
Day 3	4	8	8	66	0.34	0.10	−0.44	−0.50
Day 4	30	3	13	48	0.70	0.50	−0.43	−0.32
Combined	87	18	50	237	0.64	0.38	−0.47	−0.37

Table 6.20: Daily and combined performance of Kitsune alerts for the ISOT-CID Phase 1 dataset.

Table 6.20 shows that Kitsune had a very high number of errors, and in all cases performed close to or worse than a random classifier. When focusing only on its predictive power, a combined precision of 0.64 was obtained. The combined results summarize these findings with an F1 score of 0.38, a BM of −0.47 and an MCC of −0.37.

After performing the belief propagation on the 4 days of the dataset, there was a significant improvement in the performance of the model when comparing those numbers to the base Kitsune performance. Table 6.21 shows the results.

Day	TP	TN	FP	FN	F1	BM	MCC
Day 1	53	14	4	7	0.91	0.66	0.63
Day 2	95	15	3	21	0.89	0.65	0.50
Day 3	66	14	2	4	0.96	0.82	0.78
Day 4	73	14	2	5	0.95	0.81	0.76
Combined	287	57	11	37	0.92	0.72	0.62

Table 6.21: Daily and combined performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Kitsune alerts.

As with the previous experiment, the model showed good performance, with a large reduction in the number of errors in all days. The absolute performance was slightly worse

than that of the AEN with Snort alerts, but the gains obtained when compared with the raw alerts were higher. When looking at the combined results, the gains obtained were 0.54 for F1 score, 1.19 for BM and 0.99 for MCC.

As for the parameter adaptation scenario, Figure 6.4 summarizes the performance and dispersion of each metric as the days progress for the 100 runs with the solid lines showing the median performance. The median values, as well as the adapted parameters values for each day, are also shown in Table 6.22.

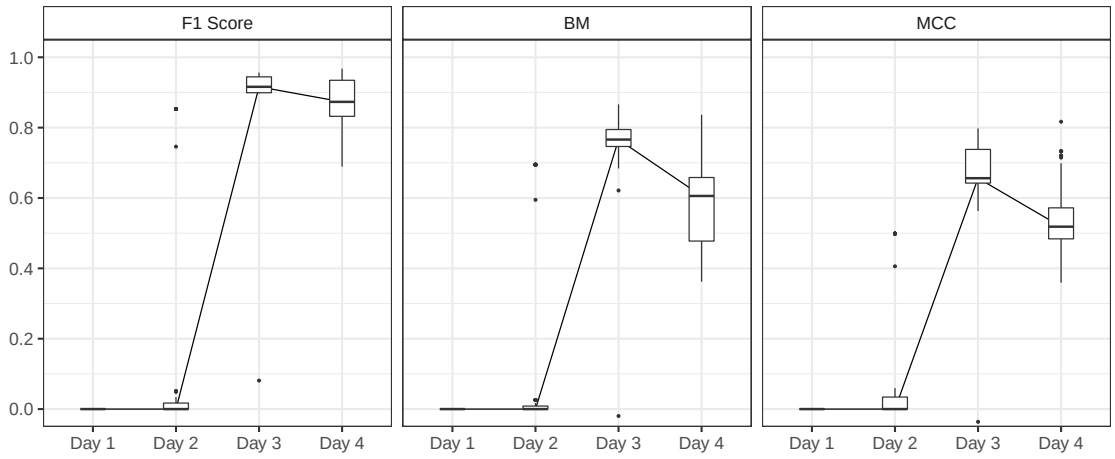


Figure 6.4: Summary of the performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Kitsune alerts and parameter adaptation. The solid lines plot the median values.

Day	P(M)	P(T)	TP	TN	FP	FN	F1	BM	MCC
Day 1	0.20	0.20	0	18	0	60	0.00	0.00	0.00
Day 2	0.41	0.28	0	18	0	116	0.00	0.00	0.00
Day 3	0.72	0.37	60	15	1	10	0.92	0.77	0.66
Day 4	0.90	0.34	62	14	2	16	0.87	0.61	0.52

Table 6.22: Median daily performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with Kitsune alerts and parameter adaptation.

The results showed behaviour similar to the previous experiment, with a performance that was initially poor but that started trending upward and reached close to the same level as the first scenario by the third day before decreasing slightly on day 4. On day 2, the dispersion (excluding outliers) was still very low with the median performance continuing to be as poor as day 1. However, there were some outlier results that reached levels comparable to those of the first scenario. This shows that in most cases, the adaptation delta for that single iteration was not enough to overcome the low initial values of the parameters but it was

possible to reach acceptable performance depending on the rewards obtained. On day 3, the median performance markedly increased, with the best runs achieving better performance than that of the first scenario, and only a single outlier showing poor performance.

Finally, the median performance of the fourth day was once again below that of day 3, likely for the same reasons as the previous experiment. The dispersion observed was high, but there were no very poor performance as with day 3. There was also a single result on day 4 that was better than the result of the scenario with parameter adaptation. Combined with day 3, this points to a high variability of the rewards generated for each run and shows that, on average, the parameters could still reach more optimal values. In effect, this illustrates the possible gains that can be obtained by applying the parameter adaptation mechanism.

6.5.2.3 Results with Combined AEN Detector Alerts

The individual performance of the detectors for the ISOT-CID Phase 1 dataset are shown in Table 6.8 for the fingerprint matching and in Table 6.12 for the anomaly detection.

On one hand, the ensemble classification using the *and* rule resulted in the same predictions as the fingerprint matching. This outcome means that all hosts classified as malicious by fingerprint matching were also classified as malicious with the anomaly detection. On the other hand, but for the same reason, the ensemble classification using the *or* rule resulted in the same predictions as the as anomaly detection.

In contrast, the performance of the belief propagation mechanism when inputted with alerts from both detectors is shown in Table 6.23.

Day	TP	TN	FP	FN	F1	BM	MCC
Day 1	50	15	3	10	0.88	0.67	0.60
Day 2	92	16	2	24	0.88	0.68	0.51
Day 3	61	14	2	9	0.92	0.75	0.66
Day 4	71	15	1	7	0.95	0.85	0.75
Combined	274	60	8	50	0.90	0.73	0.61

Table 6.23: Daily and combined performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with combined AEN detector alerts.

The results show that the belief propagation model was able to combine the good performance elements of the two detectors. It was able to correctly classify almost the same number of malicious hosts as the anomaly detection mechanism while at the same time only misclassifying a small number of benign hosts as malicious. The combined difference was 4 fewer true positives and 31 fewer false positives as the anomaly detection mechanism, and 161 more true positives and 7 more false positives as the fingerprint matching mechanism.

