RECONSTRUCTING ALERT TREES FOR CYBER TRIAGE

by

ERIC FICKE, B.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Ravi Sandhu, Ph.D., Co-Chair
Shouhuai Xu, Ph.D., Co-Chair
Xiaoyin Wang, Ph.D.
Greg B. White, Ph.D.
Mimi Xie, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2022

DEDICATION

Homini Sapienti
Live long and prosper.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Shouhuai Xu, for his guidance, patience and commitment to excellence throughout the course of my studies. I would also like to thank him, as well as the rest of the committee members, for their time and critique of my dissertation. It is only with such earnest feedback that I am able to improve myself and my contributions to the academic community. I would also like to thank the many collaborators that have enabled me to grow in academic excellence over the course of my studies. Finally, I would like to thank UTSA and the Department of Computer Science for providing the institutional resources necessary for the pursuit of this research. I could not have completed this endeavor without the support of all of those mentioned here.

The research reported in this Dissertation has been supported in part by NSF Grants #1736209, #1814825 (#2122631) and #2115134, ARO Grant #W911NF-17-1-0566, and AFRL Grant #FA8750-19-1-0019.

This research was conducted in collaboration with Dr. Raymond Bateman (ARL South-Cyber) and undergraduate students Matthew Herrada (UTSA), Jake Mueller (UCCS), and Matt Brady (UCCS), as well as high school student Caleb Cox (LPHS).

May 2022

RECONSTRUCTING ALERT TREES FOR CYBER TRIAGE

Eric Ficke, Ph.D.
The University of Texas at San Antonio, 2022

Supervising Professors: Ravi Sandhu, Ph.D. and Shouhuai Xu, Ph.D.

Cyber defense operators are often confronted with a large amount of data, such as alerts generated by intrusion detection systems. Much of this data is misleading or even counterproductive in the pursuit of effective and efficient defense. The field of Cyber Triage aims to pinpoint threats to a network and provide defenders with the data pertinent to these pursuits. This turns out to be a challenging task because many cyber attacks are conducted in multiple steps and cannot be matched by existing cyber defense tools, which tend to focus on specific hosts or network links. Towards bridging this gap, this Dissertation presents a systematic study on the innovative notions of *alert paths* and *alert trees*, which present a given set of seemingly unrelated alerts in a meaningful structure. Specifically, the Dissertation makes three contributions: (i) it investigates how to formulate alerts into alert paths to make sense of them; (ii) it investigates how to formulate alerts into alert tress to most systematically represent the corresponding threats; (iii) it investigates how to reduce the sizes of these trees without losing useful information, in order to make it more feasible to visualize alert trees. For these purposes, the Dissertation presents suites of algorithms, which are validated via real-world datasets.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Dissertation Motivation

This research, in the realm of cybersecurity, aims to facilitate the process of cyber triage through network analysis with a focus on algorithmic efficiency.

1.1.1 Cyber Triage

The field of cyber triage aims to facilitate incident response by collecting information about threat on the network. Specifically, triage begins when the defender receives intelligence that an attacker has been active somewhere on the network. This activity could include reconnaissance, exploit attempts or a confirmed compromise. In any case, the triage team must investigate applicable resources, such as logs and network monitors, in order to determine the scope and severity of the attack. This is necessary before the response team can take meaningful action against the threat.

This Dissertation targets the problem of cyber triage from an algorithmic perspective, proposing systems to track threats as they cross the network. Using alerts from security devices, these systems prepare the defender to diagnose threats by returning alert paths and trees on demand, showing the scope of a given threat on the network.

1.1.2 A Gap in Cyber Triage

In real-world cyber defense operations, it is well known that defenders are are often overwhelmed with a large number of alerts, which are generated by the employed defense tools, such as intrusion detection systems (see, e.g., [1, 2]). Moreover, many of these alerts are erroneous, resulting from widespread weaknesses of the defense tools in question. As a consequence, defenders become fatigued by the false positives and begin to simply ignore alerts. When real threats go undefended, they are given time to embed themselves in systems and propagate further into the network. By the time active threats are identified, it may be too late to find the original vector of compromise. This highlights a big gap between what defenders are *given* and what defenders *need*. Specifically, it is

not sufficient to evaluate the current status of systems when they are compromised, but defenders need to access the history of the system. This Dissertation investigates a new approach to tracking suspicious activity on the network.

1.1.3 The Challenges

As mentioned, the overwhelming volume of activity on a network makes it difficult to monitor current threats, let alone past threats. This means that efficiency is of utmost importance when addressing this problem.

Additionally, the semantics of a particular attack or set of attacks is often highly relevant to the threat it poses to a network. This makes it difficult to get meaningful results from anomaly detectors, which often lack interpretability.

Finally, the size and architecture of many networks make it difficult for defenders to intuitively grasp the meaning of threats as they relate to various network components. This highlights the need for strong visualization techniques to model such threats.

1.1.4 Existing Approaches to Related Problems

To help defenders leverage the useful but hidden alerts in their cyber operations, there is a need for techniques that can distill alerts into a meaningful structure that can be immediately used by defenders. To this end, there have been many studies on fusing alerts, which aim to achieve some of the following: (i) reduction in the volume of alerts produced, while maintaining the position and timing with some level of precision, (ii) providing more holistic alerts which establish patterns of threats on a network, and (iii) identifying anomalous alerts that may have special relevance to the general state of the network. Other studies form graphs of host-based activity, aiming to: (i) identify suspicious processes on a computer, and (ii) track known threats on a computer.

These approaches have limited success because, when analyzing, visualizing or modeling attack paths, they investigate specific examples and do not consider the case of hundreds or even thousands of concurrent, co-located attack/alert paths. As a result, they may be able to form com-

pelling narratives about a particular threat, but do not see widespread use because of their limited scope.

In light of this observation, the work in this Dissertation focuses on optimizing the efficiency of alert path and tree reconstruction, conducting rigorous analysis of the asymptotic complexity of these systems while maintaining relevance to both the network and individual systems.

1.2 Dissertation Research Methodology

The overall methodology that is used in the Dissertation study is a particular kind of Data Analytics for analyzing *dynamic* data, meaning that the data is collected over a period of time and the time dimension is explicitly considered rather than assumed away [3–6]. This perspective is dubbed *Algorithmic Data Analytics*, and is complementary to other kinds of Data Analytics, such as: *Statistical Data Analytics*, which primarily use statistical models and techniques to analyze data (see, e.g., [7–14]); *Machine Learning-based Data Analytics*, which primarily use machine learning techniques to analyze data (see, e.g., [15]).

The Algorithmic Data Analytics approach designs algorithms to analyze data. The design of algorithms is driven by the needs from a cyber defense perspective, such as application- or purpose-specific representation or identification of cyber attacks. Since algorithms often deal with context-specific forms of data, a necessary preprocessing step is to design appropriate data structures to represent the data in question, which itself may not be trivial. For example, a data structure that is suitable for representing the kinds of data pertinent to the present Dissertations would be *temporal-spatial graphs*. As will be presented in the subsequent chapters, there are various ways to model such data, each with its own set of strengths and weaknesses. Thus the precise details of the data structures and algorithms with which they are used is critical in their ability to achieve their goals.

1.3 Dissertation Contributions

The Dissertation makes three contributions, which are centered at helping defenders make sense of the alerts generated by Network-based Intrusion Detection Systems (NIDSs), by turning intractable

amounts and orientations of alerts into a structured representation for automated or semi-automated reasoning.

Chapter 2 introduces the concept of alert trees in order to reconstruct multi-step attack paths. Named for its purpose in Alert Path Identification in computer Networks (APIN), this model focuses on cyber triage and prioritizes efficiency and high-yield threats, rather than an exhaustive search. In this pursuit, it also introduces the concept of *threat score*, contrasting the existing notion of *vulnerability score*. Threat score measures an attacker's attempts to exploit weaknesses in a network, rather than the existence of those weaknesses. Combining the path and tree identification algorithms with the threat score heuristic, APIN is able to quickly model many significant threats to the network, in order to provide a reliable starting point for incident response teams.

Chapter 3 proposes an alternative model of alert trees, along with its own method for reconstructing them. This approach conducts Automatic Cumulative Reconstruction of Alert Trees (AutoCRAT), and offers better network coverage and more consistent runtimes than APIN, but sacrifices general efficiency. The work also defines a more general form of threat score which is not specific to a graph structure.

Chapter 4 targets a challenge arising from the previous two chapters, in which some of the resulting alert trees are too big for defenders to view comfortably. Some trees can be as large as 8000 nodes, defeating the point of the model. To solve this problem, this chapter explores three methods for reducing the size of alert trees by removing redundancies and uninteresting components. These methods are evaluated for their ability to reduce tree size and to maintain the relevant information embedded in the trees.

The three contributions are coherent in the following sense: APIN and AutoCRAT offer two methods for constructing alert trees from network alerts, and Chapter 4 offers the means to ensure that the resulting alert trees are optimized for use by the defender.

CHAPTER 2: APIN: ALERT PATH IDENTIFICATION IN COMPUTER NETWORKS

Identifying the scope of a network attack can be difficult with limited information about the nature of the attack. The urgent nature of this problem highlights the importance of automating the cyber triage process. Because of this, it is important to investigate new methods for mapping and quantifying the threat posed by an attack, in order to prioritize actions during incident response. To this end, this chapter proposes a framework for automatic Alert Path Identification in computer Networks (APIN) by leveraging observable malicious behaviors to quantify the threat score of a set of attacks. Using two academic datasets, experimental results show that APIN is able to quickly reconstruct paths that offer meaningful insight into the nature of multi-step threats on the network, given only reasonable restrictions on network size and structure. These insights would not be possible with only existing tools, such as IDSs, and human analysts would require significant time and expertise to obtain the same findings without APIN's guidance.

2.1 Introduction

In cyber triage, a stage of incident response, the defender needs to identify the location and extent of damage that an attacker has been able to inflict. Since many cyber attacks are conducted via multiple steps [16–18], the *attack paths* that have been exploited by attackers must be identified quickly and with high certainty. Additionally, it is important that the information provided to human defenders has clear cyber security meaning and, if possible, suggests a mitigation approach [19, 20]. For these reasons, this paper aims to identify all potential attack paths (modeled as alert paths) with respect to a known or suspected target, and to rank the most significant attacks based on the magnitude of threat to the network and the temporal relationships between attacks.

2.1.1 Chapter Contributions

This chapter makes the following contributions. First, it introduces the concept of alert paths for the purpose of cyber triage. Second, it proposes a method for reconstructing alert paths from network intrusion detection system (NIDS) alerts. Named for its purpose in Alert Path Identification in computer Networks, APIN also quantifies the notion of a *threat* score, as distinguished from a *vulnerability* score, although the two terms have been used interchangeably in some of the literature. The framework utilizes output from existing network monitors such as intrusion detection systems (IDSs) and incorporates existing database software for efficiency. This approach achieves *explainability*, *resistance* to some attacks, and can be *automated*. Third, in order to demonstrate the effectiveness of the framework, two case studies show APIN's performance on two widely-accepted datasets. Experimental results demonstrate the model's ability to identify paths quickly and accurately, given reasonable restrictions on network size and architecture.

2.1.2 Related Works

The problem of identifying alert paths is closely related to the formulation of attack narratives [1] with respect to some cyber attack models (e.g., cyber kill chains [17, 18]) and the formulation of attack stories [21]. Contrasting these studies, APIN aims to achieve *automated* formulation and reconstruction of attacks that are described in diverse collections of data.

There have been studies on reconstructing Distributed Denial-of-Service (DDoS) attacks (via probabilistic packet marking) [22], malware infections on hosts and command-and-control [23–25], and network attack paths (via similarities) [26]. Contrasting these studies, APIN seeks to identify and *rank* attack paths, since real networks naturally have many discrete paths of various significance.

Some other works have taken a vulnerability-based approach to analyzing attacks [27–34]. However, vulnerability information is often incomplete or out of date, since new vulnerabilities are found and published daily [35]. Hoping to avoid this limitation, we instead use a *threat-based* approach, which may be better able to identify exposures. The identification of attack paths falls

into the broader context of cybersecurity data analytics [12, 15, 36–39].

2.1.3 Chapter Outline

The rest of this chapter is organized as follows. Section 2.2 details the proposed solution framework, including pseudocode. Section 2.3 demonstrates the usefulness of the framework using two research datasets. Section 2.4 discusses implications of the chapter and Section 2.5 concludes the chapter.

2.2 Framework

The terminology used in this chapter is given in Table 2.1.

Table 2.1: Terms used throughout the chapter on APIN.

Term	Meaning / Usage
Attacker	A computer which behaves maliciously. This does not imply ownership of the system, merely the ability to make it perform some action.
Victim	A computer which has been targeted by an attack. This does not imply compromise of the computer.
Alert Path	Series of computers on a network which begin at some attacker (the origin) and terminate at some victim (the target).
Path Link	A tuple of computers in a network (i.e., an attacker and a victim) which belong to some alert path and which are connected by at least one edge directed to the victim.
NTS	Node Threat Score, corresponding to a specific computer within the network.
PTS	Path Threat Score, corresponding to an alert path and denotes the combined threat against every system in the path.
Origin	Node from which an attacker began executing its strategy (or where this was first observed)
Target	Node which is presumed to be the objective of an attacker's mission. If specified, this node is always at the end of an alert path.

2.2.1 Problem Statement

In this chapter, a computer network is modeled as a graph $G = (V, E)$, where V is the set of vertices or nodes (representing IP addresses), E is the set of arcs or directed edges between nodes (representing suspicious communications). Each edge $e \in E$ consists of a unique tuple

$(src, dst, time, SID)$, which identifies a suspicious communication between a source IP address (src) and a destination IP address (dst) at a certain time ($time$) and indicates the type of suspicious activity via a signature identifier (SID). SIDs may be provided by intrusion detection systems or other defense tools.

Given a set of security events (e.g., IDS alerts), where an *event* is represented as an edge, these security events can formulate a graph $G = (V, E)$ as follows: vertices are extracted from each event such that both the src and dst represent vertices in the graph. The edge's SID is added to the src vertex's list annotation of $SIDs_{out}$ and the corresponding dst vertex's list annotation of $SIDs_{in}$. The dst address is added to the src vertex's list annotation of $neighbors_{out}$, which contains the set of neighbors against which it has initiated attacks. Likewise, the src vertex is added to the dst vertex's corresponding list annotation of $neighbors_{in}$, which contains the set of neighbors that have initiated attacks against it. Edges are naturally converted directly from alerts because each alert indicates an attack waged from src againt dst.

The *research problem* is to extract and rank the observed alert paths within a network based on the perceived threat of each path. The ranking of these paths should be *generalizeable* so that it can be used in any network without manual tagging and training, and *robust* so that attackers cannot easily force paths to be ranked in the wrong order (i.e., higher ranked alert paths are attacked more heavily).

2.2.2 Solution Framework

Figure 2.1 highlights the proposed framework, known as APIN, per the title of this work. The framework has 3 stages. The first stage uses the alerts provided to construct the graph and quantify the node threat score (NTS) for each node. If no suspected target is provided, this stage will also select some potential targets or origins based on NTS. The second stage is to identify possible alert paths with respect to a given node. This stage uses the network communication data from the aforementioned graph G. The third stage aggregates the NTSs in each path to obtain the path threat score (PTS). These paths are ranked according to PTS so that the paths with the most significant

threat can be handled first.

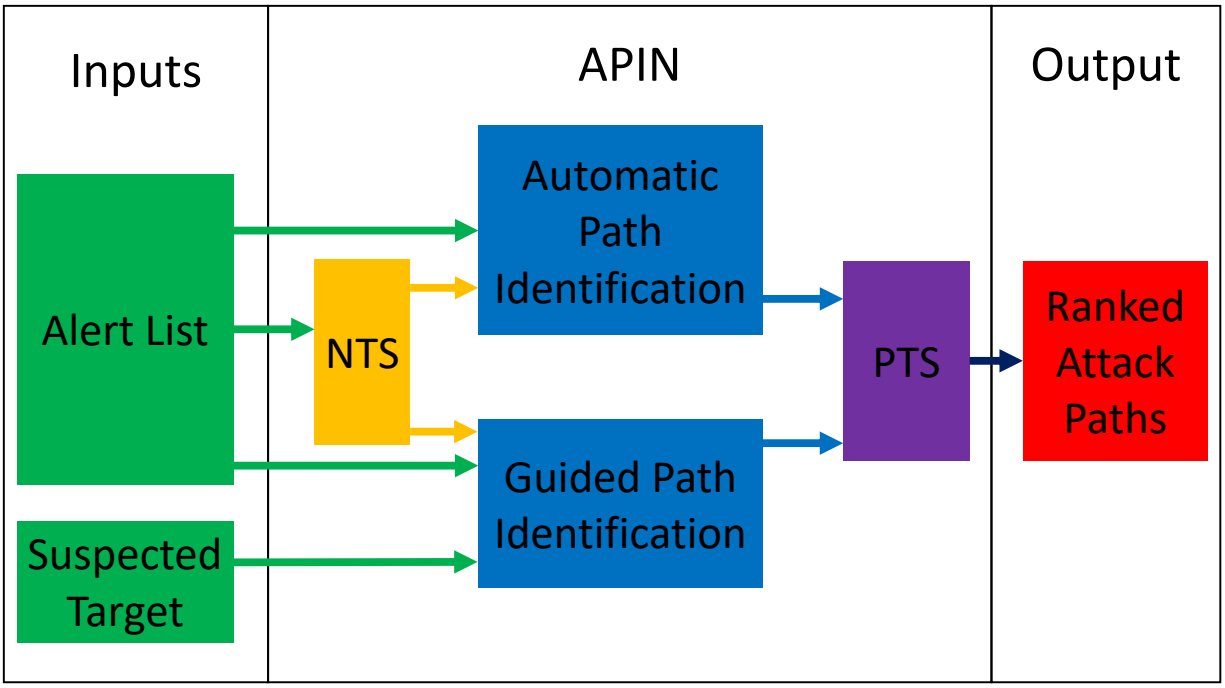

Figure 2.1: An overview of the APIN framework. NTS refers to the calculation of node threat score. PTS refers to the calculation of path threat score.

