INFORMATION-FUSION-BASED METHODS TO IMPROVE THE DETECTION OF ADVANCED CYBER THREATS

BY

ATUL BOHARA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor William H. Sanders, Chair
Professor Klara Nahrstedt
Associate Professor Matthew Caesar
Dr. Jordi Ros-Giralt, Reservoir Labs

ABSTRACT

The increasing adoption of information and communication technologies in every aspect of modern life has made the security of networked systems more crucial. Their growing size and complexity have provided adversaries with a larger attack surface leading to numerous breaches in recent years that have undermined the confidentiality and availability of such systems. Thus, it is essential to improve security solutions to protect systems against malicious threats.

Intrusion detection is an essential strategy that, together with intrusion prevention and response, make systems more resilient against malicious access. The challenges that are faced while developing intrusion detection systems (IDSes) are manifold. First, the malicious actors are continuously revising their tactics in using the victim's infrastructure against itself. Next, to address the threats, organizations need to employ many layers of security products. The information generated by these products poses significant technical and processing overheads. Finally, security systems need to adapt according to the nature of the target network and constraints of the services delivered.

In this dissertation, we improve the detection of advanced cyber threats that use intelligent planning and persistent actions in compromising large networked systems. In particular, we design and implement threat detection methods that utilize information fusion. Information fusion guides the analysis and incremental refinement of monitoring information to obtain more accurate detections and smaller volumes of alerts.

The general framework of the presented methods is as follows. We collect security monitoring information by using a range of host- and network-level monitors. We then refine that monitoring information by identifying and extracting useful features. The features then drive anomaly-based and specification-based detection of attacks to provide alerts and improve visibility into the target network.

We develop techniques that apply to general networked systems. However, to make the discussion concrete and reason about our design decisions, we have adopted two types of target systems: an enterprise IT network and a power grid substation network. These systems offer different types of architectures and security requirements, encompassing a wide variety of networked systems. Nevertheless, the possible types of attacks are similar.

We set out to detect vectors of initial compromises, such as network scans, network-layer distributed denial-of-service, and malware presence on hosts. In our approach, we combine the host-level context, which is captured by monitors such as system logging deployed on

individual hosts, with the network-level context captured by monitors such as firewalls, and we use the aggregated profile in detecting anomalous behavior. The detection of abnormal behavior uses unsupervised cluster analysis. We devise a method to order the anomalous clusters in terms of their likely maliciousness, which can help a security administrator prioritize the clusters to investigate manually. Our experiments using an enterprise network dataset demonstrate that our approach has higher accuracy of detection than any individual monitors alone. Further, our completely unsupervised approach detects more attacks and generates a smaller volume of alerts than a state-of-the-art rule-based IDS, Snort.

We then introduce a novel technique to detect malicious lateral movement (LM). The LM attackers use the already compromised entities (e.g., hosts, accounts, and services) as stepping stones for reaching critical segments. The process of expansion usually happens in conjunction with command and control (C&C) to gather internal system structure information and carry out a damaging action. To effectively detect such attacks, we first build a graph-based model to represent the current state of the network. Guided by the model, we identify the essential features of C&C and LM activities and extract them from internal and external communication traffic. Driven by the analysis of the features, we propose to use an ensemble of multiple anomaly detection techniques to identify compromised hosts. Our experiments using enterprise network traces show that our approach can detect the attacks with high accuracy and a low false-alarm rate even when the attacker's behavior is similar to benign behavior.

We then study the advanced attack detection when it is in the last stage before launching a harmful action. With false data injection on IEC 61850-compliant substations as our use case, we design and implement a system to detect the attack within the strict timing constraints. We first develop an algorithm to identify poisoning attacks on GOOSE protocol. The algorithm performs a highly-stateful analysis of traffic to reason about ongoing communication's properties in the context of protocol specifications. We then use a novel combination of whitelisting, specification-based analysis, and physical behavior attributes to detect with high accuracy a broad class of false data injection attacks. Our experiments using substation network traces show that the system can identify attacker-injected messages even if they resemble benign communication patterns. We discuss software and hardware bottlenecks, devise a systematic approach to improve our IDS's performance, and demonstrate its applicability to high-speed protection-related communication.

To my mother and wife for their love and support.

ACKNOWLEDGMENTS

I want to begin by thanking my advisor, Professor William H. Sanders, for his consistent support throughout my Ph.D. journey. In addition to advising my research, he supported me during the good and bad times of my life during these years. His strong work ethic and broad scientific vision have been a great source of inspiration. I consider myself very fortunate to work with him.

In addition to my advisor, I would like to thank my other committee members, Professor Klara Nahrstedt and Professor Matthew Caesar, for their insightful feedback and guidance on my dissertation work. I am sincerely honored to have them on my doctoral committee. I would also like to thank my collaborator at Reservoir Labs, also a member of my Ph.D. committee, Dr. Jordi Ros-Giralt, for his detailed feedback and hands-on guidance. My internships with Jordi and work with him on technical papers were great learning experiences.

I would like to thank the current and former members of the performability engineering research group (PERFORM) and Information Trust Institute. I want to thank Ahmed Fawaz, Uttam Thakore, Carmen Cheh, and Mohammad Noureddine for being wonderful friends and collaborators. I am thankful to Jenny Applequist for the many long hours of work she has graciously spent helping me with my writing. I would like to thank Varun, Gabe, Michael, Ben, Ken, Brett, Ron, Ryan, and David for the lively discussions around the office. I am grateful to Alfonso Valdes and Ghada Elbez, with whom I collaborated on the work included in this dissertation.

I want to express my deepest indebtedness to my family. I am thankful to have been raised among such a loving and caring family. My mother encouraged me to learn, inspired me to work harder, and showed me the right way by doing it herself. I am proud of my mother. I am also thankful to my grandparents and my father, who have sacrificed a lot for our education and well-being. I thank my wife Nikita for her unconditional love and constant support. She has always placed my work and happiness before everything else. We are fortunate to have our daughter Anika who has made the last two years of my Ph.D. a lot more memorable. I thank my brothers and sister for their encouragement throughout my life and career. My brother Alok always takes the first step and makes it easy for me to follow. My brother Ankit and sister Sushmita are kind, all-around caring, and ingenious researchers, who have supported me unconditionally.

I extend my warmest thanks to my friends and colleagues at the Coordinated Science Lab, including Subho, Arjun, Saurabh, Krishnakant, and Yoga, who made my time more

productive and enjoyable. My friends in Urbana, Shubhanshu & Shivangi, Ravi & Harshala, Himanshu, and Ram Sai ensured I was never alone in this graduate school journey. Finally, I would like to offer my gratitude to my friends from back home, Dev, Deepika, Abhishek, Vikash, and Sheesh, for their thoughts and phone calls, and for being there when I visited.

FUNDING SOURCES AND COPYRIGHT

- Chapter 2 is based on work published in A. Bohara, U. Thakore, and W. H. Sanders. "Intrusion detection in enterprise systems by combining and clustering diverse monitor data." In Proceedings of the Symposium and Bootcamp on the Science of Security (HoTSoS '16). Association for Computing Machinery, New York, NY, USA, 7–16. DOI: https://doi.org/10.1145/2898375.2898400.

- The following copyright notice adheres to Chapter 3, "©2017 IEEE. Reprinted, with permission, from A. Bohara, M. A. Noureddine, A. Fawaz, and W. H. Sanders. "An unsupervised multi-detector approach for identifying malicious lateral movement." In Proceedings of the 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS '17), Sept. 2017, 224–233. DOI: https://doi.org/10.1109/SRDS.2017.31.

- The material in Chapters 2 and 3 is based in part on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084.

- The material in Chapters 4 and 5 is based upon work supported by the Department of Energy's Office of Cybersecurity, Energy Security, and Emergency Response and the Department of Homeland Security's Security Science & Technology Directorate under Award Number DE-OE0000780.

DISCLAIMERS

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The current worldwide increase in reliance on *online services* has dramatically increased the importance of networked systems. Such systems include enterprise networks, cloud data centers, and industrial control systems. As the dependence on those systems grows, so does the risk of malicious access to them. Over the past decade, numerous cyber attacks have undermined networked systems' security for financial or political gains [1, 2, 3, 4, 5].

A common strategy of adversaries is to carry out large-scale breaches by means of persistent, targeted campaigns. Malicious actors enter victim systems by evading existing prevention and detection systems. Afterward, these actors establish a presence and persistence in the network by compromising multiple entities. The process of expansion usually happens in conjunction with command and control communications to gather internal system structure information, exfiltrate sensitive data, and ultimately carry out damaging actions [6, 7, 8].

Since the attack vectors are endless, and security solutions are imperfect, intrusion prevention systems alone are incapable of protecting such systems. Consequently, massive breaches continue to undermine the confidentiality and availability of enterprise networks [1, 2, 3] and safety-critical networks [4, 5, 9]. Therefore, system security needs intrusion detection as an additional layer of defense to address the attacks that defy the prevention systems.

Their increased size and complexity has made the protection of networked systems more challenging. In particular, first, organizations typically employ a wide variety of security tools that generate highly verbose logs and require significant resources to process the information. Next, the high levels of interconnectedness make it challenging to identify the spread of a successful compromise within the internal network. Finally, the malicious subjects are continuously evolving by increasing their sophistication in using the victim system's infrastructure against itself. In that context, the adoption of the standard OS and networking services as the threat vectors requires continuous improvements in attack detection solutions.

This dissertation focuses on detecting advanced cyber threats that use intelligent planning and persistent actions in compromising large networked systems. We design and implement intrusion detection methods to address different stages of an advanced attack. The methods use concepts from *information fusion* to address the challenges associated with the detection of such attacks. They analyze monitoring information to incrementally refine it and obtain improved decisions, i.e., more accurate detections, smaller volumes of alerts, and better support for constraints specific to target platforms.

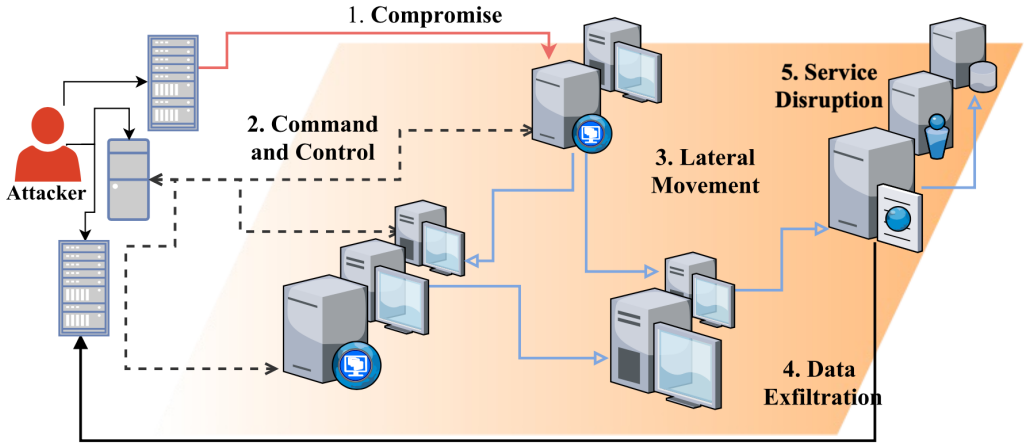


Figure 1.1: Overview of advanced cyber threats. We abstract the steps into five high-level stages.

1.1 ADVANCED CYBER THREATS AND CHALLENGES

We consider network-based multi-stage attacks by external actors. We refer to these attacks as *advanced cyber threats*. Such attacks are popularly known as *advanced persistent threats (APTs)*. Typically, such attacks are intended to steal proprietary intellectual property or disturb mission-critical services. Although they infect fewer entities than mass malware (e.g., botnets, worms) do, the estimated cost of data loss due to APT incidents is significantly higher. For example, the expected losses incurred in the data breaches at Target Corporation and Anthem Inc. were $248 million and $115 million, respectively [1, 10]. Microsoft Advanced Threat Analytics estimated an average cost of $3.8 million per data breach in 2014 [11]. The cyber attack on the Ukrainian power grid in 2015–16 left about 225,000 consumers without power for about six hours at enormous cost [4].

As shown in Figure 1.1, we abstract the lifecycle of an advanced cyber threat into the following stages: (i) initial compromise, (ii) command and control, (iii) lateral movement, (iv) data exfiltration, and (v) service disruption. These stages are essential parts of several models of the lifecycle of an APT [7, 12, 13] and other previously studied incidents [1, 8, 14].

Initial compromise refers to the first incursion into a target network. It involves external reconnaissance, delivery of malware through either vulnerability exploits or social engineering, and establishing of backdoor communication.

Subsequently, the attacker establishes *command and control* (C&C) channels between the victim machines and a set of external servers, to maintain active control over the compromised hosts. During C&C, a collection of compromised devices send automated periodic beacons to attacker-controlled servers, awaiting future instructions. The commands are in-

tended to compromise other hosts in the network or to gather intelligence and sensitive data. The Internet Relay Channel has traditionally been the preferred communication tool for C&C, but attackers have started opting for more commonly used protocols, such as hypertext transfer protocol secure (HTTPS), file transfer protocol (FTP), and secure shell (SSH), to evade easy detection. For a more robust communication infrastructure, attackers may use dynamic domains and P2P server architectures.

In conjunction with C&C, the attacker attempts to move laterally to widen the compromised region. That movement may be advanced through various tactics, including internal reconnaissance, credential stealing, vulnerability exploitation, and privilege escalation. The exploit techniques that are used to achieve such objectives are ever-changing and evolving. Examples include utilization of application deployment software to deploy malware, exploitation of a vulnerability to escalate privileges, and use of pass-the-hash to bypass standard authentication steps [6].

The last step includes the actual execution of malicious actions, whether they perform data exfiltration, service disruption, or physical damage.

In this dissertation, we address the problem of detecting such attacks with high accuracy and a low false-alarm rate and generating alerts with information that will enable further actions to limit damage.

1.1.1 Challenges

In this section, we discuss the challenges in detecting advanced cyber threats on large networked systems. We also briefly mention how we address them, which we will discuss in detail in the following chapters.

High Verbosity of Monitoring

Security monitoring information is high-volume (i.e., a large amount of data), high-velocity (i.e., high speed of data generation), and high-variety (i.e., a wide range of data sources and types). Volume, velocity, and variety are known as the *3 V's of big data* [15]. These 3 V's of big data apply perfectly to the security monitoring that forms the basis of intrusion detection. Therefore, the development of intrusion detection approaches for large networked systems faces the challenges that a typical big data analysis task does. In particular, the highly verbose logs require significant computing resources and security administrator person-hours. Without some refinement methods that can produce concise, actionable insights, the resources can be overwhelmed. That can lead to high operational costs and missed alerts, as

seen in past incidents [1, 16]. In this dissertation, we design an unsupervised cluster analysis approach that ingests highly verbose monitor data and generates concise descriptions of any attacks present in the system.

Limited Awareness of the Network State

Because the advanced cyber threats involve lateral movement, it is challenging to identify appropriate countermeasures, even after the detection methods have discovered an ongoing attack. For example, if a set of computers are infected by an attack at a given time, what other systems and services should be considered vulnerable? How does the current state affect the critical services running in the network? These types of questions are challenging to answer since the networked systems could be very densely connected. Standard network services such as single sign-on and network file shares can introduce nontrivial reachabilities, making it more challenging to identify the region of spread. In this work, we design a simple graph-based system model that allows analysis of the current security posture of a network for lateral movement activities and thus enables effective identification of compromised regions.

Attack Characteristics

New attack techniques are invented very frequently, so malicious actors often succeed in evading traditional signature- or anomaly-based detection systems. Attackers can stay undetected over prolonged periods by using conventional operating system and networking services as their attack vectors [7, 8, 14]. For example, C&C communications can happen over HTTP(s) channels, bypassing firewalls. Malicious lateral movement can make use of services like Remote Desktop Protocol and Windows Update while avoiding the generation of intrusion detection system alarms. Although there are models that provide a high-level understanding of the progression of such attacks [6], the attackers are not bound to follow any particular model in compromising a target system [14, 17]. There is therefore an ongoing demand for research on novel ways to define and detect anomalies concerning advanced cyber threats. In this dissertation, we design an attack-vector-independent method that correlates diverse indicators to identify malicious lateral movement and thus enable the protection of the system when the attack is still early in its lifecycle.

Overall, the detection of advanced cyber threats is challenging. We hypothesize that we can counter the challenges by using information fusion to improve situational awareness and detection techniques. In this dissertation, we evaluate that hypothesis. To introduce the methods that we develop in this work, we first discuss the relevant concepts related to

information fusion and present a high-level architecture of our approach.

1.2 INFORMATION FUSION FOR INTRUSION DETECTION

Information fusion can be defined as a combination of information from multiple sources to obtain improved information and more specific inferences than could be achieved by the use of a single source. In the context of intrusion detection, improved information means alerts of higher accuracy, more relevance, and less volume [18, 19]. The application of information fusion to intrusion detection is grounded in various mathematical and heuristic techniques from multiple domains, including statistics, artificial intelligence, digital signal processing, operations research, and information theory [20, 21].

Research on fusion techniques has roots in defense applications such as battlefield surveillance. Traditionally, those applications have used data fusion (sometimes known as *sensor fusion*) to obtain higher reliability and fewer detection errors in multisensor environments. For instance, the JDL[1] data fusion model and Boyd's OODA control loop [22] stand as reference frameworks for such research. (OODA stands for "Object, Orient, Decide, and Act.") Researchers have revised those frameworks to pull them out of the defense context and make them useful for other application domains [23, 24].

Although the terms *information fusion* and *data fusion* are used interchangeably in the literature, we make a subtle distinction between them. We consider *data fusion* to refer to the techniques that deal with raw sensor data (e.g., field measurements). In contrast, *information fusion* involves combining and refining relatively more processed information (e.g., network logs and cyber security information).

A conceptual framework for information fusion in intrusion detection was developed by T. Bass [25]. His general scheme aims to provide a representation of computer system events and security threats at the level of human understanding. The framework is structured in levels ranging from 0 through 4, which incrementally convert the data to information and eventually to knowledge. The levels are Data Refinement, Object Refinement, Situation Refinement, Threat Assessment, and Resource Management. In the context of that framework, we can describe the existing research in this field in terms of the following two categories:

1. Data and object refinement approaches aim to remove unwanted data, convert the available information to more concise formats, and improve the performance of intru-

[1] JDL (Joint Directors of Laboratories), a US Department of Defense government committee overseeing US defense technology R&D; the original data fusion model was created by the Data Fusion Group of the JDL in 1991.

sion detection systems. For example, [26] performs information reduction by automatically correlating different indicators of the same event. Similarly, network traffic aggregation [27] and controlled packet dropping [28] are effective fusion techniques in improving the performance of traffic anomaly detection.

2. Situation refinement and threat assessment methods use information fusion to detect intrusions and analyze results. Examples of this class are (1) classifier combination methods that utilize an ensemble of classifiers to improve accuracy [29, 30, 31, 32], and (2) alert correlation techniques that aim to provide a higher-level description of the attacks by correlating outputs of different detectors [33, 34, 35, 36].

Intrusion detection can be considered a pattern recognition task, that involves data acquisition, data preprocessing, feature selection, model selection, classification, and result analysis. That pipeline of stages fits perfectly with the continuous refinement strategy of information fusion. Therefore, information fusion applies well to cyber threat detection. More specifically, our insight that information fusion can improve intrusion detection relies on the following observations.

Multiple Sources

Advanced cyber threats are intelligent and coordinated. They can be launched from either outside or inside, and usually spread over multiple network domains. They can affect several systems such as user workstations and servers, and access a variety of information, such as user accounts, network services, and internal databases. Therefore, to detect indicators of such attacks, the defense mechanism needs to utilize information from multiple sources. Such information might include vulnerability databases, network operation records, host-level events, and network service logs. Thus fusion can strategically refine the widespread information and extract the most relevant attributes for detection.

Diverse Formats

The information generated at multiple sources is bound to have diverse formats. Networks are inherently designed in a hierarchical architecture in which large systems are divided into smaller components. For instance, the TCP/IP stack, which is the most widely used protocol suite in data networks, is implemented in a layered structure, i.e., physical, data-link, network, transport, and application layers. Similar hierarchies are present in the physical architectures of hosts and networks. Therefore, the diverse information obtained from

those elements is usually at different levels of abstraction. As an example, information from host-level and network-level tools will have different abstraction levels. We therefore need techniques for mining correlated patterns from the diverse logs.

Human Level of Representation

Even though modern systems are increasingly adopting automated intrusion response, human operators are still crucial throughout the process. The primary reason is that since cyber attacks are carried out by humans, it is easiest to comprehend detection-related knowledge by using human intelligence. Hence, low-level events and anomalies should be correlated to produce higher-level patterns, making information fusion a natural choice.

In this dissertation, we develop methods that mainly map to situation refinement and threat assessment levels of the information fusion framework. Next, we present the architectural considerations, followed by details about how our approaches work.

1.2.1 Architecture

We present a high-level architecture of the information-fusion-based intrusion detection approach in Figure 1.2. The architecture shows the incremental refinement of information at multiple levels. It starts with the monitors at the lowest level, which are deployed in a system at different locations. A *monitor* is a host-level or network-level tool that observes specific system events and generates log records. We then use that information to represent the current state of the network with a system model. The model's development depends upon the detection task at hand. For example, different models are built to detect initial compromise or to detect lateral movement activities. Next, we carry out feature extraction and selection, which are guided by the system model. In the process, we identify, extract, and select relevant attributes from the monitoring information. Finally, at the highest level, we perform threat assessment and result analysis. In particular, anomaly-based and specification-based attack detection methods are used at this level.

Monitoring Information

Monitoring the operation of a system is essential in detecting intrusions. Information collected by the monitors can be used to estimate the current state and inform the models that identify malicious events. In this dissertation, we develop data-driven techniques that

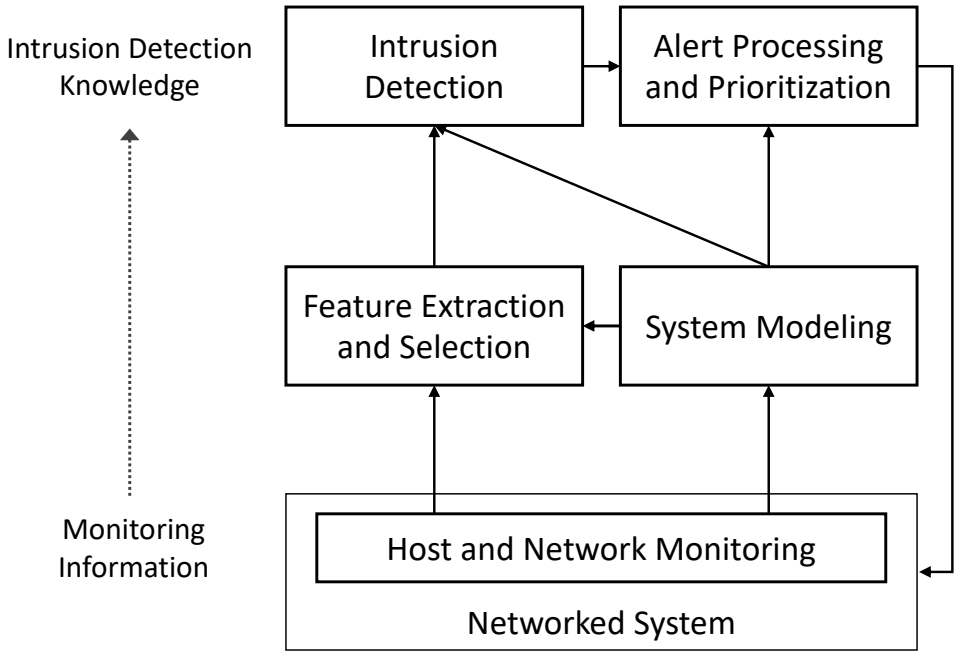


Figure 1.2: The general scheme of our information-fusion-based intrusion detection approach.

rely on empirical models. As we address the problem of detecting multiple indicators of attacks, we need to utilize a variety of information to construct such models.

Monitors can broadly be classified into two categories: network-based and host-based. Network-based monitors have the advantage of having a wide field of vision. However, such monitors, specifically intrusion detection systems (IDSes), are prone to evasion and ambiguity attacks. *Evasion* involves bypassing of a security device and delivery of an exploit to the target network, without detection. *Ambiguity* refers to situations in which the network monitor cannot determine whether a network packet will be accepted. Those situations can arise because of varying end-system behavior and underlying network uncertainties. Host-based monitors, on the other hand, do not face ambiguity problems. They impose significant performance overhead on the host, degrading the host's ability to perform necessary business functions [37]. Minimizing of the host-level overhead is critical in general, but resource-constrained environments, such as substation networks, limit what can be deployed on end devices even more strictly.

The methods that we develop rely on the insight that host-based and network-based monitor data can be used together to provide additional context for improved intrusion detection. Since the advanced threats use techniques whose behavior may seem normal to an individual monitor, an efficient way to detect these attacks is to look at the pattern of events across different monitors.

In particular, the systems presented in this thesis use the following types of information:

1. Host-level activity: Records of host-level activity provide the *local* view. We use authentication logs and kernel-level process information (e.g., [38]) from each host.

2. Network operation logs: Network operations reveal a wider view of the system, including network architecture and user interactions. We analyze the communications between users and devices captured in firewall logs, network authentication logs, and logs generated by the Zeek network security monitor [39] while analyzing live traffic.

3. Protocol specifications: Network protocol specifications define how the entities in a system should interact. Attacks are, however, possible within the specified behaviors. Nevertheless, it is valuable to understand the protocol behavior and develop models to analyze anomalies. In particular, in our approach to protecting electrical substation networks, we use the specifications of the generic object-oriented substation events protocol [40].

Deployment Considerations

Although the presented techniques ingest information that originated at multiple levels of abstraction, we deploy the resulting solutions at the network level. Examples of such deployment locations are a software-defined networking (SDN) controller, an internal gateway, or a firewall. We adopt a network-level deployment as opposed to a host-level deployment for the following reasons. First, since external actors launch the majority of attacks over the network, a network-level implementation would provide the capability to detect and stop the attacks before they infect host machines. Second, we can achieve a lower overall resource overhead by deploying a detection system at one central location instead of at every device in the network. Finally, the network-level deployment requires fewer changes to the existing infrastructure than host-level tools would. The last two considerations are particularly important in industrial control networks, e.g., electric substations in which the host machines are special-purpose devices with limited software and hardware capacity.

In addition, our detection methods are designed to work out-of-band such that they do not interrupt or block the ongoing communication. Among the methods, the approach to detecting false data injection attacks on electric substation networks also works in real-time in order to meet the strict timing requirements of control data exchanges.

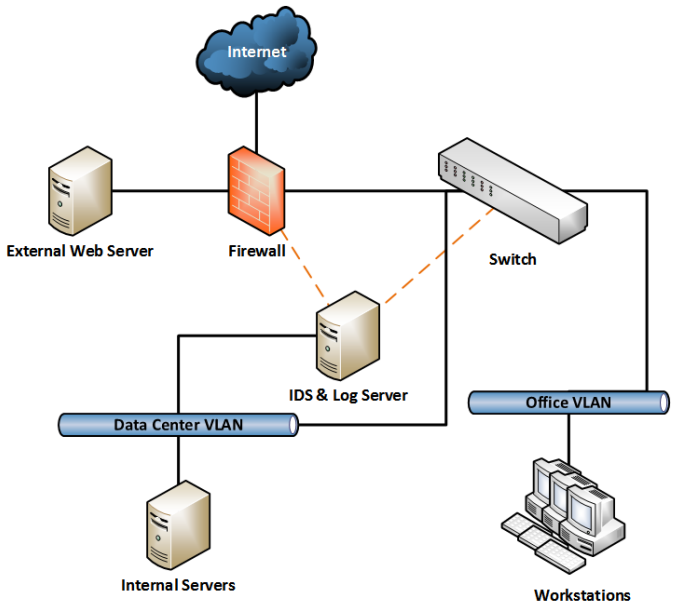


Figure 1.3: An example architecture of an enterprise network.

1.2.2 Networked System Case Studies

In this dissertation, we aim to develop techniques that apply to general networked systems. However, to make the discussion concrete and help us reason about the adopted design decisions, we work with two specific systems: an enterprise network and a smart grid control network. Together, they represent a wide variety of networked systems. Although they offer different types of architectures and security requirements, the kinds of attacks that are possible are similar for the two.

Enterprise Network

An enterprise network corresponds to the IT infrastructure used by corporate entities to deliver their services over the Internet. Physically, such a network is composed of workstations, servers, and networking elements, as shown in Figure 1.3. Logically, the network is divided into domains by using virtual local area networks (VLANs), which roughly reflect the corporate departments. Users with different roles and rights interact with the infrastructure by using network services and communication.

The goals of the security assurance process in enterprise settings include the continuous availability of the service and the confidentiality of both corporate secrets and consumer records. Accordingly, security threats primarily aim to undermine availability and confidentiality. A successful compromise can lead to both monetary and social damage.