That represents a clear improvement when compared with each detector individually, and consequently, with the two ensemble classifications discussed previously.

Moving to the parameter adaptation scenario, Figure 6.5 summarizes the performance and dispersion of each metric as the days progress for the 100 runs with the solid lines showing the median performance. The median values and the adapted parameters values for each day are also shown in Table 6.24.

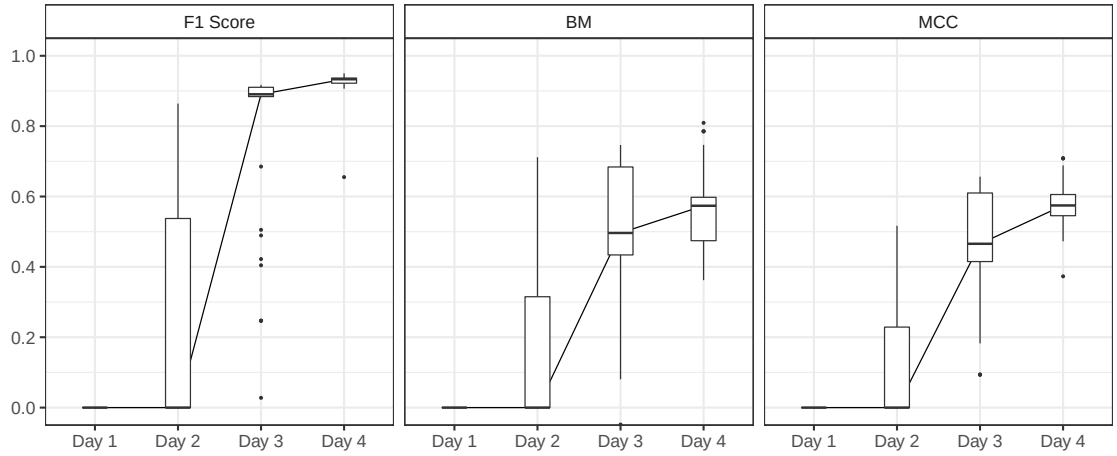


Figure 6.5: Summary of the performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with combined AEN detector alerts and parameter adaptation. The solid lines plot the median values.

Day	P(M)	P(T)	TP	TN	FP	FN	F1	BM	MCC
Day 1	0.20	0.20	0	18	0	60	0.00	0.00	0.00
Day 2	0.46	0.44	0	18	0	116	0.00	0.00	0.00
Day 3	0.75	0.69	61	10	6	9	0.89	0.50	0.47
Day 4	0.91	0.88	74	10	6	4	0.93	0.57	0.57

Table 6.24: Median daily performance of the belief propagation mechanism for the ISOT-CID Phase 1 dataset with combined AEN detector alerts and parameter adaptation.

The median results obtained for the first two days were poor, with no host classified as malicious. This was expected for day 1 given the initial parameters not being adequate for this dataset. On day 2, while the parameters trended in the right direction, the single adaptation was not enough to result in an improved median performance. However, there is a large dispersion of results, even with high values obtained on some runs.

In contrast, the median performance obtained for days 3 and 4 are high, reaching close to the same level as the first scenario. On day 4, the median number of correctly classified

malicious hosts was higher than in the first scenario, but the median number of false positives was higher as well. As with the Snort experiments, this indicates that the median parameters likely overshoot their optimal values, once again an artifact of the aggressive learning rate and momentum used. On this day, the dispersion was low.

The observed dispersion of the results of the last two days was mostly high but there was a clear reduction between day 3 and day 4, which points to a convergence of the adapted parameters on different runs.

6.5.3 CIC-IDS2017

For this dataset, two sets of experiments were run, one inputting the Kitsune alerts into the graph and another inputting alerts from both the fingerprint matching and the anomaly detection mechanisms. Given the very small prevalence of malicious hosts in this dataset, as shown in Table 6.14, the $P(M)$ and $P(M_D|M_S)$ selected for the experiments using this dataset were 0.05 and 0.10, respectively.

6.5.3.1 Results with Kitsune Alerts

The generation of Kitsune alerts followed the same process previously described, with the difference that the training data were not cleaned because there were no packets labelled as malicious. Also, because more packets were available, more data were used for training. Table 6.25 shows the sizes of the training data for each phase per day.

Day	FM	AD	TC	Total
Day 1	200000	1800000	450000	2450000
Day 2	200000	1800000	3800000	5800000
Day 3	200000	1800000	1600000	3600000
Day 4	200000	1800000	3000000	5000000

Table 6.25: Daily Kitsune training data size (number of packets) for each phase for the CIC-IDS2017 dataset.

Table 6.26 shows the performance of the generated Kitsune alerts when classifying hosts as benign or malicious. The table shows that Kitsune is skewed towards classifying hosts as malicious. Consequently, Kitsune was able to correctly classify all malicious hosts (a 100% true positive rate), but misclassified many benign hosts. A consequence of this behaviour was the very low precision obtained.

This imbalance can be further observed in the good results for the BM compared to the bad results obtained for the F1 score, MCC and precision. This dichotomy is explained by

Day	TP	TN	FP	FN	PPV	F1	BM	MCC
Day 1	1	6660	1837	0	5.4E-4	0.01	0.78	0.02
Day 2	1	7192	1824	0	5.5E-4	0.01	0.80	0.02
Day 3	2	6802	1741	0	0.001	0.01	0.80	0.03
Day 4	10	6662	1659	0	0.006	0.01	0.80	0.07
Combined	14	27316	7061	0	0.002	0.01	0.79	0.04

Table 6.26: Daily and combined performance of Kitsune alerts for the CIC-IDS2017 dataset.

how each metric uses the different classification values, either directly (TP, TN, etc.) or as rates (TPR, TNR, etc.) and either only as dividends or also as divisors. As such, while according to the BM, Kitsune had high performance levels, the MCC metric reveals that the performance was almost equivalent to that of a random classifier.

After calculating the performance of the IDS, the belief propagation was performed for each of the 4 days of the dataset. Table 6.27 shows the results.

Day	TP	TN	FP	FN	F1	BM	MCC
Day 1	1	8486	11	0	0.15	0.99	0.29
Day 2	0	9005	11	1	0.00	0.00	0.00
Day 3	2	8543	0	0	1.00	1.00	1.00
Day 4	8	8317	4	2	0.73	0.80	0.73
Combined	11	34351	26	3	0.43	0.78	0.48

Table 6.27: Daily and combined performance of the belief propagation mechanism for the CIC-IDS2017 dataset with Kitsune alerts.