Node Threat Score

In this stage, APIN parses the security events and builds a database each for nodes and edges. Nodes are defined by the IP addresses named as the src or dst of the events. They are annotated with the quantity of incident edges by type (i.e., SID), either outbound or inbound (for the src and dst, respectively). Within the database, nodes are indexed according to address, ITS, and number of neighbors in and out. Edges are defined by the tuple described in Section 2.2.1, which is a representation of the corresponding security event. Within the database, edges are indexed according to src, dst and $time$, for each combination of the 3 terms. The combination of all three is indexed twice, to facilitate both a chronological and a reverse-chronological query, as necessary. The 'nodes' and 'edges' databases comprise $G = (V, E)$.

NTS is the first building block in alert path analysis. Intuitively, it measures the scale and cost of attacks against a node. In this case, attack cost represents the difficulty of generating and

launching diverse attacks. This value must be calculated for each computer in an alert path. The parameters used in the NTS formula are chosen as follows:

- Inbound alert diversity (D_{in}). This captures the idea that adaptive attacks by skilled attackers will result in various alarms being triggered. It is calculated as the number of inbound alert types, or 1, if there are no inbound alerts.

- Outbound alert diversity (D_{out}). As with inbound alert diversity, this captures the idea of an adaptive attacker, but is distinguished in the case of server-side attacks, data exfiltration and similar alerts. It is calculated as the number of outbound alert types, or 1, if there are no outbound alerts.

- Inbound alert scale by type (S_{in}). This provides a generic and intuitive measurement of the threat against a node. It is calculated as the geometric mean of the number of inbound alerts of each type present, or 1, if there are no inbound alerts. Alert type is defined by its signature identifier (SID).

Our initial attempt to formulate NTS also used the metric of outbound alert scale, but this was found to produce heavily skewed results with the highest values held exclusively by nodes which had conducted network scans (since these produce an inordinate amount of alerts relative to other attack types). This not only puts the focus on nodes with relatively routine activity (even though scans are indeed suspicious), but enables attackers to easily push nodes in their control to the top of the NTS ranking. Because of these observations, we removed outbound alert scale from the formula, making it resistant to the "noisy network scan diversion." For the same reason, we choose to define S_{in} using the *geometric mean* of the number of inbound alerts of each type present, rather than the *arithmetic mean*. This is because some attacks are cheaper than others (in terms of configuration complexity, time of execution, etc.), so attackers could more easily manipulate the arithmetic mean – inflating the threat score – by producing more cheap attacks (such as scans). Given this discussion, we define NTS as follows:

Definition 1 (Node Threat Score (NTS))**.** *The NTS of a node is the weighted geometric mean*

of the node's inbound alert diversity, outbound alert diversity, and inbound alert scale by type. Specifically,

$$NTS(x) = \sqrt[W]{D_{in}^{w_1}(x) \cdot D_{out}^{w_2}(x) \cdot S_{in}^{w_3}(x)}, \tag{2.1}$$

where $W = w_1 + w_2 + w_3$ *and each weight is configurable.*

For our case studies, we used the default values of 1 for each weight. If no reference node (i.e., suspected target) is provided, the 10 nodes with the highest NTS are selected and used in turn as reference nodes, as both target and origin (i.e., path identification is run 20 times). This means that the use of a reference node can significantly improve the runtime of APIN, but that the reliability of the corresponding results are wholly dependent on the reliability of the reference node.

Identifying Alert Paths

In principle, alert paths can be reconstructed with respect to a *time-based* or *node-based* approach. In the time-based approach, the idea is to parse each arc in reverse-chronological order and add newly identified nodes to the DAG as appropriate. This process is repeated for each known or suspected node in V. In the node-based approach, arcs are indexed by their source and destination, then each victim node's adjacency list is parsed significantly faster than in the time-based approach. Indexing needs only to occur once for each arc.

APIN also includes a configurable blacklist for the case that some nodes (e.g., honeypots) need not be examined. This is also useful for nodes which have a high cardinality of neighbors (such as DNS servers and broadcast addresses), which can cause an exponential increase in the runtime of the algorithm. This effect imitates that of attempting to identify all possible alert paths in a fully-connected graph, as discussed below. Nodes excluded for this reason should be manually inspected.

Because of its apparent disadvantage to the node-based approach, pseudocode for the time-based approach is omitted. Similarly, because the origin-centered algorithm follows the same logical flow as the target-centered algorithm (albeit in reverse), it is also omitted. The target-centered node-based approach is given in Algorithm 1. As shown in the pseudocode, APIN constructs a tree

Algorithm 1 Target-Centered APIN with Node Indexing

Input: Target_Address, $G = (V, E, Z)$, Blacklist
Output: Alert_Paths $= ([V])$

root $\leftarrow$ NewTree
root.$time \leftarrow TIME\_MAX$
New_Leaves $\leftarrow \{root\}$
while New_Leaves has nodes **do**
  Candidates $\leftarrow$ New_Leaves
  New_Leaves $\leftarrow \emptyset$
  for candidate $\in$ Candidates **do**
    Query edges with $dst =$ candidate and $time <$ candidate.$time$
    Sort Query_Result in reverse-chronological order
    for edge $\in$ Query_Result **do**
      if edge.$src \notin$ Blacklist $\cup$ candidate.$ancestors \cup$ candidate.$children$ **then**
        New_Leaf $\leftarrow \{src =$ edge.$src, time =$ edge.$time\}$
        Add New_Leaf to candidate.$children$
        Add New_Leaf to New_Leaves
Alert_Paths $\leftarrow \emptyset$
for leaf $\in$ root.$leaves$ **do**
  path $\leftarrow$ leaf.$ancestors$
  Add path to Alert_Paths
return Alert_Paths

of all the nodes which connect to the target, noting the time at which they do so. This preserves the temporal dependency between two attacks (i.e., a secure node cannot be used to conduct an attack before that node itself has been attacked). Once the tree has been constructed, the paths extending down to each leaf are the possible paths the attacker could have taken, with the leaves being the corresponding origin.

The time complexity of the algorithm is heavily dependent on the connectedness of the graph. The worst-case complexity is for a graph that is fully-connected such that each node has $|V|$ connections to *each* other node, with connections interleaved methodically to ensure that, at each branch of the tree, every node has an edge to every other node. In this case, the complexity is $\mathcal{O}(|V|^3)$. Alternatively, the worst case complexity as defined by the number of edges is $\mathcal{O}(|E|^2)$. That is, the worst-case complexity is $\min\{\mathcal{O}(|V|^3), \mathcal{O}(|E|^2)\}$, depending on the density of graph $G = (V, E)$. For sparsely connected graphs, many nodes will likely never enter a given tree, resulting in a significantly better expected runtime. For a reasonably secure network, attacks may be sparse, or at least concentrated around certain attackers or victims.

Path Threat Score

We introduce the concept of path threat score (PTS) to quantify the threat posed against a given path as a whole. The purpose of this definition is to gain insight into the attacker's intent and/or objective. In part, this can be inferred based on the amount of resources directed at individual computers in a path (given by the NTS). Additionally, the length of a path may be a partial indicator of how well-defined the attacker's objective may be (e.g., if their first target is their only target, perhaps the attacker has some inside knowledge about the location of data they seek to compromise). Based on these principles, we define PTS as follows:

Definition 2 (Path Threat Score (PTS))**.** *The PTS of an alert path is the sum of the NTS of all nodes in the given alert path.*

Threat score is conceptualized independently of path identification in Figure 2.2. Specifically, this figure highlights the fact that even though NTS may be calculated for each node individually,

PTS requires both the path structure and the NTS of each node in that path. For this reason, NTS may be stored in the database, but PTS may not.

The PTS calculation is designed to rank the paths containing those computers most targeted by an attacker, even if not all attacks incident to nodes in that path follow the path precisely. For example, an attacker may conduct some attacks from a variety of external nodes (e.g., using spoofing or a botnet). In this case, it is more important to model the threat against the target than to precisely determine which attacker realized the compromise of that node. Because of this, the measurement of ITS for nodes in a path may be impacted by attacks to/from computers not in the path.

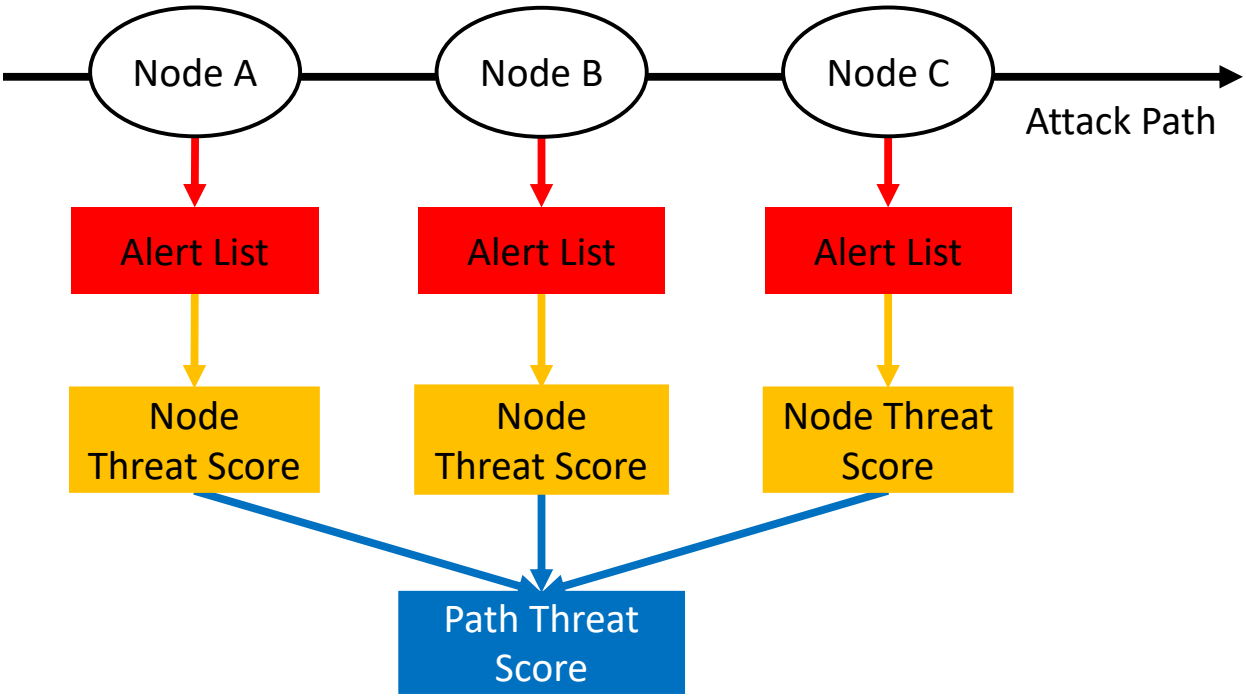

Figure 2.2: Threat Score Calculation

2.3 Evaluation with Real Data

This case study evaluates the framework using two research-oriented network traffic datasets, DARPA99 [40] and CSECICIDS2018 [41]. The former, despite major criticism and age, remains one of the most-referenced datasets for IDS evaluation today, while the latter offers a significantly

improved attack landscape, volume of data, and more robust data description.

2.3.1 Configuration

Because the alert paths are derived from the output of an existing IDS (in this case, Suricata 4.0 [42]), it is important that the IDS is properly configured for the network in question. These experiments make minimal configuration changes, with the hope that APIN will be useful to analysts even without advanced expertise and the expectation that many enterprise networks already incorporate their own custom configurations. Specifically, the $home\_net$ variable is specified according to the architecture given in the datasets, and the well-reputed Emerging Threats ruleset [43] provides rule specifications for Suricata. However, policy-based rules are not applicable to these experiments because the datasets do not specify what software use policies were implemented during the data collection. These rules are thus excluded.

The experiments were processed using an Intel Xeon X5650 (2.67GHz) CPU running ESXi 6.5.0, with two allocated VMS, one for the APIN driver and one for the database. The driver was allocated 2 cores, 32 GB RAM, and 48 GB HDD and ran Ubuntu 18.04.3 LTS Desktop. The database was allocated 4 cores, 16 GB RAM, and 100GB HDD and ran Ubuntu 16.04.2 LTS Server. They were attached to the same virtual subnet.

2.3.2 DARPA99 results

The 16,616 alerts from Suricata's output for DARPA99 were converted from text to JSON format in 1.06s. The graph of 431 nodes was constructed and indexed in 8.70s. APIN completed in 0.67s. This runtime is well within acceptable bounds.

Because of the architecture used to collect the DARPA99 data, APIN produced limited results. First, the simplicity of many of the attacks limits the ability to identify multi-step attacks. The attacks conducted in the original experiment seem to consist exclusively of a single link, where the attacker stopped the attack once the target was compromised and began a new attack, rather than pivoting between multiple internal nodes. Specifically, all of the paths identified by APIN

contained one node in the "172.16.112.0/20" subnet and one node from another subnet or external network. In this case, APIN is not useful. Nevertheless, understanding this limitation leads to the following insight.

Insight 1. *Research datasets designed to model network attacks should include multi-step attacks.*

2.3.3 CSECICIDS2018 results

According to the data description provided by its authors, the CSECICIDS2018 dataset contains a network of 450 benign nodes and a separate network of 50 attacker nodes [41]. Processing revealed an unusually high connectedness for a network of this size. Specifically, 406 nodes had over 1000 inbound neighbors (i.e., those which produced alerts when processed by Suricata) over the span of data collection, with the highest reaching 5992. This phenomenon seems to be the result of either mass spoofing by the attacker network (which is not clearly detailed in the data description) or of some third-party interference during the experiment. In this particular case, 11 of the 14 documented victims were in the top 406 highly-connected nodes. This complexity prevents the real-time parsing of the alert trees, which grow cubically. Nevertheless, this limitation offers some insight:

Insight 2. *High-granularity network segmentation is important for the efficient analysis of alert paths.*

Following this observation, a blacklist removed the nodes with over 1000 inbound neighbors, preferring a partial result over a prohibitive runtime.

The 3,323,426 alerts from Suricata's output for CSECICIDS2018 converted from text to JSON format in 3m10s. APIN constructed and indexed the graph of 97,873 nodes in 29m16s, and completed its queries in 42.08s. Given that alert processing and graph construction can proceed during data collection (which spanned a week), this runtime is well within reasonable bounds for the size of the dataset.

Another phenomenon APIN revealed is the frequent occurrence of attempts to probe or exploit the vulnerability known as EternalBlue (MS17-010). This appeared in 4 of the top 5 paths and

many besides, and was triggered by several different Emerging Threats signatures: [1:2025649:2], [1:2025992:1], [1:2025650:2]. Because the publishers of the dataset do not describe any attacks using EternalBlue or otherwise targeting SMB (the protocol which EternalBlue targets), this supports the suspicion that some of the traffic in the dataset was produced by external sources. If this is the case, it may have implications for the validity of ground truth for other experiments (such as training machine-learning based IDSs). This leads to the following insight:

Insight 3. *Datasets collected from networks with internet access should have strict controls over gateway traffic, on par with those used in production networks.*

The highest-ranked path (with a PTS of 34.31) contains five nodes (A-E) across four links. Its detail are below. Alert descriptions have been modified to improved readability, and SIDs have been included for reference. External IP Addresses have been truncated to preserve anonymity.

1. 103.aaa.aaa.aaa (A) to 172.31.67.46 (B)
    - (1:2102465:9) SMB share access
    - (1:2102466:9) SMB unicode share access
    - (1:2025649:2) ETERNALBLUE Probe MS17-010 (MSF style)
    - (1:2025992:1) ETERNALBLUE Probe MS17-010 (Generic Flags)
2. 172.31.67.46 (B) to 103.ccc.ccc.ccc (C)
    - (1:2025650:2) ETERNALBLUE Probe Vulnerable System Response MS17-010
3. 103.ccc.ccc.ccc (C) to 172.31.66.112 (D)
    - (1:2102466:9) SMB unicode share access
    - (1:2102465:9) SMB share access
    - (1:2025649:2) ETERNALBLUE Probe MS17-010 (MSF style)
    - (1:2025992:1) ETERNALBLUE Probe MS17-010 (Generic Flags)

4. 172.31.66.112 (D) to 54.eee.eee.eee (E)

    - (1:2016149:2) Session Traversal Utilities for NAT (STUN Binding Request)

The attacker probes node B in link 1, which is verified as vulnerable in link 2. This sequence may be indicative of a reflected attack (in which the response calls back to an IP distinct from the one used to initiate the attack), of a two-part attack (in which the attacker probes from node A and launches the full exploit from node C), or of two probes from distinct attackers. APIN does not determine the precise meaning of this interaction because of the nature of the model. Specifically, because the model is directional and acyclical, APIN only describes one side of each connection. In any case, the defender can manually verify the incident now that APIN has ranked its threat score appropriately. Note that the output given above does not show the temporal relationship between alerts, except insofar as at least one alert of a given link must precede at least one alert of all following links.

2.4 Chapter Discussion

The present study has several limitations, which should be addressed in future studies. First, the path identification algorithms are limited by the ability of existing IDSs to identify malicious and anomalous traffic. False-negatives from these devices may prevent APIN from identifying certain links, resulting in paths that are too short. However, because path links require only a single edge to be included, but are ranked according to threat score, False negatives from IDSs may cause negligible harm in the APIN model if there are other correctly-identified attacks along the same link as the missed ones.

Second, false-negatives and false-positives from the IDS(s) used may reduce accuracy in the calculation of NTS for affected nodes (and therefore PTS for affected paths). However, poor accuracy in the source data can be mitigated in part by careful configuration of IDS parameters. During the design and testing of APIN, policy-based rules posed particular difficulty as, prior to reconfiguration, they imposed a false-positive rate of 82.2% in the CSECICIDS2018 dataset.

Third, although NTS and PTS are objective standards, they are subject to certain parameters, such as the timespan of data collection, which must be consistent for the metrics to preserve their meaning between various samples or datasets. A simple solution to this could be to define a specific amount of time for which to consider, but at present such a definition would be arbitrary and unprincipled. This remains for future work.

Fourth, the above also makes it difficult to share precise intelligence between different organizations. However, since the sharing of this sort of intelligence between different networks is often subject to privacy concerns anyway, and since the alert paths can be abstracted with relative ease (e.g., replacing an IP address with "DNS server"), this limitation is not prohibitive to the model's usefulness.