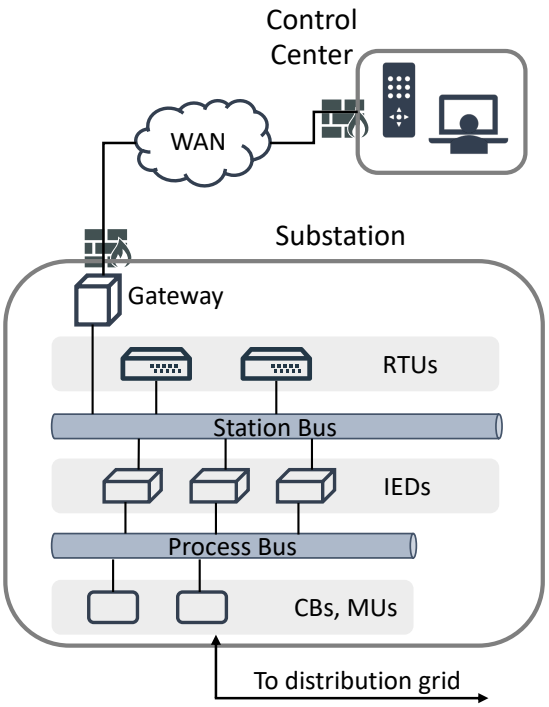


Figure 1.4: An example architecture of a power distribution substation network.

In this work, we evaluate the intrusion detection methods with enterprise network data sets. In particular, we use host-level system logs and network-level firewall and IDS logs to examine the efficacy of initial compromise detection. Next, we use network flow records that represent both internal and external communication and network authentication logs to develop and evaluate the lateral movement detection mechanism.

Power Grid Substation Network

The second networked system studied in this dissertation corresponds to the cyber infrastructure used to monitor and control electric energy delivery. In particular, we work with a substation network that uses communication among intelligent electronic devices (IEDs) to ensure safe and continuous electricity delivery.

Figure 1.4 shows a high-level architecture of a typical substation network. The IEDs interact with power grid sensors and actuators, such as circuit breakers (CBs) and measurement units (MUs), to analyze the current state of the system and send commands to keep the system in a safe state. Communication is achieved by dividing the local area network into the station bus and a process bus. Remote telemetry units (RTUs) enable the exchange of local control-related data with a remote control center over the wide area network (WAN).

The gateway consists of access control and remote access functions.

Substation networks differ from enterprise systems in some key respects that need specific attention. First, the services provided by substation networks are time-critical because they control the delivery of electricity. Second, communication is mainly device-to-device and follows a more defined model than that in enterprise networks. Finally, the networking stack consists of some protocols that are specific to the power grid. In this dissertation, we develop a method for network-level detection of false data injection attacks, keeping in mind the timing requirements of substation networks.

1.3 DISSERTATION CONTRIBUTIONS

It is our thesis that *by utilizing information from multiple sources, data-driven methods can improve the ability of a network to detect sophisticated attacks.*

Our specific contributions in this dissertation are the following:

- We developed an intrusion detection technique that uses unsupervised clustering algorithms to identify malicious behavior within large volumes of diverse security monitor data.
- To enable effective detection of lateral movement attacks, we introduced a new approach based on graph-based modeling of the network's security state and correlation of diverse indicators of anomalous host behavior.
- For the protection of power grid substations against false data injection attacks, we developed a specification-based system to detect the attacks within communication's stringent time constraints.
- We extended the coverage of false data injection detector by designing and implementing a novel combination of whitelisting, specification-based analysis, and physical behavior attributes.

The contributions of this dissertation are summarized as follows.

1.3.1 Initial Compromise Detection

It is crucial to detect attacks before they establish a widespread presence in a network. We have developed a novel approach to detecting, with high accuracy, common vectors for initial compromise, including flooding-based network attacks and the presence of malware

that alters host behavior. First, we created a simple method for joining together information generated by security monitors with diverse formats. We then extracted a set of features from network-level and host-level security logs that aid in detecting malicious host behavior and flooding-based network attacks. We the applied clustering algorithms to the separate and joined logs and used statistical tools to identify anomalous behaviors captured by the logs. Finally, we evaluated the approach on an enterprise network dataset. Our method correctly identifies and prioritizes abnormal behaviors in the logs by their likelihood of maliciousness. By combining network and host logs, we are able to detect malicious behavior that cannot be detected by either log alone.

1.3.2 Malicious Lateral Movement Detection

Attacks that can successfully infiltrate a network attempt to achieve persistence by using lateral spread and privilege escalation. We found that the operations used to obtain greater persistence generate noisy traces as the attacker jumps around in the network. However, since that movement happens mainly by using benign services, traditional intrusion detection systems do not generate alerts. In fact, the behavior might resemble a network administrator's actions, further complicating the detection of malicious activities.

To address those challenges, we present an unsupervised anomaly detection approach to identifying malicious lateral movement early in its lifecycle, even when benign lateral movement, (e.g., admin tasks) are also present. The method relies on graph-based modeling of the target system's security state and the correlation of different indicators of anomalous host behavior. First, we identify important features of command-and-control and lateral movement activities and extract them from internal and external communication traffic. Driven by the analysis of the features, we use multiple anomaly detection techniques to identify compromised hosts. The methods include principal component analysis, k-means clustering, and median absolute deviation-based outlier detection. We performed experiments using an enterprise network dataset and injected traces of attacks to demonstrate the high accuracy of attack detection.

1.3.3 Detection of False Data Injection within Strict Timing Requirements

Next, we addressed the detection of an attack in the final stage of its lifecycle, i.e., service disruption. In particular, we present a case study in protection of International Electrotechnical Commission (IEC) 61850-compliant substations. A substation is considered the edge of the power grid control network, and an adversary can carry out actions there to cause a

power outage or equipment damage. Many successful attacks on such systems have shown that field devices often execute malicious commands without verifying the issuer or the context [4, 41]. The lack of monitoring and control in the cyber infrastructure was a primary enabler of those attacks. Since the end devices in these systems are special-purpose, low-resource machines, a lot of cyber security and monitoring needs to happen at the network level.

Modern substations use the generic object-oriented substation events (GOOSE) protocol for the high-speed exchange of protection-related events. An adversary can inject carefully crafted GOOSE messages to impact the grid's availability. Such attacks are called *GOOSE poisoning attacks*. We present algorithm, implementation, and architectural considerations for efficient and practical detection of such attacks. Our approach relies on protocol specifications to identify violations concurrently to the data transmission. We evaluated its accuracy in detecting attacks by using a synthesized dataset that represents a substation network. Finally, we provide a systematic approach to assessing bottlenecks, improving performance, and demonstrating that the presented system has low overhead and meets GOOSE's timing constraints.

1.3.4 Extending the Coverage of the Detection of False Data Injection

Having addressed the GOOSE poisoning attacks, next, we studied detection for more comprehensive false data injection on GOOSE publications. In particular, we developed a system to cover the detection of attacks that, in addition to injecting malicious traffic, can tamper with the in-transit messages. The attacks within that threat model include access-control violations, protocol semantic violations, GOOSE poisoning, and GOOSE data manipulations.

Our system relies on a coordinated analysis of whitelisting, protocol semantics, specifications, and physical-behavior attributes to identify such advanced attacks with high accuracy. Besides, it does not generate false positives on benign power-system disturbances. We implemented the system by using the performance improvement techniques mentioned above. The system could achieve a response time suitable for online analysis of the most critical GOOSE messages, which is an improvement over the existing state-of-the-art solutions.

1.4 DISSERTATION ORGANIZATION

The rest of this dissertation is organized as follows. Chapter 2 addresses the detection of attacks before they establish a widespread presence. Chapter 3 showcases a multi-detector

approach for identifying the malicious lateral movement of an attack. Chapter 4 presents our approach to detecting malicious data injection on electric substation networks, focusing on high-speed communication requirements. Chapter 5 details an extension we made to the false data injection detector in order to improve its coverage of attack detection. Chapter 6 concludes the dissertation with a discussion of the generality of the presented approaches, lessons learned, and potential future work. We include some additional work on the dissertation topics that are not directly relevant to the research objectives in the Appendix.

CHAPTER 2: INITIAL COMPROMISE DETECTION

Intrusion detection using multiple security devices has received much attention recently. The large volume of information generated by these tools, however, increases the burden on both computing resources and security administrators. Moreover, attack detection does not improve as expected if these tools work without any coordination.

In this chapter, we present a simple method to join information generated by security monitors with diverse data formats. We develop a novel intrusion detection technique that uses unsupervised clustering algorithms to identify malicious behavior within large volumes of diverse security monitor data. First, we extract a set of features from network-level and host-level security logs that aid in detecting malicious host behavior and flooding-based network attacks in an enterprise network system. We then apply clustering algorithms to the separate and joined logs and use statistical tools to identify anomalous usage behaviors captured by the logs. We evaluate our approach on an enterprise network data set, which contains network and host activity logs. Our approach correctly identifies and prioritizes anomalous behaviors in the logs by their likelihood of maliciousness. By combining network and host logs, we are able to detect malicious behavior that cannot be detected by either log alone.

2.1 SUMMARY OF CONTRIBUTIONS

Securing large computer networks and distributed systems is becoming increasingly challenging because of the growing scale and complexity of these systems. Security breaches continue to occur, despite the use of several layers of protection. Such compromises result in unauthorized access to sensitive information and misuse of computing resources, leading to significant losses to organizations, both financially and socially [42, 43, 44]. Statistics published in Symantec's 2015 Internet Security Threat Report indicate a significant increase in both vulnerabilities and attack attempts over the last year [45]. There is ongoing demand for research on better intrusion detection and analysis methods.

Manifold problems make the task of securing a large networked system very difficult. New attack techniques are invented very frequently. Most of the sophisticated attack techniques try to resemble normal behavior very closely, and thus are very difficult to detect. The large scale and high complexity of large enterprise systems make intrusion detection even more challenging. Furthermore, monitoring system-wide activities for the purpose of intrusion detection results in volumes of diverse monitor data that easily overwhelm security experts

and online intrusion detection systems [16]. As a result, in many cases, intrusions are detected long after significant losses have already been incurred.

In this work, we try to address some of the aforementioned problems and take a significant step towards improving intrusion detection. We present an approach to detect intrusions by identifying patterns of anomalous usage of an enterprise system based on the data captured in diverse monitors deployed in the system. We do not rely on any labeling of monitoring data, and instead use completely unsupervised machine learning techniques.

Recent research has shown that host-based and network-based monitor data can be used together to provide additional context for improved intrusion detection [25, 37, 46, 47]. Many modern attacks use techniques that seem normal to individual monitors. One of the effective ways to detect these attacks is to look at the combined pattern of events across different monitors.

In our approach, we combine the host-level context, which is captured by monitors such as system logs that are deployed on individual hosts, with the network-level context, which is captured by network-level monitors such as firewall and network intrusion detection systems, and we use the aggregated profile in detecting anomalous behavior.

First, we introduce the Visual Analytics Science and Technology (VAST) 2011 Mini Challenge 2 dataset, which contains network-level and host-level logs and a set of injected attacks. We present a threat model that describes a set of attack types that can be detected using anomaly detection algorithms and encompasses many of the attacks in our dataset. We then extract meaningful features from the log data, and use the features to combine the network-level and host-level data in a way that facilitates anomaly detection over the joined data. Subsequently, we perform clustering on the log data to identify usage profiles, which we classify as normal or anomalous based on statistical tests. We devise a method to order the anomalous clusters in terms of their likely maliciousness, which can aid a security administrator in prioritizing which clusters to investigate manually. Finally, we show the efficacy of our approach by evaluating it on the VAST dataset.

More specifically, we make the following contributions:

Identify important features: We reason about and identify a good set of features to detect flooding-based network attacks and suspicious host behavior in our dataset. We aggregate and combine the data from network-level and host-level monitors based on the extracted features.

Detect anomalies: We apply the k-means and DBSCAN clustering algorithms to detect different behavioral patterns in system logs and firewall logs. We use statistical techniques to choose parameters for the clustering algorithms to minimize the amount of

tuning required by a domain expert. We present a metric to analyze the output of the clustering algorithms to identify anomalous clusters, and demonstrate its efficacy on our dataset.

Support intrusion detection: We introduce a technique to prioritize the anomalous clusters in order of likely maliciousness based on analysis of the distributions of the features in each cluster, and show that the intrusions present in the dataset are captured by the top few anomalous clusters.

The rest of the chapter is organized as follows. Section 2.2 discusses the dataset that we examine and describes the types of attacks that we want to detect. We provide an overview of the complete approach in Section 2.3. We discuss feature extraction and selection in Section 2.4 and our detailed intrusion detection approach based on clustering in Section 2.5. We provide the experimental results to evaluate the efficacy of our approach in Section 2.6. Related work and conclusions are provided in Section 2.7 and Section 2.8 respectively.

2.2 DATASET DETAILS AND THREAT MODEL

In this section, we introduce the dataset that we examine and describe the types of attacks that we want to detect using our unsupervised anomaly detection approach.

2.2.1 Dataset

In this chapter, we work with the dataset published by the Visual Analytics Community for Mini Challenge 2 of the VAST 2011 Challenge [48]. The dataset contains over 1.5 GB of enterprise network and host logs for network operations performed on the workstations and servers of a fictional freight shipping company over three days. We analyzed the logs from the firewall (Cisco Adaptive Security Appliance 5510), intrusion detection system (Snort), and Windows operating system security event logs (Windows 2008 Server).

The firewall logs in the dataset contain attributes about communications between internal and external hosts, such as source and destination IP address, communication protocol, and destination service. The Windows server security logs, which we refer to as *system logs*, record events such as valid and invalid logon attempts, authentication events, and subject and target user IDs. In the dataset's scenario, the Snort IDS has been configured to generate alerts based on the default Snort rule set.

The dataset is accompanied by a document that describes the attacks that were injected into the data and the monitors in which evidence for the attacks appears. The dataset is

Table 2.1: Attacks present in the Snort logs in the dataset and incidence of alerts of each type.

Attack Type	Number of instances
Message Flooding	927
Decoy Portscan	34
Distributed Portscan	23
Portscan	5
Portsweep	16,640
TCP Window Scale Attack	4,686

injected with the following types of attacks: denial of service from external hosts, a worm installed on some internal hosts, port scans by internal hosts, a socially engineered e-mail sent by an outside network address, a remote desktop connection violating company policy from outside network, and the appearance of a suspicious host on the enterprise network. We utilize the information in the dataset documentation to evaluate our intrusion detection approach, as explained in Section 2.6.

The distribution of attack types detected by Snort is depicted in Table 2.1. We use the Snort logs to gain insights about attacks on the system and to evaluate our intrusion detection technique.

2.2.2 Threat Model

Our threat model consists of two categories of security incidents in an enterprise network environment. The first category of attacks includes network scan attacks and flooding-based network attacks. Examples of such attacks include port scan attacks, in which a malicious subject scans a critical server in the network to gather information about the services running on different ports; port sweep attacks, in which a malicious subject scans multiple hosts for a single type of service; and network-layer Denial-of-Service (DoS) or Distributed DoS attacks, in which a malicious subject attempts to disrupt service availability by overwhelming the system. The goal of these attacks is to disrupt a network-based service, with a range of possible motivations, such as revenge or financial gain.

The second category of attacks that we aim to detect is the presence of malware on a host in the network, specifically viruses and worms. These types of malware can change the behavior of an infected host by running suspicious processes, scanning the network, or attempting to authenticate to other hosts in the network. Such malware can cause serious damage to the enterprise by stealing information, disrupting service, or using the machines for illicit purposes.

The goal of our work is to identify hosts that exhibit behavior that deviates significantly from normal usage patterns in order to detect the aforementioned types of attacks. We do not wish, however, to explicitly profile normal usage; rather, we want to learn the profiles in an unsupervised manner from the data. In later sections, we present our approach to detecting anomalies without explicit profiling.

2.3 OVERVIEW OF APPROACH

Since we know that there are different types of attacks within the data, but do not have labels for the individual log entries, we cannot use supervised algorithms to classify the data. Furthermore, there is no guarantee that the types of attacks taking place in the system will remain static. We must therefore use a learning algorithm that can discover host behavior patterns in the data in an unsupervised way.

Clustering is an unsupervised machine learning technique especially suitable for identifying structure in unlabeled data. It partitions a given set of objects into subsets of similar objects and separates dissimilar objects into different clusters. We use clustering algorithms in our approach, as they can identify behavior patterns in the data by using self-similarity and can distinguish behavior patterns that could signify anomalous behavior.

First, we identify the features that are most useful for detecting the types of attacks described in our threat model. Next, we apply dimensionality reduction and clustering on the features we extract from the dataset to discover clusters in the data, and then analyze the feature distributions of the clusters to identify potential anomalous behavior. Finally, we tag the anomalous clusters with the possible types of attacks they represent, which can then be analyzed manually by a security administrator to decide the best course of action to take. We compare the results of our approach with the ground truth provided with the dataset and evaluate the efficacy of our approach.

2.4 FEATURE EXTRACTION

2.4.1 Feature Extraction

We perform feature extraction and clustering on only the firewall and system logs in the dataset. Since Snort requires a rule set to generate meaningful alerts, it relies on a system administrator to select an adequate rule set, which presents the possibility of missing alerts. Snort also aggregates network traffic before generating some alerts, so it provides results

comparable to those of our own analysis. Therefore, we use Snort only to validate the results of our approach.

We choose features based on both the attacks discussed in our threat model and domain knowledge of the enterprise network on which the dataset is based. Domain knowledge includes the types and locations of critical servers and acceptable resource usage policies. We identify the following four categories of features, which we use to capture usage behavior and join the data across different monitors.

Identification Features

We use the IPv4 address as the unique identifier of each host in the network. In this dataset, the source IP can uniquely identify an attacker machine, and the destination IP can uniquely identify the target of an attack. For both the firewall logs and system logs, for each one-minute time window, we aggregate the raw log entries for a given IP address into a single feature vector. Thus, every data point can be uniquely identified with the IP address and the timestamp. All of the features described subsequently are extracted and aggregated for a distinct pair of an IP address and a timestamp. As we show in the next section, these identification features are very useful in correlating cluster analysis results with intrusions.

One important point to note is that in doing the time-based aggregation of features, we assume that the time is synchronized across all hosts in the system and across all monitors. We account for the time difference in the firewall logs and system logs in our dataset manually, but in other systems, it could be accomplished using a centralized network time server.

Network Traffic-Based Features

We extract the following features related to network communication from the firewall data: the number of unique source and destination ports used, the number of transmission control protocol (TCP) connections built and torn down, and the number of distinct administrative services used (e.g., Telnet, finger, FTP). We use the source IP address to identify the feature vectors during cluster analysis.

Service-Based Features

To incorporate the diversity of services accessed and the types and locations of machines contacted by a host in our analysis, we extract the following features from the firewall data

for each source IP address: the number of accesses to workstations; the number of accesses to the domain controller servers, which also provide the domain name system (DNS) service; the number of accesses to database servers; and the number of accesses to any other type of server.

Authentication-Based Features

Finally, since the system log data comes from the domain controller server, which services authentication requests for all workstations and servers in the network, we extract features from the system log data by counting the following properties of system log events: failed logon attempts, logon attempts using explicit credentials, special privilege assignments to new logons, computer account changes, generation of Kerberos authentication tickets, New Technology LAN Manager (NTLM) authentication attempts, administrative logons, anonymous user logons, anonymous target user names, local interactive logons, remote desktop logons, requests for session keys, port numbers that are either zero or blank, and logons by distinct process IDs. We aggregate the log entries by the source IP address of the authentication request for all events for which the source IP is given in the log.

2.4.2 Analysis of Features

After extracting the features, we analyze the distribution of values each feature takes in the dataset to understand the relative importance of the features and their possible effect on the clustering algorithms. Specifically, we look at the shape, variance, and modality of the distributions.

First, we examine the network traffic and service-based features extracted from the firewall data. Figure 2.1 shows the empirical cumulative distribution functions (CDFs) for a subset of the features for the firewall data aggregated by source IP address. The features take many different distributions, but our first observation is that many of them are not Gaussian and are right-tailed. It is also important to note that a significant proportion of the probability mass is on very small values for most of the features. For example, for the first feature, which represents the number of unique destination IP addresses visited per minute by a source, almost 75% of the data points have a value of 1. As a result, we expect that the tails of the distributions—that is, the data points that take high values—will be of most interest in the clustering. We also expect that the clustering approaches will place the vast majority of points into a few dense clusters.

Next, we examine the authentication-based features extracted from the system log data.

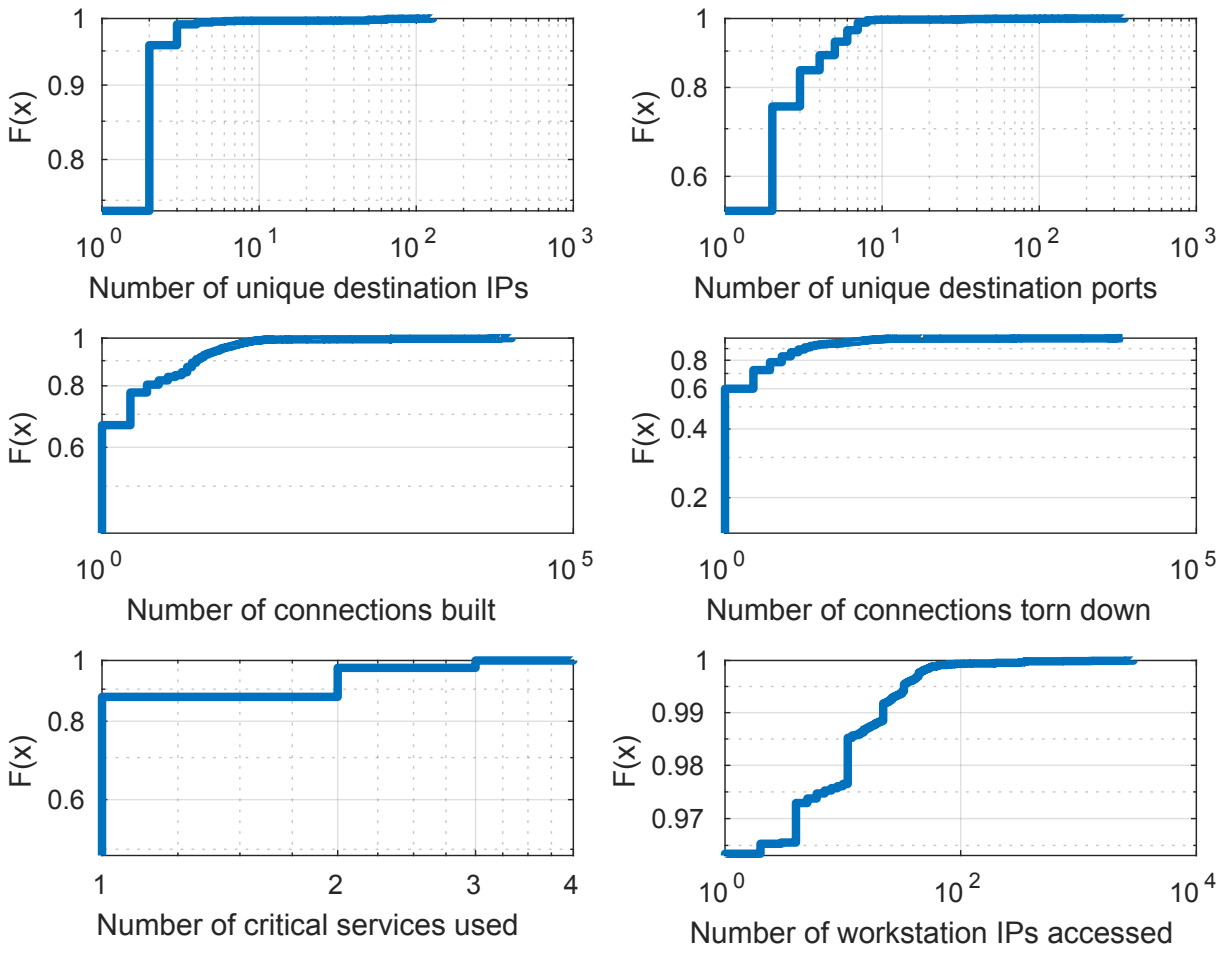


Figure 2.1: Empirical cumulative distribution functions (CDFs) for selected features in the firewall data aggregated by source IP address, shown with log scales for both the X and Y axes.

Figure 2.2 shows the empirical CDFs for a subset of the features for the system log data aggregated by source IP address. As with the firewall data, the features take many different distributions, but many are not Gaussian and right-tailed. The vast majority of the probability mass for each feature (over 95%) is on the values 0 or 1. For the system log data, we therefore expect even more densely packed clusters than for firewall data.

Through examination of our data, we observe that dimensionality reduction by Principal Component Analysis (PCA) prior to clustering does not provide useful results. Briefly, PCA projects data in high-dimensional space onto axes called *principal components*, each of which point in the direction of maximum variance given the variance already captured by preceding components. By projecting the data onto the first few principal components, it is possible to reduce the dimensionality of the data while retaining the bulk of the variance in the data.

In our data set, however, variance does not describe the expressivity of features well.

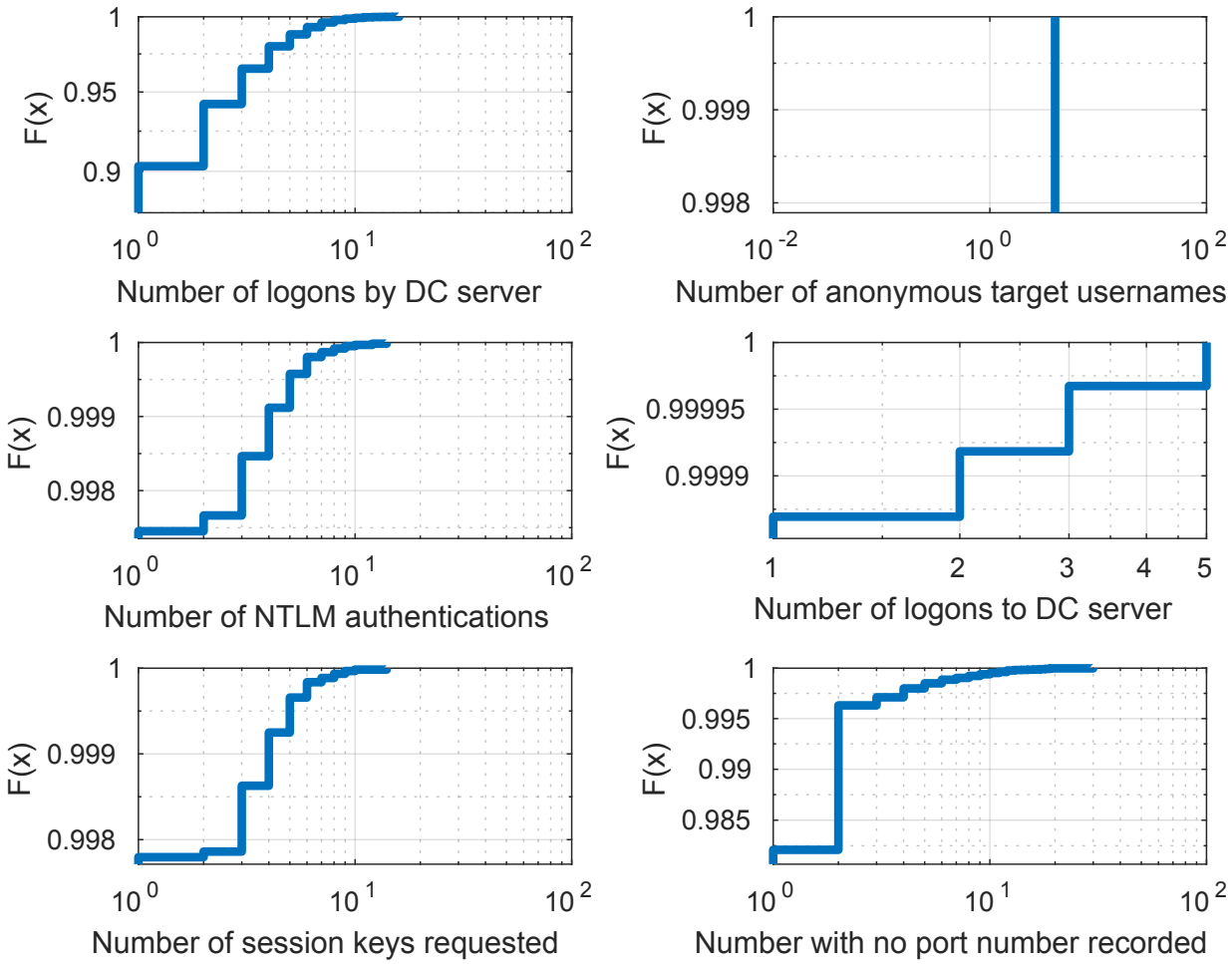


Figure 2.2: Empirical cumulative distribution functions (CDFs) for selected features in the system log data aggregated by source IP address, shown with log scales for both the X and Y axes.

First, many of the features in our data have constricted ranges, so variance can be artificially limited by the range of the feature values. As an example, the number of possible source ports in TCP is 2^{16}, so the features corresponding to the number of distinct source ports used by a host will be limited to the range $[0, 2^{16}]$. Additionally, the number of failed logon events, which is a strong indication of malicious behavior, takes a small range of values and therefore has low variance compared to the number of successful logons, which has a much higher variance but is relatively unuseful in detecting attacks.

Furthermore, we wish to detect anomalous behavior, which can manifest as a small number of data points with outlying values. However, the principal components determined by PCA that represent most of the variance in the data may not capture such behavior. For example, the first three principal components for the firewall data, which together capture over 99%

of the variance in the data, represent only the number of TCP connections built, the number of TCP connections torn down, and the number of unique source ports used. However, in our analysis of the clustering results, we find that the number of destination IP addresses contacted is the primary distinguishing feature for one of the attacks in the dataset, and clustering on the basis of the data projected onto the first few principal components would miss this attack altogether.