As can be seen in Table 6.27, the total number of errors was much smaller than what was obtained by Kitsune, with significant improvements observed in most cases. Of note is day 3 where a perfect classification was obtained by the AEN model.

On the other hand, on day 2, where the malicious host was not classified correctly, the improvements were not clear or were negative. The explanation for this comes from the very small prevalence of malicious hosts in the dataset, which, as discussed earlier, means that misclassifying the single malicious host of the day, resulted in a 0%, true positive rate. That, in turn, affected all reported metrics.

Generally speaking, for this prevalence, the candidate best classifiers will be those that are skewed towards only a few positive classifications because it is not possible to have a high absolute number of errors when there is a strong tendency to classify hosts as benign.

When looking at the gains obtained compared to the base Kitsune classification, the

metrics in which Kitsune did not obtain good results (F1 score and MCC) are those that had, on average, the best improvement. Conversely, the results for the BM metric are somewhat dissonant, depending on how the few malicious hosts of each day were classified.

Moving to the parameter adaptation scenario, Figure 6.6 summarizes the performance and dispersion of each metric as the days progress for the 100 runs with the solid lines showing the median performance. The median values, as well as the adapted parameters values for each day, are also shown in Table 6.28.

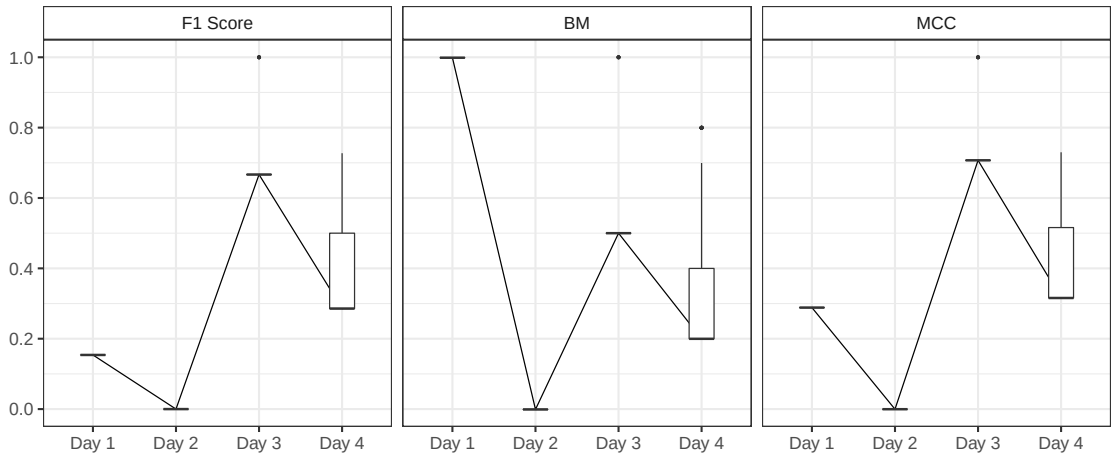


Figure 6.6: Summary of the performance of the belief propagation mechanism for the CIC-IDS2017 dataset with Kitsune alerts and parameter adaptation. The solid lines plot the median values.

Day	P(M)	P(T)	TP	TN	FP	FN	F1	BM	MCC
Day 1	0.20	0.20	1	8486	11	0	0.15	0.99	0.29
Day 2	0.12	0.12	0	9007	9	1	0.00	0.00	0.00
Day 3	0.03	0.03	1	8543	0	1	0.67	0.50	0.71
Day 4	0.01	0.01	2	8319	2	8	0.29	0.20	0.32

Table 6.28: Median daily performance of the belief propagation mechanism for the CIC-IDS2017 dataset with Kitsune alerts and parameter adaptation.

Contrary to the first two parameter adaptation experiments, this one had an initial performance in line with the experiment of the first scenario. This shows that the initial parameters were acceptable for this dataset. On day 2, the median performance was worse than on day 1 because the single malicious host of the day was not classified correctly. That mirrored the results observed in the first scenario, however, in this case, there were fewer false positive classifications so the final performance was slightly better. On day 3, there was an improvement in the median performance observed when compared to day 2, although

that was below the perfect result obtained in the first scenario. Only a few outliers were able to reach that result.

On day 4, the median results were further below those of the first scenario when compared to day 3. Once again, this can be explained by the very small prevalence of malicious hosts in the dataset such that in most runs the rewards would contain no malicious hosts or at maximum 1 host resulting in $\mathcal{O}(P(M))$ and $\mathcal{O}(P(T))$ close to zero. This resulted in a clear trend towards zero, or in effect ε since adaptation was restrained to that value, for both parameters. In fact, on day 4, both parameters reached ε on the median run, which in these experiments were set to 0.01, because these runs were never rewarded with a malicious host in one of the preceding days. As a point of comparison, the results of all the runs that were rewarded a malicious host were better, with the results of few runs that reached $P(T) \gtrsim 0.25$ being on the same level of performance as the first scenario. This same phenomenon explains the very low dispersion observed on days 2 and 3.

6.5.3.2 Results with Combined AEN Detector Alerts

The individual performance of the detectors for the CIC-IDS2017 dataset are shown in Table 6.10 for the fingerprint matching and in Table 6.13 for the anomaly detection.

As with the previous dataset, the ensemble classification using the *and* rule resulted in the same predictions as the fingerprint matching, while the ensemble classification using the *or* rule resulted in the same predictions as the anomaly detection. This again shows that hosts that were classified as malicious by fingerprint matching were also classified as malicious with the anomaly detection.

In contrast, the performance of the belief propagation mechanism when inputted with alerts from both detectors is shown in Table 6.29.

Day	TP	TN	FP	FN	F1	BM	MCC
Day 1	1	8486	11	0	0.15	0.99	0.29
Day 2	0	9005	11	1	0.00	0.00	0.00
Day 3	2	8543	0	0	1.00	1.00	1.00
Day 4	8	8317	4	2	0.73	0.80	0.73
Combined	11	34351	26	3	0.43	0.78	0.48

Table 6.29: Daily and combined performance of the belief propagation mechanism for the CIC-IDS2017 dataset with combined AEN detector alerts.