2.5 Chapter Summary

This chapter introduced an empirical approach for modeling multi-step attacks. The model is based on the concept of threat score, which includes node threat score and path threat score. The model also includes algorithms to identify potential alert paths, including the option to specify a known or suspected target or origin, and alternatively, can automatically identify high-threat nodes as well as paths targeting or originating from them. The model depends on the assumption of certain reasonable parameters which must be met in the source network in order for the model to be useful and efficient. The experiments demonstrated that given those parameters, the model is capable of identifying significant attacks, including several that had not been identified in the original dataset. The *metrics* that are used here to quantify threat score are *generalizeable* to many datasets (although not universal to each simultaneously), *resistant* to some manipulation by attackers (although possibly not fully robust), and *explainable* in plain language. The *algorithms* for identification and ranking of alert paths are also *explainable* and can be *automated*.

CHAPTER 3: AUTOCRAT: AUTOMATIC CUMULATIVE RECONSTRUCTION OF ALERT TREES

When a network is attacked, cyber defenders need to precisely identify which systems (i.e., computers or devices) were compromised and what damage may have been inflicted. This process is sometimes referred to as *cyber triage* and is an important part of the incident response procedure. Cyber triage is challenging because the impacts of a network breach can be far-reaching with unpredictable consequences. This highlights the importance of *automating* this process. This chapter introduces AutoCRAT, a system for quantifying the breadth and severity of threats posed by a network exposure, and which can be used to prioritize cyber triage activities during incident responses. Specifically, AutoCRAT conducts AUTOmatic Cumulative Reconstruction of Alert Trees, which track network security events emanating from, or leading to, a particular computer on the network. A case study validates the usefulness of AutoCRAT by using a dataset from a research testbed. Experimental results show that the prototype system can reconstruct alert trees efficiently and can facilitate data visualization in both incident response and threat intelligence analysis.

3.1 Introduction

In cyber incident response, the defender needs to precisely identify what happened to the network in question, including: how did the attacker propagate through the network, what was the attacker's intent, and where and how much damage did the attacker inflict? Since attackers may target a large portion of a network, the defender must quickly and effectively determine the range of their impact. This includes the possible routes that the attacker used to enter and propagate through the network, referred to as *alert paths* and which may be aggregated into *alert trees*.

This turns out to be a difficult task for two reasons. First, for any amount of incoming alerts, the number of alert paths that need to be examined grows quadratically. This is the problem of *efficiency*. Second, without examining all possible alert paths, it is possible that the defender will overlook the actual attack path. This is the problem of *coverage*, which is closely related to false

negatives in intrusion detection systems. These problems are highly relevant, given the frequency of cyber attacks throughout the internet. This problem also remains largely open as it often takes days or even weeks for the defender to properly diagnose a cyber incident.

In order to help defenders effectively and efficiently respond to cyber incidents, the research community needs to investigate principled solutions to tackling this problem. This motivates the present study, which aims to automatically facilitate cyber triage.

3.1.1 Chapter Contributions

This chapter offers three main contributions, as follows. First, it more rigorously defines the concept of an *alert tree*, which is necessary to make certain optimizations to the reconstruction process. Key differences from existing concepts include the incorporation of name labels for computers and ID labels for alerts, and the storage of alerts as annotations to endpoints rather than as individual arcs.

Second, it proposes AutoCRAT, a method for reconstructing alert trees from the output of a Network Intrusion Detection System (NIDS). The novel features of the method are characterized as follows: (i) *Continuous alert ingress*, meaning that it can continuously process streams of alerts reported by security devices. This is important for real-world employment where security devices constantly produce alerts, and contrasts existing works which look at all the data at once. (ii) *Efficiency*, meaning that it can quickly reconstruct alert paths and trees on demand. Our asymptotic analysis shows that in the worst-case scenario, graph maintenance scales cubically with the number of alerts, while tree reconstruction scales quadratically or log-linearly, depending on the type of tree. (iii) *Complete graph coverage*, meaning that network analysis is able to include alert paths across the entirety of the network, which was not possible with existing approaches because of runtime limitations. This is validated in the case study, as follows.

Third, a case study demonstrates the usefulness of the notion of alert tree and the alert tree reconstruction method using a dataset which is collected from a realistic cyber attack testbed, namely CSE-CIC-IDS2018, as published by the University of New Brunswick's Canadian Institute

for Cybersecurity. Experimental results show that AutoCRAT was able to analyze the data within the timeframe which the data was collected, incorporating a larger portion of the data than previous works.

3.1.2 Related Work

The existing literature can be broken into two categories: cyber attack modeling and attack reconstruction.

Attack modeling. The problem of attack modeling has been approached using alert correlation [44, 45] and clustering [46]. The present study moves a step further by aiming to make sense of a set of alerts through the notion of alert paths, which are more comprehensive than alert correlation because they explicitly model spatiotemporal relationships. This means they may be useful in mapping the attack to a particular model (e.g., [18, 47, 48]), many of which have explicit happens-before dependencies. Similarly, attack paths have been studied for their usefulness in predicting attack outcomes [49–55]. Attack graphs have also been used to model the propagation of attacks through a computer network [31–34,56,57]. These studies are focused on static network evaluation or attack prediction, rather than reconstruction of actual attack patterns.

Reconstruction of multi-step attacks. One work that reconstructs multi-step attacks is MAAC [58]. This model assumes a four-stage attack model, in which steps of an attack operation can only be assembled if stage 3 alerts are found. This means that false negatives in alert production are more likely to induce false negatives in the model. Although MAAC assembles an alert graph, it only identifies paths of length one with respect to the network. Specifically, there may be many stages of attacks within a single host, but only one stage may cross the network to another host. Thus the attack paths identified by MAAC are better described as host-paths than network-paths. The same observation applies to [59].

Another reconstruction method is MIF [60]. MIF uses a supervised machine learning algorithm to reduce graph size, then produces a risk-state graph to track network attacks. Edges are ordered by start times only, which may induce false positives when two paths overlap. During

reconstruction, the model uses a recursive depth-first-traversal to build paths. MIF achieves high classification accuracy for attacks, but no complexity analysis is given. On LLDoS 2000 (containing 60 hosts), MIF processes one million (upsampled) flows in 3m24s. It processed CICIDS2017 for accuracy but did not give runtimes.

Another reconstruction method is APIN [61]. APIN builds an alert graph using raw alerts and extracts alert paths with respect to a particular node using a chronological traversal. It includes complexity analysis and runtimes for DARPA 99 and CSE-CIC-IDS2018.

Alert prioritization. The concept of alert trees benefits from relevant studies of alert prioritization [62–64]. This process ranks alerts according to their severity or associated risk. These rankings are useful to the concept of alert trees because they enable more intuitive tree interpretation. For example, visualizing the colors of nodes in a tree based on the ranking (i.e., prioritization) can help defenders identify hotspots in the network. Alert prioritization does not impact the algorithmic reconstruction of alerts, just their presentation (at least in this and known works).

Intrusion detection. Alert trees depend on input from intrusion detection systems (IDSs), either network- or host-based (NIDS and HIDS, respectively). IDSs have been studied extensively, and have achieved high levels of accuracy in lab settings (e.g., [60, 65]). However, such works have been criticised heavily on account of the base-rate fallacy, poorly-representative environments, limited attack scope, and weak ground truth [66–69]. IDSs are ineffective in practice because of alert volumes, false positives and alert interpretability [20, 70, 71].

3.1.3 Chapter Outline

The rest of the chapter is organized as follows. Section 3.2 discusses the research problem and formalizes the concepts that are used in the rest of the chapter. Section 3.3 presents the AutoCRAT system, including its architecture and core functions for solving the research problem. Section 3.4 presents the results of applying AutoCRAT to a real-world dataset. Section 3.5 discusses the advantages and disadvantages of AutoCRAT and the present study. Section 3.6 concludes the chapter.

3.2 Problem Statement

This section formally defines the problem addressed in this chapter. The terms used are given in Table 3.1.

Table 3.1: Terms used throughout the chapter on AutoCRAT.

Term	Meaning / Usage
Defender	System operator or network administrator
Security Device	NIDS, HIDS, or other network monitor which produces alerts as the input to the system
Alert	Event indicating the presence of an attack
Arc	An alert (graph) or pair of computers (alert tree). Denoted $\alpha \in \mathcal{A}$, with $A \subset \mathcal{A}$.
Endpoint pair	The ordered pair $(source, destination)$ used to refer to a group of arcs. Endpoint set denoted E.
Alert Path	Sequence of nodes and accompanying arc sets with partial happens-before ordering. Alert path set denoted $P(\mathcal{A})$ or P.
Origin	First node in a path
Target	Last node in a path
Alert Tree	Model of alert paths with common origin or target. Denoted $T(\mathcal{A}, v)$
AutoCRAT	The proposed model for tracking network events.

The rest of the section is organized as follows. First, the problem setting and informal problem statement are given. Then, concepts are introduced in order to formalize the problem; most notably *alert graph*, *alert path*, and *alert tree*. Finally, the formal research questions are defined.

3.2.1 Informal Problem Statement

Consider an enterprise network, which consists of computer systems and security devices (e.g., Network Intrusion Detection Systems or NIDSs), and is managed by one or more network administrators, hereafter referred to as the *defender*. Note that the concept of an enterprise network is generally applicable to many types of computer networks, including IoT networks, mobile networks, etc. Computers on the enterprise network may be the target of cyber attacks from some *attacker*, which may come from inside or outside enterprise network. Network traffic is monitored by security devices, which produce alerts when they observe an attack. A successful attacker may

conduct secondary attacks (known as *lateral movements*) against other computers within the network by leveraging previously compromised computers. The term *multi-step attack* refers to such a sequence of attacks. These attacks pose threats to the network and must be identified, understood, and mitigated.

Given the setting discussed above, this chapter aims to develop an understanding of multi-step attacks as they pertain to a particular network. This naturally leads us to several intuitive questions, which will form the basis of the Research Questions (RQs). The present focus is on NIDS-like security devices, leaving potential extensions to future work. These intuitive question are:

1. What routes could an attacker have taken to get from one computer to another?
2. What is the scope of a given attack operation, as represented by an alert or group of alerts?
3. For some attack, how may the attacker have used lateral movements to traverse the network and finally set up the attack?
4. What is the most efficient way to achieve each of the above goals?

These questions are intuitive but not precise enough for treatment. In order to answer them, they must be clarified by formally defining the relevant concepts.

3.2.2 Concept Formalization

In order to formalize the IQs mentioned above, alerts and their relationships to network objects must be modeled appropriately. This begins with the input of a stream of alerts generated by security devices, denoted $\mathcal{A}$. In the context of network defense, this is composed of alerts:

Definition 3 (Alert)**.** *An alert α is generated by a security device corresponding to a communication between a $source$ computer and a $destination$ computer and can be described as a tuple $\alpha = (source, destination, time, ID)$, where $source$ and $destination$ are the* endpoints *of the alert, $time$ represents the timestamp at which the communication begins and ID is the alert identifier given by the security device (e.g., signature identifier or remote logon type).*

Alerts compose more complex objects, namely *alert graphs*, *alert paths* and *alert trees*.

Alert Graph. Given a set of alerts, an *alert graph* can structure them in a meaningful way. This alert graph is modeled as a labeled multidigraph, which allows vertices and arcs to have common labels. This is important because alert (i.e., arc) IDs belong to a pre-defined set, which have specific cybersecurity meanings, and because alert tree construction will expand loops into branches (later in this section), which formally requires that distinct nodes have common identifiers. Specifically, an alert graph is defined as:

Definition 4 (Alert Graph)**.** *Given a set of alerts $\mathcal{A}$, an alert graph is a labeled multidigraph $G(\mathcal{A}) = (\Sigma_V, \Sigma_A, V, A, s, t, \ell_V, \ell_{ID}, \ell_{time})$, where vertex $v \in V$ represents a computer on the network, arc $a \in A$ represents an alert produced by a security device, Σ_V denotes the set of computer labels (such as IP addresses), Σ_A denotes the set of alert labels (i.e., α's ID), $s : A \to V$ maps arcs to their source vertex (i.e., $\alpha.source$), $t : A \to V$ maps arcs to their target vertex (i.e., $\alpha.destination$), $\ell_V : V \to \Sigma_V$ maps vertices to their labels, $\ell_{ID} : A \to \Sigma_A$ maps arcs to their alert labels, and $\ell_{time} : A \to \mathbb{N}$ maps alerts to the set of natural numbers, representing the time at which they occurred.*

Alert graphs can be used to identify alert paths and trees.

Alert Path. Given an alert graph $G(\mathcal{A})$, the concept of *alert path* describes a sequence of vertices through which an attack is observed (as indicated by alerts). However, it is important to realize that the standard definition of *path* in graph theory does not sufficiently represent a multi-step attack. This is because the alerts representing a multi-step attack may consist of multiple repeated arcs between a pair of vertices (for example, when an attack must make multiple connections before it succeeds, thus resulting in multiple arcs). Thus, the requirement for arcs belonging to an alert path are as follows.

A path is associated with a set of arcs rather than a sequence. The arc set is composed of all arcs traversing between adjacent nodes in the alert path. This arc set is subdivided into subsets which form a sequence, where subsets are grouped according to the ordered endpoints along path.

Within the arc set, there must be a valid sequence of arcs, one each from consecutive arc subset, such that they appear in chronological order.

This divergence from the standard usage of *path* leads us to define alert paths based on their vertices rather than their arcs, which instead *result from* the defined sequence of vertices. This leads to the following definition:

Definition 5 (Alert Path)**.** *Given an alert graph $G(\mathcal{A})$, an* alert *path is denoted $p = (V^p, A^p)$, where $V^p = (v_1^p, \ldots, v_n^p)$ is a sequence of unique vertices in V; A^p is a corresponding set of arcs in A, for which there must exist a sequence of arc sets $(A_1^p, \ldots, A_{n-1}^p)$ such that $A_i^p = \{\alpha \in A : s(\alpha) = v_i^p \wedge t(\alpha) = v_{i+1}^p\}$, and $A^p = \bigcup_{i=1}^{n-1} A_i^p$, and there must exist some sequence $(\alpha_1, \ldots, \alpha_{n-1})$ such that for each $i \in [1, \ldots, n-1]$, $\alpha_i \in A_i^p \wedge i < n-1 \rightarrow \ell_{time}(\alpha_i) < \ell_{time}(\alpha_{i+1})$.*

Given an alert path p with $V_p = (v_1^p, \ldots, v_n^p)$, define v_1^p as the *origin* and v_n^p as the *target*. Given $\mathcal{A}$, let $P(\mathcal{A})$ denote the set of all alert paths in $G(\mathcal{A})$. A path $p \in P(\mathcal{A})$ may be uniquely identified by V^p, allowing A^p to be inferred from context.

Alert Tree. An alert tree represents a set of alert paths $P(A)$ (where $A \subset \mathcal{A}$) with a common *reference vertex*. With respect to these paths, the reference vertex is either the origin or the target, but not both. Trees with a common origin are called *forward trees* and trees with a common target are called *backward trees*, because of the partial happens-before relationship of alerts in the tree. Because the graph $G(\mathcal{A})$ is a labeled multidigraph, it is possible that cyclical subgraphs will result in sets of paths $P(A)$ such that they cannot naturally be combined to form trees under graph theory. For example, the graph constructed from the set of paths $P = \{(a, b, c), (a, c, b)\}$ contains a cycle (b, c, b) and is therefore not a tree. However, if duplicate nodes are replaced by unique nodes with the same identifier (in Σ_V), this loop formation can be artificially prevented. Consider the previous example. The second path might become (a, c', b') such that $\ell_V(c') = \ell_V(c)$ and $\ell_V(b') = \ell_V(b)$. This results in a tree with the arcs $(a, b), (b, c), (a, c'), (c', b')$, such that the modified graph forms a tree, which can be used to extract the relevant computer identifiers. The advantage of the tree structure over an arbitrary graph is that one can visualize the *temporal dependence* of the arcs based on their height within the tree. Since temporal dependence is an important feature of alert paths,

it is thus natural to visualize sets of alert paths using alert trees rather than alert graphs. Formally, alert trees are:

Definition 6 (Alert Tree)**.** *Given an alert graph $G(\mathcal{A})$, a reference vertex $\hat{v} \in V$, and a set of paths $P' = \{p \in P(\mathcal{A}) : v_1^p = \hat{v}\}$, an alert tree is a labeled digraph denoted $T(\mathcal{A}, \hat{v}) = \{\Sigma_{V,T}, V_T, A_T, \ell_{V,T}\}$, where $\Sigma_{V,T} = V$, $V_T = V \times P'$,$a \in A_T = (v, v') \in V_T$ such that $\exists p \in P', i \in [1, \ldots, |V^p|]$ for which $v = v_i^p \wedge v' = v_{i+1}^p$, and $\ell_{V,T} : V_T \rightarrow \Sigma_{V,T}$ maps vertices to their corresponding nodes in V.*

A forward alert tree is rooted at reference vertex $\hat{v}$ and represents paths $p \in P'$, such that every vertex in p has a corresponding vertex $v \in V_T$ for which all of the ascendants of v (denoted $asc(v)$) are labeled with the vertices in the path. Specifically, $\forall i \in [1, \ldots, |asc(v)|], \ell_{V,T}(asc(v)_i) = v_i^p$.

A backward alert tree T_{bwd} is a reversed forward tree in the following sense: reference vertex $\hat{v}$ is the target rather than the origin of all of the corresponding alert paths, while the vertex identifiers must match vertex descendants rather than ascendants.

3.2.3 Research Questions

Given the preceding formalisms, the intuitive questions can be transformed into RQs as follows.

1. What routes could an attacker have taken to get from one computer to another? This question asks for the set of all alert paths with a specified origin and target. This leads to:

   RQ1: *Given a set of alerts $\mathcal{A}$, a known attack origin v_{origin}, and a known attack target v_{target}, produce the set of attack paths $P' = \{p \in P(\mathcal{A}) : v_1^p = v_{origin} \wedge v_{|V^p|}^p = v_{target}\}$.*

2. What is the scope of a given attack operation, as represented by an alert or group of alerts? This question asks for the depth and breadth of access that an attacker achieved corresponding to an alert (or attack). This naturally suggests that one must design an algorithm to build a forward alert tree that is rooted at the given alert. This leads to:

   RQ2: *Given a set of alerts $\mathcal{A}$ and a known attack origin v_{origin}, produce the forward alert tree $T_{fwd}(\mathcal{A}, v_{origin})$.*

3. For some attack, how may the attacker have used lateral movements to traverse the network and finally set up the attack? This question asks for the set of multi-stage attacks that may have led to the compromise of a computer. This naturally suggests that one must design an algorithm to identify a backward alert tree that is rooted at the given alert.