As a result, we perform clustering on the features without performing dimensionality reduction first. We do, however, refine the feature set before doing the clustering by looking at the correlation between different features and their relative importance for clustering, as we discuss in Section 2.4.3.

Also, to ensure that the clustering algorithms are not influenced by the difference in ranges in the different features, we normalize the data prior to clustering such that all features have a range of $[0, 1]$. We use min-max normalization, which performs a linear transformation on the original data values to scale them into a new range. For each feature value v, the new value v_{new} is calculated using Equation 2.1, where v_{max} and v_{min} are the maximum and minimum values of the feature v and new_{min} and new_{max} are endpoints of new range of values (0 and 1 in our case) respectively:

$$v_{new} = \frac{(v - v_{min})(new_{max} - new_{min})}{(v_{max} - v_{min})} + new_{min} \tag{2.1}$$

2.4.3 Feature Selection

We extract all the features described above and then use two different techniques to refine the feature set to use for clustering. First, if the features are strongly correlated to each other, they will contain redundant information and increase the complexity of the clustering algorithms. To remove this redundancy, we keep only one feature from each set of strongly correlated features. For example, every time there is an anonymous logon, the subject domain name is recorded as "NT Authority". These two features (subject user name is anonymous and subject domain name is NT Authority) essentially carry the same information. So we only use one of these two features to indicate an anonymous user logon.

We use the Pearson correlation coefficient to measure the linear dependence between each pair of feature vectors. We consider a pair as strongly correlated if the correlation coefficient is greater than 0.99. We believe that the Pearson correlation coefficient will provide fairly accurate estimation for correlation between features because the ranges of all the features are bounded and the size of our dataset is moderately high.

Second, if the features do not contribute towards clustering, then we discard them. For example, the number of logon events in each time interval is an important feature to characterize each host in the network, but in this dataset, it does not help distinguish between clusters. We prune such features before performing our clustering experiments.

To determine which features to prune, we run a clustering algorithm and analyze the average normalized feature scores in each cluster, which we define later in Section 2.5.3. We identify the features having almost the same average score in each cluster, and discard them for the final clustering experiments.

2.5 INTRUSION DETECTION USING CLUSTER ANALYSIS

In this section, we describe our intrusion detection approach in detail. First, we apply clustering algorithms to the data to discover clusters of similar data points, and then analyze the joint distributions of features in each cluster to identify the types of behavior associated with each cluster. We hypothesize that different types of attacks will manifest as different distributions of the features in the data, and data points corresponding to different attack types will cluster together because of their similarity with each other or their dissimilarity with other clusters. Based on the feature distribution analysis of the clusters, we then identify anomalous clusters and sort them by their dissimilarity with clusters that represent normal behavior. Finally, we manually analyze the anomalous clusters and identify the types of attacks they represent.

2.5.1 Overview of Algorithms Used

We use the k-means and DBSCAN clustering algorithms to classify different host behavior profiles. k-means is a centroid-based clustering algorithm that partitions a given dataset into k clusters, such that each observation belongs to the cluster with the closest mean. For our analysis, we apply Lloyd's basic k-means algorithm [49] and use Euclidean distance as the distance metric for clustering.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [50] creates clusters on the basis of the density of clusters formed. The DBSCAN algorithm starts with a random point and adds it to a cluster if a certain minimum number of points lie within a certain radius of the point. The algorithm then iteratively grows the cluster by similarly considering all points in the radius and repeating until no more points are added. After the assignment of points to one cluster completes, a new arbitrary point is selected, and the process is repeated. This algorithm also marks outlier points that lie alone in low-density

regions. For our analysis, as with k-means, we use the Euclidean distance as the distance metric for the DBSCAN algorithm.

We chose these two algorithms because they are simple and widely used, and we hypothesize that given the right set of features and analysis techniques, they can yield good results. Since the algorithms use different approaches to cluster points, we expect them to work well in different types of situations. k-means is relatively simpler and computationally faster than DBSCAN, but it is sensitive to noise and works well only when the clusters are spherical (i.e., have equal variance in all dimensions) and have a roughly equal number of data points. DBSCAN, on the other hand, can generate arbitrarily shaped clusters by identifying connected regions of high density, and is designed to ignore noisy points. We compare the results of each algorithm on our dataset and make decisions on which to use based on the goodness of the clusters they generate.

In addition to running the two clustering algorithms on the firewall data and system log data separately, we join the two data sources by performing an inner join for values that share source IP address and timestamp values, and run clustering on the joined data as well. We hypothesize that clustering on joined data will reveal attacks that would not be evident from either data set independently.

2.5.2 Parameter Selection for Clustering Algorithms

Parameter Selection for k-means

The k-means algorithm requires a single parameter: the number of clusters assumed to be in the data, k. To determine the number of clusters to use for k-means, we examine the cluster tightness and the separation between clusters.

To choose the number of clusters that best represents the data, we start by choosing a range of values for k that would likely best fit the data. For each candidate value of k, we run the k-means algorithm for a fixed number of iterations and take the cluster means that minimize the *within-cluster sum of distances (WCSD)*, which is the average sum of Euclidean distances from each point to the center of the cluster to which it is classified. We use this procedure to account for the initialization bias encountered with k-means, which could otherwise lead the k-means result to reside in a local optimum.

Then, to test the appropriateness of the value of k, we look at the silhouette values [51] of all points in the data set. The silhouette value of a point quantifies how well it belongs to the cluster to which it is assigned as compared to all other clusters. For a point $\mathbf{x}$, let $a(\mathbf{x})$ represent the average distance from $\mathbf{x}$ to all other points in its cluster, and $b(\mathbf{x})$ represent

the average distance from $\mathbf{x}$ to all other points in the cluster for which such distance is minimized (the next closest cluster). Then, the silhouette value of $\mathbf{x}$, $s(\mathbf{x})$, is given by

$$s(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\left(a(\mathbf{x}), b(\mathbf{x})\right)} \tag{2.2}$$

The values of $s(\mathbf{x})$ can range from -1 to 1, and values closest to 1 indicate clusters that are tight and well-separated. Therefore, we choose the value of k for which the run with the smallest WCSD has the maximum average silhouette value for all points.

Parameter Selection for DBSCAN

The DBSCAN algorithm requires two input parameters: ϵ, the neighborhood radius for each point, and $minPts$, the minimum number of points required to form a cluster. We use the guidelines presented in the original DBSCAN paper [50] along with evaluation of the average silhouette values to compute suitable values for the parameters.

According to [50], the value of $minPts$ should be at least $D + 1$, where D is the number of dimensions of the dataset used for clustering. The authors recommend choosing a larger value of $minPts$ as the size of the dataset or the amount of noise in the dataset increases. After deciding on a reasonable estimate for the value of $minPts$, we examine the *sorted k-distance graph* for our dataset to determine the value of ϵ. The k-distance graph depicts the set of distances from each data point to its k^{th}-nearest neighbor, sorted in decreasing order of magnitude, where $k = minPts$. A good value of ϵ is one for which the k-distance graph shows a strong bend or *knee point*.

Using the approach given above, we estimate a small range of values for each of the parameters. For each pair of $(\epsilon, minPts)$, we then run the DBSCAN algorithm on the dataset and compute the average silhouette score as explained above for k-means clustering, ignoring the outlier points. We choose the pair of parameter values corresponding to the best average silhouette score. In this manner, we select the parameter values by considering the properties of our dataset, the properties of the DBSCAN algorithm, and the quality of clusters generated. As with k-means, we use the Euclidean distance between the data points to measure the similarity for clustering.

2.5.3 Cluster Analysis and Classification

After running k-means and DBSCAN with the parameter values selected by the methods described above, we analyze the clusters obtained to identify those that correspond to

anomalous behavior. For each clustering algorithm and log type, we examine the cluster sizes and distribution of hosts within the clusters to distinguish between "normal" clusters and "anomalous" clusters.

For each cluster c in the data set, let $\mathcal{S}_c$ represent the set of points in cluster c, and let $\mathcal{H}_c$ represent the set of unique hosts, identified by source IP address, in the cluster. We consider a cluster c to represent normal behavior in the system if $|\mathcal{S}_c| \geq s$, where s is a minimum threshold for the number of points, and $|\mathcal{H}_c| \geq h$, where h is a minimum threshold for the number of unique hosts. That decision was motivated by our observation that the distributions of feature values for both the firewall and system log data have very high probability mass at low values, so most of the normal data points will cluster into a few densely packed clusters containing most of the hosts. Anomalous behavior represents either 1) a large number of attacking hosts acting maliciously over a short period of time, as in the case of a denial-of-service (DoS) attack, which would result in a small value of $|\mathcal{S}_c|$, or 2) a small number of attacking hosts acting maliciously over an arbitrary period of time, as in the case of a port scan, which would result in a small value of $|\mathcal{H}_c|$.

Indeed, we found that for all data types in our dataset, over 80% of the data points always fell into the first two or three clusters, and that these clusters individually contained well over half of the hosts. Based on those observations, we chose the values of $s = S/k$ and $h = H/k$, where k is the number of clusters, $S = \sum_{c=1}^{k} |\mathcal{S}_c|$ is the total number of data points, and $H = \left|\cup_{c=1}^{k} \mathcal{H}_c\right|$ is the total number of hosts in the system. Intuitively, a cluster will be considered normal if more than a proportional fraction of the data points and hosts fall within the cluster. In our actual data, the normal clusters contained much more than the threshold value of data points and almost always contained all of the hosts.

We represent the sets of normal and abnormal clusters with the following equations:

$$\mathcal{N} = \{c : |\mathcal{S}_c| \geq s \wedge |\mathcal{H}_c| \geq h\} \tag{2.3}$$

$$\mathcal{A} = \{c : |\mathcal{S}_c| < s \vee |\mathcal{H}_c| < h\}. \tag{2.4}$$

Next, we compute the *normalized average feature value vector*, $\hat{\mathbf{f}}^c$, for each cluster c as follows:

$$\mathbf{f}^c = \sum_{\mathbf{x} \in \mathcal{S}_c} \frac{\mathbf{x}}{|\mathcal{S}_c|} \tag{2.5}$$

$$\hat{\mathbf{f}}^c = \frac{\mathbf{f}^c}{\max_i \mathbf{f}_i^c} \tag{2.6}$$

where i is the feature index, S_c is the set of points in cluster c, and F is the number of features. For each cluster, $\mathbf{f}^c$ is the centroid of cluster c, and $\hat{\mathbf{f}}^c$ is the centroid normalized by the supremum norm of the centroid. $\hat{\mathbf{f}}^c$ describes the features that are active in a given cluster and makes it easy to compare their relative strengths, as the maximum value in $\hat{\mathbf{f}}^c = 1$. We use $\hat{\mathbf{f}}^c$ to understand the feature distributions for each of the clusters and to determine which clusters are more likely to be malicious, as we describe in the next section.

2.5.4 Intrusion Detection

The last step in our approach is to determine which of the clusters representing anomalous behavior actually describe *malicious* behavior.

Using the $\hat{\mathbf{f}}^c$ vectors, we compare the joint distributions of features for each anomalous cluster with those for normal clusters. Specifically, we compute the difference between the feature distributions of two clusters c and c' by computing the L_1 distance between the $\hat{\mathbf{f}}$ values for each cluster. We define the *cluster difference*, $D_{c,c'}$ as follows:

$$D_{c,c'} = \left\| \hat{\mathbf{f}}^c - \hat{\mathbf{f}}^{c'} \right\|_1 \tag{2.7}$$

In our analysis of the dataset, we observed that the feature vectors were sparse; that is, only a few features took nonzero values in each cluster, and of those, only a few had high values. Furthermore, since we removed highly correlated features before clustering, the cluster difference for different clusters was not small unless the clusters themselves were geometrically close to each other. Thus, we posit that anomalous clusters with the greatest difference from normal clusters in terms of normalized feature distributions are more likely to be malicious.

As a result, we define the *cluster normalcy*, Δ_c, for an anomalous cluster c as the cluster difference between the anomalous cluster and its closest normal cluster. In other words,

$$\Delta_c = \min_{c' \in \mathcal{N}} D_{c,c'}. \tag{2.8}$$

We ultimately provide a prioritized list of the anomalous clusters, ordered by decreasing value of Δ_c, to the security administrator to manually evaluate. By separating the clusters by behavior, reducing the clusters to those that are anomalous, and prioritizing the clusters by likely maliciousness, we significantly simplify the work that must be performed by the administrator.

Given the administrator's input, we envision the possibility of training a classifier that

could automatically determine whether a new data point is likely to be malicious. We discuss the idea briefly as part of future work in Section 2.8.

2.6 EXPERIMENTAL EVALUATION

To experimentally evaluate our approach, we implemented our feature extraction code in Python and ran all of the machine learning and cluster analysis on the data in MATLAB. We used MATLAB's stats toolbox implementation of k-means and our own implementation of DBSCAN.

As mentioned in Section 2.2.1, a ground truth document was provided with the dataset that contains the attacks that were injected into the data and the monitors in which evidence for the attacks appears. Analysis of this document and the challenge description showed that because of the network configuration of the dataset's system, the firewall was not able to observe all of the port scanning/sweeping attacks that took place in the dataset. As a result, we evaluated our approach only with those attacks detectable using only the firewall logs and system logs. This yielded the following list of attacks: denial of service (DoS) from hosts 10.200.150.201 and 10.200.150.206–209, worm likely installed on hosts 192.168.2.171–175, port scans by hosts 192.168.2.174–175, a socially engineered e-mail sent by host 10.200.150.6, a remote desktop connection violating company policy from host 10.200.150.201, and the appearance of a suspicious host with IP address 192.169.2.151.

Based on our threat model, we further narrowed the set of attacks we aim to detect to those representing network flooding attacks and suspicious behavior by hosts. Any attack that was a direct violation of policy or required investigation of raw packet data (such as the socially engineered e-mail) would not be detectable under our threat model, or indeed with just the firewall and system log data and no other information. Thus, the set of attacks we expected to be able to detect decreased to DoS from hosts 10.200.150.201 and 10.200.150.206–209, worm likely installed on hosts 192.168.2.171–175, and port scans by hosts 192.168.2.174–175.

To evaluate the ability of our approach to identify the attacks actually present in the dataset, for each anomalous cluster, we examine the IP addresses of the hosts in the cluster, the timestamps of the entries, and the distribution of feature values, and compare these to the ground truth document. If the hosts and timestamps in the cluster correspond to a known attack in the ground truth document, we consider the cluster to detect the attack.

From the firewall data, as described in Section 2.4, we extracted a total of 12 features. Two of the features—IP address and timestamp—were identification features, but the remaining 10 features were of use for clustering: unique destination IPs, unique source ports, unique destination ports, connections built, connections torn down, critical services used, work-

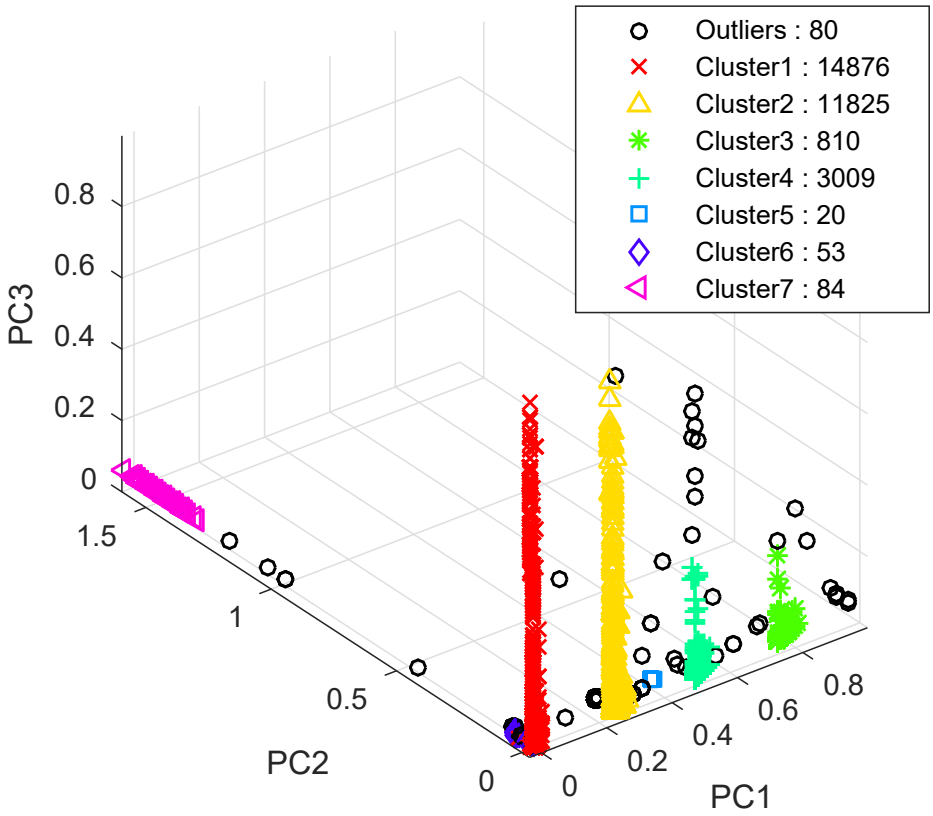


Figure 2.3: DBSCAN clusters for the firewall data for parameter values $\epsilon = 0.15$ and $minPts = 21$, projected to the first three Principal Components (PC) for visualization purposes. Clustering was performed in the original 10-dimensional space.

stations accessed, DNS server accesses, database server accesses, and other critical server accesses. For each feature, we aggregated the value of the feature per minute for each source IP.

From the system log data, we extracted a total of 36 features, of which the same two as for the firewall were identification features. After we ran our feature selection approach from Section 2.4.2, the number of uncorrelated features was reduced to 20, and all of them were used for clustering.

2.6.1 Detecting Flooding-Based Network Attacks

Between the two types of monitors we used in our analysis, flooding-based network attacks manifest most directly in firewall data. Because external hosts cannot communicate with the domain controller according to the dataset scenario, hosts in the 10.200.0.0/24 subnet do not appear in the system log data at all, so the only way for us to detect attacks from

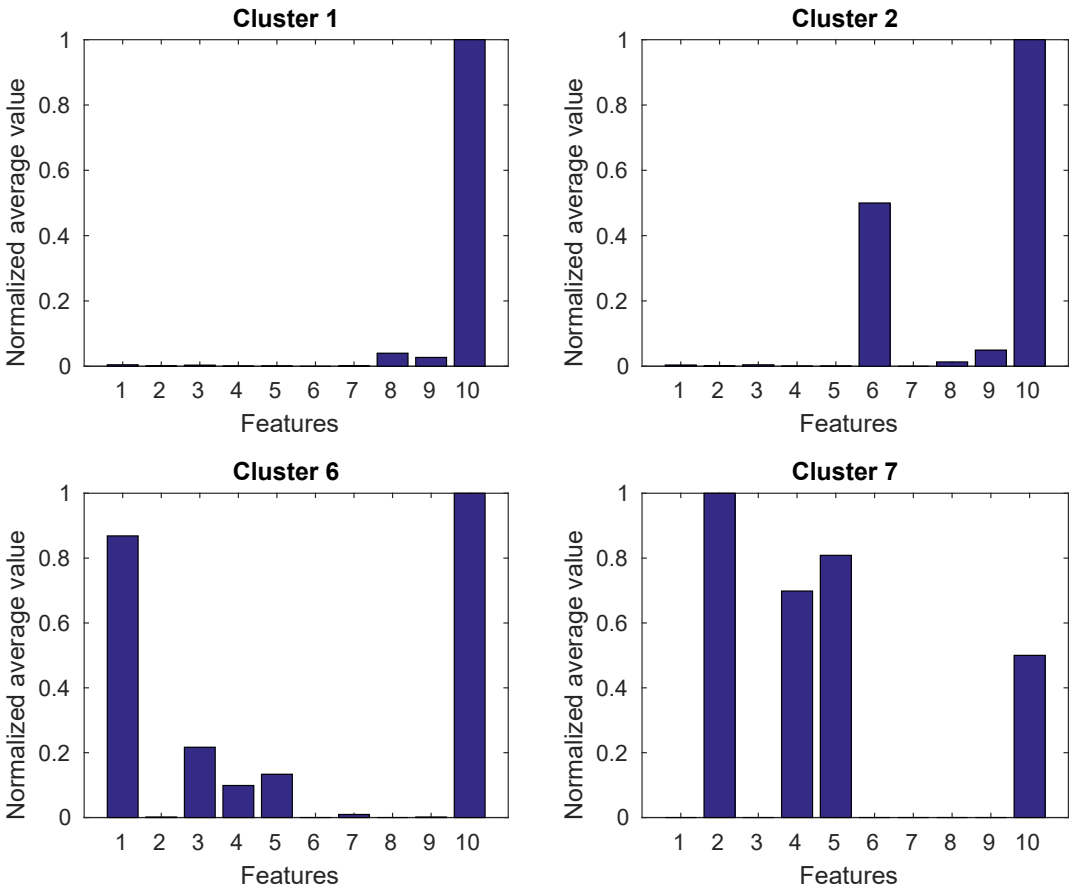


Figure 2.4: Normalized average feature values for four DBSCAN clusters on the firewall data.

outside hosts with our approach is by analyzing the firewall logs.

Extracting per-minute data points from the firewall logs yielded over $150,000$ data points, which exceeded the computational and memory capabilities of our machines when running DBSCAN. To accommodate, we uniformly randomly downsampled the data to 20% of its original size before running both clustering algorithms. Since we downsampled uniformly, the overall distribution of points did not change. Furthermore, the types of attacks we aim to detect generate multiple data points in the data set, so there was no threat that important clusters would disappear from the dataset. We also adjusted the parameters of DBSCAN to account for the decreased density of the dataset.

We ran the parameter value selection algorithms described in Section 2.5.2 on the data set to obtain the optimal parameters for the clustering algorithms. The parameter values we obtained were $k = 8$, $\epsilon = 0.15$, and $minPts = 21$. The results of running DBSCAN on the firewall logs with said parameters are shown in Figure 2.3. With the parameters specified, DBSCAN identified 6 clusters and a small set of outlier points. While it is not evident from

the graph, the clusters are actually quite distant from each other in the higher-dimensional space in which clustering was performed.

We found that k-means performed poorly on the firewall data, clustering together data points that did not have similar feature distributions and absorbing smaller clusters into their larger, neighboring clusters. That was likely due to the non-Gaussian distribution of the data points and the large variance in the number of points in each cluster. DBSCAN, on the other hand, performed exceptionally well. For the remainder of this analysis, we use the DBSCAN results.

Next, we analyzed the size of each cluster and the number of unique hosts in each. Applying Equations 2.3 and 2.4, we found that clusters 1 and 2, which together represented over 86% of the data points, were classified as normal, and all others were classified as anomalous. We then performed the cluster difference analysis and ordered the clusters based on their Δ_c values, as described in Section 2.5.4. For the firewall data, excluding the outliers, cluster 6 was considered most likely to be malicious, followed by clusters 5, 3, and 4. This analysis relied on the computation of normalized average feature vector for each cluster, as shown in Figure 2.4.

When we analyzed the hosts and timestamps for each of the anomalous clusters, we found that the order given by the cluster normalcy-based prioritization exactly matched the ground truth, which showed that clusters 6 and 5 corresponded to attacks and clusters 3 and 4 were not malicious.

Cluster 6 contained only the IP addresses 10.200.150.201 and 10.200.150.206–209 and had very high feature values for the number of unique source ports, number of connections built, and number of connections torn down. The cluster corresponds to outside attackers' performing a DoS attempt on one of the servers inside the network. The timestamp values for the points in cluster 6 correspond to some of the times that the hosts were performing the DoS attack.

Cluster 5 contained only the IP addresses 192.168.2.174–175 and 192.168.1.6, and had very high feature values for the number of destination IPs accessed. Examining the timestamp values for the points showed that while 192.168.1.6 appeared only at three very different times in the cluster, the other two hosts appeared over 20 times within the same 4-hour period. The cluster corresponds to two hosts inside the network that were performing port scans of other machines in the network in the time window in which the two hosts appeared in the cluster. It is likely that 192.168.1.6, which is a mail server, was classified into this cluster because it sent mails to many hosts at the three times it appeared in the cluster, and thus had a feature distribution similar to that of the port scanners.

For the firewall data, our approach performed incredibly well, separating out the two

attacks—DoS from the external network and port scan from the internal network—and providing those two clusters as those most likely to be malicious. As a sanity check, when we compare the performance of our completely unsupervised approach to that of Snort, we find that Snort detected the same two attacks in the dataset, but required an administrator to sift through many lines of logs.

2.6.2 Detecting Suspicious Host Behavior

To detect suspicious behavior of internal hosts and to find the attacks we mention above, we experimented with both system log data alone and system log data combined with firewall data. As was the case with the firewall data alone, and for many of the same reasons, k-means did not perform well in cleanly separating clusters by host behavior. Therefore, here, too, we focus on the clustering results we obtained from DBSCAN.

First, we performed DBSCAN clustering on the system log data alone. Because of the size of the data, we uniformly randomly downsampled the data to 50% of its original size. Running the parameter selection algorithm yielded $\epsilon = 0.15$ and $minPts = 30$, and running DBSCAN on the system log data with those parameters yielded five clusters and a set of outliers. Performing the analysis described in Sections 2.5.3 and 2.5.4, we found that cluster 1 was classified as normal, and all others as anomalous. Of those, only cluster 3, which had the highest Δ_c value, represented a malicious host.

Next, we performed DBSCAN clustering on the combined firewall and system logs, which were joined by the approach explained in Section 2.5.1. The optimal parameters for the joined data were $\epsilon = 0.25$ and $minPts = 20$, for which the results of DBSCAN are shown in Figure 2.5. There are four clusters and one group of outlier points. While the clusters do not seem separable in the graph, they are quite distant in the higher-dimensional space in which clustering was performed.

Performing the analysis described in Sections 2.5.3 and 2.5.4, we found that cluster 1 represented normal behavior, and clusters 2, 3, and 4 were classified as anomalous. In order to analyze the anomalous clusters for attacks, we computed the normalized average feature vector for each cluster, as shown in Figure 2.6. Features 1 – 18 correspond to the features extracted from system log data, and features 19 – 28 correspond to those extracted from the firewall log.

When we ordered the anomalous clusters by their cluster normalcy values, we found that cluster 2 was most likely to be malicious, followed by clusters 4 and 3. Based on the unique IP addresses in cluster 3, we concluded that it contained only benign hosts. However, in line with the prioritization of the clusters, clusters 2 and 4 corresponded to the attacks present

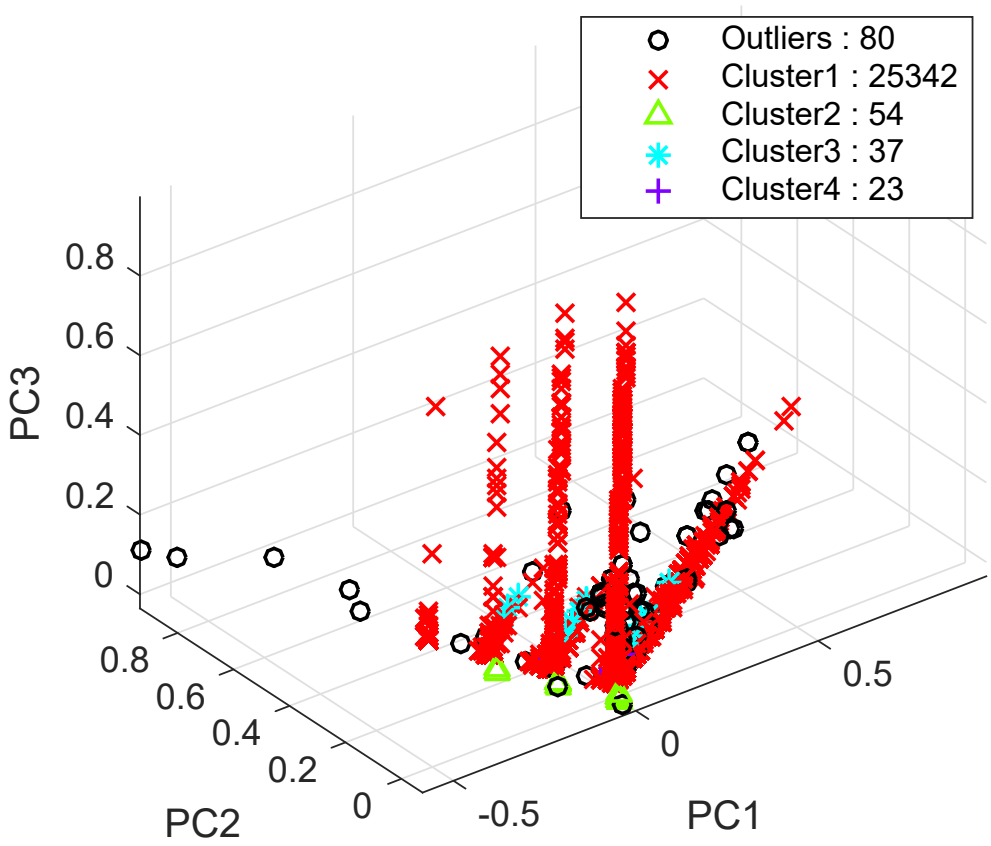


Figure 2.5: DBSCAN clusters for the combined firewall and system log data for parameter values $\epsilon = 0.25$ and $minPts = 20$, projected to the first three Principal Components (PC) for visualization purposes. Clustering was performed in the original, 28-dimensional space.

in the dataset.