The results show that the belief propagation mechanism was able to reduce the number of classification errors compared with the two detectors. Specifically, the model was able to increase or equal the number of true positives compared with the fingerprint matching

mechanism while reducing the number of false positives compared with the anomaly detection mechanism. Of note are days 2 and 3. On day 2, the model failed to identify the single malicious host of the day, which resulted in metric values of 0. In contrast, on day 3, the model obtained a perfect result, with fewer false positives than both detectors. finally, the performance metrics obtained for the other two days were mostly better than the ones obtained by the other detectors. These results are the same as the ones obtained with the Kitsune alerts, which points to a strong influence of the graph structure and of the parameter values used for this dataset.

Moving to the parameter adaptation scenario, Figure 6.7 summarizes the performance and dispersion of each metric as the days progress for the 100 runs with the solid lines showing the median performance. The median values, as well as the adapted parameters values for each day, are also shown in Table 6.30.

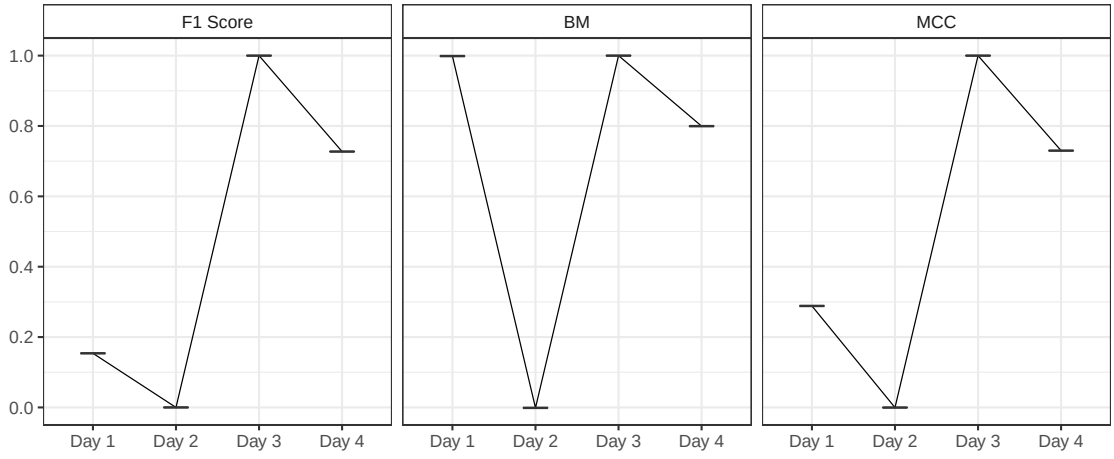


Figure 6.7: Summary of the performance of the belief propagation mechanism for the CIC-IDS2017 dataset with combined AEN detector alerts and parameter adaptation. The solid lines plot the median values.

Day	P(M)	P(T)	TP	TN	FP	FN	F1	BM	MCC
Day 1	0.20	0.20	1	8486	11	0	0.15	0.99	0.29
Day 2	0.12	0.12	0	9005	11	1	0.00	0.00	0.00
Day 3	0.03	0.03	2	8543	0	0	1.00	1.00	1.00
Day 4	0.03	0.01	8	8317	4	2	0.73	0.80	0.73

Table 6.30: Median daily performance of the belief propagation mechanism for the CIC-IDS2017 dataset with combined AEN detector alerts and parameter adaptation.

The median results obtained in this scenario were the same as the first scenario. This

shows that the initial parameters were acceptable for this dataset but also reinforces the notion that for this dataset, the graph structure exerts a strong influence on the results that are hard to be changed by the alerts inputted. Moreover, there is no dispersion of the results over the different runs even if there is a dispersion in the parameters used over different runs, which shows that the adapted parameters trended to acceptable values and implies that the other parameters that were not adapted exert a stronger influence on the results of this dataset than the ones that were adapted.

6.6 Running Times

This section presents running times samples of the six main algorithms involved in the experiments for all days of both datasets so their time complexity can be discussed. Specifically, those are the graph creation, fingerprint matching, anomaly detection, threat horizon generation, threat graph generation and belief propagation. The data and the parameters used are the same as the ones used in the experiments.

The computer used had an Intel Core i7-10750H processor with 6 cores and 12 threads, and 32 GB of RAM. To generate values that are useful as a baseline of performance, all algorithms were executed single-threaded.

For the graph creation, the values represent the time spent processing the data and creating the graph elements but exclude reading data from disk and contacting external services, such as DNS servers, or reading external data from cache. The data used in this case includes network (TCPdump) and log data but not any alerts.

The values of the fingerprint matching show the execution time of matching all fingerprints in sequence and collecting results but excludes any processing of the results such as generating alerts that can be fed into the graph.

Likewise, the values of the anomaly detection show the execution time of extracting all features, binning them and calculating the respective anomaly scores but excludes any processing of the results.

For the final three algorithms, the alerts from the fingerprint matching and the anomaly detection mechanisms were added to the graph.

The threat horizon generation values represent the time required to generate both the forward and the reverse threat horizons for all hosts in the graph sequentially.

For the threat graph generation, the values correspond to the execution time of creating the threat graphs of all hosts in the graph sequentially given that the threat horizons are already created and in memory.

Finally, the values of the belief propagation algorithm represent the time required to

propagate the beliefs in all threat graphs sequentially given that the threat graphs are already created and in memory.

The values for the ISOT-CID Phase 1 dataset and for the CIC-IDS2017 dataset are presented in Table 6.31 and Table 6.32, respectively.

Day	Data Size	GC	FP	AD	TH	TG	BP
Day 1	3.3 GB	15.2s	0.26s	0.14s	0.18s	0.21s	0.86s
Day 2	4.5 GB	27.7s	0.34s	0.26s	1.28s	0.62s	2.95s
Day 3	803 MB	1.5s	0.28s	0.21s	0.14s	0.23s	1.08s
Day 4	969 MB	3.2s	0.17s	0.13s	0.13s	0.15s	0.61s

Table 6.31: Running time samples for the ISOT-CID Phase 1 dataset. GC stands for graph creation, FP for fingerprint matching, AD for anomaly detection, TH for threat horizon generation, TG for threat graph generation, and BP for belief propagation.

Day	Data Size	GC	FP	AD	TH	TG	BP
Day 1	10.3 GB	42s	6.8s	1.4s	2h28m20s	21m42s	47m11s
Day 2	12.5 GB	7m30s	6.7s	1.4s	2h14m40s	23m42s	53m22s
Day 3	7.8 GB	41s	6.6s	1.6s	1h58m58s	21m16s	52m14s
Day 4	8.3 GB	30m59s	7.5s	2.0s	2h16m09s	33m20s	1h15m39s

Table 6.32: Running time samples for the CIC-IDS2017 dataset. GC stands for graph creation, FP for fingerprint matching, AD for anomaly detection, TH for threat horizon generation, TG for threat graph generation, and BP for belief propagation.