   RQ3: *Given a set of alerts $\mathcal{A}$ and a known attack target v_{target}, produce the backward alert tree $T_{bwd}(\mathcal{A}, v_{target})$.*

4. What is the most efficient way to achieve each of the above goals? Of course, this is a significant optimization problem. However, the asymptotic complexity and practical runtime efficiency of the proposed algorithms can give insight when compared to those of existing models. This leads to:

   RQ4: *Analyze the asymptotic runtime and storage complexities of the proposed methods with respect to their input alerts $\mathcal{A}$.*

3.3 The AutoCRAT System

In order to address the research questions formalized in Section 3.2.3, we propose the AutoCRAT system, with its architecture highlighted in Figure 3.1. It has four components: the *database*, the *core functions*, and the *data management* and *data retrieval* interfaces.

3.3.1 Database

The database stores the edges, alerts, and paths derived from G as follows. The vertices $v \in V$ do not require indexing, so they are not stored explicitly in the database.

Arcs and Alerts: The arc collection of the database stores pairs of arc endpoints. Arcs are stored as annotations to the arc endpoints. This enables more efficient retrieval of alerts, which are always considered in the context of the other alerts with the same endpoints. This is demonstrated in Section 3.2.2.

Alert Paths: Alert paths are stored in their own collection. Each path $p \in P(\mathcal{A})$ contains a list of

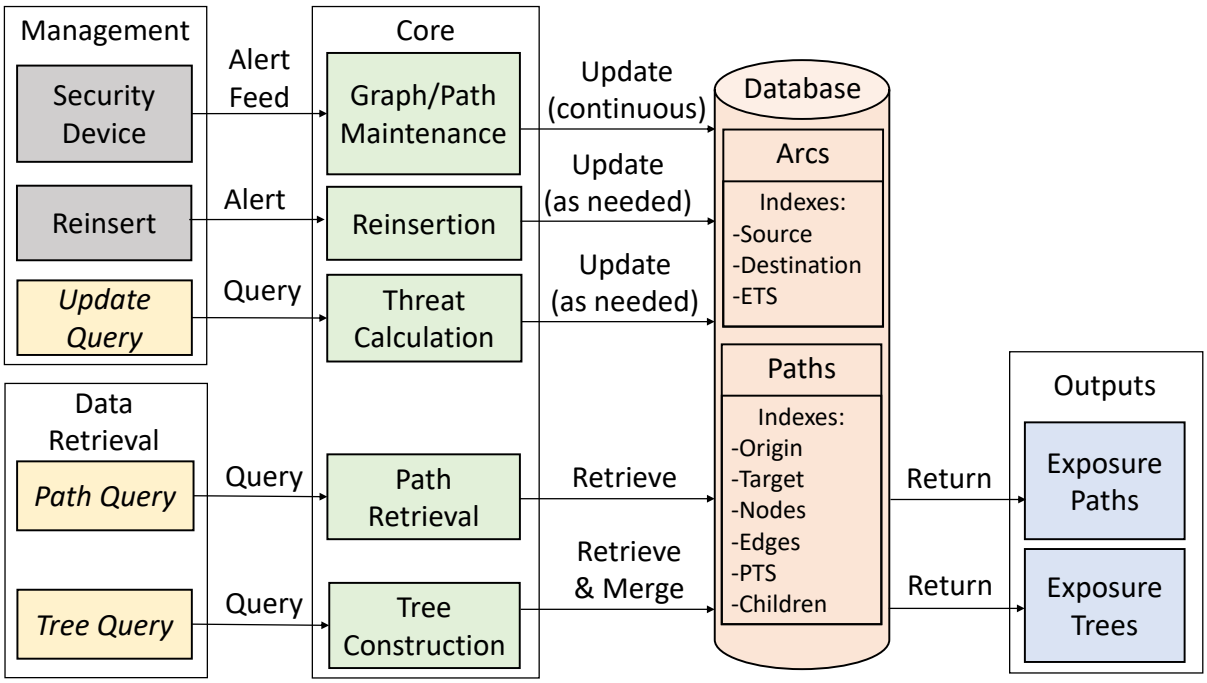

Figure 3.1: The architecture of the AutoCRAT system

nodes, the corresponding node pairs (to facilitate retrieval via pairwise indexing), the path's PTS, and a list of child vertices (denoted $Children^p$) that have been used to produce lengthened copies of the path, $\{v \in V : (\exists p' \in P(\mathcal{A}), \forall v_i^p \in V^p) v_i^p = v_i^{p'} \wedge |V^{p'}| = |V^p| + 1 \wedge v = v_{|V^{p'}|}^{p'}\}$. This list is used to facilitate path validation, as discussed later in this section. Every time a path is lengthened, the new path is added as its own object in the database, in order to facilitate indexing.

Indexing: Each collection is indexed so that its objects may be efficiently found and retrieved from the database. Our approach uses multiple types of indices, including individual indexes (on a single field), compound indexes (on two or more fields) and multikey indexes (on a set or sequence). Some of these index types are only available in certain kinds of databases, such as MongoDB, which is used in our implementation. This limits the interoperability of our approach, or risks compromising the efficiency of accessing some objects in the database.

The arc collection is indexed by arc endpoints (each individually as well as together in a compound index), and by the arc's threat score (discussed later in this section). The path collection has individual indices on the origin, the target, and the path's threat score (also discussed later). It also

has multikey indexes on the vertex sequence and the arc endpoint sequence. Additionally, the path collection has one compound index, covering the target and the list of child vertices (as a multikey index).

Duplicate paths are prevented by validating new arcs against this list of children in the path's so-called parent (i.e., $p' \in P$ such that $V^{p'} = V^p - (v^p_{|V^p|}) \wedge v^p_{|V^p|} \in Children^{p'}$. Subpaths are stored as their own objects because this allows them to be more efficiently indexed. Specifically, we index the path's origin and target, because multikey indexes do not preserve order. This means that, if a multikey index was used on the path nodes, then an attempt to retrieve paths with a certain origin and target would need to parse all paths containing both the origin and target in any position, then trim the path appropriately. In this case, the runtime optimization of multikey indexing of path vertices trades off with storage complexity (i.e., storing more paths).

On the other hand, it is efficient to use a multikey index to index to children of a particular path, because queries that search for children need only find a single child (i.e., the $destination$ of the alert being inserted). This means that the ordering of children in a multikey index is unimportant.

3.3.2 Core Functions

Alert Graph and Path Maintenance

As new security events arrive, AutoCRAT needs to incorporate them into the relevant databases. Algorithm 2 presents the pseudocode of this functionality as follows: (i) It first checks if the new arc includes existing endpoints. If so, it adds the arc to the endpoint object as an annotation; otherwise, it inserts a new endpoint object annotated with the arc (lines 1-13). (ii) It queries the database to find paths which end at the source of the event (line 14). (iii) The algorithm copies the paths found, appending the alert's $destination$ onto the copies. If the new paths already exist in the database or are cyclical, they are discarded. The original paths are annotated with a list of children to facilitate this check on future inserts, and the new paths are inserted (lines 15-27).

Storage complexity. In the worst-case scenario, every insertion of a new alert α will add a new endpoint object and path, and lengthen the set of existing paths that terminate at the source of

Algorithm 2 Alert Graph and Path Maintenance

Input: $\alpha, G(\mathcal{A}), P, Children^{p \in P}$
Output: $G(\mathcal{A} \cup \{\alpha\}), P, Children^{p \in P}$

if $\nexists \alpha' \in A : s(\alpha) = s(\alpha') \wedge t(\alpha) = t(\alpha')$ **then**
  $V^{\hat{p}} \leftarrow (s(\alpha), t(\alpha))$ ▷ Create $\hat{p}$ via $V^{\hat{p}}$
  $Children^{\hat{p}} \leftarrow \emptyset$
  $P \leftarrow P \cup \{\hat{p}\}$
$A \leftarrow A \cup \{\alpha\}$
$V \leftarrow V \cup \{s(\alpha), t(\alpha)\}$
$Candidate\_Paths \leftarrow \{p \in P : V^p_{|V^p|} = s(\alpha)\}$ ▷ Find paths to lengthen using "end" index
for $c \in Candidate\_Paths$ **do**
  if $t(\alpha) \notin V^c$ **then** ▷ Prohibit cycles
    if $t(\alpha) \notin Children^c$ **then** ▷ Prohibit identical twins
      $p' \leftarrow \{V^c, A^c\}$
      $V^{p'} \leftarrow V^{p'} \cup (t(\alpha))$
      $A^{p'} \leftarrow A^{p'} \cup \{\alpha\}$
      $Children^{p'} \leftarrow \emptyset$
      $Children^c \leftarrow Children^c \cup \{t(\alpha)\}$
      $P \leftarrow P \cup \{p\prime\}$
return $G = (V, A), P, Children^{p \in P}$

the inserted arc. This will give us a storage complexity in $\mathcal{O}(|E| + |A| + |P|)$. Now suppose one inserts the first arc, with endpoints (v_1, v_2), into a database. This will produce a single path: (v_1, v_2). A second insertion of (v_2, v_3) adds a new path and lengthens the existing path, resulting in three paths: (v_1, v_2), (v_2, v_3), and (v_1, v_2, v_3). Clearly, each arc adds at most one new endpoint object (exactly one in this case), leading to $|E| \in \mathcal{O}(|\mathcal{A}|)$. This means that each new arc adds up to $|E|$ new paths of length $|E|$ or less, leading to $|P| \in \mathcal{O}(|\mathcal{A}|^2)$. This is demonstrated in Table 3.2.

Table 3.2: Worst-case storage complexity after i arc insertions. As each new arc α_i can add as many as i paths, the worst-case number of paths is $|P| = \frac{1}{2}|\mathcal{A}|^2 + \frac{1}{2}|\mathcal{A}| \in \mathcal{O}(|\mathcal{A}|^2)$.

Arcs	1	2	3	4	$\lvert\mathcal{A}\rvert$
Endpoint pairs	1	2	3	4	$\lvert\mathcal{A}\rvert$
Paths	1	3	6	10	$\sum_{i=1}^{\lvert\mathcal{A}\rvert}(i)$

Runtime complexity. Under the same worst-case scenario, the insert algorithm must access every path in the database, $|P|$, copy each, adding one endpoint, and insert the copies, plus one new path with two nodes. Then the worst-case runtime (in terms of the number of accesses to the database)

is $2 \cdot |P| + 1 \in \mathcal{O}(|P|) \in \mathcal{O}(|\mathcal{A}|^2)$.

Threat Score

The threat posed by a given set of alerts $A \subset \mathcal{A}$ can be measured according to its threat score (TS). There can be many definitions for TS, and identifying the "best" definition is orthogonal to the focus of the present chapter. Since any "good" definitions can be incorporated into AutoCRAT in a plug-and-play fashion, one example definition is sufficient to demonstrate the idea. This example of TS is the geometric mean of the number and diversity of alerts in a set, and is used to estimate the risk posed by the associated attacks. It can be naturally extended to specify the TS of a pair of endpoints (Endpoint Threat Score — ETS) or a path (Path Threat Score — PTS). Specifically:

Definition 7 (Threat Score (TS))**.** *The TS of a set of alerts is defined as:*

$$TS(A) = \sqrt{|\{\sigma \in \Sigma_A : (\exists \alpha \in A), \sigma = \ell_{ID}(\alpha)\}| \cdot |\{\alpha \in A\}|}. \tag{3.1}$$

The ETS of an endpoint pair $(source, destination)$ *can be calculated as* $TS(\{\alpha \in \mathcal{A} : s(\alpha) = source \wedge t(\alpha) = destination\})$. *The PTS of a path* p *can be calculated as* $TS(A_p)$.

In order to identify the endpoints and paths in the database with the highest threat, AutoCRAT must periodically update ETS and PTS for endpoints and paths, respectively. In order to do this, the defender submits an update query on demand. The update function is simple: first, it calculates ETS for every endpoint object, then it calculates PTS for every path. To reduce the number of accesses to the database, a copy of each endpoint object's arc annotations is cached during the ETS calculations, in order to facilitate the PTS calculations.

Runtime complexity analysis. The runtime of this approach is based on the following: (i) calculating ETS for all endpoints requires the parsing of each arc (which each belong to exactly one endpoint annotation), meaning the runtime of updating is in $\Omega(|\mathcal{A}|)$; (ii) calculating PTS for all paths requires the parsing of each endpoint object a number of times equal to the number of paths in which it appears. This is analyzed in Table 3.3, which shows that the worst case number of

appearances of an endpoint object in the set of paths is in $\mathcal{O}(|E^2|)$. Since there are $|E|$ endpoint objects, this results in a worst-case combined runtime in $\mathcal{O}(|A| + |E|^3)$.

Table 3.3: Worst-case number of paths that contain a particular endpoint. The i^{th} endpoint can only belong to $i \cdot (|E| - i + 1) = i \cdot |E| - i^2 + i$ paths. Then the worst-case is in $\mathcal{O}(|E|^2)$, and the worst-case number of endpoint objects across the set of all paths is $\sum_{i=1}^{|E|} i \cdot (|E| - i + 1) = \frac{1}{6}|E|(|E| + 1)(|E| + 2) \in \mathcal{O}(|E|^3)$.

Endpoint i	Insertion 1	2	3	4	5	$\|E\|$
1	1	2	3	4	5	$\|E\|$
2	0	2	4	6	8	$2 \cdot (\|E\| - 1)$
3	0	0	3	6	9	$3 \cdot (\|E\| - 2)$
4	0	0	0	4	8	$4 \cdot (\|E\| - 3)$
5	0	0	0	0	5	$5 \cdot (\|E\| - 4)$
$\|E\|$	0	0	0	0	0	$\|E\|$
i	0	0	0	0	0	$i \cdot (\|E\| - i + 1)$

ETS is also used in the visualization of attack paths. Specifically, the color of an alert tree node n represents the normalized ETS of $(parent(n), n)$ for forward trees or $(n, parent(n))$ for backward trees. Vertex colors range from red to black, where the vertex in the tree with the highest ETS is red and the root is black (along with any other vertices with $ETS = 1$). In RGB (i.e., hexadecimal) notation, colors range from 0x000000 (black) to 0xFF0000 (red), such that colors may be compared ordinally to mimic the comparison of ETS. This is demonstrated in figure 3.2, showing the value of the ETS measurement in presenting salient information to the defender.

Reinsertion

In some cases, the defender may need to find paths that were unavailable at the time of data collection (e.g., if the NIDS produces a false negative which is later corrected). Because alerts are otherwise inserted chronologically (by assumption), a different algorithm must perform such a retroactive insertion ("reinsert"). This algorithms is given in Algorithm 3.

Runtime Complexity. The asymptotic runtime is dominated by $|p^{pre}| \cdot |p^{post}|$. Because cycles about α are removed (lines 2,4), $P^{pre} \cap P^{post} = \emptyset$. This is bounded by $(\delta \cdot |P|) \cdot ((1 - \delta) \cdot |P|)$, where $\delta < 1$ is the proportion of the larger of the two path sets relative to $|P|$. This simplifies to

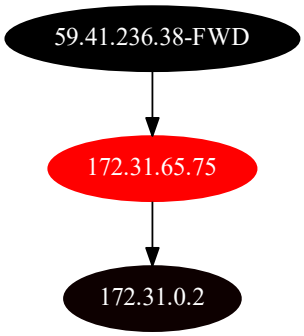

Figure 3.2: Example alert tree coloring, where the root is black, the child is red (ETS 179.10), and the grandchild is very nearly black (color code 0x0D0000 and ETS 10.49).

Algorithm 3 Alert Reinsertion

Input: $\alpha, G(\mathcal{A}), P$
Output: $G(\mathcal{A} \cup \{\alpha\}), P$

$P^{pre} \leftarrow \{p \in P : V^p_{|V^p|} = s(\alpha)\}$
$P^{pre} \leftarrow P^{pre} \setminus \{p \in P^{pre} : t(\alpha) \in V^p\}$
$P^{post} \leftarrow \{p \in P : V^p_1 = t(\alpha)\}$
$P^{post} \leftarrow P^{post} \setminus \{p \in P^{post} : s(\alpha) \in V^p\}$
$V \leftarrow V \cup \{s(\alpha), t(\alpha)\}$
$A \leftarrow A \cup \{\alpha\}$
$V^{\hat{p}} \leftarrow (s(\alpha), t(\alpha))$ ▷ Create $\hat{p}$ via $V^{\hat{p}}$
$P \leftarrow P \cup \{\hat{p}\}$
for $p \in P^{pre}$ **do**
    for $p\prime \in P^{post}$ **do**
        if $V^p \cap V^{p\prime} = \emptyset$ **then** ▷ Prohibit Cycles
            $V^{New\_Path} \leftarrow V^p \cup V^{p\prime}$ ▷ Create New_Path via V^{New_Path}
            if New_Path is valid given A **then**
                $P \leftarrow P \cup \{New\_Path\}$
return G, P

$(\delta - \delta^2) \cdot |P|^2$, for which the maximum value of the first term (when $\delta = \frac{1}{2}$) is $\frac{1}{4}$. Finally, the worst-case runtime is $\frac{1}{4}|P|^2 \in \mathcal{O}(|P|^2) \subseteq \mathcal{O}(|\mathcal{A}|^4)$.

Storage Complexity. The storage complexity is the same as for the regular maintenance function, $\mathcal{O}(|\mathcal{A}|^2)$.

3.3.3 Data Management

This interface is responsible for feeding a stream of alerts into the AutoCRAT system. The streams of alerts are produced by, and received from, security devices such as NIDSs. They are processed by the Graph and Path Maintenance module, which belongs to the Core Functions component and incorporates the newly arrived alerts into the alert graph and paths.

3.3.4 Data Retrieval

In order to identify paths and trees, one makes the appropriate query. Paths are stored in the database themselves, while trees must be reconstructed using the stored paths.

Path Retrieval

Path retrieval requires that the graph and path databases have been properly maintained. Refer to section 3.3.2 for the detailed maintenance function. The path retrieval process then involves a simple query to the database, which stores each path individually. This query leverages the $(source, destination)$ compound index, which efficiently retrieves the appropriate paths from the database. The retrieval function executes with a runtime efficiency of $\mathcal{O}(|P|)$, where $P \subseteq P(\mathcal{A})$ is the set of paths to be retrieved.