Cluster 4 contained only the IP addresses 192.168.2.174–175 and had very high feature values for the number of connections built and torn down. The cluster corresponds to the two hosts' performing port scans on the internal network that were detected by the firewall. Note, however, that this cluster did not erroneously contain the mail server.

Cluster 2 was the only cluster that had relatively higher values for both system log and firewall log features. The cluster had high average values for the number of anonymous target usernames used, the number of NTLM authentications, and the number of session keys requested. This cluster contained only the IP address 192.168.2.172. From the features for which it had high values, it can be inferred that the host was trying to make connections and log on to other hosts on the internal network. While the dataset did not explicitly mark this host as being infected, this situation indicates a very high probability that the host had been infected by a worm.

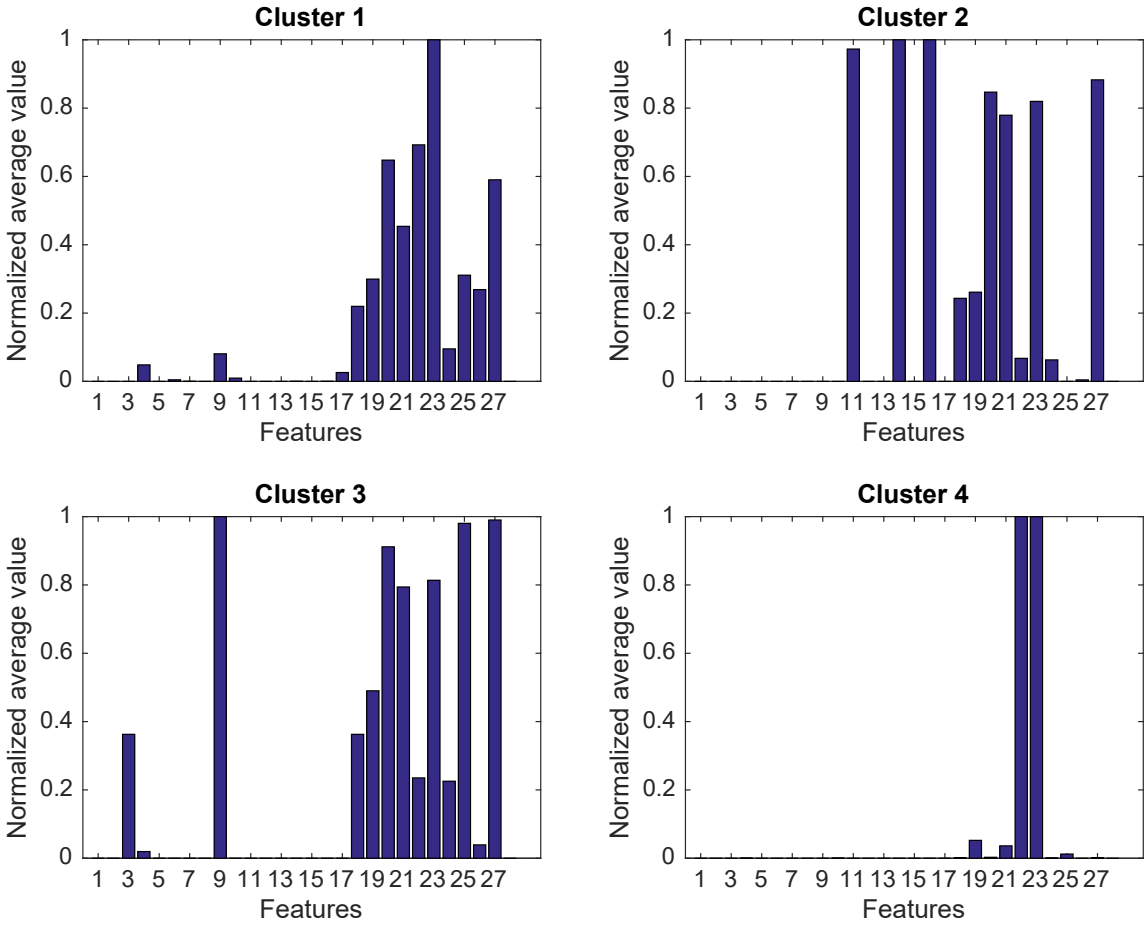


Figure 2.6: Normalized average feature values for four DBSCAN clusters on combined firewall and system log data.

Finally, as a sanity check, we compared clusters 2 and 4 against the attacks detected by the Snort IDS. While the Snort logs captured the port scan attack, they did not capture the malicious behavior of host 192.168.2.172. That behavior was detectable only through clustering on the combined data, which demonstrates the value of our approach of clustering on joined data instead of looking at each type of data source separately.

2.6.3 Discussion

As evidenced by the results in Sections 2.6.1 and 2.6.2, our approach performed well on the dataset for the types of attacks we describe in our threat model. Without labeling or preprocessing the data by hand, we were able to extract generic, time-aware features, use clustering to identify anomalous behavior, and prioritize the anomalous behaviors in order of decreasing likelihood of maliciousness. For each of the log types in the dataset, our approach

correctly identified and prioritized the anomalous clusters corresponding to attacks.

Furthermore, our approach provides the system administrator with additional visibility into the behavior of hosts in the system. By looking at the average feature distributions for each cluster in the dataset, the administrator can determine how different sets of hosts behave in the system and can easily drill down into the anomalous hosts to determine which ones are behaving maliciously.

Since the features we extract describe generic network-based and authentication-based attributes, we believe that our approach can generalize well to any attack that dramatically changes the behavior of a host. For example, we believe that with the same set of features, it would be possible to detect brute-force attempts or data exfiltration attacks.

However, our approach cannot detect all types of intrusions with the same level of accuracy. Intrusions that occur silently, such as zero-day attacks, and those that do not disturb the outward behavior of the host significantly, such as privilege escalation attacks, would not cause the host to appear anomalous.

As detailed in Section 2.6, we base the evaluation of our intrusion detection technique on the ground truth document describing all the injected attacks in the dataset. We do not claim that there is no possibility of the presence of attacks other than the ones that are given in that document.

With additional features and more diverse sources of information, it might be possible to expand the types of attacks supported by our approach. We leave such exploration to future work.

2.7 RELATED WORK

2.7.1 Unsupervised Anomaly Detection

Machine learning-based approaches have been widely adopted in the computer security and intrusion detection fields. These approaches can be classified into two categories: 1) artificial intelligence techniques such as supervised (e.g., classification) and unsupervised (e.g., clustering) learning algorithms, and 2) computational intelligence methods such as genetic algorithms, artificial neural networks, and fuzzy logic. A detailed survey of these techniques is provided in [52].

Clustering is an unsupervised machine learning technique, especially suitable for identifying structure in unlabeled data. It partitions a given set of objects into subsets of similar objects and keeps dissimilar objects into different clusters. We apply clustering and dimensionality reduction on features extracted from security monitoring data to find clusters

related to abnormal events, which might also consist of intrusions. We identify these clusters by analyzing distributions of different features in each cluster.

Unsupervised anomaly detection in enterprise network systems is not itself a novel idea. Significant research has been done in the domain of network traffic anomaly detection using techniques including PCA-based anomaly detection [53] and entropy and clustering-based anomaly detection [54]. Anomaly detection based on cluster analysis is also used in [55] and [56] to find intrusions in enterprise network environments.

We analyze a diverse set of monitoring data from different security monitors to detect anomalies and tag them as possible intrusions. We also provide a formal approach for selecting parameters for the different algorithms we use. Furthermore, we directly work with raw logs captured from different security monitors and do not rely on any commercial security information and event management (SIEM) system. These contributions differentiate our work from previous related work [55, 56].

Another distinguishing feature of our work is that we do not need a model or normal profile of the system. Some of the prior work [57, 58, 59] that relies on modeling of normal system behavior to detect anomalies can fail to model a good reference profile in a malicious environment like the one represented by our dataset. In contrast, we adopt a completely unsupervised technique for detecting anomalies. We also reason about the relative importance of various features in detecting certain types of anomalies, very similar to the work done in [60], but we use an unlabeled dataset for experimentation.

2.7.2 Information Fusion

Our work is also related to information fusion techniques for security [25, 61]. However, most of those prior solutions either are too complicated to be used in practice, or need a significant amount of model training or parameter setting.

Within the information fusion domain, our work is also related to the topic of alert correlation, which aims to convert security information into knowledge that is more intuitive and understandable to human administrators. Alert correlation techniques find non-trivial relationships between alerts from multiple sources and group them together to satisfy a variety of goals, such as summarizing IDS logs into higher-level incidents or improving the detection of collaborative attacks [33, 34, 35, 36]. These techniques operate on IDS data, which are higher-level than the data we use, and perform correlation using heuristics or predefined rule sets. In contrast, we categorize data points into normal or anomalous clusters based solely on the similarity of individual data points to each other and the feature distributions of the clusters formed.

2.8 CONCLUSION

In this chapter, we presented an approach to identifying anomalous behavior in unlabeled system log and network log data using unsupervised machine learning. Our approach and analysis are based on the monitors available in the VAST 2011 Mini Challenge 2 data set. We first established a threat model and extracted meaningful features from the log data that aided in attack detection. We then described a method to combine the data using features we have defined that allowed us to perform anomaly detection over the joined network and host log data. Next, we used the k-means and DBSCAN clustering algorithms to categorize the data into different usage profiles, and we compared the feature distributions of different clusters to identify anomalous behavior. We then introduced a cluster difference metric that we used to prioritize the anomalous clusters based on their likely maliciousness. Finally, we manually analyzed the clusters to correlate them with known attacks and evaluate our approach. We found that our approach performed very well for the data set, detecting all attacks present. Using the joined network-level and host-level data, we could detect attacks that are not detectable with either data source alone.

Our fusion-based intrusion detection achieved better accuracy than that gained when using a single source, and produced a small volume of alerts. Attacks were captured in four clusters that collectively contained less than 1% of the total data points. The work in this chapter has been peer-reviewed and was published in [62].

CHAPTER 3: MALICIOUS LATERAL MOVEMENT DETECTION

In this chapter, we continue Chapter 2's theme of improving the detection of advanced threats through analysis and correlation of information from multiple monitors. Having presented an approach for accurate detection of the vectors of initial compromise, next, we focus on the command and control and lateral movement stages of the attack.

Lateral movement-based attacks are increasingly leading to compromises in large private and government networks, often resulting in information exfiltration or service disruption. Such attacks are often slow and stealthy and usually evade existing security products. To enable effective detection of such attacks, we present a new approach based on graph-based modeling of the security state of the target system and correlation of diverse indicators of anomalous host behavior. We believe that irrespective of the specific attack vectors used, attackers typically establish a command and control channel to operate, and move in the target system to escalate their privileges and reach sensitive areas. Accordingly, we identify important features of command and control and lateral movement activities and extract them from internal and external communication traffic. Driven by the analysis of the features, we propose the use of multiple anomaly detection techniques to identify compromised hosts. These methods include Principal Component Analysis, k-means clustering, and Median Absolute Deviation-based outlier detection. We evaluate the accuracy of identifying compromised hosts by using injected attack traffic in a real enterprise network dataset, for various attack communication models. Our results show that the proposed approach can detect infected hosts with high accuracy and a low false positive rate.

3.1 SUMMARY OF CONTRIBUTIONS

There has been a surge in targeted cyber attacks that use persistent and stealthy actions to compromise victim organizations, usually high-profile companies or government agencies. Such attacks are popularly known as *advanced persistent threats (APTs)*. Typically, the goal of APT actors is to steal proprietary intellectual property or disturb mission-critical services. Although such attacks infect fewer entities than mass malware (e.g., botnets, worms) does, the estimated cost of data loss due to APT incidents is significantly higher. The expected losses incurred in the 2013 Target Corporation data breach and the 2011 RSA data breach were $248 million and $66 million, respectively [1, 63].

APT actors enter target systems by evading existing signature- or anomaly-based detection and prevention systems. Afterward, these actors establish a presence and persistence in

the network by compromising multiple accounts and hosts. This process of expansion and persistence of the APT is commonly known as *lateral movement (LM)*. Lateral movement usually happens in conjunction with *command and control* (C&C) communications to gather internal system structure information, guide the expansion process, and ultimately cause data exfiltration [7, 12].

The challenges involved in detecting APTs are multifold. First, attack tools are constantly evolving and adapting to changes in cyber defense technologies [6]. Attackers can stay undetected over prolonged time periods by using traditional operating system and networking services as their attack vectors [7, 8, 14]. Second, the attackers are not bound to follow any particular defined progression in compromising a target system [14, 17]. Finally, there is a lack of data sources about the actual behavior of attackers during the lateral movement stage, which limits the research community's ability to effectively build and evaluate models of lateral movement-based attacks and their defense mechanisms. Numerous successful incidents show that the current security solutions fall short in addressing these challenges [1, 8, 14].

We observe that regardless of the attack vectors that the attackers choose to adopt, compromised machines necessarily have to communicate with external C&C servers. The malware also needs to spread laterally in the network to reach the inner segments that are not exposed to the outside Internet. In fact, C&C and lateral movement are common, and necessary, patterns of attack behaviors; they have persisted in studied APTs over the past seven years [64].

In this chapter, we present an attack-vector-independent approach to detect lateral movement-based attacks (e.g., APTs) in an enterprise network system. We concentrate on the broader patterns of interactions that an attacker needs to have with the system after gaining initial entry. To analyze diverse indicators of C&C and LM compromises in a collective way, we propose a graph-based model of the target system, which we call a *host communication graph (HC graph)*. The HC graph represents the internal communication between target system hosts, along with the data collected from C&C and LM monitors.

First, we extract features from the HC graph to evaluate both the C&C and lateral movement-specific disruptions. Next, we use the features to develop two anomaly detection methods. In the first method, we identify C&C infected hosts and use principal component analysis (PCA) and k-means on the lateral movement-related features to find hosts that behave much like the infected hosts. In the second method, we perform extreme value analysis on the data transformed by PCA to identify hosts infected by malicious lateral movement. Since the two anomaly detectors use diverse indicators to determine infected hosts, we combine their outputs to generate the final set of compromised hosts. We use a parameter-based averaging ensemble method. We mix different proportions of the flagged

hosts from the two detectors, and, for each host, compute the average label value. Our approach is unsupervised, since we do not use the true labels to train the ensemble approach. Therefore, we measure the accuracy of detection by changing the threshold of the average label value above which we consider a host to be compromised.

For our experiments, we used a dataset of normal network operations from the Los Alamos National Lab's enterprise network [65]. To inject attack traces in the dataset, we developed a model for C&C and LM activities using the susceptible-infected-susceptible (SIS) virus spread model [66]. The model implements a small set of tunable parameters that allowed us to simulate different types of LM activities, including both benign administrative tasks and malicious APT traffic. We used these parameters to evaluate our method's sensitivity to changes in the attacker's behavior. In particular, we varied the number of hosts that were part of the malicious LM and the number of hosts that were infected by C&C malware and then computed the true and false positive rates (TPR, FPR) of detection in each setting.

On average (computed over all experiment runs), we achieved a TPR of 88.7% (confidence interval 1.9%) and an FPR of 14.1% (confidence interval 3.6%). Even in the atypical case when the attack had compromised almost all the hosts in the network, the TPR realized was 82.6%. Analysis of the ensemble detector's accuracy done by varying the classification threshold of the average label value provided insights that can be used by security analysts to tune the detectors.

More specifically, we make the following contributions:

Feature identification and extraction: We identify characteristic patterns of malicious lateral movement and C&C, and represent them as statistical features that we extract from diverse security monitors, mainly NetFlow, LM, and C&C monitors (Sections 3.4.1, 3.4.2).

Anomaly detection: We use the proposed features with an ensemble of unsupervised anomaly detection techniques to build an automated method to identify compromised hosts in the target network (Section 3.4.3).

Attack trace generation: We provide a method for generating malicious as well as benign lateral movement traces, based on the SIS virus spread model. We injected the generated traces in an enterprise system dataset consisting of internal network traffic (Section 3.5).

Trace-based evaluation: We perform trace-based simulations to evaluate the proposed system's accuracy in detecting compromised hosts for two attack communication models (Sections 3.5, 3.6).

The rest of this chapter is organized as follows. We present our problem statement and threat model in Section 3.2. We provide an overview of the proposed system in Section 3.3. We then describe the technical details of our lateral movement-based attack detection in Section 3.4. We present the results of our experimental evaluation in Section 3.5, followed by a discussion of several important characteristics of our approach in Section 3.6. Finally, we address related work in Section 3.7 and conclude the chapter in Section 3.8.

3.2 THREAT MODEL

In this work, we consider network-based APT attacks on enterprise systems from external actors, mainly involving C&C and lateral movement. We refer these attacks as lateral movement-based attacks. We address the problem of detecting such attacks with high accuracy and a low false alarm rate.

We abstract the lifecycle of lateral movement-based attacks into the following stages: (i) initial compromise, (ii) command and control, (iii) lateral movement, and (iv) data exfiltration or service disruption. These stages are essential parts of several models of a lifecycle of an APT [7, 12, 13] and previously studied incidents [1, 8, 14].

Initial compromise consists of reconnaissance, delivery of malware, and establishing of backdoor communication. Subsequently, the attacker establishes C&C channels between the victim machines and a set of external servers, with the goal of maintaining active control over the compromised hosts. In conjunction with C&C, the attacker attempts to move laterally to widen the compromised region. The last step includes the actual execution of the malicious actions, whether they are data exfiltration, service disruption, or physical damage.

In this work, we particularly focus on C&C and lateral movement. During C&C, a set of compromised machines send automated periodic beacons to attacker-controlled servers, awaiting future instructions. These commands intend to compromise other hosts in the network or to gather intelligence and sensitive data. The Internet Relay Channel has traditionally been the most preferred communication tool for C&C, but attackers have started opting for more commonly used protocols, such as HTTP(S), FTP, and SSH, to evade easy detection. For more robust communication infrastructure, attackers may use dynamic domains and P2P server architectures.

Attacker lateral movement consists of several tactics including internal reconnaissance, credential stealing, vulnerability exploitation, and privilege escalation. In fact, the exploit techniques that are used to achieve such goals are ever-changing and evolving. Examples of such techniques include use of application deployment software to deploy malware, exploitation of vulnerability to escalate privileges, and use of pass-the-hash to bypass standard

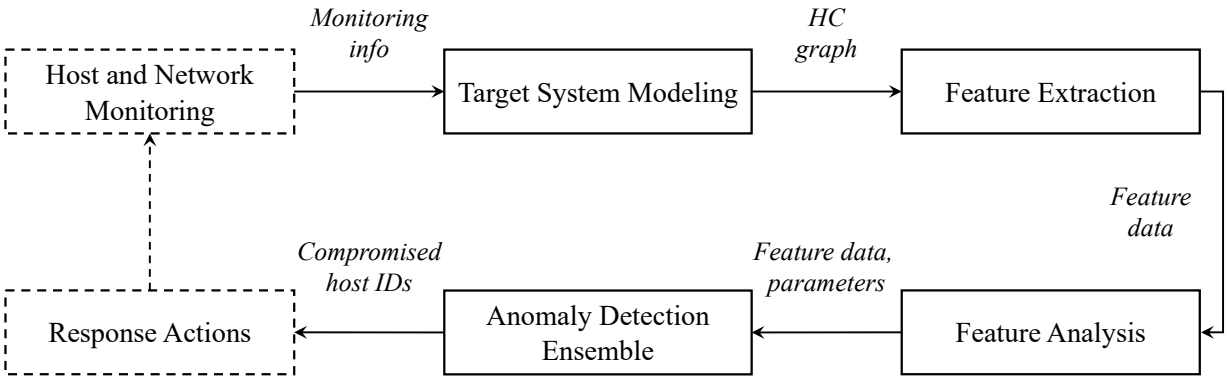


Figure 3.1: High-level architecture of our approach to detecting malicious lateral movement. We mainly focus on the modules shown in boxes with solid lines.

authentication steps [6].

We believe that since the C&C and lateral movement are closely correlated during the attacks, we can better defend systems by analyzing these two stages in a holistic way. Although significant research has been done separately on the detection of initial compromise, C&C, and data exfiltration, unfortunately, there has been relatively little research effort on the detection of attacker lateral movement. Detecting attacks in C&C and lateral movement stages can thwart an APT campaign and prevent actual damage.

3.3 OVERVIEW OF APPROACH

3.3.1 Overview

To detect the coordinated malicious activity of C&C and lateral movement, we need to construct an estimate of system-wide state such that the different types of indicators emerging from several hosts can be correlated with one another. Additionally, to act upon the system-wide state over long time windows, we need to keep the volume of information restricted, because such attacks can persist in the victim systems from over a few weeks to several months [11].

Figure 3.1 shows different components of the proposed system. We assume that the network under consideration is equipped with monitoring tools to record host- and network-level information. In this chapter, we do not build these monitors, but use the information recorded by the monitors to model the current state of the target system. We refer to this model as an *HC graph*. We use the HC graph to extract features related to C&C and LM behavior. We then standardize these features and analyze their distributions to perform

anomaly detection. In the anomaly detection step, we use a combination of different types of statistical anomaly detection techniques to detect the set of compromised hosts. The ensemble of multiple anomaly detectors allows us to achieve high accuracy while keeping the false positive rate low. Finally, we can use the results of attack detection to implement different types of attack response actions.

In what follows, we explain the three types of monitors that we use in the proposed system. The remaining modules of the proposed system are described in Sections 3.4, 3.5, and 3.6.

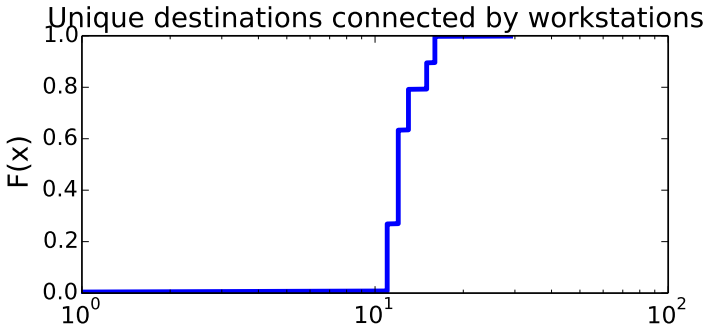
3.3.2 Monitors

Network Flow Monitor

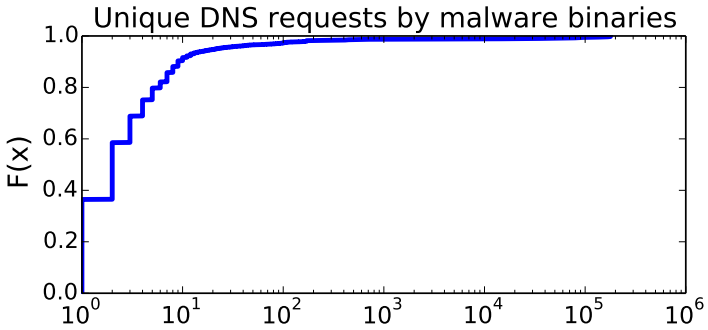
A network flow monitor processes internal network traffic and generates flow records. For our purposes, we assume that such a monitor produces a time series consisting of source and destination machines, as well as ports information. Examples of such monitors are an internal firewall, a network IDS, and a NetFlow-capable internal router.

C&C Monitor

A C&C monitor produces evidence that a host is involved in external network communication with a potential C&C server. We believe that such external traffic can be distinguished from normal traffic in a typical enterprise network. To test and validate this hypothesis, ideally, we would use a dataset consisting of both typical enterprise communications and possible C&C connections with a perfect labeling of each network flow to identify it to one of the two classes. However, such dataset is difficult to obtain in practice. Therefore, we studied the external communication behavior of hosts in two datasets, the VAST Challenge 2013 MC3 (VAST) [67] and the GT malware passive DNS data daily feeds (GT) [68]. Figure 3.2a shows the cumulative distribution of the number of unique destinations contacted by hosts over a three-day period in the VAST dataset, which has no APT-like attacks in it. We found that the workstations most often connect to a small number of external destinations. We observed that the majority of probability mass (between the first and third quartiles) lies between 8 and 17 unique destinations. Figure 3.2b shows the distribution of the number of unique DNS requests made by malware-infected hosts over the three-day period (Jan. 17–19, 2017) in the GT dataset. The malware binaries try to resolve thousands of domain names, which indicates that the infected hosts are likely to make a large number of unique external connections. We conclude that the outward communication behavior of



(a) Number of distinct destinations contacted by workstations in VAST 2013 network.



(b) Number of distinct DNS queries made by malware binaries in GT-MPDNS dataset.

Figure 3.2: Empirical CDFs.

infected hosts can be distinguished from the behavior of typical enterprise hosts.

We thus assume the presence of C&C monitors that analyze network communication of the hosts and assign a suspiciousness score to each host; we call it the `c_score`. As an example, we can compute the `c_score` simply as the number of connections that the host makes to unique destinations over the observed period. Discussion of design and analysis of the C&C monitors is beyond the scope of this work, which can be found in [69, 70].

LM Monitor

We can characterize lateral movement by a set of related chains of communications between different hosts in the network. An LM monitor creates traces of connections that are chained together based on their dependence on each other. Two types of monitors are prominently used to construct these chains: (i) network-level monitors and (ii) host-level monitors. In [71], the authors propose to correlate anomalous hosts with network events and establish attack causality in the context of epidemic-spreading attacks such as worms and botnets. Fawaz et al. [72] provide a low-overhead way to obtain a more refined view of

host behavior. This approach relies on analyzing the hosts' kernel-level activities and thus infers the connection causations more accurately. This approach is shown to work better in the lateral movement context since, in such attacks, an attacker may use known services and paths to spread through the system [72]. Therefore, in our work, we use this system as our LM monitor. To build some more context for our discussion, next, we provide a summary of that monitor.

The LM monitor [72] uses the insight that lateral movement resembles a targeted walk on a network such that sequential pairs of steps are causally related. One can track such walk in order to collect network-wide lateral movement data. To do so, one needs to detect a series of ordered network connections between hosts. To establish such order with high accuracy, one can monitor inter-process communications and create causations between incoming and outgoing connections on each host across the system.

In this approach, monitoring and fusion agents are arranged hierarchically across the system. At the lowest level, a host agent analyzes kernel-level activities to infer causation relationships between incoming and outgoing network connections. The higher-level agents, such as those at a network domain's level or the global level, use abstracted data from host-level agents, and generate a system-wide lateral movement graph.

In essence, by using kernel-level information to establish a dependency between the incoming and outgoing connections, this approach performs more accurately than those that just use timing information or port numbers. Also, distributed fusion enables lateral movement detection in large systems by spreading the storage and processing overhead among multiple domains. We refer the reader to [72] for more details on the working of such a monitor.

The techniques mentioned above can establish dependence relations between different network connections with varying levels of accuracy. They do, however, fall short of distinguishing between benign and malicious hosts. The size and complexity of modern systems and the lack of knowledge about attacker activities make the detection of malicious chains challenging. Our approach addresses these challenges by correlating lateral movement and C&C indicators to identify malicious events.

3.3.3 Target System Model

We use a graph-based model of the target system to combine the information collected by the three monitors: internal network communication, C&C, and lateral movement activity. We represent the model as a directed graph $G(V, E)$, called the *host communication* (HC) graph. The set of vertices, $V = \{1, 2, \ldots, n\}$, represents the hosts in the system. The set of edges, $E \subseteq V \times V$, represents the network communication between the hosts. E consist of

all types of communications happening in the network, including normal system operation and attacker actions. In practice, an HC graph can be generated, for example, by analyzing NetFlow records or firewall logs of the internal network traffic.

We define a vertex labeling function as $f_c : V \rightarrow [0, 1]$, which labels every vertex of the graph G with its `c_score` between 0 and 1. The `c_score` quantifies the extent to which a particular node is under C&C attack. This approach allows us to model different C&C attack patterns by generating different `c_score` distributions over the graph G.

We next define a lateral movement graph as $l(V_l, E_l) \subseteq G$ such that $\exists u, v, w \in V_l$ if the connections (u, v) and (v, w) are chained together by the LM monitors. Graph l represents a separate LM behavior. $\mathcal{L}$, which is the set of all these graphs, represents all types of lateral movements in the system. It is important to note here that we do not distinguish between benign and malicious lateral movement at this point.

The HC graph thus allows us to combine diverse indicators and construct a system-wide state.

3.4 DETECTING LATERAL MOVEMENT-BASED ATTACKS

The proposed lateral movement detection approach is as follows. First, we identify the features that are most useful for detecting the attacks. We then extract the features using data collected via the monitors and the HC graph. We standardize the feature values and analyze their distributions and finally use an ensemble of two anomaly detectors to identify compromised hosts.

3.4.1 Feature Extraction

We extract the following features to represent the current state of every host:

C&C Score

The first feature extracted ($\mathbf{f}_{cs}$) is the `c_score` of a host that is generated by the C&C monitor. Using $\mathbf{f}_{cs}$, we aim to identify suspicious external communication of hosts.

Lateral movement-related features

The next set of features is based on the lateral movement graphs that are generated by the LM monitor. Each lateral movement graph is a subgraph of the overall host communication

graph G.