As can be seen when comparing the two tables, the algorithms are very fast for the relatively small number of hosts of the ISOT-CID Phase 1 dataset but can become extremely slow when dealing with the bigger graphs of the CIC-IDS2017 dataset.

Graph creation is clearly not proportional to the size of the data but instead grows according to the size and centralization of the generated graph. That means that given two datasets with the same size of data, the graph creation time of the one that results in a graph with greater order and size will be slower. Moreover, given two graphs of the same size but different levels of centralization, that is one having all nodes with similar degrees and the other having a few nodes with high degree centrality compared to the rest of the nodes, the creation time of the more centralized graph will be slower.

The other algorithms had very distinct behaviours. Both the fingerprint matching and the anomaly detection performed well on both datasets, with the increase in execution time

appearing to be logarithmic in relation to the order and size of the graph even with the general subgraph isomorphism problem being known to be an NP problem. That is explained by the characteristics of the proposed fingerprints and of the anomaly features and by the use of indexes that speed up both algorithms.

In contrast, the three belief propagation algorithms demonstrated a quadratic or even exponential complexity, particularly the threat horizon generation. This algorithm is bounded by the diameter of the graph, which means that defining depth limits to the algorithm can greatly speed it up although that can impact the correctness of the resulting threat horizons. Other techniques like maintaining partial caches of the threat horizons can also speed up the algorithm but at the cost of extra memory. Moreover, because the generation of each threat horizon is independent, this task can easily be parallelized. The expected speed-up in this case is proportional to the number of CPUs employed with the upper bound being the generation of the threat horizon with the largest size, normally the ones of the host with the greatest degree. It is also possible to maintain the threat horizons and update them together with the graph such that they are already built when needed. The downside of this approach is that each graph update operation will be slower.

Finally, the threat graph generation and belief propagation algorithms have a similar complexity and are clearly more efficient than the threat horizon generation algorithm. They are bounded by the order and the size of the threat horizons and threat graphs, respectively. Moreover, both algorithms can be parallelized in the same manner and with the same end results as the threat horizon generation. The belief propagation algorithm can be further parallelized following the fork-join model such that in each iteration, the task is forked for each node and joined at the end. In this case, the join allows for synchronization and for verifying whether the algorithm has converged or not.

6.7 Summary

In this chapter, the experimental evaluation of the three proposed detection mechanisms was presented and the results discussed. The three detection mechanisms were evaluated individually and in combination using two different datasets, namely the ISOT-CID Phase 1 dataset and the CIC-IDS2017 dataset. Moreover, the efficiency of the six main algorithms involved in the experiments was discussed.

The next chapter provides the concluding remarks and discusses the future work.

Chapter 7

Conclusion

7.1 Contribution Summary

This dissertation proposed a new knowledge graph model, called the AEN, that leverages data from both the traditional security ecosystem and beyond the organization perimeter to capture the activities and relationships of network agents as well as their inherent dynamicity and uncertainty, and through that, increase situational awareness of the threat environment and allow detecting, responding and investigating sophisticated and stealth attacks.

As a theoretical foundation, the model combines the traditionally separate concepts of the dynamic graph (described in Subsection 2.3.2) and of the uncertain graph (described in Subsection 2.3.1) into a single dynamic uncertain graph concept that underlies a probability model (described in Section 3.5) which takes into consideration the uncertainty regarding each piece of data and assigns a probability of correctness value to each element in the graph.

To validate those capabilities, three unsupervised intrusion detection mechanisms were proposed and evaluated individually and in combination using two different datasets, namely the ISOT-CID Phase 1 dataset and the CIC-IDS2017 dataset.

The first detection mechanism proposed is a signature-based scheme, described in Section 4.1, that employs an isomorphic subgraph matching algorithm to search for graphical attack patterns, called attack fingerprints, in the graph. As a proof of concept, fingerprints for scanning, DoS and password guessing attacks were provided.

The evaluation of this mechanism was presented in Section 6.3 and produced promising results, particularly considering the limited number of fingerprints available and the specific types of errors encountered. For the ISOT-CID dataset, it obtained a combined precision of 0.99 and a combined sensitivity of 0.35, while for the CIC-IDS2017 dataset, it obtained a combined precision of 0.44 and a combined sensitivity of 0.57. Ultimately, they demonstrate that this detection method is capable of identifying known attacks and is particularly suited

to identifying stealth attacks, which is a weakness of traditional signature-based intrusion detection systems.

The second scheme proposed is an anomaly detection mechanism, described in Section 4.2, that involves calculating anomaly scores based on the bits of meta-rarity metric introduced by Ferragut et al. [32] for 15 statistical features and underlying distributions extracted from the AEN graph.

The results obtained in the evaluation of this scheme, which were presented in Section 6.4, were particularly encouraging for the CIC-IDS2017 dataset, with a BM of over 0.99 and an MCC of 0.46. This shows that the scheme has a high capacity for detecting anomalous behaviour when there was a low prevalence of malicious elements in the network.

The final detection mechanism proposed, described in Chapter 5, leverages the alerts from different IDSs that have been inputted into the graph as IOCs. It works by deriving graphs akin to MRFs from the main AEN graph and performing a probabilistic inference on the derived graphs using a modified belief propagation method that leverages the probability model of the graph. Furthermore, it contains a "human-in-the-loop" online parameter adaptation mechanism (described in Section 5.3) based on the stochastic gradient descent algorithm, which reduces the initial burden of selecting the system's parameters and allows for frequent adaptation of parameters in dynamic environments.

The evaluation of this mechanism was presented in Section 6.5 and involved five sets of experiments: three using the ISOT-CID Phase 1 dataset, one with Snort alerts, one with Kitsune alerts, and one with alerts from both the fingerprint matching and the anomaly detection mechanisms that were fed back into the model, plus two using the CIC-IDS2017, one with Kitsune alerts, and another with the combined alerts from the fingerprint matching and the anomaly detection mechanisms.

Each set of experiments was performed under two different scenarios. The first used pre-defined parameters selected to simulate parameter definition from expert knowledge and no parameter adaptation. The second used best-guess parameters but employed parameter adaptation.