Tree Reconstruction

Tree reconstruction requires reassembly of many paths into a tree which represents either the attack surface exposed to an attacker or the attack vectors exposing a target, depending on the type of query. In either case, the user must specify a node to act as the root of the tree and the tree's

direction, and the module handles the rest.

The tree reconstruction algorithm takes as input a reference node to act as the tree's root, a direction (i.e., forward or backward), G, and P. It begins by selecting all paths which have the reference node as their origin (for forward trees) or target (for backward trees), using the approach described above. It then parses each path, adding each edge as a node of the tree if it was not already added from a previous path.

Runtime complexity. The worst-case scenario for the tree reconstruction algorithm is the same as the insert algorithm as discussed above, in which each subsequent alert produces a new endpoint which extends that of the previous alert. Specifically, at most $|E|$ paths are rooted at a given node, and their maximum path length is $|E|$. This results in a final worst-case runtime of $\mathcal{O}(|E|^2) \subseteq \mathcal{O}(|\mathcal{A}|^2)$.

3.4 Case Study

Our experiments were run using Ubuntu 20.04 with 192GB of RAM, 2 cores of an Intel Xeon Gold 6242 CPU @2.80 GHz, and a 200GB HDD. These resources were shared among AutoCRAT functions and the corresponding MongoDB database, which was installed on the same computer to eliminate variability imposed by network conditions.

3.4.1 Dataset

The dataset chosen for the case study was published by the University of New Brunswick, and is referred to as CSE-CIC-IDS2018 [41]. It contains data collected over the course of 9 days, during which multiple distinct attack scenarios were executed against the network. The environment was connected to the internet during the experiments, thus real-world attacks can also be observed in the data. Suricata 4.0 [42] preprocessed the packet capture (PCAP) files from the dataset using the corresponding Emerging Threats signature set [43], to produce a set of 3,323,426 alerts, of which 19,921 were strictly internal to the network. Alerts were converted into JSON objects to conform to AutoCRAT's expected format and sorted chronologically before being fed into the database.

3.4.2 Experimental Results

Database construction for CSE-CIC-IDS2018 took 35h14m07s, resulting in 1,053,710 edges and 3,591,217 paths. Efficiency and accuracy (in terms of graph coverage, which may impose false negatives) are analyzed relative to the model from the preceding chapter, APIN. Path selection for APIN was done using its relevant heuristics. The comparison is shown in Table 3.4.

Table 3.4: Comparison of the proposed AutoCRAT model with APIN. Query runtimes are the average of 10 runs. *APIN only ranks nodes, while AutoCRAT scores endpoints and paths. The models do not directly rank these objects, so they are selected based on applicable node and endpoint rankings. These inherited rankings may not be accurate with respect to other metrics but offer a reasonable baseline.

	APIN	**AutoCRAT**
Build DB	29m43s	13h42m41s
Rank Objects*	49s	1h00m29s
Top 100 paths	52s	32ms
Top 20 trees	52s	2m42s
Coverage (nodes)	99.6%	100%
Coverage (events)	3.4%	100%
DB size	637 MB	1.1 GB

APIN sacrifices coverage in order to improve runtime. This is necessary because its tree retrieval time suffers extraordinary slowdown in the presence of highly connected nodes. Even though APIN only excludes .4% of the vertices in the graph, 96.6% of the arcs in the graph are adjacent to these vertices, and are effectively blacklisted from tree reconstruction, meaning that the set of paths (and trees) that can be reconstructed is incomplete. In AutoCRAT, the connectedness slowdown problem is solved by shifting the bulk of the work into the pre-processing stage. This results in a much faster path retrieval time and full graph coverage at the cost of maintenance time. However, this pre-processing time remains feasible in practice since 9 days of data (constituting approximately 164 hours of activity) are processed in under 14 hours. Graph coverage is important because low coverage induces false negatives in the path and tree reconstruction. This means that APIN is vulnerable to DoS attacks, which may allow an attacker to conceal their attack paths.

In order to compare the two models under comparable conditions, the original dataset was

Table 3.5: Comparison of the proposed AutoCRAT model with APIN, using only the internal events. Query runtimes are the average of 10 runs. *APIN only ranks nodes, while AutoCRAT scores endpoints and paths. The models do not directly rank these objects, so they are selected based on applicable node and edge rankings. These inherited rankings may not be accurate with respect to other metrics but offer a reasonable baseline.

	APIN-internal	**AutoCRAT**-internal
Build DB	9s	35s
Rank Objects*	0.28s	5s
Top 100 paths	3s	23ms
Top 20 trees	3s	1.97s
Coverage (nodes)	0.6%	0.6%
Coverage (events)	0.6%	0.6%
DB size	2.9 MB	2.4 MB

reduced by filtering nodes outside the network as well as edges which crossed the border of the network, and each model was ran again with the reduced version. This resulted in only 1994 edges and 3019 paths. Details are shown in Table 3.5 This resulted in a coverage of 0.6% for both nodes and paths, much closer to the effective coverage of APIN for the full graph (i.e., one sixth of the original coverage). Given the intuition that internal nodes are far more relevant to defenders, the loss of node coverage is not important in this case (although one may note that some of the nodes originally filtered by APIN were in fact internal nodes). This reduction greatly improved the performance of AutoCRAT, resulting in query times and storage efficiency that outperformed the competition.

These experimental observations are consistent with the asymptotic analysis given in Sections 3.3 and 2.2. This is shown in Table 3.6.

Table 3.6: Comparison of the proposed AutoCRAT model with APIN, on the basis of asymptotic complexity. V is vertices, A is alerts, E is endpoints, and P is paths. *APIN only ranks nodes, while AutoCRAT scores endpoints and paths.

	APIN	**AutoCRAT**
Build DB	$\mathcal{O}(1)$	$\mathcal{O}(\|A\|^2)$
Rank Objects*	$\mathcal{O}(\|V\| + \|A\|)$	$\mathcal{O}(\|A\|A + \|E\|^3) \subseteq \mathcal{O}(\|A\|^3)$
Retrieve paths	$\mathcal{O}(\|V\|^3 + \|A\|^2)$	$\mathcal{O}(1)$
Retrieve trees	$\mathcal{O}(\|V\|^3 + \|A\|^2)$	$\mathcal{O}(\|A\|^2)$
Reinsert	$\mathcal{O}(1)$	$\mathcal{O}(\|P\|^2) \subseteq \mathcal{O}(\|A\|^4)$
DB size	$\mathcal{O}(\|V\| + \|A\|)$	$\mathcal{O}(\|E\| + \|P\|) \subseteq \mathcal{O}(\|A\|^2)$

The above discussion demonstrates trade-offs between pre- and post- processing times and between processing time and accuracy. Specifically, Insights 4 and 5 state:

Insight 4. *While attaining the same accuracy as APIN, AutoCRAT front-loads its processing time into building and maintaining paths so that it can retrieve them more quickly.*

Insight 5. *For both models, reducing the volume of data processed improves both processing time and database size, but sacrifices accuracy (as measured by coverage).*

To demonstrate how the alert tree structure and threat score heuristic may be useful in practice, example forward and backward trees are shown in Figures 3.4 and 3.3, respectively. These trees were selected for their size; some trees produced had well over 1000 vertices and would not be legible in the present format. This problem is a challenge left to future work, as the present focus is efficiency of reconstruction.

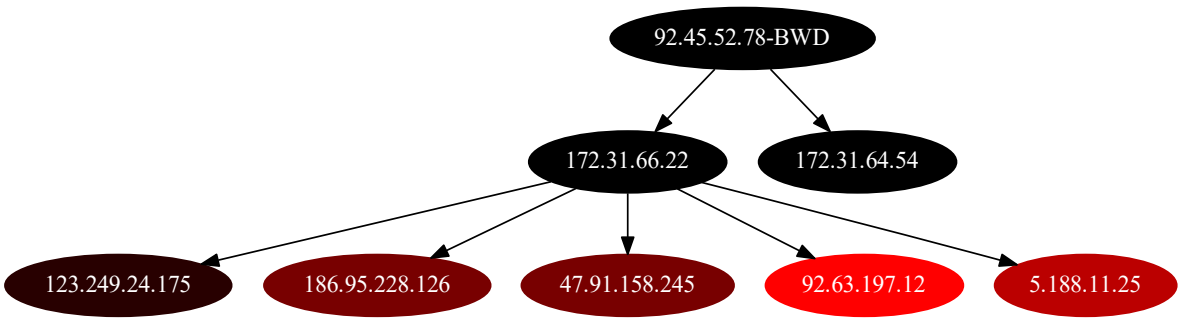

Figure 3.3: A backward tree containing eight vertices. Of the six leaves (i.e., path origins), the reddest vertex, which represents the endpoint $(92.63.197.12, 172.31.66.22)$ scored an ETS of 7.35 with 54 alerts sharing a single ID.

Answering RQ1

In order to answer RQ1, one must retrieve a set of paths corresponding to a known attack origin and a known attack target. Assuming the database maintenance has kept up with the alert stream, this can be accomplished with a query to the database utilizing the $(source, destination)$ index.

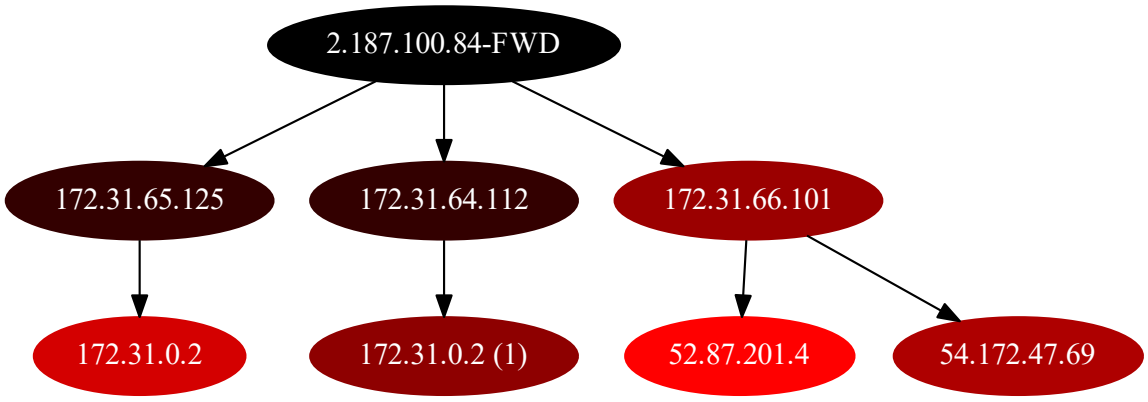

Figure 3.4: A forward tree containing eight vertices. Of the four leaves (i.e., path targets), the reddest vertex, which represents the endpoint $(172.31.66.101, 52.87.201.4)$ scored an ETS of 5.92 with 35 alerts sharing a single ID.

Because subpaths are also stored in the database, it only needs to retrieve paths that start and end with the origin and target, respectively. This efficiently returns a list containing exactly the required corresponding paths, without the need to parse the paths to truncate them at the proper destination. The path retrieval function is described in Section 3.3.4.

Answering RQ2

In order to answer RQ2, one must reconstruct the forward alert tree corresponding to a particular origin. This process begins with retrieving all of the paths beginning at the specified node, leveraging the $source$ index, then proceeds by passing these paths to the tree reconstruction function, which arranges them based on their relationships to each other. This function is described in Section 3.3.4.

Answering RQ3

In order to answer RQ3, one must reconstruct the backward alert tree corresponding to a particular target. Similar to RQ2, this process retrieves the appropriate paths from the database and passes

them to the tree reconstruction function. This function is described in 3.3.4.

Answering RQ4

The asymptotic runtime efficiency of each AutoCRAT function is as follows. alert insertion: $\mathcal{O}(|\mathcal{A}|^2)$, threat calculation: $\mathcal{O}(|\mathcal{A}| + |E|^3) \subseteq \mathcal{O}(|\mathcal{A}|^3)$, alert reinsertion: $\mathcal{O}(|P|^2) \subseteq \mathcal{O}(|\mathcal{A}|^4)$, path retrieval: $\mathcal{O}(|P|) \subseteq \mathcal{O}(|\mathcal{A}|^2)$, tree construction: $\mathcal{O}(|E|^2) \subseteq \mathcal{O}(|\mathcal{A}|^2)$.

Likewise, the storage efficiency of each function. alert insertion: $\mathcal{O}(|\mathcal{A}|^2)$, threat calculation: $\mathcal{O}(|E|) \subseteq \mathcal{O}(|\mathcal{A}|)$, alert reinsertion: $\mathcal{O}(|\mathcal{A}|^2)$, path retrieval: $\mathcal{O}(|P|) \subseteq \mathcal{O}(|\mathcal{A}|^2)$, tree construction: $\mathcal{O}(|E|^2) \subseteq \mathcal{O}(|\mathcal{A}|^2)$.

3.5 Chapter Discussion

Now we discuss the assumptions, limitations, and novel features of AutoCRAT. This discussion presents a roadmap for future studies in improving AutoCRAT.

Assumptions. The present study makes the following assumptions:

1. IDS Correctness. AutoCRAT's correctness will be impacted by mistakes in the IDS, including false positives and false negatives. In extreme cases, these will result in corresponding errors in AutoCRAT's output, but in the case where errors occur along paths with true positives, the penalty will only reduce the accuracy of the TS calculations.

2. Server-Side Attacks. The model assumes that attacks are always initiated by the malicious node, and that the exposure flows in the same direction as the initial connection. This assumption may be violated in client-side attacks, such as drive-by downloads. In this case, an advanced security device may be able to reverse the order of nodes in the alert during preprocessing, preserving attack semantics.

3. Serialization. AutoCRAT's path maintenance algorithm assumes that events are inserted sequentially. This means that it may be difficult to parallelize its execution. Since parallelization is a powerful tool of efficiency, this problem may impact viability of the methods

in practice. However, it may be possible to parallelize some alert insertions if the adjacent nodes are disparate relative to sequential alerts. This investigation is left to future work.

4. TS Semantics. Because TS has not been studied robustly, the quality of trees produced based on its usage is difficult to evaluate. However, as long as the above assumption of IDS correctness remains within reasonable bounds, all paths produced are real paths, but ranking and reference node selection may not result in the "best" tree possible. The problem of ranking alerts remains an open research problem, and the alert tree model can be adapted to many such methods.

Limitations. The present study has the following limitations.

1. Addressing. AutoCRAT assumes that each computer has a single, unique, and static address. It may be extensible to accommodate computers with multiple IP addresses (e.g., by aliasing node names before inserting them into the database or when preprocessing alerts). In the case of a segmented network with private subnets, some computers on disparate subnets may have matching addresses (e.g., 192.168.1.1). This case may be harder to accommodate.

2. Multi-key Indexing. The efficiency of the path maintenance algorithm depends on the assumption that the database is capable of efficiently indexing elements of an array. This restricts the interoperability of the framework to certain kinds of databases (or risks efficiency loss during some operations).

3. Experiment Scope. The experimental environments applied AutoCRAT to NIDS alerts. Although it can be used with any address-based events, this should be evaluated more thoroughly in other use cases (such as process and file system tracking).

Novel Features. The AutoCRAT model incorporates multiple features that have not previously been combined in the study of network defense.

1. The model incorporates multi-step attacks, which include lateral movements secondary to a compromise. This is important because attacker objectives often cannot be accomplished after a single compromise.

2. The model incorporates muti-target attacks, in which a single attacker produces multiple attack paths with different targets. This is important because it does not require the assumption that an attacker has a single target.

3. The model incorporates multi-source attacks, in which a single victim belongs to paths produced by different attackers. This is important because it does not require the assumption that a single attacker is active on the network.

4. The model incorporates alerts as a multigraph, which allows multiple instances of an edge to exist in the graph. This is important because it allows alert trees to show the temporal relationship between its edges without using a dynamic animation.

5. The TS metric provides the property of *monotonicity*, meaning that a set of alerts will always yield a higher TS than its subsets. Specifically, $A \subset A' \rightarrow TS(A) < TS(A')$. This is important because it means that for any of the alert structures used in this study, the TS of that structure will only grow over time and is resistant to manipulation by a crafty attacker.

3.6 Chapter Summary

This chapter introduced the AutoCRAT system for modeling and tracking multi-step network attacks as indicated by alerts generated by security devices. The key concepts behind AutoCRAT are those of alert graphs, alert paths, and alert trees. The technical contributions include data structures and algorithms for efficiently representing and constructing alert paths and alert trees, as well as asymptotic storage and runtime complexity analysis. This study is useful to cyber defenders because it quantifies threats against a network and its components and presents them in an intuitive form that is easy to understand. The case study based on an implementation of AutoCRAT and a research dataset shows that AutoCRAT can efficiently reproduce alert paths and trees, keeping pace with alerts produced on a testbed network. The chapter is a significant step towards automating cyber triage with and risk quantification, which remains an important and elusive problem.

CHAPTER 4: ALERT TREE REDUCTION AND VISUALIZATION

Alert trees are used to show the collection of possible routes that may have been used by a cyber attacker or attackers to compromise a computer or set of computers. In real networks, these trees can become unmanageable in size, containing as many as 8,000 nodes. With such overwhelming amounts of data, it is difficult for cyber defenders to pinpoint network hotspots in order to prioritize defensive maneuvers. This raises the need to reduce strain on defenders by minimizing the amount of non-critical information that is presented to them. To this end, this chapter proposes several methods, as well as a novel data structure, for modifying alert trees in order to reduce visual strain on defenders. The methods are evaluated using a real-world dataset, which demonstrates that they are effective at reducing redundancy while limiting collateral information loss.

4.1 Introduction

Alert trees give a spatiotemporal representation of observed attack activity on a network. This visualization technique allows defenders to track threats and identify hotspots in a network. However, when alert trees become to large and complex, they become difficult to interpret, inhibiting the process of cyber triage and thereby incident response. In order to mitigate this limitation, this work investigates several methods for for reducing the size of alert trees. Although this process inherently results in the loss of information in the tree, this trade-off may be necessary for the sake of usability of the tree. This work measures this trade-off with respect to the reduction methods proposed.

4.1.1 Related Works

Alert and Attack Trees. Alert trees are closely related in structure of attack trees [49–51, 72]. While attack trees are predictive, alert trees are retrospective, showing paths that have already been accessed. Attack trees have been studied fairly broadly, while alert trees still have much to study.