The second feature is the number of lateral movement graphs ($\mathbf{f}_{num}$) passing through a host. We formulate the extraction of $\mathbf{f}_{num}$ as a set of matrix operations. Consider a lateral movement graph $l = (V_l, E_l) \in \mathcal{L}$ represented by an $n \times n$ adjacency matrix M, where n is the number of nodes in the graph G. We compute the feature vector $\mathbf{f}_{num}$ as follows:

$$\mathbf{f}_{num}(i) = \sum_{l \in \mathcal{L}} (\bigvee_{j} \mathbf{M}_{ij} \vee \bigvee_{j} \mathbf{M}_{ji}),\ i, j \in \{0, 1, \ldots, n-1\} \tag{3.1}$$

Equation 3.1 first determines whether a host i is part of a chain l. To do that, it performs a Boolean sum of all the elements in i^{th} row and i^{th} column of the matrix $\mathbf{M}$ to generate a binary output. A result 1 then indicates the node is part of chain l. Finally, to obtain the number of chains that pass through the node i, the equation computes an arithmetic sum of the binary outputs for all $l \in \mathcal{L}$. By following the same process for each host, we obtain the $(n \times 1)$ vector $\mathbf{f}_{num}$.

Next, we extract four features, (i) $\mathbf{f}_{mean}$, (ii) $\mathbf{f}_{range}$, (iii) $\mathbf{f}_{iqr}$, and (iv) $\mathbf{f}_{mad}$, corresponding to the mean, range, interquartile range (IQR), and median absolute deviation (MAD) of the lengths of lateral movement graphs, respectively. The motivation for including these features is that in addition to the disparity induced in the number of graphs passing through different hosts, attacker lateral movement is also likely to cause variations in the sizes of these graphs. We believe that these four features will enable us to adequately summarize the properties of lateral movement graphs to separate benign and malicious lateral movements.

We start by computing mean and range to account for the central tendency and dispersion of the distribution of lengths of lateral movement graphs, respectively. We define the length of a lateral movement graph by the number of vertices ($|V_l|$). To compute $\mathbf{f}_{mean}$, we sum together the lengths of the graphs passing through each host and then use the $\mathbf{f}_{num}$ feature vector to compute the average. Similarly, for each host h, $\mathbf{f}_{range}$ is simply the difference between the max and the min values of the lengths of the chains passing through h.

$\mathbf{f}_{mean}$ and $\mathbf{f}_{range}$ help in comparing characteristics of different hosts, but they are overly sensitive to outliers. Also, since the distribution of lengths of lateral movement graphs passing through a given host may not be a Gaussian distribution, standard deviation (or variance) will not be a suitable statistic to compute. To obtain the insights missed by $\mathbf{f}_{mean}$ and $\mathbf{f}_{range}$, we also compute IQR and MAD.

The IQR of a sample, also known as the 25% trimmed range, is defined as the difference between the 75th and 25th percentiles of the sample. Compared to the range, which is based solely on the two most extreme values, IQR measures the dispersion of the central 50% of

samples, and provides insights that cannot be obtained from the range alone.

The MAD of a dataset is defined as the median of the absolute values of the differences between the individual samples and the overall median. For a host h, let $\mathbf{len}_h$ be the vector of the length of the LM chains passing through h. We compute the value of feature $\mathbf{f}_{mad}$ as follows:

$$\mathbf{f}_{mad}(h) = med\left(\{|len - med(\mathbf{len}_h)|, \text{for } len \in \mathbf{len}_h\}\right) \tag{3.2}$$

where $med()$ is the median of a sample. In summary, we use the information in the host communication graph to extract six $n \times 1$ feature vectors $\mathbf{f}_{cs}$, $\mathbf{f}_{num}$, $\mathbf{f}_{mean}$, $\mathbf{f}_{mad}$, $\mathbf{f}_{iqr}$, and $\mathbf{f}_{range}$.

3.4.2 Feature Analysis

After the extraction, we scale each feature vector using its maximum absolute value. This method does not shift the centroid of the data, and thus, it does not eliminate the sparsity present in the feature vectors. Since our feature vectors have all positive values, the scaling results into feature values within the range $[0, 1]$.

Figure 3.3 shows the distributions of the scaled feature vectors for experiment configuration E1.3 (ref. Table 3.2). The distributions of the features $\mathbf{f}_{cs}$, $\mathbf{f}_{iqr}$, $\mathbf{f}_{range}$, and $\mathbf{f}_{mad}$ show a positive skew as the mass of distribution is concentrated on the left of the plots. The distribution of $\mathbf{f}_{mean}$, on the other hand, has a negative skew. A possible reason for the negative skewness of $\mathbf{f}_{mean}$ may be the fact that a larger variability in the lengths of chains affects the mean more than it affects the IQR and MAD. A few very long chains (probably benign) can significantly skew the distribution of the mean. For different types of attack, the compromised hosts can be in different regions. For example, if an attacker compromises a small number of hosts with a C&C malware, these hosts are then likely to have larger values of $\mathbf{f}_{cs}$ (e.g., greater than 0.5 in Figure 3.3a). As another example, if the attacker generates several LM chains of different lengths, then the hosts that have higher values of $\mathbf{f}_{num}$, $\mathbf{f}_{iqr}$, and $\mathbf{f}_{range}$ are more likely to be compromised, since the normal nodes should have fewer chains, most of which will be of similar lengths.

Next, we perform PCA to analyze the correlation among the features and reduce the dimensionality of the dataset. During all the experiments we performed, we could project the data to a two-dimensional space (using the first two PCs) while retaining more than 95% of the original variance in the data. In addition to improving the performance of analyses, the lower-dimensional data also helps in visualizing the distributions. We use these two properties to build an anomaly detection method and select values for different

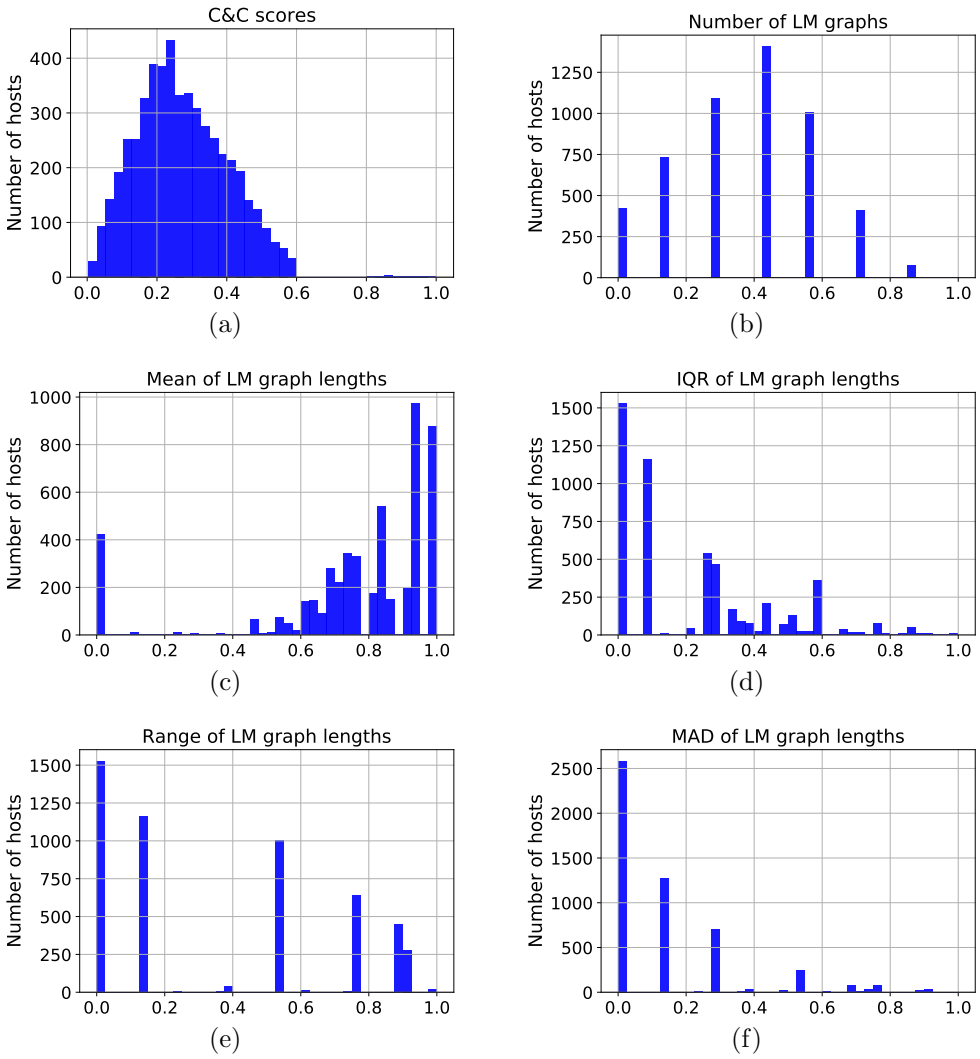


Figure 3.3: Distributions of scaled feature values computed using the host communication graph from a network of $5,151$ computers that we used in the experiments.

thresholds required during the process. We provide more details on the specific experimental configurations in Section 3.5.

3.4.3 Anomaly Detection Methodology

We now present an approach to classify the network hosts as normal or compromised. Doing so involves two important considerations. First, we aim to build unsupervised techniques because of the unavailability of labeled datasets of network traffic. Second, we seek a classification method that performs accurately even when attacks deviate from the assumed pattern. Although we use an assumed threat model as a guiding template to build the de-

fense mechanism, we do not make strong assumptions related to the size of attack or the sequence of the attacker's actions.

The traditional anomaly detection methods that assume that the size of the anomaly (e.g., the number of compromised hosts) is significantly smaller than the size of normal activity do not fit directly in our case [53, 73, 74]. We thus propose to use a combination of multiple anomaly detection algorithms to achieve high accuracy and robustness while facing variations in attacker behavior. Intuitively, the approach is first to find the hosts that exhibit at least one of the two types of suspicious indicators—high `c_score` or high disparity in lateral movement chains—and then use the relative similarity of hosts to identify compromised hosts that do not directly show any of the suspicious indicators. Our hypothesis is that the features will help us perform both tasks.

Anomaly Detection using PCA and k-means

First, we perform PCA using the five lateral movement-based features. In the evaluation dataset that we use, the first two principal components (PCs) capture more than 95% of the original variance in the data. Therefore, we project the original data to a two-dimensional space. We then use the k-means algorithm to form clusters of similar data points. To select the optimal value of k, which is the number of assumed clusters in the dataset, we use silhouette analysis [51]. The silhouette value of a data point is a measure of how well it lies within its cluster as compared to the other clusters. We start by choosing a range of possible values of k. For each value of k, we perform the clustering and compute the average silhouette score of each cluster, which can range from -1 to 1; values closer to 1 indicate well-separated and cohesive clusters. We thus choose the value of k that results in the maximum average silhouette score for all points.

The next task is to identify clusters that contain compromised hosts. Since we perform clustering using lateral movement-related features, we use suspicious C&C behavior for the identification. We use MAD-based anomaly detection on $\mathbf{f}_{cs}$. MAD has a 50% breakdown point; the estimate remains bounded when fewer than 50% of the data points are replaced by arbitrary numbers. Because of this high stability in the presence of outliers, MAD can be used as a tool for anomaly detection [75]. The MAD-based outlier score is computed as follows:

$$score(i) = \frac{|\mathbf{f}_{cs}(i) - median(\mathbf{f}_{cs})|}{MAD(\mathbf{f}_{cs})} \tag{3.3}$$

Equation 3.3 generates a numeric score for each point in the vector $\mathbf{f}_{cs}$. We then flag all the points that have values greater than a threshold (θ_{mad}) to represent anomalies. Finally,

we identify the clusters that cover at least 90% of the anomalous points and output all the hosts that are part of these clusters.

Anomaly Detection using PCA and Extreme Value Analysis

In the second method, we identify anomalous hosts using indicators of suspicious lateral movement. Our first insight is that the compromised hosts will have a large number of chains of different lengths passing through them. Similar observations have been presented in [71, 76, 77]. For example, Sekar et al. [71] found that the communication graph generated by normal hosts is sparse and significantly follows a client-server model. Therefore, a presence of diverse graph-structures in the host communication pattern is suspicious. Such behavior will result in large values for features $\mathbf{f}_{num}$, $\mathbf{f}_{iqr}$, and $\mathbf{f}_{range}$. However, it is relatively difficult to reason about the values in three-dimensional space, and since these three features are highly correlated, we perform PCA. We find that the first PC alone captures more than 90% variance. We project the data on the first PC and analyze the distribution. Doing this allows us to easily filter out the points that have large values on the first PC. These points correspond to the anomalous hosts. Specifically, we consider the points that have values greater than a threshold ($\theta_{pc.low}$) to be anomalous.

Our second insight is that the benign hosts should be part of either no chains or benign chains that originate from an administrative activity. From the viewpoint of $\mathbf{f}_{mean}$ and $\mathbf{f}_{mad}$, these two cases should result in either very small or very large values for these features. For example, admin workflows such as automated scripts in Windows Management Instrumentation and patch management via Windows Update Server would result in large graphs originating from the admin server. Conversely, the suspicious hosts should be found in the middle portion of the distributions. Following the same approach as in the previous case, we project data on the first PC after doing PCA on $\mathbf{f}_{mean}$ and $\mathbf{f}_{mad}$. This provides us with the ability to easily identify extreme values. Specifically, we compute the mid-range of the data projected on the first PC and flag as anomalous all the hosts that lie within a threshold ($\theta_{pc.mid}$) distance from the mid-range.

The final output of the second anomaly detector is the intersection of the two sets of hosts obtained above. These hosts exhibit both indicators of a suspicious lateral movement.

Combining the Outputs of Two Anomaly Detectors

We presented two detectors that use different types of indicators to identify anomalous hosts. Since (i) we do not have access to the true labels to train the anomaly detection and

(ii) we cannot assume the number of compromised hosts for either C&C or lateral movement attacks, we need to rely on an ensemble of the two detectors to achieve high accuracy.

Algorithm 3.1 shows the proposed parametric ensemble method. Our approach is inspired by the work presented in [78]. The algorithm takes as input (i) the two sets of anomalous hosts ($\mathcal{A}$ and $\mathcal{B}$); (ii) the number of hosts in the network (N); and (iii) a step size ($step$). The main parameter of the algorithm is $frac$, which denotes the fraction of points to select from $\mathcal{A}$ and, correspondingly, the $(1 - frac)$ fraction of points to select from $\mathcal{B}$.

The approach to select the points from $\mathcal{A}$ (in line 10) is based upon the *goodness* of each point, which is equal to the silhouette score of the point with respect to the clusters formed by the first detector. Therefore, *goodness* is a score within the range $[-1, 1]$, and a value closer to 1 is better. We then select remaining points from set $\mathcal{B}$ at random (line 15). For every point selected by this method, we increment the corresponding value in the output set $\mathcal{F}$. The variable *step* decides the number of times the main loop executes, which is equal to $\lfloor \frac{1}{step} \rfloor$. When the algorithm ends, the values that we get in $\mathcal{F}$ indicate the number of times a point was flagged as anomalous. Finally, to classify a point as normal or compromised, we use the threshold θ_{avg}. A smaller value of θ_{avg} would classify a large number of hosts as compromised, and would also lead to a higher false positive rate. During the evaluation of the proposed approach, we used the results of the ensemble algorithm with different values of θ_{avg} to analyze the trade-off between accuracy and the false alarm rate.

Parameter Selection for Anomaly Detection Algorithms

In addition to deciding an optimal value of k for clustering, we need the following parameters for anomaly detection. The first is the threshold of the MAD-based outlier score (θ_{mad}) above which we consider a `c_score` anomalous. We always use a value 2 for θ_{mad}. Since MAD is a very robust measure of dispersion, a value of 1.5 or above for the outlier score (Eq. 3.3) works well in most applications [75]. By choosing a value of 2, we pick a relatively strict threshold and ensure that the flagged points are really outliers. Next, the second anomaly detector needs two thresholds, $\theta_{pc.low}$ and $\theta_{pc.mid}$. To decide on good values of these two thresholds, we perform PCA on the dataset for different settings and analyze the distribution of the data projected on the first PC. We find that a value of 0.25 for each of the two thresholds provides a good separation of different groups of points.

Algorithm 3.1 Anomaly Detection Ensemble

Input: $\mathcal{A} \leftarrow$ detector1 output; $\mathcal{B} \leftarrow$ detector2 output; N; *step*
Output: $\mathcal{F}$

$l_1 \leftarrow |\mathcal{A}|$
$l_2 \leftarrow |\mathcal{B}|$
$frac = 1.0$
$\mathcal{F} = [0]_{1 \times N}$
while $frac \geq 0.0$ **do**
    $n_1 \leftarrow frac * l_1$
    $tmp \leftarrow$ getBest $(\mathcal{A}, n_1)$
    for all $i \in tmp$ **do**
        $\mathcal{F}[i] \leftarrow \mathcal{F}[i] + 1$
    end for
    $n_2 \leftarrow (1 - frac) * l_2$
    $tmp \leftarrow$ getRandom $(\mathcal{B}, n_2)$
    for all $i \in tmp$ **do**
        $\mathcal{F}[i] \leftarrow \mathcal{F}[i] + 1$
    end for
    $frac \leftarrow frac - step$
end while

3.5 EXPERIMENTAL EVALUATION

We performed experiments to evaluate the accuracy of the proposed anomaly detection approach in identifying compromised hosts. The experiments used internal network flow logs obtained from the Los Alamos National Lab's (LANL's) Comprehensive Muti-Source Cyber-Security Events dataset [65]. The complete dataset contains anonymized traces of 58 consecutive days from LANL's enterprise network, which is composed of 17,684 host computers. We used network flow logs only for the first day; they include a total of 8,096 active hosts and 73,150 network flows among the hosts. The dataset does not contain any traces of lateral movement-based attacks. We used the LANL NetFlow logs and injected C&C and lateral movement traces to generate the HC graph.

3.5.1 Host Communication (HC) Graph Generation

During the analysis of the NetFlow records, we found that out of a total of 8,096 hosts, only 5,151 were part of a single connected component. More than 99% of the remaining 2,945 hosts did not connect to other hosts over the one-day period. Therefore, we considered the large connected cluster of $n = 5151$ hosts and constructed a communication graph consisting

only of the NetFlow information. Furthermore, since the proposed approach does not require knowledge of port numbers and flow sizes, we aggregated all the flows between a pair of hosts as a single edge in the generated NetFlow graph.

Next, we augmented the NetFlow graph with simulated attack traces. As mentioned earlier, we assume the presence of benign LM activity in the network, such as patching and Windows Management Instrumentation traffic. Our distinction between benign and malicious LM traces was based on the observation that the APT attackers are more targeted towards escalating their privileges and moving deeper into the network. The attackers' spread in the network is slower than that of benign administrative tasks because the attackers need to avoid detection by having their malicious traffic masquerade as benign traffic.

To further reason about the graph structure generated from the malicious LM, we analyzed the attacker's behavior in commonly-used LM tactics [6]. We found that most of the LM tactics would lead to a general graph structure, one which is not restricted to any specific topology, e.g., a simple chain, a closed chain, or a star. Examples of those tactics are Apple scripts, application deployment software, vulnerability exploitation, pass-the-hash (ticket), and remote file copy. We found only a few tactics to produce a star graph structure, such as logon scripts and taint shared content. Also, none of the tactics were restricted to a chain-like structure. With that analysis, we concluded that to generate traces of malicious LM, we would need to consider a model that allows simulating a general graph structure.

Now we describe the approach for generating $\mathcal{L}$ chains of communication, out of which we consider $\mathcal{M} \subseteq \mathcal{L}$ to be malicious. Each chain $l \in \mathcal{L}$ is a subgraph of the NetFlow graph generated earlier. We generate the communication chains (both benign and malicious) according to the continuous-time N-intertwined *susceptible-infected-susceptible* (SIS) model [66], which is widely adopted by researchers for modeling the spread of epidemics and computer worms and viruses [79].

Given a communication graph $G(V, E)$, we represent the state of each vertex $v \in V$ with a Bernoulli random variable $X_v \in \{0, 1\}$ such that for each chain $l(V_l, E_l) \in \mathcal{L}$, $X_v = 1$ iff $v \in V_l$. A node $v \in V$ is part of a chain l at time t with probability $p_v(t) = P[X_v(t) = 1]$, and is healthy with probability $1 - p_v(t)$. Once a chain l passes a node v, spread from v to a neighboring node w occurs at an exponential rate of β_m for malicious chains and β_b for benign chains. The spreading rate per edge is thus a Poisson process with rates β_m and β_b for malicious and benign chains, respectively. In fact, the N-intertwined SIS model can be represented as a continuous-time Markov chain (CTMC) with $2^{|V|}$ states. We assume that $\beta_b > \beta_m$ since attackers tend to be slower and stealthier than administrators as previously discussed.

Figure 3.4 shows a sample lateral movement chain, along with its equivalent CTMC.

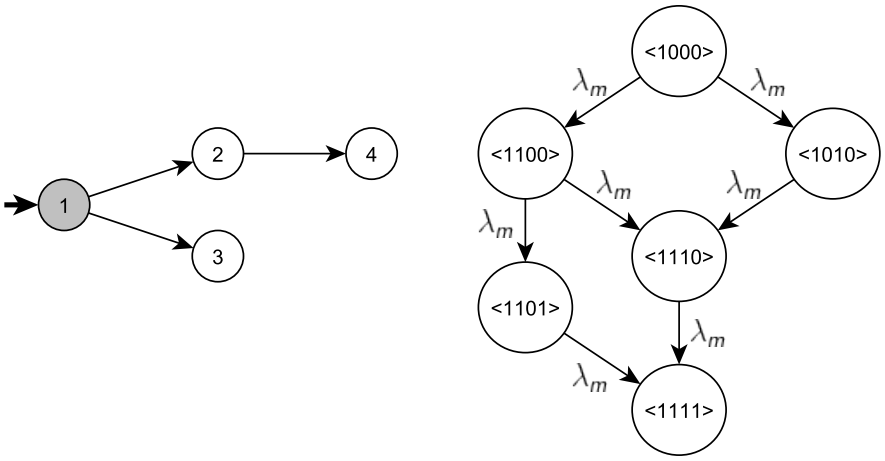


Figure 3.4: Example of an LM chain and its corresponding CTMC.

The initial state corresponds to the attacker's establishing a point of entry into the system through node 1 (or an admin's starting of a machine, in the case of a benign chain). The CTMC then captures the attacker's possible moves and corresponding rates, according to the N-intertwined SIS model. The attacker can compromise either node 2 or node 3, each with a rate of compromise λ_m. For a fully connected graph with n nodes, the size of the CTMC is 2^n, since from any node, the attacker can compromise any other node. However, in practice, network graphs are not fully connected (given segmentation into domains), and thus the state space would be significantly reduced. For the example shown in Figure 3.4, since the attacker can only reach node 4 from node 2, the state where node 4 is compromised cannot be reached from any state where node 2 is not compromised.

To capture the correlation between C&C and lateral movement, we start the trace generation with an initial distribution of `c_scores` of hosts, such as uniform random or triangular distribution. During the lateral spread, we adopt a probabilistic update of `c_score` values. In particular, we set the `c_score` of a newly infected destination node as the maximum of its old `c_score` and the source's `c_score` with a probability p_{cs}. Different values of p_{cs} allow us to simulate different types of C&C behaviors of attackers. We also make sure that each node $v \in \mathcal{V}$ that has a very high `c_score` (e.g., 0.8 or more) has at least one chain passing through it.

We show the attack trace generation in Algorithm 3.2. The simulation approach that we used was inspired by the work presented in [80]. At any time step t, let k be the number of vulnerable nodes to compromise; the simulation would then generate a random variable τ with an exponential distribution with rate $k \times \beta_m$ for a malicious chain, and $k \times \beta_b$ for benign chains. We would then advance the simulation time to $t + \tau$, sample a node w to compromise from the k susceptible nodes uniformly at random (i.e., each node would have

Algorithm 3.2 LM Chain Generation

Input: $G(V, E)$, v_0, λ, $is\_malicious$
Output: $C(V_c, E_c)$

$Visited(v) := \bot \quad \forall v \in V$
$V_c := \phi, \quad E_c := \phi, \quad C :=< V_c, E_c >$
$Visited(v_0) := \top$
$\mathcal{E} := \{e \in E \mid e = (v_0, w) \in E\}$
Let $len = length(\mathcal{E})$
while not done **do**
    Sleep for $\tau \sim Exp(len \times \lambda)$
    Choose $e = (u, w)$ target edge from $\mathcal{E}$ w.p. $1/len$
    if $is\_malicious$ **then**
        Compromise w w.p. $p_c(w)$
        if not successful **then**
            continue
        end if
    end if
    $Visited(w) := \top$
    $V_c := V_c \cup \{w\}$
    $E_c := E_c \cup \{e\}$
    $\mathcal{E} := \mathcal{E} \setminus \{e' = (v, w) \in \mathcal{E} \text{ for any } v \in V_c\}$
    $\mathcal{E} := \mathcal{E} \cup \{e \in E \mid e = (w, v) \in E \wedge (\neg\, Visited(v))\}$
end while

a probability of $1/k$ of being selected), and then add w and its corresponding edge to the generated chain. All neighbors of the newly compromised node w would then be added to the set of susceptible nodes. We note that in the case of a malicious chain, the compromise of the node w could fail with probability $1 - p_c(w)$, so we restart the simulation from the same state at time $t + \tau$.

Attack trace generation thus generates an HC graph consisting of network flows, lateral movement chains (both benign and malicious), and `c_scores`.

3.5.2 Results

We parsed the LANL dataset and implemented in Python the code for attack trace generation, feature extraction, feature analysis, and anomaly detection. We used the implementations of PCA and k-means provided by Python's scikit-learn machine learning module [81].

As we mentioned in Section 3.4.3, the traditional anomaly detection methods (e.g., PCA-based and clustering-based) do not directly fit since we cannot assume the number of com-

Table 3.1: Summary of the simulation parameters

Parameter	**Parameter**
Number of nodes ($n = 5,151$)	Initial `c_score` distribution*
Benign rate ($\beta_b = 0.8$)	Initially compromised nodes ($k = 15$)
Malicious rate (β_m)*	Compromise success probability (p_c)*
Update score probability (p_{cs})*	Simulation time ($T = 10$)
Simulation runs ($runs = 100$)	

Table 3.2: Configurations for the first experiment (E1)

Configuration	**E1.1**	**E1.2**	**E1.3**	**E1.4**	**E1.5**
β_b/β_m	8.0	4.0	2.0	1.3	1.0
# nodes in benign LM	4369	4418	4342	4325	4294
# nodes in malicious LM	66	326	1403	3594	4504

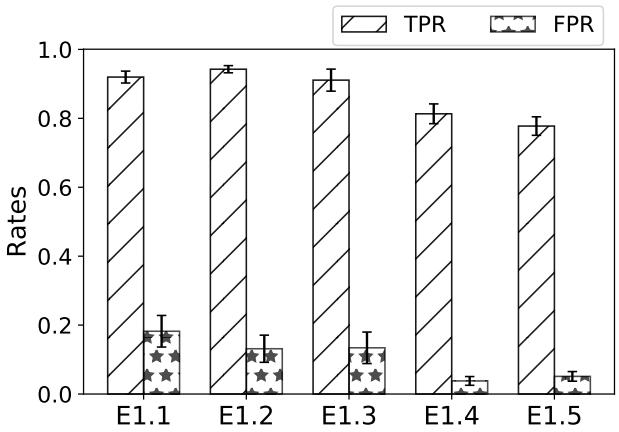
promised hosts to be very small as compared to the number of benign hosts. Moreover, to the best of our knowledge, there are no unsupervised approaches proposed previously to detect lateral movement-based attacks. Therefore, we were unable to compare our results with other methods. Instead, we focused on measuring the effect of changing attacker behavior on the accuracy of detecting compromised hosts. We evaluated the true and false positive rates (TPR and FPR) in two different experimental configurations (configs) that we created by using different values of the model parameters marked with an asterisk in Table 3.1. To compute TPR and FPR, we used the ground truth obtained from the attack injection. We consider a host to be benign only when no malicious chain traverses it.

Varying the Number of Nodes in Malicious LM Chains

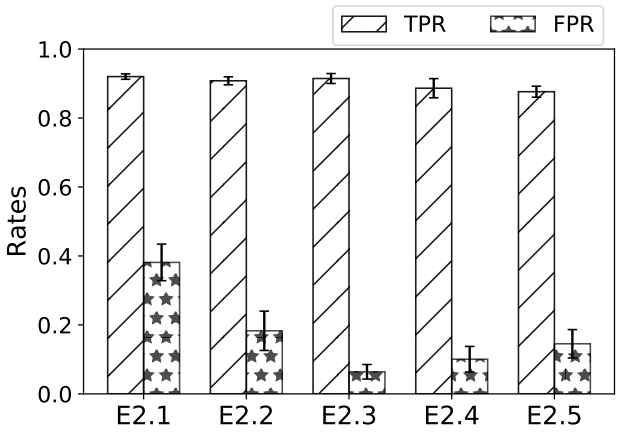
In the first experiment, we varied β_m while keeping β_b fixed. We also set both p_c and p_{cs} equal to 0.2. This experiment corresponds to the different speeds with which the attacker moves laterally by compromising network hosts. We show the resulting configs in Table 3.2.

The proposed approach was able to detect the compromised hosts with high accuracy, as shown in Figure 3.5a. For example, there were 66 compromised hosts in config E1.1, out of which 60 hosts were detected by the proposed approach. In config E1.3, in which a larger number of hosts were part of malicious chains, the average FPR was only 13.13%, while an average TPR of 91.08% was still achieved.

Although the average FPR is relatively high in these cases (i.e., 18.21% in E1.1 and 13.40% in E1.3), we can tune the classification threshold (θ_{avg}) of the ensemble algorithm to decrease the FPR. We implemented the ensemble approach from Algorithm 3.1 with $step = 0.1$ to



(a) Changing the number of hosts in malicious LM chains.



(b) Changing the number of C&C-infected hosts.