Under the first scenario, when looking into the experiments using Snort and Kitsune alerts, the scheme obtained large gains in detection performance compared to the individual IDSs by themselves in all three experiments. The combined gains for the BM and the MCC metrics were respectively 0.45 and 0.46 for the ISOT-CID Phase 1 dataset with the Snort alerts, 1.19 and 0.99 for the same dataset with the Kitsune alerts, and 0.44 and -0.01 for the CIC-IDS2017 dataset with the Kitsune alerts.

The two experiments using alerts from the other two AEN detection mechanisms show that the scheme was able to combine the good performance elements of the two detectors

by increasing the number of true positives compared with the fingerprint matching mechanism while reducing the number of false positives compared with the anomaly detection mechanism.

Under the second scenario, the experiments showed that the online parameter adaptation mechanism is capable of quickly reaching near-optimal parameters under most circumstances.

Some limitations of the proposed model and of the detection mechanisms are as follows:

- The high computational cost required to build and maintain the AEN graph.
- The amount of human effort needed to create the attack fingerprints.
- The high computational cost required to search for isomorphic subgraphs given the high complexity of the subgraph matching algorithms, although that is largely mitigated by the characteristics of the proposed fingerprints and by the use of indexes at the cost of extra memory requirements.
- The binning of the values of the anomaly features which can result in suboptimal distributions in some cases.
- The high computational cost required to extract the threat graphs and perform the belief propagation algorithm on them.
- The requirement for human analysts to provide labels useful for the adaptation mechanism.

7.2 Future Work

The future work will involve several possible improvements to the model, such as:

- Explore in greater depth the identification and quantification of associations between different network elements.
- Introduce metrics and algorithms for pruning data that becomes outdated or stale.
- Investigate other detection mechanisms.
- Improve and extend the proposed fingerprint database to add other types of attacks, particularly those that are traditionally harder to detect, such as HTTP request smuggling.
- Increase the feature space of the anomaly detection mechanism by introducing more features, which can help improve detection accuracy, particularly in environments that have a high prevalence of malicious hosts.

- Implement more advanced classification rules as part of the anomaly detection mechanism.
- Employ adaptive bin width values according to the value range of each given variable to improve the fitness of the bin distributions.
- Explore other mechanisms of reinforcement learning to better deal with very imbalanced datasets like the CIC-IDS2017.
- Conduct further evaluations using other datasets, such as the ISOT-CID Phase 2 dataset and the CSE-CIC-IDS2018 dataset [15].

Bibliography

[1] Eugene Albin. A comparative analysis of the snort and suricata intrusion-detection systems. Master's thesis, Naval Postgraduate School, 2011.

[2] Abdulaziz Aldribi, Issa Traore, and Belaid Moa. *Data Sources and Datasets for Cloud Intrusion Detection Modeling and Evaluation*, pages 333–366. Springer International Publishing, Cham, 2018.

[3] Abdulaziz Aldribi, Issa Traoré, Belaid Moa, and Onyekachi Nwamuo. Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking. *Computers & Security*, 88, 2020.

[4] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[5] Edward G Amoroso. *Fundamentals of computer security technology*. PTR Prentice Hall New Jersy, 1994.

[6] Michael Artz. Netspa: a network security planning architecture. Master's thesis, Massachusetts Institute of Technology, 2002.

[7] Richard Bejtlich. What APT is (and what it isn't). *Information Security*, 12(6):20–24, 2010.

[8] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Márk Félegyházi. The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971–1003, 2012.

[9] Parth Bhatt, Edgar Toshiro Yano, and Per M. Gustavsson. Towards a framework to detect multi-stage advanced persistent threats attacks. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pages 390–395, April 2014.

[10] Monowar H. Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. Surveying port scans and their detection methodologies. *The Computer Journal*, 54:1565–1581, 2011.

[11] Mitko Bogdanoski, Tomislav Suminoski, and Aleksandar Risteski. Analysis of the syn flood dos attack. *International Journal of Computer Network and Information Security (IJCNIS)*, 5(8):1–11, 2013.

[12] Pruet Boonma and Juggapong Natwichai. Reliable cluster on uncertain multigraph. In *2015 18th International Conference on Network-Based Information Systems*, pages 494–498, September 2015.

[13] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

[14] Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Slow dos attacks: definition and categorisation. *International Journal Trust Management in Computing and Communications*, 1:300–319, 2013.

[15] Canadian Institute for Cybersecurity. Cse-cic-ids2018 on aws: A collaborative project between the communications security establishment (CSE) & the canadian institute for cybersecurity (CIC).

[16] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Measuring temporal lags in delay-tolerant networks. In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 209–218, May 2011.

[17] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, April 2012.

[18] Lei Chen and Changliang Wang. Continuous subgraph pattern search over certain and uncertain graph streams. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1093–1109, August 2010.

[19] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In Bart De Decker and André Zúquete, editors, *Communications and Multimedia Security*, pages 63–72, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[20] Yanbei Chen, Xiatian Zhu, Wei Li, and Shaogang Gong. Semi-supervised learning under class distribution mismatch. In *AAAI*, 2020.

[21] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 2020.

[22] David D. Clark and Susan Landau. The problem isn't attribution: it's multi-stage attacks. In *ReARCH '10*, 2010.

[23] Tyler Cody. A layered reference model for penetration testing with reinforcement learning and attack graphs. In *2022 IEEE 29th Annual Software Technology Conference (STC)*, pages 41–50, 2022.

[24] Gregory F Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

[25] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1367–1372, 2004.

[26] Marc Dacier, Yves Deswarte, and Mohamed Kaâniche. Quantitative assessment of operational security: Models and tools. *Information Systems Security*, pages 179–86, 1996.

[27] Fangfang Dai, Yingwu Hu, Kangfeng Zheng, and Bin Wu. Exploring risk flow attack graph for security risk assessment. *IET Information Security*, 9:344–353, 2015.

[28] David J. Day and Benjamin M. Burns. A performance analysis of snort and suricata network intrusion detection and prevention engines. In *ICDS 2011*, 2011.

[29] Marco De Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O de Vivo. A review of port scanning techniques. *ACM SIGCOMM Computer Communication Review*, 29(2):41–48, 1999.

[30] P. L. Dobruschin. The description of a random field by means of conditional probabilities and conditions of its regularity. *Theory of Probability and Its Applications*, 13:197–224, 1968.

[31] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[32] Erik M. Ferragut, Jason A. Laska, and Robert A. Bridges. A new, principled approach to anomaly detection. *2012 11th International Conference on Machine Learning and Applications*, 2:210–215, 2012.

[33] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. *Proceedings of the 2018 International Conference on Management of Data*, 2018.