The term *hypotree* has been used to refer to an altered subtree structure in [73], but is otherwise absent from the literature. The present usage is not inconsistent with this one. However, it may seem to imply that its inverse is a *hypertree*, which has been used to denote an unrelated concept [74–77]. For the purposes fo the present work, it is sufficient to exclusively use the one-way relationship of hypotree.

Graph Visualization. Network visualization has been used to present data to defenders for the purposes of cyber triage [72, 78–80]. These visualizations use various types of chart, including pie, line, and spiral charts. Metrics used to rank and color graphs vary, as alert trees contain multiple types of data such as the type, number of attacks observed, vertex connectedness, and number of paths [72, 81]. Some libraries used for graph visualization include Tulip [82], Graphviz [83], and Pajek [84].

In the context of graph visualization, an important property of visualizations is scalability [85], which is the focus of the present work. Another desireable property is planarity, in which no edges in a graph overlap [86]. This is important because it makes it easier to follow links. Planarity is guaranteed in alert trees but not alert graphs.

Alert Aggregation. Visualization reduction is closely related to alert aggregation [59, 87, 88]. Alert aggregation can be used to reduce alert cardinality, improving efficiency with the potential trade-off of accuracy. Contrasting these, the present work aims to optimize efficiency without sacrificing precision.

Information Loss. Information loss in graphs has not been studied in depth. During data anonymization, information loss has been studied in [89]. This work used a statistical approach, measuring changes incurred during the anonymization process. Another approach used perturbation cost to estimate information loss [90]. Note that information loss is not the same as data loss, which measures the compromise of data by cyber attackers.

4.1.2 Chapter Contributions

Contributions of this chapter include several methods for optimizing the visualization of alert trees, as well as the definition of a novel data structure related to alert trees. This data structure may also have independent value with respect to other types of trees. Specifically, this work describes and provides pseudocode for the following optimizations:

1. Merging Sibling Leaves
2. Merging Similar Sibling Branches
3. Truncating Hypografts (a novel data structure)

Finally, a case study demonstrates the usefulness of each algorithm by applying them to a well-known dataset. Metrics measured include the reduction in visual strain as well as its trade-off in lost information.

4.1.3 Chapter Outline

The rest of the chapter is organized as follows. Section 4.2 introduces the research problem and defines important terms. Section 4.3 details the methods used. Section 4.4 gives details of the case study. Section 4.4.3 presents the results of the case study. Section 4.5 discusses strengths and weaknesses of the work, and Section 4.6 concludes the work.

4.2 Problem Formalization

This section introduces concepts and terms used to frame the problem at hand.

4.2.1 Setting and Terminology

This chapter investigates the problem of intuitive and efficient threat tracking. Consider an enterprise network, which consists of computers, networking devices, and security devices (e.g., intrusion detection systems), and is managed by an administrator or administrators, referred to as

the *defender*. A cyber *attacker* may come from inside or outside the enterprise network, conducting malicious actions. Once the attacker establishes a foothold in the network (through exploits or other means), the attacker conducts lateral movements to compromise more computers in the enterprise network. These attacks leave footprints that can be detected by security devices, to form *alert paths*. The first computer in an alert path is known as the *origin* and all other computers are considered *victims*. The final computer may also be called the path's *target*.

When multiple paths branch out from a single node, these can be formulated into an *alert tree*. Using graph theory terminology, computers in an alert tree are referred to as *vertices*. Each connection between computers forms an *arc* (or directed edge) within the tree. Alert trees may be forward-looking or backward-looking, such that the root of the tree belongs to all of the tree's paths as either the origin or target, respectively.

The concept of alert trees is important because they serve a critical role in facilitating incident response. Specifically, alert trees help defenders intuitively understand the scope of an attack in terms of the breadth of network impact and focal points thereof. Note that that an alert path does not necessarily give a precise account of an attacker's activity. This is for the following reasons: (i) some attacks will undoubtedly fail, leaving links that merely show the effort of the attacker against a particular target; (ii) some attacker addresses may be spoofed or reflected, such that the source of the connection is some tertiary node not visible to network monitors; (iii) security devices may have false positives or negatives; and (iv) lastly, some exploits may target client applications such as web browsers, resulting in attack links that are effectively inverted (i.e., the source of the attack may be compromised rather than the destination). Because of these phenomena, the precise meaning of arcs in an alert tree is that the target has been exposed to the origin, which may or may not indicate compromise.

The visualization of alert trees has presented some significant limitations. Firstly, alert trees have been shown to be particularly large, with some cases resulting in over 5000 nodes, in under a week of attacks [61]. Obviously, it is difficult for a defender to look at such a tree and immediately zero in on its hotspots. At the same time, simply pruning elements of tree in a naïve attempt

to simplify it may result in significant information loss (i.e., the defender does not readily see some portion of the relevant data). Thus, the focus of this chapter is *reducing visual strain while minimizing information loss.*

4.2.2 Intuitive Problems

The above discussion naturally leads to several intuitive needs regarding alert tree visualization. Specifically;

1. Trees must be reasonably sized for viewing by defenders.
2. Valuable information must be preserved.
3. Relevant information must stand out.

These problems highlight some of the limitations in the literature, and inspire us to design and implement the methods proposed in this chapter. First, the concepts used in this chapter must be formally defined.

4.2.3 Alert Path and Alert Tree

The core of this work is the reduction operations on alert paths and trees. These concepts are used throughout the chapter, while the novel concept of *hypotree* is used only for one reduction type. As such, alert paths and trees are defined here, and hypotree will be defined in Section 4.3.

Alert path. Intuitively, an *alert path* describes a series of attacks traversing one or more edges, which may have been used by an attacker to conduct a multi-step attack against a network.

Definition 8 (Alert Path)**.** *Given a graph $G = (V, E)$, an alert path $p = (nodes, edges)$; where $p.nodes = (v_1, v_2, \ldots, v_\ell) \subseteq V$, such that $\forall v, v' \in p.nodes$, $v = v'$; and $p.edges = ((v_1, v_2), (v_2, v_3), \ldots, (v_{\ell-1}, v_\ell$ such that $e \in p.edges \rightarrow e \in E$*

Alert Tree. An alert tree is composed of nodes with parent/child relationships. Each node has a name and a color that represents some metric used to show the importance of a node. This metric

could be any of a variety of measurements, which may include asset value, asset vulnerability, threat score, etc. This chapter will use the threat score (TS) metric, although the model is metric agnostic. It is sufficient to note that node colors range from red to black (i.e., in hexadecimal notation: 0xFF0000 to 0x000000), where red indicates a higher value of the relevant metric, denoting a higher importance. This is demonstrated in Figure 4.1, and will be elaborated further in Section 4.3. These nodes are used to construct an alert tree based on the following definition.

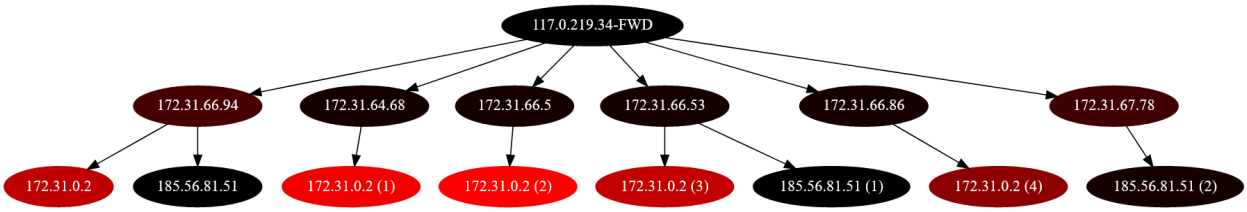

Figure 4.1: An example alert tree. Threat scores are indicated by the node's color, where red is the highest score.

Definition 9 (Alert Tree)**.** *An alert tree t is an arborescence (i.e., a rooted directed acyclic graph (DAG) where each node is accessible from the root by a unique sequence of $ancestors$), rooted at a particular node $t.root$, and for which each node n is annotated by name (denoted $n.name$) and color (denoted $n.color$). Sibling nodes (i.e., nodes which are adjacent to a common ancestor) may not share the same name, and nodes may not share a name with any of their ancestors.*

Alert trees come in two logical forms: forward and backward. For any node n_f in a forward tree, an edge $(n_f.parent, n_f)$ indicates an attack from $n_f.parent$ to n_f. Conversely, for any node n_b in a backward alert tree, an edge $(n_b.parent, n_b)$ indicates an attack from n_b to $n_b.parent$. These are formulated respectively to show the scope of victims that a particular attacker may have targeted, and the scope of attackers that may have targeted a particular victim.

It is also worth noting that alert paths can be reconstructed from the alert tree by extracting a node's ancestors. Note that in the case of backward alert trees, the ancestors must be reversed to retrieve the proper alert path.

4.2.4 Formalizing Intuitive Problems as Research Questions

Based on the preceding formalisms, the intuitive problems become rigorous Research Questions (RQs) as follows.

1. Given an alert tree, modify the tree to reduce visual strain.

2. Given a potential tree modification, minimize the amount of information lost.

3. Given an alert tree, identify salient information and highlight it for defenders.

Extraneous nodes may have names or substructures that are identical to others in the alert tree, and thus represent redundant data. Removing this redundant data can improve usability of the alert tree. This motivates RQ1 and RQ2.

Research Question 1: How much can one reduce alert tree size by merging similar nodes?

Research Question 2: How much can one reduce alert tree size by removing duplicate nodes?

While modifying an alert graph, the removal and merging of nodes can reduce the amount of information available to the defender. Specifically, when merging nodes, one should maximize the ratio of complexity reduction to information loss. Similarly, when removing duplicate nodes, one should preserve to location information of the nodes that were removed. This motivates RQ3.

Research Question 3: How can one preserve the information lost in the solutions to RQ1 and RQ2?

Salient information can represent a wide variety data, such as threat score, asset value, etc. One way to represent such data is to color-code the relevant graph elements. This motivates RQ4.

Research Question 4: How can one highlight salient information in an alert graph without increasing visual strain on the user?

4.3 Methods

Many portions of a given alert tree can present duplicate or extraneous information to the viewer, overwhelming their ability to extract meaningful insights. This chapter proposes several methods

for modifying the tree's presentation in order to focus on the most salient information within it. Specifically, these are merging sibling leaves, merging similar sibling branches, and truncating hypotrees, as highlighted in Figure 4.2.

Each of the three base functions can be used on its own to reduce a given alert tree. These represent reduction schedules one, three and five, respectively. However, because the nodes they merge or remove may overlap, it is generally unsafe to apply more than one reduction at a time. The only exception is the *merge sibling leaves* reduction, which may be applied after either of the other two base reductions because it does not create conflicts with them. This forms reduction schedules two and four. Once the trees have been reduced, they are exported and rendered into images. The remainder of this section describes the base functions.

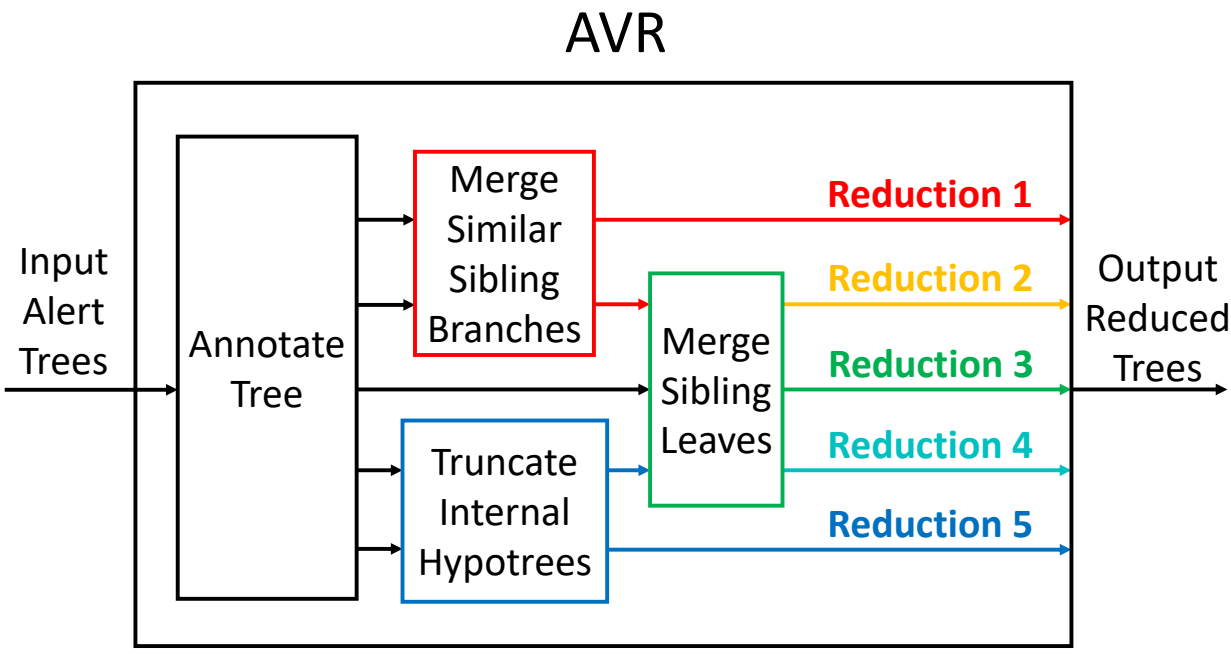

Figure 4.2: Reductions overview. Each of the primary-colored boxes represents a reduction module, while colored arrows represent a reduction schedule. Alert trees are passed through modules as specified by the reduction schedule.

4.3.1 Model Inputs

The model requires alert trees as inputs, as defined in Section 4.2. Specifically, it reads JSON objects from a file in the appropriate format. Once the trees are imported, it annotates them to facilitate the reduction algorithms and sends them to the appropriate functions.

4.3.2 Merging Sibling Leaves

High volume, low yield information can be reduced by merging sibling leaves, preserving information about the severity of attacks against the merged leaves according to the TS of the merged leaves. This also adopts the color of that specific node.

This approach uses a breadth-first traversal to iterate over the tree, parsing the list of children for each internal node in order to identify and remove its leaves. While parsing the leaves, it documents the highest TS within the set of leaves, then replaces the leaves with a single node, showing the number of leaves merged and applying coloring according to the TS selected above.

Algorithm 4 gives pseudocode for the merge sibling leaves function.

Algorithm 4 Merge Sibling Leaves

Input: $Root$
Output: $Root$, $Archives$

$Node\_Queue \leftarrow (Root)$
while $Node\_Queue \neq ()$ **do**
    $Current\_Node \leftarrow Node\_Queue.\text{next}()$
    $Colors \leftarrow \emptyset$
    $To\_Be\_Merged \leftarrow \emptyset$
    for each $Child \in Current\_Node.children$ **do**
        if $Child.children = \emptyset$ **then** ▷ This is a leaf
            $Colors \leftarrow Colors \cup \{Child.color\}$
            $To\_Be\_Merged \leftarrow To\_Be\_Merged \cup \{Child\}$
    $Node\_Queue \leftarrow Node\_Queue \cup Current\_Node.children$
    $New\_Name \leftarrow |To\_Be\_Merged| +$ " Merged Nodes"
    $New\_Color \leftarrow \max_{color \in Colors}$
    if $|To\_Be\_Merged| > 1$ **then**
        $Current\_Node.children \leftarrow \{(New\_Name, Current\_Node, \emptyset, New\_Color)\}$
return $Root$

4.3.3 Merging Similar Sibling Branches

Duplicate information can be reduced by merging similar sibling branches. This allows the viewer to more quickly identify common patterns within the tree. The method for merging similar sibling branches is given in Algorithm 5.

In order to merge identical branches, the algorithm conducts a breadth-first traversal of the tree, recursively comparing sibling branches. When it identifies two siblings with identical subtrees, it combines them into a single node, preserving the shared form of the subtree. It then labels the new node with the number of siblings that were merged, applying the same color from the sibling with the highest ETS.

Branches are compared using a hash function, which produces a tuple

$$H(root) = (root, (H(node)|node \in root.children)),$$

which incidentally produces $H(leaf) = (leaf, ())$ for leaf nodes. For comparison purposes, each group of siblings is assumed to be sorted consistently throughout the tree. This allows for easier comparison of subtree hashes and is trivially enforced in the implementation.

4.3.4 Truncating Hypotrees

Intuitively, a *hypotree* is a tree which resembles a portion of another tree (i.e., its *hypertree*), where the two trees have identical roots. This relationship is distinct from the concept of a subtree, which constitutes a branch of its supertree. By contrast, a hypotree may be missing individual nodes or branches relative to its hypertree.

Recall that each node n in a tree is annotated with a name and a color. These are denoted by $n.name$ and $n.color$, respectively. Additionally, the concept of a node's *ancestry* (denoted $n.ancestors$) is derived from its relationship to other members of the tree. This intuitively results in a sequence of nodes, starting with the root, followed by each successive child leading toward n, and ending with n's parent. In any given tree, each node has a single ancestry. Furthermore, if

Algorithm 5 Merge Similar Sibling Branches

Input: $Root$
Output: $Root$

$Node\_Queue \leftarrow (Root)$
while $Node\_Queue \neq ()$ **do**
    $Current\_Node \leftarrow Node\_Queue.\text{next}()$
    $Unique\_Hashes \leftarrow ()$
    $New\_Children \leftarrow ()$
    for each $i \in [1, 2, \ldots, |Current\_Node.children|]$ **do**
        if $Current\_Node.children_i.children \neq \emptyset$ **then** ▷ Only check siblings that are not leaves
            $Current\_Hash \leftarrow H(Current\_Node.children_i)$
            if $\exists Hash \in Unique\_Hashes : Hash_2 = Current\_Hash_2$ **then**
                $Hash_3 \leftarrow Hash_3 + 1$
                $Hash_1.name \leftarrow Hash_3 +$ " Merged Nodes"
                if $Current\_Hash_1.color > Hash\_1.color$ **then**
                    $Hash_1.color \leftarrow Current\_Hash_1.color$
                for each $j \in [1, 2, \ldots, |Current\_Hash_2.children|]$ **do**
                    if $Current\_Hash_2.children_j.color > Hash_2.children_j.color$ **then**
                        $Hash_2.children_j.color \leftarrow Current\_Hash_2.children_j.color$
            else
                $Unique\_Hashes \leftarrow Unique\_Hashes \cup (Current\_Hash_1, Current\_Hash_2, 1)$ ▷ Here the third element stores the number of merged nodes
                $New\_Children \leftarrow New\_Children \cup (Current\_Node.children_i)$
    $Current\_Node.children \leftarrow New\_Children$
    $Node\_Queue \leftarrow Node\_Queue \cup Current\_Node.children$
return $Root$

there are no identical siblings in the tree, each node's ancestry is unique in the tree. Based on this, we define hypotree in Definition 10.