Figure 3.5: TPR and FPR of detecting compromised hosts.

generate ten sets of output labels. We then varied the θ_{avg} from 0.1 to 1.0 and computed TPR and FPR in each case. We show the results for config E1.3 as an receiver operating characteristic (ROC) curve in Figure 3.6a. For the case in which a host was considered compromised once it had been flagged as anomalous by all ten runs of the ensemble detector ($\theta_{avg} = 1.0$), the average TPR and FPR were 54.84% and 1.52%, respectively. Both the TPR and FPR increased as we decreased the threshold by 0.1 in each experiment. Finally, with $\theta_{avg} = 0.1$, we got TPR = 92.39% and FPR = 14.06%. These results provide insights that will help us select a desirable balance for TPR and FPR.

Varying the Number of C&C-Infected Nodes

Table 3.3 shows the configs generated when we increased the probabilities of successfully moving to a host (p_c) and compromising it with C&C malware (p_{cs}). We used $\beta_b/\beta_m = 4.0$.

Table 3.3: Configurations for the second experiment (E2)

Configuration	**E2.1**	**E2.2**	**E2.3**	**E2.4**	**E2.5**
$p_c = p_{cs}$	0.2	0.4	0.6	0.8	1.0
# nodes in benign LM	4407	4360	4494	4410	4448
# nodes in malicious LM	87	342	850	2072	2422

Here we model the attackers that can install C&C malware on several hosts, which would then start connecting to potential C&C servers. Therefore, the identification of compromised hosts mainly happens through use of outlier `c_scores`. The number of hosts that are part of malicious LM chains, however, does not increase as fast as in experiment E1.

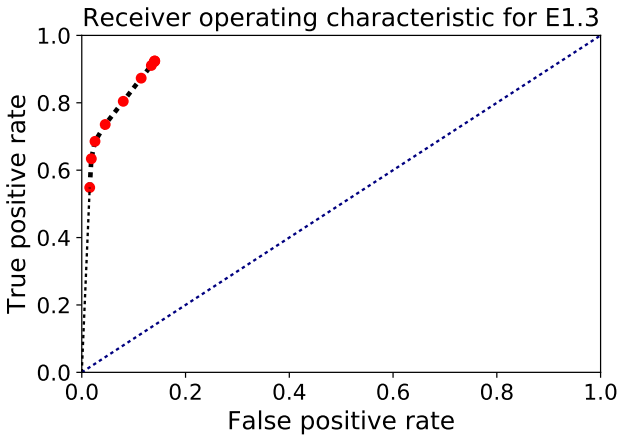
We show the results for average TPR and average FPR in Figure 3.5b, and a ROC curve for varying the classification threshold θ_{avg} for config E2.3 in Figure 3.6b. We achieved an average TPR of 90.10% across all the configs. The average FPR is relatively high in config E2.1 when both the C&C and the lateral movement indicators are not strong. The FPR goes to an acceptable level (e.g., 6.39 in E2.3 and 10.05 in E2.4) because the C&C indicators are visible in the network.

In summary, by using an ensemble of the two anomaly detectors, we achieved an overall average TPR (computed over all experiments) of 88.7% even when attacks happened with different intensities. We achieved low FPR when at least one type of suspicious indicators was present in the network. The classification threshold is a tunable parameter, and we can adjust it to decrease the FPR at the cost of a small decrease in the TPR.

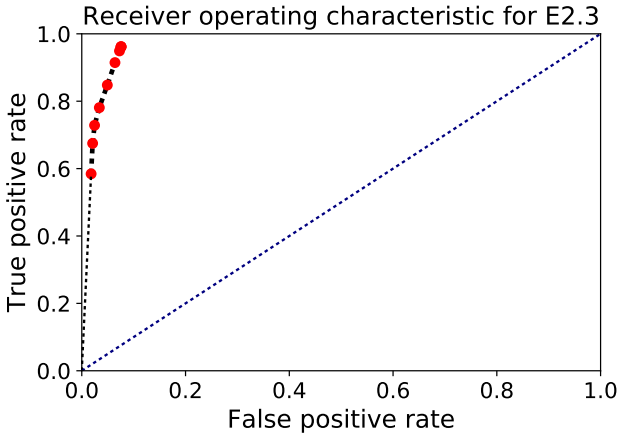
3.6 LIMITATIONS AND POTENTIAL SOLUTIONS

The proposed approach can identify compromised hosts by using simple statistical features related to C&C and LM indicators. However, we lack context information from a real-world enterprise system. We believe that we can further improve the approach by including information related to topology and segregation architecture, users, and different host-types. Access to such information would allow us to white-list the known normal behavior, extract features related to the underlying network, and evaluate the performance of the proposed approach under dynamic network workload.

Lateral movement occurs after a successful initial compromise of the system; a well-equipped attacker may be able to act quickly on a target, even before sufficient data have been collected for our system to detect the attack. We can easily extend the approach and push the anomaly detection to early in the progression of an APT. Specifically, we can use the features to detect targeted malicious e-mails [82] and inbound network scan traffic [83]



(a) Config E1.3.



(b) Config E2.3.

Figure 3.6: ROC plots with changing average label classification threshold from 1.0 to 0.1.

to correlate the indicators of initial compromise with those of lateral movement.

Like any other network-based anomaly detection system, the proposed approach would fail to detect an attack that results in no behavioral changes. For instance, if an attacker can perfectly replicate typical enterprise communication behavior during its C&C and regular network services during its LM, the approach will not be able to detect it. However, we believe that it is infeasible for an attacker to hide all the malicious traces and go undetected.

Finally, we address the problem of the lack of validation data by using the proposed attack trace generation method. This approach allows us to model a number of attacker behaviors and test our approach. The proposed anomaly detection shows good performance since (i) the feature extraction is based on simple matrix operations and (ii) the sizes of the feature vector are bounded by the number of hosts in the network. However, we want to test the time it takes to detect an attack from the point when the initial compromise happened. We leave such evaluation to future work.

3.7 RELATED WORK

3.7.1 Advanced Threat Detection

Researchers have primarily focused on detecting individual stages of an APT. For example, Amin et al. [82] used a random forest classifier to identify coordinated and persistent campaigns of malicious emails that are used by APT actors to facilitate computer network exploitation. Smutz et al. [84] used features related to document metadata and structure to detect malicious PDF documents. Opera et al. [70] proposed a C&C detection system that identifies compromised hosts and suspicious external domains by using a belief propagation algorithm. Hagberg et al. [76] studied authentication graphs in a large network to assess the risk with respect to credential-hopping attacks. Johnson et al. [85] presented an approach to measure the potential vulnerability of a network to LM attacks. Chen et al. [86] proposed to integrate multiple graph-based feature using PCA and graph dictionary learning to detect cyber intrusions.

3.7.2 Correlation of Anomaly Detectors

Several recent papers have also explored the idea of using diverse information to detect botnet and worm-based attacks. To detect bot-infected hosts, BotHunter [83] uses correlation of multiple network-level anomaly and signature-based IDSes, whereas BotMiner [87] performs clustering and correlation of indicators observed at different hosts. Sekar et al. [71] proposed to temporally correlate host-based anomaly detection results with network-related events to establish causality among the events that happen during the propagation of computer worms.

Unlike the epidemic-spreading attacks, APT campaigns are slow and silent, particularly in the phases after initial compromise. APT actors maintain a low profile while moving laterally within a victim system. Therefore, we need to build defense methods specifically tailored towards APTs. Our approach correlates the features related to C&C and lateral movement and detects compromised hosts by using unsupervised anomaly detection methods.

3.7.3 Response

To respond to an attack, a human security analyst or an automatic response selection module [88, 89] can use the proposed system's results. Examples of possible responses include blocking a suspected remote address, isolating a subset of hosts that are part of

malicious chains from rest of the network, and learning the target of an attack by actively disturbing the lateral movement chains and evaluating the changes in attacker's path.

3.7.4 Anomaly Detection

We detect attacks in an unsupervised manner that relies on statistical anomaly detection. The anomaly detection techniques that inspired our approach are presented in [53, 74, 75, 78]. In contrast, some of the related work has adopted supervised techniques, which suffer from the lack of training data. For example, in the work of Opera et al. [70], the detection of C&C communication relies on a supervised linear regression and requires external information about domain registration.

3.8 CONCLUSION

In this chapter, we presented an unsupervised multi-detector approach to detecting lateral movement-based attacks in enterprise systems. With it, we extracted useful features using NetFlow, C&C, and specialized lateral movement monitors. We then introduced a graph-based model to combine the diverse features to evaluate the current security state of the system. We showed that by using the proposed features with an ensemble of PCA, k-means clustering, and extreme value analysis enables the identification of compromised hosts in an unsupervised manner. The accuracy of detection was about 88.7% and always stayed above 82.6%, even in extreme deviations of the attacker from expected behavior. In each case, the number of false positives was very small. We evaluated how sensitive the approach was in detecting attacks as attacker capabilities change. The insights that we obtained can be used by researchers to test new defense mechanisms before deploying them on real systems.

In conclusion, we demonstrated the validity of the thesis statement in the context of enterprise systems; we showed that our information-fusion-based detection methods outperformed single detectors. In particular, Chapter 2 described their superior accuracy and smaller volume of alerts, and Chapter 3 described an accurate detection of a novel attack. In both cases, we utilized the correlation of diverse monitoring information and developed unsupervised anomaly detection systems. The work in this chapter has been peer-reviewed and was published in [90].

CHAPTER 4: SPECIFICATION-BASED DETECTION OF GOOSE POISONING

Having discussed ways in which we improve the detection of advanced cyber threats in the context of vectors of initial compromise (Chapter 2) and lateral movement (Chapter 3), we now discuss a method for improved detection of service disruption. In particular, we study cyber attacks on IEC 61850 substations by a man-in-the-middle adversary.

Devices in IEC 61850 substations use the generic object-oriented substation events (GOOSE) protocol to exchange protection-related events. Because of its lack of authentication and encryption, GOOSE is vulnerable to man-in-the-middle attacks. An adversary with access to the substation network can inject carefully crafted messages to impact the grid's availability. One of the most common such attacks, GOOSE-based poisoning, modifies the `StNum` and `SqNum` fields in the protocol data unit to take over GOOSE publications. In this chapter, we present ED4GAP, a network-level system for efficient detection of the poisoning attacks. We define a finite state machine model for network communication concerning the attacks. Guided by the model, ED4GAP analyzes network traffic out-of-band and detects attacks in real-time. We implement a prototype of the system and evaluate its detection accuracy. We provide a systematic approach to assessing bottlenecks, improving performance, and demonstrating that ED4GAP has low overhead and meets GOOSE's timing constraints.

4.1 SUMMARY OF CONTRIBUTIONS

The primary purpose of the power grid protection system is to minimize the impact of electrical faults. Substations in the modern power grid achieve this objective by using real-time communication between intelligent electronic devices (IEDs). In IEC 61850-compliant systems, the protection-related data exchange relies on the generic object-oriented substation events (GOOSE) protocol [40]. To meet the timing requirements of such exchanges, GOOSE, although an application-layer protocol, sits directly above the data-link layer and follows a multicast communication model.

The security of GOOSE communication is a crucial aspect of the grid's reliability. Accordingly, the IEC 62351 standard [91] was developed to fill the security gaps in GOOSE, among other protocols. However, high-speed communication, coupled with the resource-constrained substation network, has resulted in limited application of the proposed security measures [92]. In fact, IEC 62351 does not recommend the use of encryption for GOOSE. Thus, not all of the guidelines, such as recommendations that message authentications use digital signatures and that certificates be supported in configuration files, are applicable.

As a result, the substation security still relies significantly on *best practices* [93] and the implementations and configurations from individual vendors [94].

Nevertheless, researchers have explored some encryption-based measures for hardening the integrity and authenticity of GOOSE messages (e.g.,[95, 96, 97]). All of these approaches fail to meet the most stringent GOOSE transfer time unless a significant overhaul is made in IEDs' hardware [98, 99]. Another set of methods (e.g., [98, 99, 100, 101]) that can satisfy the timing constraints uses pre-shared symmetric keys, requiring a robust key management operation within a substation. Because of those obstacles, what we have today—an unencrypted channel between substation devices—makes GOOSE susceptible to man-in-the-middle (MITM) attacks. The absence of flow control and acknowledgments further increases the risk.

We primarily focus on GOOSE poisoning, a common class of MITM attacks affecting system availability [93, 94, 102, 103]. Such attacks work by injecting valid GOOSE messages that can cause IEDs to stop processing legitimate traffic and take malicious control actions.

In this chapter, we introduce *ED4GAP*, a network-level system for *E*fficient *D*etection *For* *G*OOSE-b*A*sed *P*oisoning attacks. ED4GAP monitors the substation's local area network through port mirroring on switches. It then analyzes the GOOSE messages to identify indicators of poisoning attacks concurrently to the data transmission. The identification depends on the protocol specifications, as well as the attack's needs to inject extra packets on the network. The solution uses a state machine model to find such syntactically correct but malicious packets. The model enumerates all possible transitions, both valid and invalid, from the current state of GOOSE communication. Any transitions to the invalid states are easily detected. However, a sophisticated attacker can achieve malicious goals even if limited to only benign transitions. Our approach overcomes that challenge by introducing a *staging state*. The staging state temporarily stores the information about a set of packets until a classification of "attack" or "normal" is confirmed. Finally, the system generates alerts concerning the detected attacks to enable further action.

We implement a prototype of ED4GAP using the Zeek network security monitor [39, 104]. Zeek, an open-source and highly extensible framework, has seen considerable application in power grid security [105, 106]. With Zeek, we use a protocol analyzer [107] to decode GOOSE traffic, implement the attack detection method as a policy script, and extend the logging mechanism to produce GOOSE-related logs and attack alerts. We further devise a systematic approach to identify bottlenecks and speed-up the traffic processing throughput by using the exclusive pinning of CPU cores.

In our experiments, we generated traffic by replaying a synthesized dataset [108] representing an IEC 61850 substation. ED4GAP accurately detected all the attacks and generated

corresponding alerts. Afterward, we evaluated the real-time performance of ED4GAP to ensure that it could support GOOSE timing constraints. We replayed the traffic by increasing the transmission speed to achieve a total of more than 200 experimental settings. When employed as a virtual machine on a host with an Intel Core i7 quad-core processor running at 2.6 GHz with 8 GB of RAM, ED4GAP could achieve a throughput of up to 21.98 megabits per second (Mbps) and average per-packet response time of 0.03 milliseconds (ms). For the current dataset, that throughput translated into $17,535$ packets per second. Our results demonstrated that ED4GAP could analyze traffic and detect attacks while satisfying the real-time requirements of GOOSE.

We can summarize the contributions of this chapter as follows:

Intrusion detection: We present a specification-based method for detecting GOOSE poisoning attacks even when the attacker can replicate normal behavior.

System implementation: We implement ED4GAP—a system for out-of-band concurrent traffic analysis and poisoning detection.

Performance improvement: We devise a systematic approach to finding bottlenecks and optimizing performance by varying the number and using the exclusive pinning of CPU cores.

To provide some context to the problem addressed in this chapter, next, we describe the advanced cyber threats on smart grid networks. We then present our approach and the design choices we made to detect the attacks while meeting the strict timing constraints for communication.

4.2 ADVANCED CYBER THREATS ON THE SMART GRID

The modern power grid, known as the *smart grid*, uses computer communication networks to achieve better efficiency and reliability. State-of-the-art information technology and integrated networks enable communications between sensors and actuators attached to grid elements and smart grid applications. However, that increased connectivity comes at the cost of a larger surface for cyber attacks.

Advanced cyber attacks on the smart grid have become more common in recent years. The energy sector was one of the top five industries hit by targeted attacks in 2017 [109]. The cyber attacks of 2015 and 2016 on the Ukrainian power grid alarmed stakeholders with their ability to use the victim system's own communication infrastructure against itself [4, 5]. The 2016 attack also showed the attackers' capabilities to build upon knowledge from

past incidents, greater sophistication in each attack step, and glimpses of a vendor-neutral, reusable compromise framework [110].

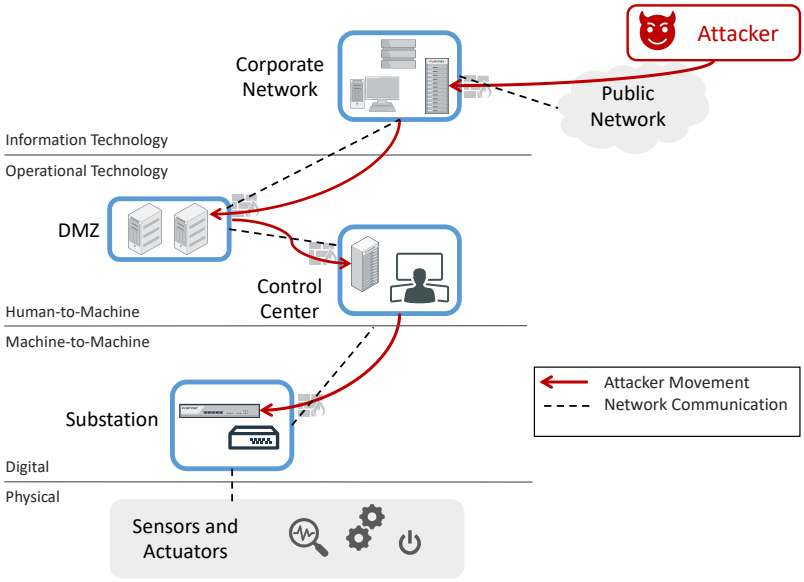


Figure 4.1: Schematic of advanced cyber attacks on the smart grid.

As shown in Figure 4.1, advanced cyber attacks on the smart grid proceed in the manner similar to what we discussed in Chapter 1. Often starting with the compromise of the corporate IT network, the attackers then use lateral movement tactics to reach critical segments. The most critical part of a smart grid network is the supervisory control and data acquisition (SCADA) network, comprising the control center and substation networks. Once inside the control network, the attackers can inject malicious commands to cause disconnection or failure of devices.

4.2.1 Example Incident

In December of 2015, A cyber attack on three regional power distribution entities in Ukraine left about 225,000 consumers without power. The attack was described as the first instance of a cyber-induced power outage ever reported in the public domain [4, 41].

The attackers began by sending spear-phishing emails that tempted some employees to open the attached documents, leading to installation of the Black Energy 3 (BE3) malware, among other tools, on corporate workstations (i.e., achieving initial compromise). The malicious tools performed internal reconnaissance and installed multiple backdoors (i.e., achieving persistence). The malicious actors then discovered and compromised Active Directory servers and obtained a large number of enterprise credentials (i.e., achieving credential ac-

cess). Using the stolen credentials, the attackers established an encrypted tunnel into the control network (i.e., achieving lateral movement). In the next step, they used the tunnel to access SCADA human machine interface (HMI) servers in the control center as well as the substations (i.e., further lateral movement). Finally, the attackers opened the breakers to cause the power outage by using the backdoor communication and compromised SCADA servers (i.e., performing the goal action on the target). The power outage action was followed by several complicating actions, such as a telephony-based denial-of-service attack, disabling of control center UPSs, corruption of firmware on serial-to-Ethernet converters at substations, and wiping of control center workstations. Throughout the entire campaign, the attackers utilized command-and-control (C&C) communication to gather intelligence about the system, move to more critical segments, and launch the malicious actions [4, 41]. Overall, the attack was thus able to interact with the systems, persist for long periods, and eventually use the cyber infrastructure against the system itself.

We note that the solutions presented in Chapters 2 and 3 would apply to smart grid corporate networks, which fit into the model of enterprise systems. Therefore, in this chapter, we will instead study attacks on the control network. In particular, we will present a system for detecting false data injection in substation networks.

4.3 PRELIMINARIES: IEC 61850 AND GOOSE

IEC 61850 is an architecture for utility automation systems, including electrical substations [111]. It guides the configuration, operation, and maintenance of intelligent electronic devices (IEDs) to achieve a truly digital substation automation system. IEC 61850 offers both business and technological gains. Regarding the business benefits, it enables interoperability among different vendors, improves operational reliability, and reduces the overall capital cost. Concerning the technological advantages, it offers common communication protocols, enables information management, and provides real-time, distributed control and event-driven architecture.

At its core, IEC 61850 standardizes the following three components:

- Data model: A semantic hierarchical object data model to describe the data points, devices, and substations.
- Communication model: Ethernet- and IP-based medium, new protocols and interfaces, and security guidelines.
- Process and format: XML-based representation, configuration language, and standard

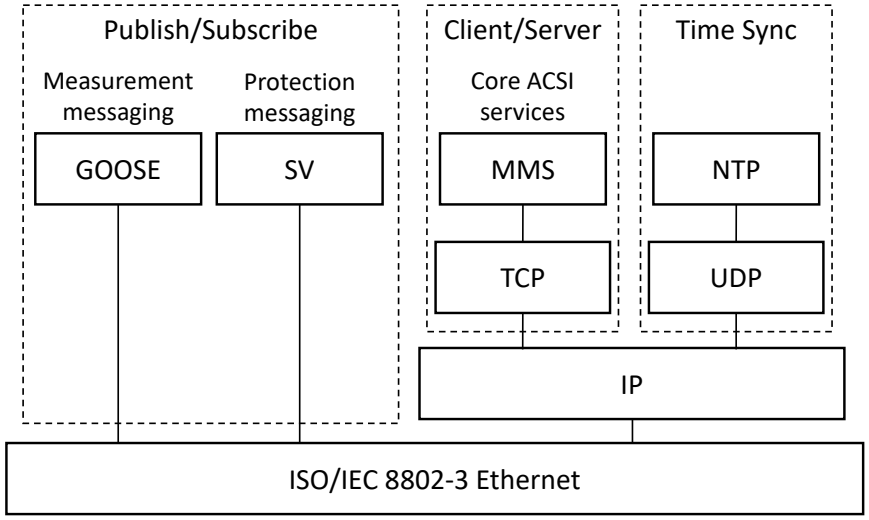


Figure 4.2: IEC 61850 protocol stack.

interchanging process.

The semantic hierarchical data model divides each physical device, i.e., an IED, into multiple logical devices. Each logical device is a collection of logical nodes implemented in one IED. A *logical node* is a named grouping of data and associated services logically related to some power system function, such as that of a circuit breaker. Each *logical node* can have a set of data and attributes. Therefore, IEC 61850 data objects follow the format of `LogicalDevice/LogicNode$DataObject$Attribute`.

For example, `Relay1/XCBR1$Pos$stVal` would indicate that "Relay 1" is a logical device controlling the position of "circuit breaker 1." The "position" can take a Boolean value, as represented by the "`stVal`" attribute.

In the context of the network, Ethernet-based communication is adopted to eliminate the need to hard-wire thousands of copper wires, making access to information more efficient. Further, IEC 61850 uses the manufacturing message specification (MMS) to implement its abstract communication model. MMS is an International Organization for Standardization (ISO) standard used in control networks to define messaging between controllers as well as between the engineering station and controllers. IEC 61850 objects and services are mapped to MMS objects and operations, respectively.

If MMS is thus used to implement the automation system, the protocol stack in a substation will look similar to Figure 4.2. The Sampled Value (SV) protocol provides a method for transmitting sampled measurements from transducers such as current transformers, voltage transformers, and digital I/O. The generic object-oriented substation events (GOOSE) protocol is responsible for the exchange of commands, alarms, and indications. GOOSE groups

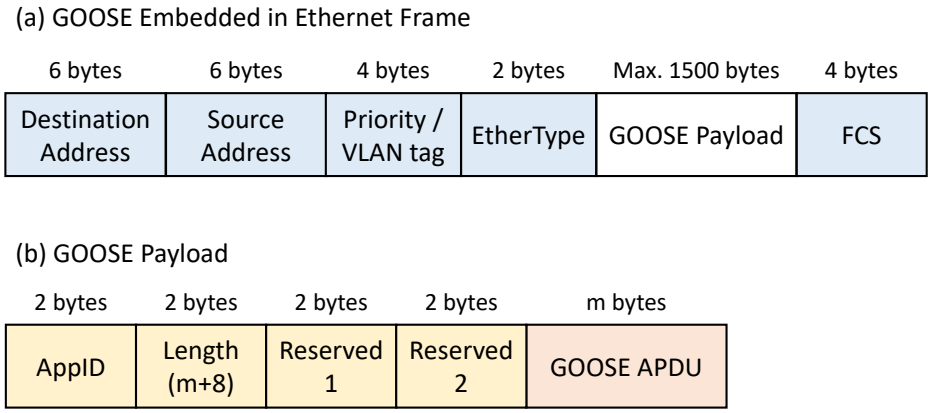


Figure 4.3: GOOSE message structure.

such data in a construct called a *dataset* and aims to achieve a strict transfer time, which is 3 ms for the most critical events, e.g., trips and blockings. To achieve the required transmission speed, GOOSE is placed directly above the Ethernet layer and follows a multicast communication model. In this model, a *publisher* sends data to a multicast Ethernet address to which multiple *subscribers* can register to receive the data.

Figure 4.3 shows the structure of a GOOSE message, encapsulated within an Ethernet frame. The `EtherType` for GOOSE is 0x88b8. A GOOSE payload has a variable `Length` of less than 1500 bytes. The variability in length comes from GOOSE's application protocol data unit (APDU). Optionally, an extended encapsulation with IEEE 802.1Q (VLAN) can be used by adding VLAN ID and priority fields to the header.

The GOOSE APDU is a sequence of another twelve fields. The values of these fields are assigned locally by the publishers. The fields include (1) `GocbRef`, a unique path-name of an instance of a control block; (2) `DatSet`, an identifier for the set of values transferred, i.e., the dataset; (3) `NumDatSetEntries`, the number of data values contained in the frame; and (4) `AllData`, a list of the actual values contained in the message. The APDU also includes two time-related fields: (1) `T`, the timestamp at which `StNum` was incremented, and (2) `TimeAllowedtoLive`, an indicator for a subscriber of how long to wait for the next packet.

To achieve communication reliability at high speed, GOOSE makes use of an enhanced retransmission mechanism instead of the usual acknowledgments. In the GOOSE scheme, represented in Figure 4.4, a message is published immediately after a new event. Afterward, the publisher retransmits the same state as long as it persists. The retransmissions happen with an exponentially increasing time separation between messages, e.g., T_1, T_2, T_3, and so on. Once the interval has reached a steady-state value of T_0, it stops changing. The values of T_1, T_0, and the exponential increment are set at the initial configuration of the devices.

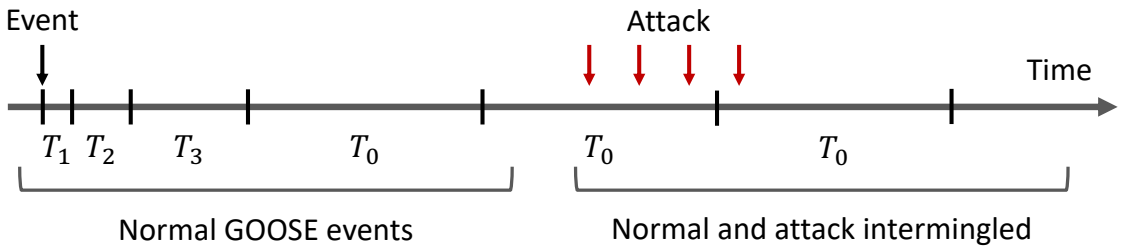


Figure 4.4: Enhanced retransmission mechanism of GOOSE; T_i is interval between two successive messages.

The GOOSE APDU also includes two 32-bit counters, namely a state number (`StNum`) and a sequence number (`SqNum`), to support the retransmission mechanism. Every new event causes a change in one or more values in the corresponding `DatSet`. The source device reacts to that change by publishing a GOOSE message with an incremented `StNum`. In the absence of any state changes, the publisher transmits identical messages, and only `SqNum` is incremented. While the values of `StNum`, `SqNum`, and `TimeAllowedToLive` are set by publishers, GOOSE subscribers use these fields to only admit new messages. More specifically, a subscriber processes messages that have a higher `StNum` or higher `SqNum` (with `StNum` unchanged) than the last message had.

4.4 THREAT MODEL

We consider an attacker who can monitor, spoof, and publish GOOSE messages within a substation. In particular, we assume that the adversary (1) knows subscribers' processing logic and the current status of GOOSE communication, (2) injects packets at the desired time-points, and (3) can choose values of `StNum` and `SqNum` (among other fields) such that the malicious traffic gets admitted. However, we trust the IEDs and assume that they behave as expected.

This threat model is relevant because of the vulnerabilities discussed earlier, in particular the lack of encryption and authentication. The former allows the attacker to observe the communication, whereas the latter causes the spoofed frames to be processed. Such attacks can cause a denial-of-service or the manipulation of devices. They can lead to severe consequences, including loss of electricity, damage to equipment, and injury to humans [102, 103, 107]. Therefore, these attacks are also known as *GOOSE poisoning*.