[34] Ni Gao, Ling Gao, Quanli Gao, and Hai Wang. An intrusion detection model based on deep belief networks. *2014 Second International Conference on Advanced Cloud and Big Data*, pages 247–252, 2014.

[35] Paul Giura and Wei Wang. A context-based detection framework for advanced persistent threats. In *2012 International Conference on Cyber Security*, pages 69–74, December 2012.

[36] Will Gragido. Lions at the Watering Hole – The "VOHO" Affair « Speaking of Security – The RSA Blog and Podcast. https://web.archive.org/web/20121212050809/https://blogs.rsa.com/lions-at-the-watering-hole-the-voho-affair, December 2012.

[37] Kapil Kumar Gupta, Baikunth Nath, and Kotagiri Ramamohanarao. Conditional random fields for intrusion detection. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, volume 1, pages 203–208. IEEE, 2007.

[38] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. *Proceedings of the 2019 International Conference on Management of Data*, 2019.

[39] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD '13*, 2013.

[40] Frank Harary and Gopal Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7):79–87, 1997.

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

[42] Kyle Ingols, Richar Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 121–130, December 2006.

[43] Georgios Ioannou, Panos Louvieris, Natalie Clewley, and Gavin Powell. A markov multi-phase transferable belief model: An application for predicting data exfiltration APTs. In *Proceedings of the 16th International Conference on Information Fusion*, pages 842–849, July 2013.

[44] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005.

[45] Farah Jemili, Montaceur Zaghdoud, and Mohamed Ben Ahmed. Intrusion detection based on "hybrid" propagation in bayesian networks. *2009 IEEE International Conference on Intelligence and Security Informatics*, pages 137–142, 2009.

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

[47] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. Distance-constraint reachability computation in uncertain graphs. *PVLDB*, 4:551–562, 2011.

[48] Rudolf E. Kálmán. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.

[49] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 66:214–235, 2016.

[50] R. Kikuchi. A theory of cooperative phenomena. *Physical Review*, 81:988–1003, 1951.

[51] Hyunjun Kim, Sunwoo Ahn, Whoi Ree Ha, Hyunjae Kang, Dong Seong Kim, Huy Kang Kim, and Yunheung Paek. Panop: Mimicry-resistant ann-based distributed nids for iot networks. *IEEE Access*, 9:111853–111864, 2021.

[52] Vassilis Kostakos. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007 – 1023, March 2009.

[53] Igor Kotenko and Elena Doynikova. Security assessment of computer networks based on attack graphs and security events. In *International Conference on Information and Communicatiaon Technology*, 2014.

[54] Brian Krebs. Chinese VPN Service as Attack Platform? – Krebs on Security. https://krebsonsecurity.com/2015/08/chinese-vpn-service-as-attack-platform/, August 2015.

[55] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the royal statistical society series b-methodological*, 50:415–448, 1988.

[56] Georgiy Levchuk, John Colonna-Romano, and Mohammed Eslami. Application of graph-based semi-supervised learning for development of cyber COP and network intrusion detection. In *Disruptive Technologies in Sensors and Sensor Systems*, volume 10206, pages 10206D–1 – 10206D–16, March 2017.

[57] Richard Lippmann and Kyle Ingols. An Annotated Review of Past Papers on Attack Graphs. Technical Report PR-IA-1, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, March 2005.

[58] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. *Journal of Computer Virology and Hacking Techniques*, 13(1):47–85, February 2017.

[59] Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognit.*, 91:216–231, 2019.

[60] Haishou Ma, Yi Xie, Shensheng Tang, Jiankun Hu, and Xingcheng Liu. Threat-event detection for distributed networks based on spatiotemporal markov random field. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1735–1752, 2022.

[61] Mandiant. APT1 – exposing one of China's cyber espionage units. https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf, 2013.

[62] MazeBolt. Layer 4 | mazebolt knowledge base.

[63] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, May 2016.

[64] Jelena Mirkovic and Peter L. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *Comput. Commun. Rev.*, 34:39–53, 2004.

[65] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. *ArXiv*, abs/1802.09089, 2018.

[66] Mitre. Brute force: Password spraying.

[67] Daesung Moon, Hyungjin Im, Ikkyun Kim, and Jong Hyuk Park. DTB-IDS: an intrusion detection system based on decision tree using behavior analysis for preventing apt attacks. *The Journal of Supercomputing*, 73(7):2881–2895, July 2017.

[68] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, 1999.

[69] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, pages 109–118, New York, NY, USA, 2004. ACM.

[70] Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1, 2010.

[71] Xinming Ou, Wayne Boyer, and Miles McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 336–345, New York, NY, USA, 2006. ACM.

[72] Xinming Ou, Sudhakar Govindavajhala, and Andrew Appel. Mulval: A logic-based network security analyzer. In *USENIX Security Symposium*, volume 8, 2005.

[73] Christof Paar, Jan Pelzl, and B Preneel. Understanding cryptography: A textbook for students and practitioners, 2010.

[74] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A weakest-adversary security metric for network configuration security analysis. In *QoP '06*, 2006.

[75] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 355–366, New York, NY, USA, March 2011. ACM.

[76] Panos Parchas, Nikolaos Papailiou, Dimitris Papadias, and Francesco Bonchi. Uncertain graph sparsification. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, March 2018.

[77] Mark Patton, Eric Gross, Ryan Chinn, Samantha Forbis, Leon Walker, and Hsinchun Chen. Uninvited connections: a study of vulnerable devices on the internet of things (iot). In *2014 IEEE joint intelligence and security informatics conference*, pages 232–235. IEEE, 2014.

[78] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. *Probabilistic and Causal Inference*, 1982.

[79] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier Science, 1988.

[80] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, NSPW '98, pages 71–79, New York, NY, USA, 1998. ACM.

[81] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. k-nearest neighbors in uncertain graphs. *PVLDB*, 3:997–1008, 2010.

[82] Darpa Internet Program. Transmission Control Protocol. RFC 793, September 1981.

[83] Thomas H Ptacek and Timothy N Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks inc Calgary Alberta, 1998.

[84] Han Qiu, Tian Dong, Tianwei Zhang, Jialiang Lu, Gérard Memmi, and Meikang Qiu. Adversarial attacks against network intrusion detection in iot systems. *IEEE Internet of Things Journal*, 8:10327–10335, 2021.

[85] Uri Rivner. Anatomy of an Attack « Speaking of Security – The RSA Blog and Podcast. https://web.archive.org/web/20110404003357/http://blogs.rsa.com/rivner/anatomy-of-an-attack/, April 2011.