Definition 10 (Hypotree)**.** *A tree T_{hypo} is a hypotree relative to a tree T_{hyper} if* $\forall n \in T_{hypo}, \exists n\prime \in T_{hyper} : n\prime.ancestors = n.ancestors$

Denote "T_{hypo} is a hypotree of T_{hyper}" as $T_{hypo} \trianglelefteq T_{hyper}$. Similarly, denote hypertree ($\trianglerighteq$); proper hypertee ($\triangleright$), a hypertree that is not also a hypotree; and proper hypotree ($\triangleleft$), a hypotree that is not also a hypertree. Recall that "hypertree" is an existing concept in other contexts, so "hypotree" is preferred where possible. Where necessary, the symbol ($\trianglerighteq$) can be used to avoid confusion, as this is specific to the current usage.

Because all nodes in hypotrees are duplicated in their respective hypertrees, they are redundant. For this reason, hypotrees are truncated in order to reduce visual strain on the viewer. This preserves the most amount of information since all edges are preserved, even if their location and number are lost. The method for truncating hypotrees is given in Algorithm 6.

This algorithm parses the tree using a breadth-first traversal, marking all nodes that share the same name. It then compares the subtrees of each set of identical nodes, preserving trees which have no proper hypertrees and truncating the rest. In the case of two equivalent hypotrees (i.e., $A \trianglelefteq B \wedge B \trianglelefteq A$), it preserves only the one appearing first in the traversal (i.e., the one closer to the root). Truncated trees contain annotations to show viewers that a hypotree of the trunk is available for reference at another part of the tree. Archives are saved in case precise information about a particular hypotree needs to be retrieved later.

4.4 Case Study

This section describes the experimental environment and results.

4.4.1 Dataset

The case study utilized CSE-CIC-IDS2018 [41], a well-known dataset collected from a testbed with both injected and wild attacks. From the network traffic, Snort [91] produced 3.3M alerts.

Algorithm 6 Truncate Hypotrees

Input: $Root$

Output: $Root$, $Archive$

$Candidates \leftarrow \emptyset$
$Unique\_Names \leftarrow \emptyset$
for each $Node \in Root.descendants$ **do**
    if $\exists Name\_List \in Unique\_Names : Name\_List_1 = Node.name$ **then**
        $Name\_List \leftarrow Name\_List \cup (Node)$
        $Candidates \leftarrow Candidates \cup \{Node.name\}$ ▷ Nodes sharing a name become candidates
    else
        $Unique\_Names \leftarrow Unique\_Names \cup \{(Node.name, Node)\}$ ▷ Here elements 2 onward are nodes with the same name
$Trunks \leftarrow (\emptyset^{|Unique\_Nodes|})$
$Trunk\_Colors \leftarrow (()^{|Unique\_Nodes|})$
for each $n \in [1, 2, \ldots, |Unique\_Names|]$ **do**
    if $Unique\_Names_n \in Candidates$ **then**
        for each $i, j \in [2, 3, \ldots, n), i < j]$ **do** ▷ Compare pairs of candidates
            if $Unique\_Names_{n,i} \lhd Unique\_Names_{n,j}$ **then**
                if $|Unique\_Names_{n,i}.descendants| > 1$ **then**
                    $Trunks_n \leftarrow Trunks_n \cup (Unique\_Names_{n,i})$ ▷ Mark hypotree i for truncation
                    $Trunk\_Colors_n \leftarrow Trunk\_Colors_n \cup (\max_{d \in Unique\_Names_{n,i}.descendants}(d.color))$
            else
                if $|Unique\_Names_{n,j}.descendants| > 1$ **then**
                    $Trunks_n \leftarrow Trunks_n \cup (Unique\_Names_{n,j})$ ▷ Mark hypotree j for truncation
                    $Trunk\_Colors_n \leftarrow Trunk\_Colors_n \cup (\max_{d \in Unique\_Names_{n,j}.descendants}(d.color))$
$Archives \leftarrow \emptyset$
for $i \in [1, \ldots, |Trunks|]$ **do**
    for $j \in [1, \ldots, |Trunks_i|$ **do**
        $trunk\_archive \leftarrow \text{copy}(Trunks_{i,j}.parent)$
        $trunk\_archive.parent \leftarrow \emptyset$
        $new\_trunk \leftarrow \text{copy}(Trunks_{i,j})$
        $new\_trunk.color \leftarrow Trunk\_Colors_{i,j}$
        $Trunk_{i,j}.parent \leftarrow trunk\_archive$
        $Archives \leftarrow Archives \cup \{trunk\_archive\}$
return $Root$, $Archives$

These were converted into a network graph and assembled into trees using an adaptation of APIN [61]. Nodes were ranked according to threat score, calculated as the geometric mean of the volume and diversity of alerts incident to the node. Trees were ranked according to the threat score of their root node.

From the resulting trees, 15 were selected to be reduced: 5 each from the top ranked, bottom ranked, and randomly selected trees. Statistics for the selected trees are given in Table 4.1.

Table 4.1: Dataset statistics for selected alert trees. Numbers given are averages. Top and bottom 5 refer to TS.

	Full size	Unique nodes	Unique edges
Top 5	9.80	7.6	8.8
Random 5	1999.0	234.4	825.6
Bottom 5	15.0	10.6	14.0

4.4.2 Evaluation Metrics

The effectiveness of the methods is demonstrated using a total of 4 metrics, including three atomic metrics and one aggregate metric. The atomic metrics are visual strain reduction (VSR), node retention (NR) and threat score retention (TSR). The latter two are derived from the notion of *information loss*, as its additive inverse (i.e., $1 - loss$). These three metrics are also combined to create a *reduction index*.

The first metric measures VSR as the number of nodes in the reduced tree relative to the full tree. The formula for VSR is given in Equation.VSR has a range of [0,1] (as a percentage) relative to the full graph (which has a VSR of 0) and is measured for each of the five reductions. 100% VSR is ideal.

In order to measure the amount of information loss for the following two, one must specify what types of information are present in the alert tree and relevant to the problem at hand. Given the present goal of intuitively visualizing cyber attack traffic in a network, the following types of information will form the basis of the metrics used: Node presence in tree and threat score information.

These metrics measure *information retention* as the number of nodes or threat scores from the full tree that remain after the reduction. These are referred to as "node retention" (NR) and "threat score retention" (TSR), respectively. Both of these measurements have a range of [0,1] (as a percentage) relative to the full graph (which has both an NR and TSR of 1) and are measured for each of the five reductions. 100% information retention is ideal.

Note that NR is not necessarily $1 - VSR$, since some reductions add supplemental nodes after pruning (e.g., merged leaves). These merged nodes increase visual strain but do not increase node retention, since they do not belong to the full tree. However, they may increase TSR, since some of the supplemental nodes inherit color codes (i.e., threat score) from the node(s) they replaced.

Naturally, one may conceive of naïve approaches to manipulate these metrics. For example, an empty tree has 100% VSR, and a full tree has 100% NR and TSR. In light of this, these metrics are combined into a *reduction index* using the harmonic mean of the three atomic metrics. The harmonic mean is most appropriate here because it is most significantly impacted by low values, favoring models that produce good results in all metrics rather than those with one excellent value and many poor values. This results in both of the above naïve approaches scoring a reduction index of 0. Contrast this with arithmetic mean, for which the naïve models would score a reduction index of 33% and 66%, respectively. This demonstrates that, compared with arithmetic mean, harmonic mean produces scores that better represent the intuitive meaning of the reduction index.

4.4.3 Results

Results of the experiments are given in Table 4.2. The merge leaves reduction consistently performed the best, with its RI leading by margins of 0.17, 0.15 and 0.33 for the top 5, random 5 and bottom 5 categories, respectively.

All reductions performed best against low-ranking trees. In this category, merge leaves saw an RI improvement of 0.25, merge branches 0.08, and truncate 0.08, relative to their next highest category.

Merge leaves tended to perform better at information retention than VSR. Merge branches

performed better in node retention than VSR or TSR. Truncate retained all node information and nearly all threat score information in every case, but struggled to reduce visual strain. In some cases, it was unable to reduce any visual strain at all.

Table 4.2: Performance of each method on trees selected by TS. VSR is average visual strain reduction, NR is average node retention, TSR is average threat score retention, and RI is reduction index, as the harmonic mean of VSR, NR and TSR.

Algorithm	VSR	NR	TSR	RI
Merge Branches (top 5)	0.243	0.539	0.278	0.313
Merge Branches (random 5)	0.352	0.553	0.254	0.349
Merge Branches (bottom 5)	0.433	0.493	0.360	0.421
Merge Leaves (top 5)	0.363	0.577	0.611	**0.489**
Merge Leaves (random 5)	0.282	0.824	0.799	**0.499**
Merge Leaves (bottom 5)	0.791	0.744	0.730	**0.754**
Truncate (top 5)	0.009	1.0	0.999	0.026
Truncate (random 5)	0.0	1.0	1.0	0.0
Truncate (bottom 5)	0.037	1.0	0.983	0.103

4.4.4 Answering Research Questions

Research Question 1: How much can one reduce alert tree size by merging similar nodes? By merging leaves, tree size can be reduced by as much 79%, and by merging branches tree size can be reduced by as much as 43%.

Research Question 2: How much can one reduce alert tree size by removing duplicate nodes? By truncating hypotrees, tree size can be reducecd by as much as 3.7%.

Research Question 3: How can one preserve the information lost in the solutions to RQ1 and RQ2? The best way to preserve information is to truncate hypotrees, which contain almost exclusively redundant information. Otherwise, merging leaves tends to retain more information than merging branches.

Research Question 4: How can one highlight salient information in an alert graph without increasing visual strain on the user? Color-coding salient information allows the tree to highlight important data such as network hotspots and threat activity. Color can be used for both nodes and edges, so NR and TSR are the metrics to look at when one needs information salience.

The novel reductions had a broad range of performance, with each one having a different strength. Since user needs will vary, it will be important to consider these differences when choosing how to handle alert trees. Meanwhile, these results are only preliminary and warrant further study.

4.5 Chapter Discussion

The methods used in this study are restrictive in that they have some areas of overlap depending on the input alert tree. Specifically, merging branches and truncating hypotrees are not independent of each other. In other words, the results of doing one before the other do not match those of the opposite ordering. This can have significant impact on the performance, although it may be reasonably to try both in alternating order and selecting the better result on a case-by-case basis. This is left to future work.

The case study is limited in that the dataset utilizes threat score to rank edges and paths. This metric has not been robustly analyzed and may not produce the best scores relative to a particular attack. However, the methods proposed in the present study need not use threat score, but could easily be adapted to rank nodes according to monetary value, vulnerability score, or other related metrics.

4.6 Chapter Summary

This work introduced several methods for reducing the size of alert trees while retaining as much information as possible. The three core functions can be used independently or combined in a total of five reduction schedules. One of the reductions includes the use of a novel data structure, the hypotree. These reductions were applied to alert trees from a research testbed dataset, and were evaluated for information retention and visual strain reduction. Results show that the reductions have varied performance relative to each other for different types of trees. This finding warrants more research into how the application of the reductions may be optimized.

CHAPTER 5: CONCLUSION

This section presents a discussion on the research reported in the Dissertation, explores future research directions, and describes the conclusions drawn from the Dissertation work.

5.1 Further Discussion

The methods contained in this Dissertation approach the problem of alert tree reconstruction from a variety of algorithmic perspectives. These algorithms have various trade-offs among them. These include a coverage/efficiency trade-off and a pre-/post- processing-time trade-off. Additionally, the assumptions that one makes about underlying data has significant impact on these trade-offs relative to the given models.

Specifically, APIN tends to prioritize efficiency and storage but not coverage. On the other hand, Autocrat tends to prioritize coverage but not efficiency. APIN has better reinsertion and pruning performance, but has difficulty handling highly-connected networks. AutoCRAT can retrieve paths immediately but takes longer on average to assemble trees. Defenders that have a strong understanding of their network environments can make good use of these trade-offs in order to optimize their system performance for their specific needs.

The reconstruction methods are dependent on incoming threat intelligence in order to begin building alert trees. This is because they start building from the root of the tree (i.e., the reference node). Building complete alert trees for every node in the network would be computationally infeasible, although some more restrictive data structure may be appropriate for this task.

5.2 Future Research Directions

There are several exciting future research directions. The first research direction is to automatically detect potential incidents in order to preemptively build the relevant alert trees. This could improve the monitoring process by freeing up the defender to prioritize other defensive actions.

The second research direction is to incorporate host-based activity into the models. The ability

to reliably link network and host activities would give highly precise information to defenders, allowing them to perceive threats at every applicable level.

The third research direction is to incorporate the studies presented in the Dissertation, and Cyber Triage in general, into the Cybersecurity Dynamics framework [3–6], which inspired the research presented in this Dissertation. The Cybersecurity Dynamics framework is driven by the need for cybersecurity quantification, which is fundamental to many applications such as cyber defense command-and-control and cyber risk management (including cyber insurance). An important consideration in the framework is the cybersecurity metrics problem, which has been challenging the community for decades [92–94]. Fortunately, there has been significant progress towards tackling this problem [36, 95–108]. This explains why Cybersecurity Metrics is one pillar of Cybersecurity Dynamics, on par with (i) the Cybersecurity First-Principle Modeling pillar for the purposes of seeking cybersecurity laws that govern the evolution of the global cybersecurity state under various kinds attack-defense interactions [28–30, 109–119] (with notable recent results in [109–111, 119]), and (ii) the Cybersecurity Data Analytics pillar [7–14].

As mentioned above, the present Dissertation falls under the Cybersecurity Data Analytics pillar, or more specifically Algorithmic Data Analytics. It also intersects with the Cybersecurity Metrics pillar because it defines and utilizes several metrics, such as threat score, which measures the scale and diversity of attacks against a target; and information loss metrics, which measure the effect of a transformation on a graph insofar as it accurately models a phenomenon.

The research presented in this Dissertation, and Cyber Triage in general, can be incorporated into the Cybersecurity Dynamics framework to enrich it as follows.

- Pertinent to the Cybersecurity Metrics pillar, it is important to systematically define a suite of metrics that are necessary and sufficient for the purposes of Cyber Triage. This would require us to, for example, define metrics to quantify the attributes that are involved in the Cyber Triage process (e.g., under what circumstances on alert can be triaged with high confidence if not certainty?). How would these new metrics enrich the SARR cybersecurity metrics framework [95] towards a more comprehensive metrics framework?

- Pertinent to the Cybersecurity Data Analytics pillar, it is important to investigate the conditional relevance of the proposed Algorithmic Data Analytics if attack event details are omitted. For example, what if alerts represent uncertain events (e.g., unreliable threat intelligence which only suspects that some activities are attacks)? What if side-channel exploits are used? Can Cyber Triage be improved by leveraging the threat prediction or forecasting (as in [8–14]?

- Pertinent to the Cybersecurity First-Principle modeling pillar, one important question is how to mathematically model the triage process so that it can be incorporated into the broader Cybersecurity Dynamics equations? How can one build models to quantify the effectiveness of Cyber Triage as a defense mechanism, for example in a fashion similar to what are described in [28–30, 108–119]?

Accomplishing the research outlined above will enhance the usefulness of the Cybersecurity Dynamics framework while deepening our understanding of cybersecurity from a holistic perspective rather than a building-blocks perspective.

5.3 Conclusion

This Dissertation discussed the notion of cyber triage, including multi-step attack reconstruction via alert paths and trees, and the reduction and visualization thereof. These problems are critical to mission assurance and cyber security practice because the complexity of modern computer systems demands a high level of expertise and many hours of diligent monitoring in order to maintain the security of mission resources. The solutions discussed in this paper have made significant progress toward formalizing and addressing this problem insofar as it applies to incident response. The models produced in this work have brought novel perspectives to these problems and will hopefully inspire many future works, propelling the field of Algorithmic Data Analytics to greater scientific advancement.

BIBLIOGRAPHY

[1] J. Mireles, J. Cho, and S. Xu, "Extracting attack narratives from traffic datasets," in *Proc. CyCon U.S. 2016*, pp. 118–123, 2016.

[2] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "HERCULE: attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016* (S. Schwab, W. K. Robertson, and D. Balzarotti, eds.), pp. 583–595, ACM, 2016.

[3] S. Xu, "Cybersecurity dynamics," in *Proc. Symposium on the Science of Security (HotSoS'14)*, pp. 14:1–14:2, 2014.

[4] S. Xu, "Emergent behavior in cybersecurity," in *Proc. HotSoS*, pp. 13:1–13:2, 2014.

[5] S. Xu, "Cybersecurity dynamics: A foundation for the science of cybersecurity," in *Proactive and Dynamic Network Defense* (Z. Lu and C. Wang, eds.), vol. 74, pp. 1–31, 2019.

[6] S. Xu, "The cybersecurity dynamics way of thinking and landscape (invited paper)," in *ACM Workshop on Moving Target Defense*, 2020.

[7] D. Li, Q. Li, Y. F. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Comput. Surv.*, vol. 55, nov 2021.

[8] Z. Fang, M. Xu, S. Xu, and T. Hu, "A framework for predicting data breach risk: Leveraging dependence to cope with sparsity," *IEEE T-IFS*, vol. 16, pp. 2186–2201, 2021.

[9] M. Xu, K. M. Schweitzer, R. M. Bateman, and S. Xu, "Modeling and predicting cyber hacking breaches," *IEEE T-IFS*, vol. 13, no. 11, pp. 2856–2871, 2018.

[10] Y. Chen, Z. Huang, S. Xu, and Y. Lai, "Spatiotemporal patterns and predictability of cyberattacks," *PLoS One*, vol. 10, p. e0124472, 05 2015.

[11] M. Xu, L. Hua, and S. Xu, "A vine copula model for predicting the effectiveness of cyber defense early-warning," *Technometrics*, vol. 59, no. 4, pp. 508–520, 2017.

[12] Z. Zhan, M. Xu, and S. Xu, "Characterizing honeypot-captured cyber attacks: Statistical framework and case study," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1775–1789, 2013.