As a concrete example, we consider relays assigned to protection zones, in which correct operation is for the relay nearest a fault to trip and signal upstream relays not to trip via a blocking signal [112], sent as a GOOSE message, as shown in Figure 4.5. An attacker can

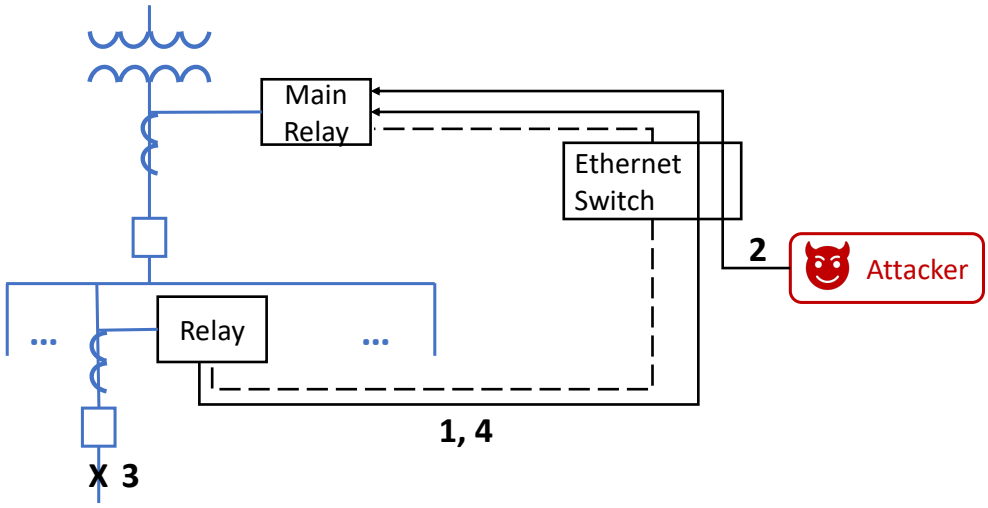


Figure 4.5: Example of a GOOSE poisoning attack on the relay blocking response; numbers indicate the message sequence.

analyze the properties of that messaging, and after a normal operation update (event 1), can send a high-`StNum` frame (event 2). After the malicious message has been processed, the upstream relays would discard any message that has a lower `StNum`, including reverse blocking (event 4) in response to a fault (event 3). As a result, the main relay might sense the fault at a later point and trip the attached breaker. Hence, the attack can cause an outage over a more extensive part of the system, i.e., trip the main circuit breaker instead of a single downstream breaker.

4.5 THE ED4GAP SYSTEM

To detect the indicators of GOOSE poisoning within a substation, first, the ED4GAP system monitors the traffic on the station bus. It can do so using port mirroring on Ethernet switches. Next, it uses the Zeek platform to decode the traffic and extract relevant features. Subsequently, it performs a specification-based security analysis to detect GOOSE poisoning attacks. Finally, it generates alerts corresponding to the detected attacks, which it can send to the substation human-machine interface or the control center for further action. In what follows, first, we discuss our design choices. We then present ED4GAP's specification-based analysis and GOOSE poisoning detection approach.

ED4GAP is a passive detection device that does not interfere with ongoing communication. Instead, it analyzes traffic data out-of-band in real-time and generates alerts about attacks. Despite that placement, in order to support high-speed GOOSE communication, it needs

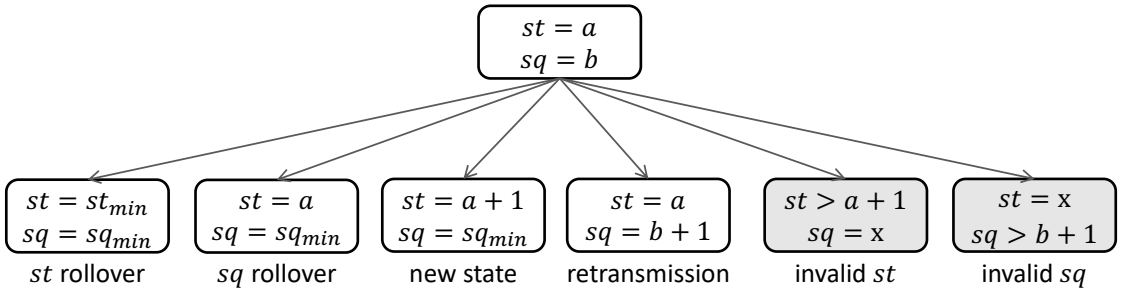


Figure 4.6: `DatSet`-level state machine, as per GOOSE specifications. Variables st and sq represent `StNum` and `SqNum`, respectively; st_{min} and sq_{min} are the minimum values that the counters can take; and x stands for any permissible value.

to ensure fast analysis. In that context, we had the following design considerations. First, the packet-analysis response time should be minimal for both benign and malicious cases. Second, the system should operate with a low overhead even with the high-speed traffic that is common in substations. Finally, the system should be easy to plug into existing environments with minimal changes.

4.5.1 Specification-Based Analysis

GOOSE protocol specifications [40] define the manner in which `StNum` and `SqNum` are used when the substation devices exchange protection-related information. To devise a method for the detection of poisoning, we started by developing a state machine model for those specifications. This model is at the level of a GOOSE `DatSet` because the `DatSet` uniquely identifies the entity responsible for every transmission.

As shown in Figure 4.6, the model enumerates all the states (starting at the values (a, b)) that the (`StNum`, `SqNum`) pair can take and that the subscriber would process. States "new state" and "retransmission" refer to an incremented `StNum` or `SqNum`, respectively, which are the most common situations in every GOOSE exchange. States"st rollover" and "sq rollover" correspond to the scenarios in which one of the counters rolls over and resets to its minimum value. Finally, the two invalid states correspond to an increment of greater than one in either of the counters; such increments are allowed and processed by subscribers.

Packets having lower-than-expected values for `StNum` or `SqNum`, i.e., $st < a$ or $sq < b$, are discarded by the subscribers. Similarly, devices typically can check and ignore syntactically incorrect frames, e.g., one with an invalid timestamp (`T`), `Length`-size mismatch, or invalid MAC address. Therefore, we do not include those states in the model. Finally, although GOOSE poisoning can manipulate data values in addition to `StNum` and `SqNum`, we do not

consider that explicitly during the detection. However, so long as the attack involves the injection of at least one GOOSE message, our approach will detect it, as we describe next.

4.5.2 Detecting GOOSE Poisoning

The basic premise of our approach is that even though the attacker can inject harmful GOOSE messages into the network, the benign publishers will continue transmitting their respective states. Therefore, we can detect the attack by logically following the benign message sequence and identifying any anomalies concerning the protocol specification.

In the context of the state model (Figure 4.6), it is relatively simple to find attacks that rely on the two invalid states. A detector can do so by looking for unusually high values of `StNum` or `SqNum` in an otherwise consistent sequence. However, an intruder can attack while staying within the four valid states, i.e., can inject packets at approximately the same times when benign packets are being transmitted and can contain expected or valid values for `StNum` and `SqNum`. Moreover, the attack may consist of more than one message closely following the GOOSE retransmissions. Our detector must therefore consider all those scenarios.

The detector's approach is to maintain a memory or state consisting of metadata extracted from GOOSE messages. The metadata necessary for the detection include `DatSet`, `TAL`, `StNum`, and `SqNum`, although in our implementation we also extract a timestamp and the source and destination MAC addresses to obtain some context that is useful for alerts. In particular, we define two states, *normal* and *staging*. The normal state points to the information of the last packet that has been marked as normal. The staging state stores metadata for the packets waiting for classification.

Next, we define a parameter T_m representing the maximum delay in the arrival of the next benign packet, which is equal to the `TAL` of the packet held in the normal state. The need for this extra variable arises because IEC 61850 only defines the allowed range for the configurations of GOOSE `TAL` and retransmission times (T_i's in Figure 4.4) [40]. The actual settings vary across vendors; for example, `TAL` is often set equal to T_0 or $2 * T_0$. Moreover, whether different GOOSE messages can have different `TAL` values is also an implementation choice.

The maintenance of the staging state relies on the following property:

Proposition 4.1. If a GOOSE poisoning attack injects a set of packets, Q, then any $q \in Q$ will be between two benign packets that are, at the maximum, T_m time apart.

Proof. The proof follows from GOOSE specifications. A publisher sends periodic messages either to retransmit a state or to notify subscribers of a new event concerning a `DatSet`.

That behavior results in a time series of events similar to the example in Figure 4.4. In this time series, the maximum delay between any two consecutive events is T_0, which needs to stay below T_m to keep the communication alive. Thus, since the attack needs to add extra packets, any given attack packet will be between two good packets, which are T_m or less apart. QED.

Algorithm 4.1 GOOSE Poisoning Detection

Input: st_{min}, st_{max}, sq_{min}, sq_{max}
Output: alerts
Initialize: $n \leftarrow (st_0, sq_0)$ ▷ stNum and sqNum of the first benign packet
$S \leftarrow \{(st_1, sq_1)\}$ ▷ stNum and sqNum of the second benign packet
$V \leftarrow \phi$
$T_m \leftarrow n.\texttt{TAL}$ ▷ `TAL` of the last benign packet
$stateChanged \leftarrow \bot$

1: $p \leftarrow$ getNextPacket ▷ stNum, sqNum, and other fields
2: **schedule**(flushStaging, T_m)
3: **while** p **do**
4: **if** nextExpectedSq(p, sq_{min}, sq_{max}) **then**
5: $S \cup \{p\}$
6: $stateChanged \leftarrow \bot$
7: **else if** nextExpectedSt(p, st_{min}, st_{max}) **then**
8: $S \cup \{p\}$
9: $stateChanged \leftarrow \top$
10: **else if** highSq(p) **or** highSt(p) **or** presentInStaging(p) **or** ($stateChanged$ **and** prevStateReccured(p)) **then**
11: $V \cup \{p\}$
12: sendAlert(p)
13: **break**
14: **end if**
15: $p \leftarrow$ getNextPacket
16: **end while**

17: **procedure** flushStaging
18: **if** $|V| > 0$ **then** ▷ Attack(s) detected since the last activation
19: **return**
20: **end if**
21: $n \leftarrow S[last - 1]$ ▷ Second to the last member
22: $S \leftarrow \{S[last]\}$ ▷ The last member
23: $T_m \leftarrow n.\texttt{TAL}$
24: **schedule**(flushStaging, T_m)
25: **end procedure**

We present GOOSE poisoning detection in Algorithm 4.1, which tracks the benign communication augmented with the normal and staging states. The algorithm is at the level of a `DatSet`. It requires as input parameters from a substation's configuration, including the minimum and maximum values for `StNum` and `SqNum`. At the setup, the normal state, n, refers to the first benign packet's metadata. The staging state, S, contains metadata for at least the next benign packet. V, an empty set, represents the information on attacks. Finally, T_m is set to the first benign packet's `TAL`, as noted previously.

The algorithm starts by scheduling FLUSHSTAGING to run at T_m time-points in the future. The activation of this procedure means that T_m time has elapsed since the last activation, and if there have been no attacks in this interval, the packets in S are benign. Therefore, the algorithm updates n, S, and T_m and schedules FLUSHSTAGING recursively (lines 18–21).

The algorithm's main body runs a loop until there are no packets to process, or an attack is detected. First, it checks whether a new packet, p, is valid, i.e., has the values for `StNum` and `SqNum` from one of the four valid states in Figure 4.6. If it determines that the packet is valid, it appends p's information to S. The procedures NEXTEXPECTEDSQ and NEXTEXPECTEDST perform the checks (lines 4, 7), also taking into account the rollover scenarios. However, if p fails to satisfy the validity conditions, the algorithm then checks whether p relates to one of the violations (line 10). Functions HIGHSQ and HIGHST correspond to unusually high values of the counters, i.e., invalid states in Figure 4.6. The third test, PRESENTINSTAGING, covers all the cases that correspond to the attacker's injection of the next expected packet. It returns *True* if a packet with the same `StNum` and `SqNum` values is present in S. Finally, PREVSTATERECURRED checks whether either the attacker or the publisher has inserted a new state, while the other is still retransmitting the old state. It compares p against the information in both n and S, to account for the situations in which the attack was injected concurrently with the transmission of the next benign packet.

On finding any violation, the algorithm appends p's metadata to V (line 11). SENDALERT (line 12) then triggers an alert that includes information such as source and destination addresses, `StNum` and `SqNum`, and timestamp. At that point, S contains the entire trace of packets that enabled the detection. A security analyst can subsequently act upon the alerts generated by the algorithm to bring the system back to a safe state and initialize n and S with new data.

The algorithm has both time and space complexities of $\mathcal{O}(s)$, where s is the size of the staging state. Theoretically, s can be large, however, since attacks and state changes are rare, its expected value is low. Finally, the algorithm satisfies the following property:

Proposition 4.2. GOOSE Poisoning Detection will detect all variants of the attack within

T_m time units from the attack's occurrence, given the benign communications continue.

Proof. Let $p_1 := (a, b)$ and $p_2 := (x, y)$ be the benign packets on the either side of an attack packet q, i.e., $t(p_1) \leq t(q) \leq t(p_2)$, where $t(.)$ is the arrival time. One of the following situations will happen:

- q injects a high `StNum` or high `SqNum` (two invalid states in Figure 4.6): The algorithm will detect that immediately (at $t(q)$).
- q takes one of the valid states: The arrival of p_2, depending on the values of (x, y), will result in either (1) a duplicate retransmission or (2) recurrence of an old state. Those two scenarios are detected at $t(p_2)$ by PRESENTINSTAGING and PREVSTATERECURRED, respectively. Since, $(t(p_2) - t(q)) \leq T_m$ (from Proposition 4.1), the time from occurrence to detection is always less than T_m. Finally, if the attacker injects multiple valid packets after q, they all remain in S until $t(p_2)$ and get detected.

QED.

Proposition 4.2 provides a *sufficient* condition for the algorithm. Even if an attack involves a bad `TAL` to cause a timeout at the subscriber, the algorithm will detect it. However, if the publisher fails to send the next benign message within the T_m interval (i.e., the current `TAL`), the communication is deemed lost. In those cases, our algorithm will need to restart, which is a limitation, especially because how the devices are reset after a timeout is not consistent across different vendors.

4.6 EXPERIMENTAL EVALUATION

We conducted experiments to assess the detection accuracy and real-time performance of the ED4GAP. The experiments used an IEC 61850 security dataset collected from a simulated power system [108]. The devices in the simulated system normally send multicast packets every second to share their statuses. Anomalous behaviors corresponding to several benign power system disturbances and attacks are also present. Overall, the dataset consists of 14 traces amounting to about 86 megabytes and $523,853$ packets. GOOSE messages comprise about 87% of the traffic, whereas the remaining 13% includes conventional protocols, such as address resolution protocol (ARP), internet control message protocol (ICMP), DNS, and dynamic host configuration protocol (DHCP).

We implemented ED4GAP using the Zeek network security monitor [39]. Zeek provides an extensible platform with built-in decoders for many protocols and a scripting language

for security-policy development. To process GOOSE traffic, we used an analyzer proposed in [107]. It allowed us to parse Ethernet and GOOSE headers as well as the APDU to extract features essential for detection. Using those features, we implemented the GOOSE poisoning detection algorithm as a Zeek script. Finally, we utilized Zeek's logging infrastructure to generate logs of the GOOSE header, APDU, and alerts.

We installed Zeek (version Bro 2.5.4) on a virtual machine (VM) running Ubuntu 16.04.6 LTS with 2 GB of RAM and four virtual CPUs. The host computer had macOS version 10.15.4 on a 2.6 GHz quad-core Intel Core i7 with 8 GB of DDR3 RAM. We used Tcpreplay [113] to feed the existing traces to an interface that Zeek was monitoring.

4.6.1 Intrusion Detection Accuracy

Of the three types of attacks present in the dataset, message suppression (MS) and data manipulation (DM) are relevant for our evaluation. They inject packets with modified `StNum` or `SqNum`; DM attacks also manipulate data values. Therefore, we consider only the MS and DM attack scenarios for the security assessment.

The dataset, however, does not cover all forms of GOOSE poisoning. There are 24 possible variations, which consider all types of violations and benign changes. The variations correspond to the two fields {`StNum`, `SqNum`}, two options for the first attack packet {a high value, the next value}, three choices for how the attacker will follow up {do nothing, retransmit, inject new state}, and two benign changes {retransmit, new state}. The original traces cover only two of the 24 attack variations. Therefore, we generated new traces that rely on the dataset's normal trace, in order to cover the remaining 22 test cases. To do so, we used Python's Scapy module [114] to inject manipulated packets at specific points in the trace.

We fed all the attack traces, including those that we created, to the ED4GAP system. We compared the output alerts with the ground-truth information present at known points in the dataset. The system could detect all the attacks and triggered alerts to identify source packets correctly.

4.6.2 Performance

In the current design, ED4GAP is non-blocking; it does analyze the traffic off mirror ports without interruption. Nevertheless, since GOOSE handles protection-related information, the security analysis must be fast. To confirm that ED4GAP meets that criterion, we evaluated its response time and throughput. In each experiment, we replayed all 14 traces and performed more than 200 trails.

ED4GAP showed an average per-packet response time of 0.06 ms. We computed the average response time by dividing the completion time by the total number of packets. Hence, we accounted for the network latency, waiting time, processing time, and log-writing time of all the protocols. Response time analysis thus indicates that the additional overhead due to ED4GAP is low.

Next, we measured the processing time in handling only the GOOSE messages, which we defined as the time taken in packet analysis, attack detection, alert generation, and writing of logs and alerts to the disk. Figure 4.7a depicts the GOOSE processing time as the speed of the input traffic increased. Both the median and the 90th percentile of the processing time (in ms) hold stable within the ranges $[0.025, 0.028]$ and $[0.038, 0.050]$, respectively. Those values are less than 2% of the GOOSE's most stringent transfer time. GOOSE processing time analysis thus provides empirical confirmation of the constant complexity of the GOOSE poisoning detection approach.

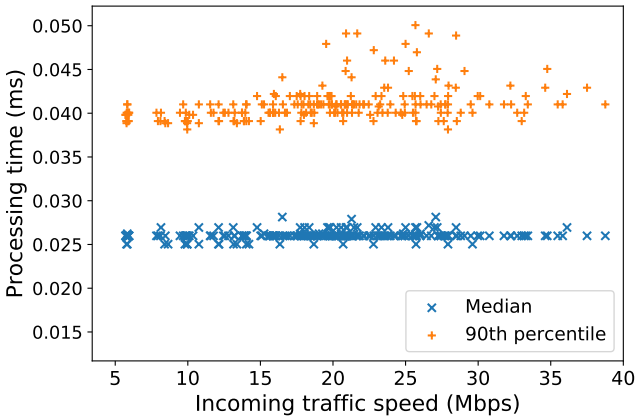
Afterward, to test how ED4GAP behaves with high-speed input traffic, we measured its throughput, i.e., the rate of *successful* message processing. Our results showed that the throughput increased almost linearly with the input speed until it hit a plateau, as plotted in Figure 4.7b. Before reaching the plateau, however, the system started dropping packets. The maximum throughput realized without any packet drop was 20.87 Mbps. Packet drop incurred beyond that speed reached up to 25.95% of the total packets transmitted at around 35 Mbps speed. We show this effect in Figure 4.7c.

The packet drop reflects the capacity of our experimental setup. The theoretical limit of the throughput of our setup was 23.30 Mbps, which we measured by inputting the traffic directly to Zeek with no network interface in the middle (using the `zeek -r` command). The limit is depicted as a green horizontal dotted line in Figure 4.7b. However, when the traffic is read from a network interface, that upper bound would not be hit. The socket layer between the interface and Zeek would add delay. Those factors led to a throughput of 20.87 Mbps in our experiments, lower than the upper-bound.

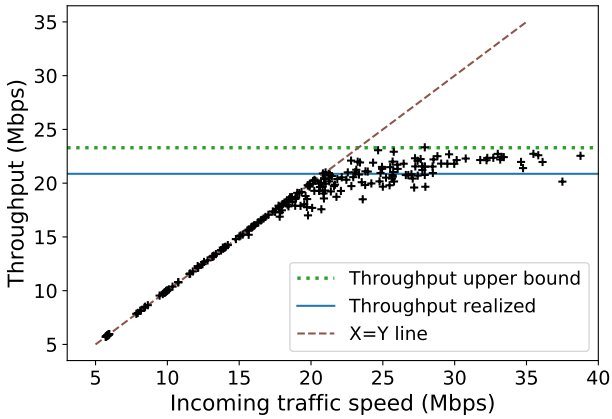
Finally, in packets per second (pps), the throughput was $16,649$, since the evaluation dataset had an average packet size of 164.30 bytes. If all the GOOSE frames were of the maximum size of 1500 bytes, and considering an additional 22 bytes for the typical Ethernet plus VLAN overhead, a throughput of about 1797 pps would still be realized.

4.6.3 Architectural Considerations

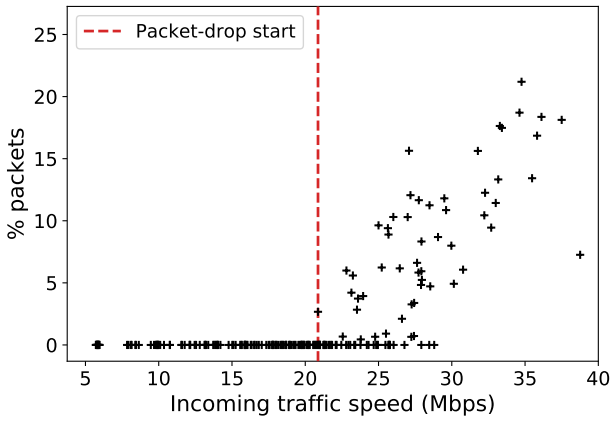
In practice, ED4GAP will be deployed at a substation server. Although the performance of such a deployment can be better than our experimental setup (a VM on a typical work-



(a) Processing time for GOOSE events.



(b) Overall system throughput.



(c) Packets dropped by Zeek.

Figure 4.7: Performance of the proposed system in processing the evaluation dataset. Each marker denotes a run of an experiment in which all 14 traces were analyzed.

Table 4.1: Effect of increasing the number of CPU cores (n).

n	Throughput (Mbps)	Response time per packet (ms)	Avg % packet drop at speed > throughput
2	12.30	0.10	25.93 ± 4.11
4	20.87	0.06	5.43 ± 1.30
6	21.15	0.06	2.58 ± 3.28

Table 4.2: Effect of pinning c cores to Zeek; $n = 6$.

c	Throughput (Mbps)	Response time per packet (ms)
1	18.91	0.07
2	21.68	0.04
3	21.98	0.03
4	21.98	0.03

station), one needs to consider several factors to speed up the execution, as we discuss next.

In general, the throughput will be better if more resources are available. Processing time, on the other hand, is a function of the complexity of the algorithm. Therefore, processing time may not decrease below some limit simply through access to additional resources.

We found that ED4GAP was mainly CPU-bound. Therefore, to increase throughput, we can either add CPUs or use multiple similar Zeek workers running in parallel. To test that hypothesis, we separately collected the performance results on deployments with two and six CPU cores, in addition to the previously discussed results with four cores. As presented in Table 4.1, the overall throughput of the system (with no packet drop) increased with the number of cores. Also shown in the table is the average percentage packet drop with its 95% confidence interval, which decreased when cores were added. That analysis shows that with fewer cores, the system got overloaded at a lower traffic-speed. Although Zeek's primary function is single-threaded, it uses multiple threads to perform secondary tasks, such as writing logs. In fact, we observed up to nine threads of the Zeek process. Therefore, the overall throughput improved when the number of cores increased.

A lower throughput indicates that Zeek was interrupted by other system processes. To analyze that effect, with a total of six cores, we pinned $c = \{1, 2, 3, 4\}$ cores exclusively to Zeek and measured the throughput and response time. Exclusion of c cores removed them from the Linux scheduler's pool, forcing it to consider only $6 - c$ cores for running other processes. Thus, Zeek could run uninterrupted on c cores. The results of that set

of experiments, shown in Table 4.2, reveal that the exclusive pinning helped improve the throughput compared to that of the unpinned setting. In summary, this analysis provides insights into system bottlenecks and can inform decisions that will improve performance.

4.7 DISCUSSION

ED4GAP's intrusion detection approach requires several input parameters, such as MAC addresses, the GOOSE `DatSet`, and minimum and maximum values for `StNum` and `SqNum`. All of that information is present in substation configuration files and can be extracted automatically.

Our analysis of bottlenecks and performance-improvement aspects has focused mainly on the effect of increasing compute resources. However, some network monitoring appliances implement various other techniques to accelerate packet processing. For instance, the R-Scope [115] network security appliance implements a set of packet-path optimization techniques that help accelerate the processing of Zeek-based security analytics, such as the ones we present in this chapter. These high-performance techniques reaffirm the viability of running the presented system in real-time according to the protocol's timing requirements.

Currently, our threat model does not consider attacks that can compromise an IED to suppress/manipulate its behavior completely. Network-level detection of such attacks would require one to find anomalies by taking into account both the cyber and physical properties of GOOSE messages. An extension of our system to incorporate such models is possible.

Finally, since the poisoning detection relies on storing traffic metadata in a staging state, a memory attack causing a denial-of-service (DoS) of ED4GAP is possible. Such an attack would need to flood the network with a large number of packets in an interval smaller than T_m. In this chapter, we do not aim to detect flooding-based DoS. However, some existing systems can help mitigate those attacks [116, 117].

4.8 RELATED WORK

4.8.1 Network-Based IDSes for Substations

Network-based intrusion detection is an effective strategy for substation security. For example, EDMAND [105] analyzes DNP3 protocol traffic flowing between a substation and a control center. It then performs anomaly-based detection of attacks on transport, operation, and content layers of the traffic. EDSGuard [106] is an application that runs at an SDN

controller and implements best-practice security requirements concerning Modbus and ARP protocols. The work in [116] describes a network IDS for IEC 61850 substations focused on detecting attacks on conventional protocols, such as ARP, ICMP, FTP, and HTTP. The design and implementation of a system for network monitoring and data collection compliant with the latest IEC 62351 guidelines are presented in [118]. The system is used to detect attacks on IEC 61850 communication by using deep learning and rule-based methods.

The negative impact of GOOSE poisoning on the availability of substation devices is presented in [102, 103]. Papers that propose detectors for such attacks include [107, 119, 120]. However, those solutions cover only a limited number of attack variants. In contrast, the presented solution relies on the GOOSE protocol specifications to detect the attack in all variations, including cases in which the attacker closely follows the normal pattern. Moreover, we uniquely provide a systematic analysis of the performance to demonstrate that the system can achieve a low overhead that is suitable for GOOSE communication.

4.8.2 Evaluation Platform

GOOSE APDU is encoded using the Abstract Syntax Notation ONE, Basic Encoding Rules (ASN.1/BER) standard for data networks. In this encoding, the data is represented in triplets of tag-length-value (TLV) fields. Systems that analyze such encoding of GOOSE and allow a deep inspection of the traffic include [102, 107].

Our experiments utilize the substation dataset developed in [108]. The dataset consists of network traces generated using a 66/11kV substation model. The model uses an IEC 61850-compliant network architecture in which interfacing equipment are placed at the process level. At bay level, 18 IEDs including Line Feeder, Transformer Feeder, Bus IED, and Under Frequency Load Shedding IED connect using Ethernet and communicate GOOSE messages.

Finally, systems that, like ours, incorporate Zeek for SCADA network monitoring and security analysis include [105, 106, 107, 121].

4.9 CONCLUSION

The secure communication of protection-related data is crucial for the end-to-end security of the smart grid. False data injection attacks, such as GOOSE poisoning in IEC 61850 substations, can undermine communication security and lead to cascading failures. In this chapter, we introduced a network-level system, ED4GAP, dedicated to detecting GOOSE poisoning attacks in modern substations. The presented system relies on the extraction of features from substation traffic, analysis of protocol specifications, and a comprehensive

examination of communication properties to detect violations and generate alerts. We have implemented our system using the Zeek platform and devised a systematic approach to evaluate its performance. Our results showed that the system could analyze traffic and accurately detect the poisoning attacks concurrently to the data transmission. Thus, it is suitable for the context of high-speed communication within a substation. The next steps will be to extend the security analysis to evaluate the cyber and physical properties of the control traffic and test the presented system in a live substation environment.

CHAPTER 5: MULTILAYER DETECTION OF FALSE DATA INJECTION

In this chapter, we continue to study false data injection attacks on a smart grid control network. In Chapter 4, we described an efficient solution for detecting GOOSE poisoning attacks, which are a common class of false data injection attacks in such networks. Now we extend the detection to include a broader range of attacks within the false data injection model.

Herein, we present a system for multilayer, real-time detection of false data injection attacks on IEC 61850 GOOSE communication. The detection approach uses information related to multiple attributes of a substation network. The information includes the substation's physical architecture, configurations, and protocol specifications, and measurements of its physical properties. With such information, we design and implement layers of detectors that address a broad class of false data injection attacks. Those layers are whitelisting, GOOSE semantic analysis, GOOSE poisoning detection, and physical behavior-based detection. We utilize a substation network dataset, including a variety of attacks, to evaluate our system's detection accuracy. We also evaluate the system's response time for the security analysis of GOOSE messages and show that our result is an improvement over the existing state-of-the-art methods.

5.1 SUMMARY OF CONTRIBUTIONS

False data injection attacks on electrical systems have been a major security concern [112, 122, 123]. Such attacks first determine the current state of a power system and then inject malicious messages or measurements to mislead the receiving devices into executing harmful commands. We focus on such attacks on IEC 61850 GOOSE communication. GOOSE is the primary enabler of automated control and protection and is fast enough to support any time-critical applications [40]. However, because of the resource-constrained environments and limited network monitoring available in today's substations, GOOSE is vulnerable to false data injection attacks [112, 118, 120].

Because of the limited processing power and mission-critical protection function of intelligent electronic devices (IEDs), host-based security analysis is not a viable option for substations. A network-level intrusion detection system (IDS) that can passively monitor communication and identify security violations can address that limitation. However, currently, there is a deficiency of such cyber monitoring and detection [118], and it has been exploited by many successful attacks [4, 5, 124, 125].