[86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[87] Arnaud Sallaberry, Chris Muelder, and Kwan-Liu Ma. Clustering, visualizing, and navigating for large dynamic graphs. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing*, pages 487–498, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[88] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[89] Petteri Sevon, Lauri Eronen, Petteri Hintsanen, Kimmo Kulovesi, and Hannu Toivonen. Link discovery in graphs derived from biological databases. In Ulf Leser, Felix Naumann, and Barbara Eckman, editors, *Data Integration in the Life Sciences*, pages 35–49, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[90] Jawwad Ahmed Shamsi, Sherali Zeadally, Fareha Sheikh, and Angelyn Flowers. Attribution in cyberspace: techniques and legal implications. *Security and Communication Networks*, 9:2886–2900, 2016.

[91] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, 2018.

[92] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *2002 IEEE Symposium on Security and Privacy (S&P'02)*, page 273. IEEE, 2002.

[93] Oleg Sheyner and Jeannette Wing. Tools for generating and analyzing attack graphs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, pages 344–371. Springer Berlin Heidelberg, 2003.

[94] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 305–316, Washington, DC, USA, 2010. IEEE Computer Society.

[95] Aditya K. Sood and Richard J. Enbody. Targeted cyberattacks: A superset of advanced persistent threats. *IEEE Security Privacy*, 11(1):54–61, January 2013.

[96] NIST SP. 800-39. *Managing information security risk, National Institute of Standards and Technology, NIST*, 2011.

[97] Frank Spitzer. Markov random fields and gibbs ensembles. *American Mathematical Monthly*, 78:142–154, 1971.

[98] Indraneel Sreeram and Venkata Praveen Kumar Vuppala. Http flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. *Applied Computing and Informatics*, 2019.

[99] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10:105–136, 2002.

[100] Gaurav Tandon and Philip K Chan. Tracking user mobility to detect suspicious behavior. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 871–882. SIAM, 2009.

[101] Rajat Tandon. A survey of distributed denial of service attacks and defenses. *ArXiv*, abs/2008.01345, 2020.

[102] Claudio Taranto, Nicola Di Mauro, and Floriana Esposito. Uncertain (multi)graphs for personalization services in digital libraries. In Maristella Agosti, Floriana Esposito, Stefano Ferilli, and Nicola Ferro, editors, *Digital Libraries and Archives*, pages 141–152, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[103] Gina C. Tjhai, Maria Papadaki, Steven Furnell, and N. Clarke. The problem of false alarms: Evaluation with snort and darpa 1999 dataset. In *TrustBus*, 2008.

[104] Nicolás García Trillos, Zachary T. Kaplan, Thabo Samakhoana, and Daniel Sanz-Alonso. On the consistency of graph-based bayesian semi-supervised learning and the scalability of sampling algorithms. *Journal of Machine Learning Research*, 21:1–47, 2020.

[105] Wojciech Tylman. Anomaly-based intrusion detection using bayesian networks. *2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*, pages 211–218, 2008.

[106] Wojciech Tylman. Misuse-based intrusion detection using bayesian networks. *2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*, pages 203–210, 2008.

[107] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[108] Oskar van Rest, Sungpack Hong, Jinha Kim, Xu Meng, and Hassan Chafi. Pgql: a property graph query language. In *GRADES '16*, 2016.

[109] Nikos Virvilis, Dimitris Gritzalis, and Theodoros Apostolopoulos. Trusted computing vs. advanced persistent threats: Can a defender win this game? In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE*

10th International Conference on Autonomic and Trusted Computing, pages 396–403, December 2013.

[110] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 13:260–269, 1967.

[111] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1242–1254, 2016.

[112] Lingyu Wang, Sushil Jajodia, Anoop Singhal, Pengsu Cheng, and Steven Noel. k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 11:30–44, 2014.

[113] Zhendong Wang, Yong Zeng, Yaodi Liu, and Dahai Li. Deep belief network integrating improved kernel-based extreme learning machine for network intrusion detection. *IEEE Access*, 9:16062–16091, 2021.

[114] Tyler Wrightson. *Advanced Persistent Threat Hacking: The Art and Science of Hacking Any Organization*. McGraw-Hill Education Group, 1st edition, 2014.

[115] Liyuan Xiao, Yetian Chen, and Carl K. Chang. Bayesian model averaging of bayesian network classifiers for intrusion detection. *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 128–133, 2014.

[116] Li Xiaoming, Valon Sejdini, and Hasan Chowdhury. Denial of service (dos) attack with udp flood. *School of Computer Science, University of Windsor, Canada*, 2010.

[117] Peng Xie, Jason H. Li, Xinming Ou, Peng Liu, and Renato Levy. Using bayesian networks for cyber security analysis. *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 211–220, 2010.

[118] Virendra Kumar Yadav, Munesh Chandra Trivedi, and BM Mehtre. Dda: an approach to handle ddos (ping flood) attack. In *Proceedings of International Conference on ICT for Sustainable Development*, pages 11–23. Springer, 2016.

[119] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8(236-239):0018–9448, 2003.

[120] Shengwei Yi, Yong Peng, Qi Xiong, Ting Wang, Zhonghua Dai, Haihui Gao, Junfeng Xu, Jiteng Wang, and Lijuan Xu. Overview on attack graph generation and visualization technology. In *2013 International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, pages 1–6, October 2013.

[121] Waleed A. Yousef, Issa Traoré, and William Briguglio. Classifier calibration: with application to threat scores in cybersecurity. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[122] Sherali Zeadally, Erwin Adi, Zubair A. Baig, and Imran A. Khan. Harnessing artificial intelligence capabilities to improve cybersecurity. *IEEE Access*, 8:23817–23837, 2020.

[123] Hao Zhang, Yongdan Li, Zhihan Lv, Arun Kumar Sangaiah, and Tao Huang. A real-time and ubiquitous network attack detection based on deep belief network and support vector machine. *IEEE/CAA Journal of Automatica Sinica*, 7:790–799, 2020.

[124] Shaojun Zhang, Jianhua Li, Xiuzhen Chen, and Lei Fan. Building network attack graph for alert causal correlation. *Computers & Security*, 27:188–196, 2008.

[125] Y. Zhang, Peisong Li, and Xinheng Wang. Intrusion detection for iot based on improved genetic algorithm and deep belief network. *IEEE Access*, 7:31711–31722, 2019.

[126] Zhaonian Zou, Hong Gao, and Jianzhong Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 633–642, New York, NY, USA, July 2010. ACM.

[127] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1203–1218, September 2010.