[13] Z. Zhan, M. Xu, and S. Xu, "Predicting cyber attack rates with extreme values," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1666–1677, 2015.

[14] V. Trieu-Do, R. Garcia-Lebron, M. Xu, S. Xu, and Y. Feng, "Characterizing and leveraging granger causality in cybersecurity: Framework and case study," *EAI Endorsed Trans. Security Safety*, vol. 7, no. 25, p. e4, 2020.

[15] X. Fang, M. Xu, S. Xu, and P. Zhao, "A deep learning framework for predicting cyber attacks rates," *EURASIP J. Information Security*, vol. 2019, p. 5, 2019.

[16] J. Navarro, A. Deruyver, and P. Parrend, "A systematic survey on multi-step attack detection," *Computers & Security*, vol. 76, pp. 214–249, 2018.

[17] Lockheed Martin, "Cyber kill chain." `http://cyber.lockheedmartin.com/solutions/cyber-kill-chain`, (Accessed July 08, 2016).

[18] Mandiant, "Apt1 report." `https://www.fireeye.com/content/dam/fireeyewww/services/pdfs/mandiant-apt1-report.pdf`, February 16, 2013 (Accessed July 08, 2016).

[19] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*, pp. 305–316, IEEE, 2010.

[20] E. Ficke, K. M. Schweitzer, R. M. Bateman, and S. Xu, "Analyzing root causes of intrusion detection false-negatives: Methodology and case study," in *Proc. IEEE MILCOM'2019*, 2019.

[21] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in *Proc. ACSAC'2016*, p. 583595, 2016.

[22] S. Saurabh and A. S. Sairam, "A more accurate completion condition for attack-graph reconstruction in probabilistic packet marking algorithm," in *Proc. 2013 National Conference on Communications*, pp. 1–5, 2013.

[23] J. Morales, M. Main, W. Luo, S. Xu, and R. Sandhu, "Building malware infection trees," in *Proc. MALWARE*, pp. 50–57, 2011.

[24] A. F. Shosha, J. I. James, and P. Gladyshev, "A novel methodology for malware intrusion attack path reconstruction," in *Lecture Notes in Social Informatics and Telecommunications Engineering*, pp. 131–140, Springer Berlin Heidelberg, 2012.

[25] J. A. Morales, A. Al-Bataineh, S. Xu, and R. S. Sandhu, "Analyzing and exploiting network behaviors of malware," in *SecureComm*, pp. 20–34, 2010.

[26] J. Tian, X. Li, Z. Tian, and W. Qi, "Network attack path reconstruction based on similarity computation," in *Proc. ICNC-FSKD*, pp. 2457–2461, 2017.

[27] P. Mell, K. Scarfone, and S. Romanosky, *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. NIST and Carnegie Mellon University, 1 ed., June 2007.

[28] W. Lu, S. Xu, and X. Yi, "Optimizing active cyber defense dynamics," in *Proc. GameSec'13*, pp. 206–225, 2013.

[29] S. Xu, W. Lu, L. Xu, and Z. Zhan, "Adaptive epidemic dynamics in networks: Thresholds and control," *ACM TAAS*, vol. 8, no. 4, 2014.

[30] G. Da, M. Xu, and S. Xu, "A new approach to modeling and analyzing security of networked systems," in *Proc. HotSoS'14*, pp. 6:1–6:12, 2014.

[31] F. Leitold, A. Arrott, and K. Hadarics, "Quantifying cyber-threat vulnerability by combining threat intelligence, it infrastructure weakness, and user susceptibility," in *24th Annual EICAR Conference*, 2016.

[32] S. Lee, S. Kim, K. Choi, and T. Shon, "Game theory-based security vulnerability quantification for social internet of things," *Future Generation Computer Systems*, vol. 82, pp. 752–760, 2018.

[33] H. Hu, H. Zhang, and Y. Yang, "Security risk situation quantification method based on threat prediction for multimedia communication network," *Multimedia Tools and Applications*, vol. 77, no. 16, pp. 21693–21723, 2018.

[34] M. Frigault and L. Wang, "Measuring network security using bayesian network-based attack graphs," in *Proc. IEEE ICSAC*, pp. 698–703, 2008.

[35] "CVE." Available from MITRE, 2020.

[36] J. Mireles, E. Ficke, J. Cho, P. Hurley, and S. Xu, "Metrics towards measuring cyber agility," *IEEE T-IFS*, vol. 14, no. 12, pp. 3217–3232, 2019.

[37] Z. Zhan, M. Xu, and S. Xu, "Predicting cyber attack rates with extreme values," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1666–1677, 2015.

[38] R. M. Rodriguez, E. Golob, and S. Xu, "Human cognition through the lens of social engineering cyberattacks," *CoRR (to appear in Frontiers in Psychology-Cognition)*, vol. abs/2007.04932, 2020.

[39] P. Du, Z. Sun, H. Chen, J. H. Cho, and S. Xu, "Statistical estimation of malware detection metrics in the absence of ground truth," *IEEE T-IFS*, vol. 13, no. 12, pp. 2965–2980, 2018.

[40] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Comput. Netw.*, vol. 34, pp. 579–595, Oct. 2000.

[41] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP*, pp. 108–116, 2018.

[42] Suricata, "Suricata | open source ids / ips / nsm engine." `https://suricata-ids.org/download/`, Mar 2018.

[43] Rules.emergingthreats.net, "Welcome to the emerging threats rule server." `https://rules.emergingthreats.net/`, 2019.

[44] S. Haas and M. Fischer, "Gac: graph-based alert correlation for the detection of distributed multi-step attacks," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pp. 979–988, 2018.

[45] K. Zhang, F. Zhao, S. Luo, Y. Xin, and H. Zhu, "An intrusion action-based ids alert correlation analysis and prediction framework," *IEEE Access*, vol. 7, pp. 150540–150551, 2019.

[46] S. C. De Alvarenga, S. Barbon Jr, R. S. Miani, M. Cukier, and B. B. Zarpelão, "Process mining and hierarchical clustering to help intrusion alert visualization," *Computers & Security*, vol. 73, pp. 474–491, 2018.

[47] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," in *2011 International Conference on Information Warfare and Security*, 2011.

[48] B. Strom, "Att&ck 101: Cyber threat intelligence," 2018.

[49] Y. Chen, B. Boehm, and L. Sheppard, "Value driven security threat modeling based on attack path analysis," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pp. 280a–280a, IEEE, 2007.

[50] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "{SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 487–504, 2017.

[51] C.-M. Chen, D. Guan, Y.-Z. Huang, and Y.-H. Ou, "Attack sequence detection in cloud using hidden markov model," in *2012 seventh asia joint conference on information security*, pp. 100–103, IEEE, 2012.

[52] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, 1998.

[53] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 245–254, 2002.

[54] P. Ning and D. Xu, "Learning attack strategies from intrusion alerts," in *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 200–209, 2003.

[55] M. Howard, J. Pincus, and J. M. Wing, "Measuring relative attack surfaces," in *Computer security in the 21st century*, pp. 109–137, Springer, 2005.

[56] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 336–345, 2006.

[57] M. Cinque, R. Della Corte, and A. Pecchia, "Contextual filtering and prioritization of computer application logs for security situational awareness," *Future Generation Computer Systems*, vol. 111, pp. 668–680, 2020.

[58] X. Wang, X. Gong, L. Yu, and J. Liu, "Maac: Novel alert correlation method to detect multi-step attack," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 726–733, IEEE, 2021.

[59] A. Nadeem, S. Verwer, and S. J. Yang, "Sage: Intrusion alert-driven attack graph extractor," in *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 36–41, IEEE, 2021.

[60] B. Mao, J. Liu, Y. Lai, and M. Sun, "Mif: A multi-step attack scenario reconstruction and attack chains extraction method based on multi-information fusion," *Computer Networks*, vol. 198, p. 108340, 2021.

[61] E. Ficke and S. Xu, "Apin: Automatic attack path identification in computer networks," 2020.

[62] K. Alsubhi, I. Aib, and R. Boutaba, "Fuzmet: A fuzzy-logic based alert prioritization engine for intrusion detection systems," *International Journal of Network Management*, vol. 22, no. 4, pp. 263–284, 2012.

[63] A. A. Ramaki, A. Rasoolzadegan, and A. G. Bafghi, "A systematic mapping study on intrusion alert analysis in intrusion detection systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–41, 2018.

[64] G. Apruzzese, F. Pierazzi, M. Colajanni, and M. Marchetti, "Detection and threat prioritization of pivoting attacks in large networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 404–415, 2017.

[65] G. Shearer, N. Leslie, P. Ritchey, T. Braun, and F. Nelson, "Ids alert prioritization through supervised learning," in *Proceedings of the NATO Specialists Meeting on Predictive Analytics and Analysis in the Cyber Domain, 10-11 October 2017, Sibiu, Romania*, 2017.

[66] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[67] A. Thakkar and R. Lohiya, "A review of the advancement in intrusion detection datasets," *Procedia Computer Science*, vol. 167, pp. 636–645, 2020.

[68] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser, "Towards the creation of synthetic, yet realistic, intrusion detection datasets," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 1209–1214, IEEE, 2016.

[69] D. Chou and M. Jiang, "Data-driven network intrusion detection: A taxonomy of challenges and methods," *arXiv preprint arXiv:2009.07352*, 2020.

[70] FireEye, "The numbers game: How many alerts is too many to handle?." https://www.fireeye.com/offers/rpt-idc-numbers-game-special-report.html, 2015.

[71] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[72] M. Angelini, N. Prigent, and G. Santucci, "Percival: proactive and reactive attack and response assessment for cyber incidents using visual analytics," in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–8, IEEE, 2015.

[73] A. V. Gerbessiotis, "An architecture independent study of parallel segment trees," *Journal of Discrete Algorithms*, vol. 4, no. 1, pp. 1–24, 2006.

[74] A. Brandstädt, V. D. Chepoi, and F. F. Dragan, "The algorithmic use of hypertree structure and maximum neighbourhood orderings," *Discrete Applied Mathematics*, vol. 82, no. 1-3, pp. 43–77, 1998.

[75] J. R. Goodman and C. H. Sequin, "Hypertree: A multiprocessor interconnection topology," *IEEE Transactions on Computers*, vol. 30, no. 12, pp. 923–933, 1981.

[76] G. Gottlob, N. Leone, and F. Scarcello, "Hypertree decompositions and tractable queries," *Journal of Computer and System Sciences*, vol. 64, no. 3, pp. 579–627, 2002.

[77] A. Schidler and S. Szeider, "Computing optimal hypertree decompositions," in *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 1–11, SIAM, 2020.

[78] A. Sethi and G. Wills, "Expert-interviews led analysis of eevia model for effective visualization in cyber-security," in *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–8, IEEE, 2017.

[79] A.-P. Lohfink, S. D. D. Anton, H. D. Schotten, H. Leitte, and C. Garth, "Security in process: Visually supported triage analysis in industrial process data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 4, pp. 1638–1649, 2020.

[80] M. J. McGuffin, "Simple algorithms for network visualization: A tutorial," *Tsinghua Science and Technology*, vol. 17, no. 4, pp. 383–398, 2012.

[81] E. Kerzner, A. Lex, C. L. Sigulinsky, T. Urness, B. W. Jones, R. E. Marc, and M. Meyer, "Graffinity: Visualizing connectivity in large graphs," in *Computer Graphics Forum*, vol. 36, pp. 251–260, Wiley Online Library, 2017.

[82] D. Auber, "Tulipa huge graph visualization framework," in *Graph drawing software*, pp. 105–126, Springer, 2004.

[83] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphvizopen source graph drawing tools," in *International Symposium on Graph Drawing*, pp. 483–484, Springer, 2001.

[84] V. Batagelj and A. Mrvar, "Pajek-program for large network analysis," *Connections*, vol. 21, no. 2, pp. 47–57, 1998.

[85] L. Hao, C. G. Healey, and S. E. Hutchinson, "Ensemble visualization for cyber situation awareness of network security data," in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–8, IEEE, 2015.

[86] I. Herman, G. Melançon, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on visualization and computer graphics*, vol. 6, no. 1, pp. 24–43, 2000.

[87] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, 2013.

[88] R. Sadoddin and A. Ghorbani, "Alert correlation survey: framework and techniques," in *Proceedings of the 2006 international conference on privacy, security and trust: bridge the gap between PST technologies and business services*, pp. 1–10, 2006.

[89] D. F. Nettleton, "Information loss evaluation based on fuzzy and crisp clustering of graph statistics," in *2012 IEEE International Conference on Fuzzy Systems*, pp. 1–8, IEEE, 2012.

[90] D. F. Nettleton, V. Torra, and A. Dries, "The effect of constraints on information loss and risk for clustering and modification based graph anonymization methods," *arXiv preprint arXiv:1401.0458*, 2014.

[91] "Snort - network intrusion detection & prevention system." `https://www.snort.org/downloads`, Mar 2018.

[92] I. R. Council, "Hard problem list." `http://www.infosec-research.org/docs_public/20051130-IRC-HPL-FINAL.pdf`, 2007.

[93] N. Science and T. Council, "Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program." `https://www.nitrd.gov/SUBCOMMITTEE/csia/Fed_Cybersecurity_RD_Strategic_Plan_2011.pdf`, 2011.

[94] D. Nicol, B. Sanders, J. Katz, B. Scherlis, T. Dumitra, L. Williams, and M. P. Singh, "The science of security 5 hard problems (august 2015)." `http://cps-vo.org/node/21590`.

[95] S. Xu, "Sarr: A cybersecurity metrics and quantification framework (keynote)," in *Science of Cyber Security - Third International Conference (SciSec'2021)*, vol. 13005 of *Lecture Notes in Computer Science*, pp. 3–17, Springer, 2021.

[96] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, vol. 49, pp. 62:1–62:35, Dec. 2016.

[97] J.-H. Cho, S. Xu, P. M. Hurley, M. Mackay, T. Benjamin, and M. Beaumont, "Stram: Measuring the trustworthiness of computer-based systems," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 128:1–128:47, 2019.

[98] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *J. Comput. Secur.*, vol. 21, no. 4, pp. 561–597, 2013.

[99] L. Wang, S. Jajodia, A. Singhal, P. Cheng, and S. Noel, "k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities," *IEEE TDSC*, vol. 11, no. 1, pp. 30–44, 2014.

[100] Y. Cheng, J. Deng, J. Li, S. DeLoach, A. Singhal, and X. Ou, "Metrics of security," in *Cyber Defense and Situational Awareness*, pp. 263–295, 2014.

[101] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues, "Model-based quantitative network security metrics: A survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2704–2734, 2017.

[102] S. Noel, , and S. Jajodia, *A Suite of Metrics for Network Attack Graph Analytics*, pp. 141–176. Springer International Publishing, 2017.

[103] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: A security metric for evaluating the resilience of networks against zero-day attacks," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 5, pp. 1071–1086, 2016.

[104] J. Cho, P. Hurley, and S. Xu, "Metrics and measurement of trustworthy systems," in *Proc. IEEE MILCOM*, 2016.

[105] L. Wang, S. Jajodia, and A. Singhal, *Network Security Metrics*. Springer, 2017.

[106] H. Chen, J. Cho, and S. Xu, "Quantifying the security effectiveness of firewalls and dmzs," in *Proc. HoTSoS'2018*, pp. 9:1–9:11, 2018.

[107] H. Chen, J. Cho, and S. Xu, "Quantifying the security effectiveness of network diversity," in *Proc. HoTSoS'2018*, p. 24:1, 2018.

[108] H. Chen, H. Cam, and S. Xu, "Quantifying cybersecurity effectiveness of dynamic network diversity," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[109] Y. Han, W. Lu, and S. Xu, "Preventive and reactive cyber defense dynamics with ergodic time-dependent parameters is globally attractive," *IEEE TNSE*, vol. 8, no. 3, pp. 2517–2532, 2021.

[110] Z. Lin, W. Lu, and S. Xu, "Unified preventive and reactive cyber defense dynamics is still globally convergent," *IEEE/ACM ToN*, vol. 27, no. 3, pp. 1098–1111, 2019.

[111] R. Zheng, W. Lu, and S. Xu, "Preventive and reactive cyber defense dynamics is globally stable," *IEEE TNSE*, vol. 5, no. 2, pp. 156–170, 2018.

[112] Y. Han, W. Lu, and S. Xu, "Characterizing the power of moving target defense via cyber epidemic dynamics," in *HotSoS*, pp. 1–12, 2014.

[113] X. Li, P. Parker, and S. Xu, "A stochastic model for quantitative security analyses of networked systems," *IEEE TDSC*, vol. 8, no. 1, pp. 28–43, 2011.

[114] M. Xu and S. Xu, "An extended stochastic model for quantitative security analysis of networked systems," *Internet Mathematics*, vol. 8, no. 3, pp. 288–320, 2012.

[115] M. Xu, G. Da, and S. Xu, "Cyber epidemic models with dependences," *Internet Mathematics*, vol. 11, no. 1, pp. 62–92, 2015.

[116] S. Xu, W. Lu, and L. Xu, "Push- and pull-based epidemic spreading in networks: Thresholds and deeper insights," *ACM TAAS*, vol. 7, no. 3, 2012.

[117] S. Xu, W. Lu, and Z. Zhan, "A stochastic model of multivirus dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 30–45, 2012.

[118] S. Xu, W. Lu, and H. Li, "A stochastic model of active cyber defense dynamics," *Internet Mathematics*, vol. 11, no. 1, pp. 23–61, 2015.

[119] R. Zheng, W. Lu, and S. Xu, "Active cyber defense dynamics exhibiting rich phenomena," in *Proc. HotSoS*, 2015.

VITA

Eric Ficke is from San Antonio, TX. He earned a B.S. and Ph.D. in Computer Science from The University of Texas at San Antonio in 2017 and 2022, respectively. His research focuses on cybersecurity, with significant use of algorithms analysis and metrics. Outside of his immediate research, he has a broad range of academic interests, including Psychology, Philosophy and Darwinism.

Eric has received awards from UTSA, AFCEA and RSA, including the UTSA Presidential Distinguished Research Fellowship, UTSA Top Scholar Program, RSA Scholar and AFCEA Cyber Security Scholarship. He is an AFCEA life member and an IEEE Member. He held a Security+ from 2014-2017. He earned the Eagle Scout award in 2010.