Motivated by those observations and challenges, we present SEEZE[1], a system dedicated to detecting false data injection attacks on GOOSE communication concurrently with data transmission. SEEZE employs a multilayer scheme to provide high coverage of detection. The layers include whitelisting, GOOSE semantic analysis, GOOSE poisoning detection, and physical behavior-based detection. The first two layers address the detection of early indicators of an attack, including malformed and semantically invalid messages as well as violations of access control. Next, the GOOSE poisoning layer covers the detection of GOOSE header manipulation attacks. Finally, physical behavior-based detection uses rules related to the physical architecture of the substation and safe operating ranges. That layer handles the attacks that involve harmful payload, either through injection of packets or by manipulation of in-transit packets.

Architecturally, SEEZE sits at a substation edge server and passively monitors the network using mirrored ports on the station bus. It uses the Zeek network security monitor [39] to collect traffic data and implement its multilayer detection method. Zeek's powerful features allow SEEZE to perform a highly stateful traffic analysis in real-time. We draw upon the successful application of Zeek in smart grid network security, particularly in substation environments. Zeek has been used for deep inspection of the control traffic [121] and security analysis of various protocols, including Modbus [106], DNP3 [105], and EtherNet/IP [126]. We note that Zeek's application to IEC 61850 substations is limited [107] because of the prevalence of Ethernet-based protocols, such as GOOSE. By incorporating Zeek as the underlying platform, we also contribute towards its more extensive application in IEC 61850 substations.

In our experiments, we used a substation network dataset [108] to test SEEZE's detection accuracy and performance overhead. The dataset contained attacks related to bad measurements and instances of simple GOOSE poisoning. Therefore, we also implemented an attack-injection module to generate traces involving attacks, including sophisticated GOOSE poisoning, GOOSE semantic attacks, and access violations. Our results showed that SEEZE could detect all the attacks with a response time of less than 5% of the most stringent GOOSE transfer delay. That response time is an improvement over the existing state-of-the-art multilayer detection systems.

More specifically, we make the following contribution.

Integrated analysis of cyber and physical properties: We propose a novel combination of security analyses, including whitelisting, protocol specification, and physical-behavior attributes, to detect false data injection attacks on GOOSE communication.

[1]SEEZE stands for *Substation Edge Enhanced with ZEek*.

Prototype implementation: We developed a prototype of SEEZE by using the Zeek network security monitor and implementing the presented intrusion detection approach.

Accuracy and performance evaluation: Using substation network traces with injected attacks, we demonstrate high accuracy and low overhead of performing security analysis by SEEZE.

This chapter is organized as follows. We describe the relevant details of GOOSE communication in Section 5.2, followed by the threat model in Section 5.3. We provide an overview of the presented system and the choice of the Zeek platform in Section 5.4. We detail the multilayer detection method in Section 5.5. We then evaluate our system's detection accuracy and performance overhead in Section 5.6. Finally, we present the related work in Section 5.7 and conclude in Section 5.8.

5.2 PRELIMINARIES

IEC 61850 utilizes a peer-to-peer communication service named *generic substation events* (*GSE*) that is associated with fast and reliable communication between IEDs. One of the formats associated with GSE is the generic object-oriented substation events (GOOSE) protocol that is responsible for the exchange of protection-related events.

5.2.1 Summary of GOOSE Specifications

GOOSE is an application-layer protocol directly encapsulated in the data-link layer, i.e., an 802.3 Ethernet frame. (Optionally, an extension with IEEE 802.1Q (VLAN) can be applied.) In such an encapsulation, a GOOSE message is embedded into the Ethernet header, i.e., into a destination MAC address (`DstMAC`), a source MAC address (`SrcMAC`), the `EtherType`, and the Ethernet trailer, i.e., a frame checksum sequence (`FCS`). IEC 61850 reserves a range of multicast MAC addresses that can be used as the destination MAC address. That range for GOOSE is 01:0c:cd:01:00:00–01:0c:cd:01:01:ff.

The Ethernet header is followed by a GOOSE header, which is a set of four fields: `AppID`, `Length`, `Reserved 1`, and `Reserved 2`. `AppID` is a two-byte-long, system-wide unique identifier of the application associated with a message. `Length` indicates the number of octets contained in the message. The maximum allowed value for the length is $1,500$ bytes. The reserved fields, each of two bytes, are for future applications and security measures. We presented details of the GOOSE frame structure in Chapter 4 (Figure 4.3).

At the application layer, GOOSE's application protocol data unit (APDU) defines twelve fields. The fields can be of variable lengths, making the total size of the APDU flexible. In the following, we describe the relevant details of the GOOSE APDU fields.

- `GocbRef` stands for "GOOSE control block reference". It contains a unique path name of a GOOSE control block within a logical node.
- `TimeAllowedtoLive`, or `TAL`, informs the subscriber of the maximum time in milliseconds within which a new GOOSE message will arrive. If there is no message before the `TAL` has elapsed, the communication is considered lost, requiring a reset.
- `DatSet` is an identifier for a group of values, i.e., status, alarms, and measurements, that are transmitted together.
- `GoID` is a user-assigned identification of the GOOSE message.
- `T` is the timestamp at which the state of the corresponding `DatSet` last changed.
- `StNum` is a counter that increments every time the state of the GOOSE `DatSet` changes.
- `SqNum` represents an incremental sequence number for the messages within each state.
- `Test` indicates whether the GOOSE message originated in a real application or a simulated test.
- `ConfRev` stands for "configuration revision" and contains the number of times the configuration of the `DatSet` has changed.
- `NdsComm` indicates whether the corresponding GOOSE control block requires further configurations.
- `NumDatSetEntries` is the number of elements contained in the `DatSet`.
- `AllData` is a list of data values.

5.2.2 Summary of GOOSE Communication

IEC 61850 categorizes substation network traffic into five classes, called the *Function Types*. GOOSE is responsible for Function Type 1, which is further divided into two subcategories: "1A Trip" and "1B Other." Class 1A is responsible for messages regarding trips and blockings and mandates a maximum transfer delay of 3 ms. Class 1B refers to other

types of protection information and can take from 10 to 100 ms to transmit a message. All of the class 1 messages, i.e., GOOSE messages, follow a layer-two (L2) multicast mechanism in transmitting protection-related events.

In the L2 multicast model, a sender or *publisher* transmits messages destined for a multicast MAC address. A receiver or *subscriber* needs to subscribe to those addresses in order to receive the data. Since many subscribers can subscribe to a single multicast MAC address, in effect, the communication occurs in a one-to-many form.

GOOSE publications involve a continuous sequence of messages that are transmitted very fast. The continuous sequence is due to GOOSE's retransmission scheme, which is intended to provide some level of reliability for the high-speed messaging. In that scheme, the publisher sends keep-alive heartbeats with a fixed interval between consecutive messages. However, when the publisher senses any change in its observed state, it instantaneously notifies the subscribers about the change. In such situations, the heartbeat interval is set to a very small value and exponentially increased until it reaches the steady-state value.

5.3 THREAT MODEL

In this work, we aim to detect false data injection attacks on GOOSE communication by a man-in-the-middle adversary. In particular, we assume that an adversary has access to the local area network of the substation and capabilities to monitor, modify, and publish valid GOOSE messages. The attacker is also assumed to have the ability to persist in the network for long time scales. Such a threat model has been part of many previously disclosed cyber attacks on electrical substations [5, 41, 125].

In particular, we consider the following scenarios. An adversary without the full knowledge of the internal network structure may inject anomalous values for one or more protocol fields. On the other hand, a well-equipped attacker can carry out more advanced attacks, such as the GOOSE poisoning that we studied in Chapter 4. In that attack strategy, the attacker injects normal GOOSE messages with specific values for `StNum` and `SqNum` to fool a subscriber to processing the injected traffic and discarding the legitimate traffic. Such an attacker can also inject packets or modify in-transit messages with malicious data values to force the receiving devices to take harmful actions.

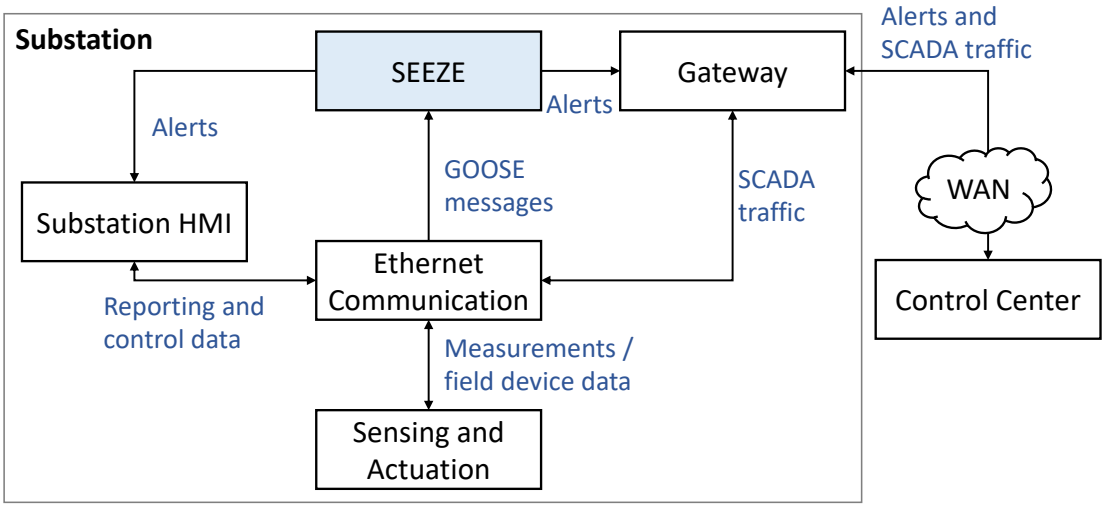


Figure 5.1: Logical placement of SEEZE in a substation network.

5.4 THE SEEZE SYSTEM

In this section, we present an overview of SEEZE, describing its placement in a substation network and its incorporation of the Zeek platform. In the next section, we will discuss in detail its intrusion detection methodology.

As shown in its logical architecture in Figure 5.1, SEEZE takes as input the substation's internal GOOSE communication. It then performs a multilayer security analysis of the traffic and generates alerts corresponding to the attacks it detects. The security analysis also requires details of the substation's physical architecture and configurations. Finally, SEEZE shares the alerts with the substation monitoring server as well as the control center.

Physically, SEEZE can be deployed at an edge device within the substation network. That device can be either an existing server, such as the one that operates the gateway application, or a machine dedicated to running security analyses. Such an edge device connects with the Ethernet switches to monitor the GOOSE traffic and the HMI server to send alerts within the substation. Outside of the substation, the edge device connects with the remote control center to share alerts.

SEEZE's placement at an edge server offers several benefits. First, the edge device can be a special-purpose, high-end substation computer, so that running of the security analysis will be more efficient than it would be on the IEDs. Next, such placement provides SEEZE with a consistent global view of the substation network without adding any additional traffic. Finally, edge placement requires no changes to IEDs, making it easy to plug our system into the substation.

5.4.1 Choice of Platform

Zeek is a Unix-based real-time network analysis framework that allows passive inspection of traffic [104]. The main factors that motivated our decision to use Zeek as the underlying platform include its extensibility, powerful features at low cost, ease of use, and successful existing applications in the smart grid domain.

Zeek offers a customizable and extensible platform, in terms of both user-specified scripts and protocol analyzers. The user-specified scripts represent a site's security policy. They can emphasize application-level semantics, track information over time (within and across flows), and perform arbitrary analysis tasks. Internally, the scripts perform semantic security analysis by handling events generated by Zeek's event engine. The event engine uses protocol analyzers to convert network traffic into higher-level, policy-neutral events.

Zeek is open source, delivers powerful functionality out of the box, and runs on commodity hardware, offering a low-cost solution. Also, Zeek can support signature-based analysis as well as other novel approaches, including semantic misuse detection, anomaly detection, and behavioral analysis. Therefore, Zeek is a better choice than the traditional signature-based IDSes, such as Snort. [127].

Another open-source framework that comes close to Zeek in terms of the functionalities is Suricata [128]. Suricata can potentially achieve a better performance than Zeek because of its multi-threaded architecture. However, Zeek is overall easier to extend with new protocol analyzers and policy scripts. We also found such aspects of Zeek better documented than those in Suricata.

Finally, Zeek has already seen extensive adoption in smart grid network monitoring and security. Such applications include the security analysis of various control protocols, e.g., Modbus [106], DNP3 [105, 121], EtherNet/IP [126], and GOOSE [107]. All of the above characteristics of Zeek made it the best choice for the development of SEEZE's attack detection approach, which we discuss next.

5.5 MULTILAYER DETECTION METHODOLOGY

The objective of SEEZE's intrusion detection is to identify syntactically valid but malicious communication. Syntactically incorrect or malformed packets are discarded by network switches as well as the IEDs, so they do not harm the security of the substation. For instance, a malformed packet, whether due to transmission errors or caused by an attacker, will not pass the integrity check performed using the `FCS`. Therefore, SEEZE aims to detect injected or modified GOOSE messages that can lead to the manipulation of subscribers. It

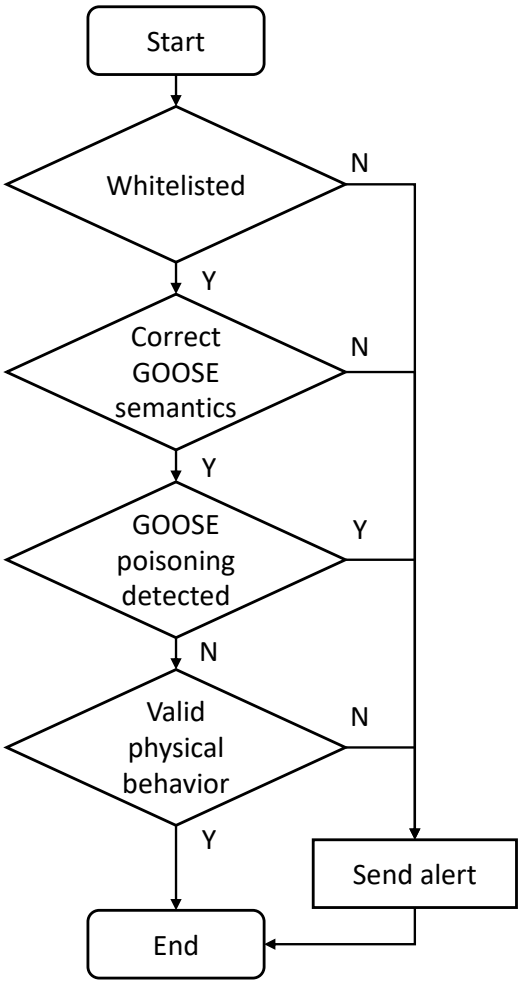


Figure 5.2: SEEZE's multilayer detection process applied to each GOOSE message.

does so by using multiple attributes of the substation traffic.

At a high level, those attributes rely on the following characteristics of the substation network: (1) well-defined communication behavior relying on physical properties, (2) a limited number of devices, and (3) relatively simple protocols. To reason about those characteristics, we extract the following types of information. First, IEC 61850-compliant systems use substation configuration language (SCL) files, which can explain the communication and data models. Second, the physical behavior attributes can come from the measurements, such as current, voltage, power, and frequency values, contained in GOOSE messages, as well as the thresholds of safe operation. Finally, protocol specifications provide the details of how communication proceeds.

In particular, SEEZE employs a multilayer method, depicted in Figure 5.2. A message first goes through the whitelisting layer, which ensures that message endpoints and GOOSE iden-

tifiers take only the known values. Next, a semantic verification layer determines whether the GOOSE-related values follow patterns consistent with the protocol specifications. Subsequently, the GOOSE poisoning layer checks the message for indicators of `StNum` and/or `SqNum` abuse. Finally, the physical behavior layer analyzes the data contained in the message to ensure they are consistent with the current state of the substation.

The sequence of the layers is such that they address a successively more sophisticated false data injection. The top two layers identify the appearance of messages that are inconsistent with the properties of GOOSE communications. However, a more equipped attacker can learn the properties of ongoing communication and launch a GOOSE poisoning attack. Such an attacker can even attempt to destabilize the system by modifying in-transit messages with wrong measurements. That attack would not reveal itself as a GOOSE poisoning and can be detected only by correlating the communication with the current physical state. Accordingly, the bottom two layers address such advanced forms of attack. Thus, the multilayer approach handles a wide variety of false data injection attacks. In the following, we discuss each of the layers in more detail.

5.5.1 Whitelisting

SEEZE uses a whitelisting strategy to check the access-control properties of GOOSE messages against the known configurations. Such a strategy involves the following components.

End-Point Whitelisting

Since GOOSE works directly on the data-link layer, the endpoints are the two MAC addresses, `SrcMAC` and `DstMAC`. Therefore, we develop lists of source and destination MAC addresses and verify each message's endpoints against these lists. Moreover, that examination will implicitly ensure that the `DstMAC` value is in the allowed range of multicast addresses, i.e., 01:0c:cd:01:00:00–01:0c:cd:01:01:ff.

GOOSE-Specific Whitelisting

GOOSE messages contain several fields that represent preconfigured information, including `GocbRef`, `DatSet`, and `GoID`. Therefore, SEEZE utilizes whitelists of those three fields and generates alerts on if it sees any unknown values. Further, it use whitelists of (`SrcMAC`, `DatSet`) and (`DatSet`, `NumDatSetEntries`) pairs to detect malicious messages with spoofed addresses.

5.5.2 Specification-Based Detection

Next, SEEZE uses GOOSE protocol specifications [40] to verify the semantic properties of ongoing messaging. Such analysis can be considered a general form of whitelisting, in which one defines not just the allowed values but also the correct behavior. In contrast with the misuse-based systems that define what is *malicious*, specification-based systems establish what is *valid*. SEEZE performs the following types of detection at this layer.

GOOSE Semantics

Verification of `T` updates: The GOOSE timestamp (`T`) is updated if and only if the `StNum` is incremented, i.e., when a state change occurs. SEEZE therefore examines successive messages for each `DatSet` to detect malicious updates of `T`.

`TAL` verification: IEC 61850 describes a range of valid `TAL` values specified by its $MinTime$ and $MaxTime$ parameters. The actual settings for those parameters vary across substations. However, given the SCL files, SEEZE can extract the range of valid `TAL` values and check whether a GOOSE message violates that range.

`Length` verification: To verify the correctness of the frame length, SEEZE performs two checks. First, it ensures that the value represented by `Length` matches with the APDU's actual size. Second, it checks whether the value of `Length` is in the allowed range of 8–1500 bytes.

GOOSE Poisoning Detection

SEEZE utilizes the GOOSE poisoning detector that we presented in Chapter 4. The detector works on the premise that even if an attacker can fool the receivers into accepting the malicious packets (and discarding the legitimate data), he/she cannot stop the benign devices from sending their updates. Therefore, the primary attack indicator that it looks for is the presence of duplicate retransmissions of a known state.

The detector analyzes GOOSE messages by using a state machine, derived from the protocol specifications, to track the values of the (`StNum`, `SqNum`) pair. The state machine describes all the acceptable transitions for the pair from its current value. If a new message represents any of the invalid transitions, it can easily be detected. However, an attack is possible even if it is limited to valid transitions. Therefore, the detector maintains a *staging state* to track information on packets waiting for a classification. The size of the staging

state is bounded by the `TAL` of the last benign packet. If a new packet's (`StNum`, `SqNum`) values match those of a packet in the staging state, an attack is identified. Otherwise, the staging state is continuously flushed to keep track of the ongoing communication. Thus, by using protocol specification as its basis, the detector can identify GOOSE poisoning even when the attack involves the injection of valid or expected packets. We have demonstrated its detection accuracy in Chapter 4.

5.5.3 Physical Behavior-Based Rules

In the last layer, SEEZE searches for anomalies in GOOSE data values. These values include measurements, alarms, and status and reveal the current physical state of the substation. In an attack-free operation, the measurements and alarms are consistent with the status of devices. However, false data injection attacks will disrupt those consistencies. To detect such violations, we rely on the stateful analysis of GOOSE traffic and incorporate the following rules.

Switching Device State

Every switching device's state correlates with the measured values. For instance, when a circuit breaker is in the *open* state, the corresponding current and voltage measurements should be almost zero. Conversely, if a breaker is in the *closed* state, the measurements should be above a positive threshold within safe ranges. Accordingly, SEEZE analyzes each `DatSet`'s measurements and associated breaker's status to look for violations of such correlations.

Protection Events

When electrical faults—such as overcurrent, overload, or under-frequency—happen, and persist for a longer time than what is deemed safe (typically, 2 to 4 s), protective relay IEDs trip the associated circuit breakers. However, sometimes breakers can malfunction and fail to execute given commands. In such cases, the associated relay's upstream and neighboring relays carry out the required opening of circuits.

Those benign disturbances are a norm in the routine functioning of electrical systems. When such events occur, the devices react in a correlated manner to keep the system safe and running. Such coordination would also be visible in GOOSE communication through the measurements, alarms, and status across different datasets.

On the other hand, as discussed above, an attack would also disrupt the consistent global state. Therefore, SEEZE monitors the measurements in every GOOSE message. Each time the measurements go out of the acceptable ranges, SEEZE looks for the associated protection events in GOOSE messages exchanged within a short time window. If it does not find any alarms generated or trips initiated by the associated relay or the upstream relay, it generates an alert.

To construct the rules mentioned above, SEEZE uses the following types of data:

1. Safe ranges of measurements, such as current, voltage, and frequency, which are typically configured when a substation is commissioned and change only when devices are added or removed.

2. The substation physical architecture, i.e., the relative placement of devices and the system elements they monitor.

3. GOOSE communication and data configurations, which are available in SCL files.

5.6 EXPERIMENTAL EVALUATION

5.6.1 Experiment Setup

We performed experiments to demonstrate that SEEZE could detect false data injection attacks with high accuracy concurrently with GOOSE transmission. For our experiments, we used network traces of a simulated IEC 61850-compliant substation [108].

Physically, the dataset's substation system contains two redundant zones, each comprising two levels: 66kV and 11kV. Figure 5.3 shows the one-line diagram of one of the two redundant zones, as described in [108]. The two zones stay disconnected from each other during the usual operation. The system consists of line feeder IEDs, transformer feeder IEDs, bus IEDs, and under frequency load shedding IEDs. The one-line diagram describes the placements of the IEDs and physical elements that the IEDs monitor.

In addition to the physical architecture, the dataset includes substation configuration language (SCL) files that provide insights into the system's network configurations. In particular, the SCL files describe the communication and data models for each IED. The communication model includes IP and MAC addresses and VLAN-related details. The data model describes logical devices, data types, and GOOSE datasets associated with the IED. The physical structure, i.e., the one-line diagram (Figure 5.3) and the network configurations,

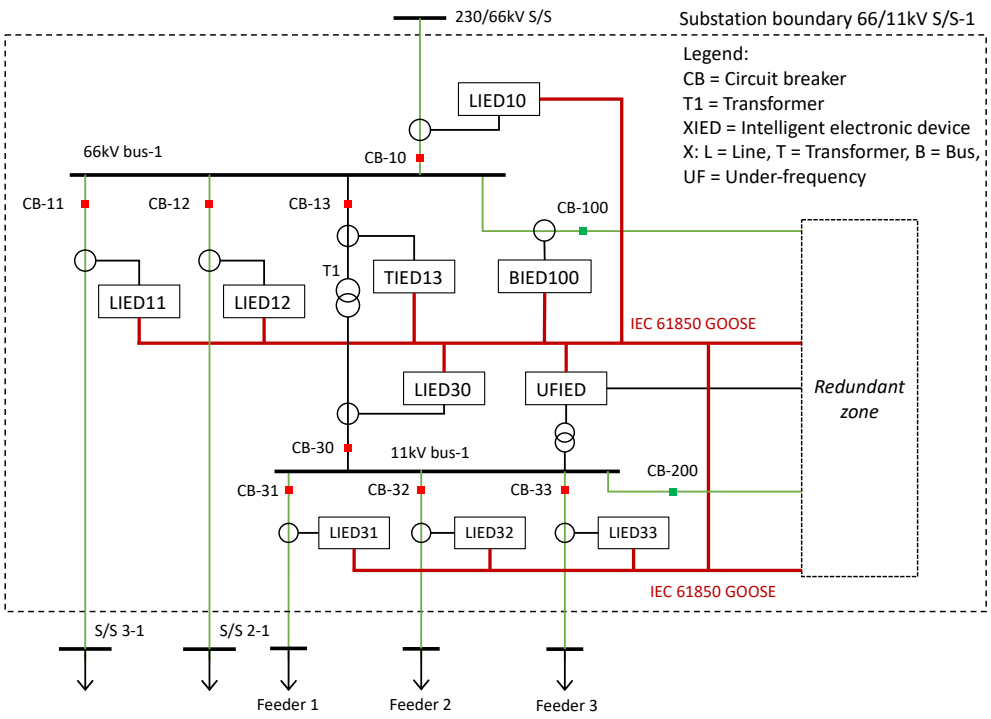


Figure 5.3: One-line diagram of one of the two redundant zones of the substation used for the generation of the dataset described in [108]

i.e., the SCL files enable us to implement the multilayer detection approach, as we will discuss in the next section.

The dataset includes traces of GOOSE communication among the IEDs. The traces represent three operational scenarios: normal operations, benign power system disturbances, and attacks. In the normal scenarios, the substation works within safe bounds with no failures. The benign disturbances include the occurrence of a fault at one of the busbars, failure of one of the circuit breakers, and occurrence of under-frequency. Those disturbances lead to a coordinated reaction among the IEDs, leading to the opening of associated breakers in order to keep the system in a safe state.

Of the different attack scenarios present in the dataset, message suppression (MS) and data manipulation (DM) attacks come under the false data injection model. The MS attacks inject packets consisting of malicious headers to attempt GOOSE poisoning. The DM attacks inject GOOSE messages with malicious payloads intended to cause instability in the system. The traces of a coordinated MS + DM attack are also present; we refer to that as a *composite* attack.

To test the accuracy of the detection of false data injection attacks not present in the dataset, we implemented an attack injection module. The module injected into the normal

communication traces GOOSE packets with the following violations: (1) unknown source and destination MAC addresses, (2) unknown `DatSet`, (3) incorrect value of `Length` field, and (4) an invalid `TAL`. We also used the GOOSE poisoning attack traces from Chapter 4's experiments.

5.6.2 Implementation

We deployed Zeek (version 2.5.4) on an Ubuntu 16.04.6 LTS VM with 2 GB of RAM and four virtual CPUs. The host computer had macOS version 10.15.4 on a 2.6 GHz quad-core Intel Core i7 with 8 GB of DDR3 RAM. To emulate live traffic analysis, we used the Tcpreplay [113] utility, running locally on the VM, to replay the dataset traces.

We implemented attack injection, GOOSE analysis, logging, and attack detection scripts. The attack injection used Python's Scapy module [114]. For logging and alerting, we utilized Zeek's Logging Framework and Notice Framework, respectively.

We applied the details of the physical model and network architecture (shown in Figure 5.3 and described in [108]) to guide the development of various detectors in SEEZE. We obtained the details of GOOSE communication and datasets from SCL files and learned safe ranges of measurements from the normal operation traces of the dataset.

First, we implemented the whitelisting-based detector by using Zeek's Input Framework [129]. Input Framework offers several crucial features. It allows asynchronous reads from files to ensure that the performance of analysis is not affected when the input files are large. It can also detect changes to the input files to enable the automatic refresh of the whitelisting information. In our implementation, we stored the whitelists, i.e., known endpoints, GOOSE identifiers, and valid pairs in text files. Input Framework then allowed us to read the information asynchronously in the attack detection script.

Next, for the semantic verification and specification-based detector, we used recommended values for the different GOOSE fields. For example, we used 10 s as the maximum value for `TAL`. We also used our implementation of the specification-based poisoning detector from Chapter 4.

Finally, we implemented the physical behavior-based detector by using the learned safe ranges and IED relationships. In particular, we specified rules indicating the correlated GOOSE datasets that represent the measurements, status, and alarms for the same power line. We also specified the upstream-downstream relationships among the datasets across different IEDs. We used the following settings for the parameters used by the detector. We set the trip threshold for the current measurements to be 5% above and below their average normal value. We used the value of 4 s for the time window within which a trip

corresponding to a disturbance should happen.

5.6.3 Detection Accuracy Results

We fed the dataset traces to SEEZE and compared output alerts with the known attack-injection points. We found that SEEZE could correctly identify all the MS, DM, and composite attacks present. Moreover, it did not generate any alerts for the normal or benign disturbance scenarios. Thus, there were no false positives for the given dataset.

Table 5.1: Alerts generated by SEEZE for DM.1 attack, showing the detection of both GOOSE poisoning and data manipulation.

No.	Fields	Values
1	Timestamp Note Message	1591926649.210878 GOOSE::Poisoning Injection of a packet with expected numbers: stNum=1, sqNum=11, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, detectionTime=1591926649.213720
2	Timestamp Note Message	1591926649.246565 GOOSE::Data_Manipulation False data injection attack: stNum=1, sqNum=11, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, allData=[380,310,310,38105,38105,38105,30000000,18500000,50.000000,0.850000], detectionTime=1591926649.248683
3	Timestamp Note Message	1591926649.570483 GOOSE::Poisoning Injection of a packet with expected numbers: stNum=1, sqNum=21, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, detectionTime=1591926649.571001
4	Timestamp Note Message	1591926649.610485 GOOSE::Data_Manipulation False data injection attack: stNum=1, sqNum=21, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, allData=[310,270,310,38105,38105,38105,30000000,18500000,50.000000,0.850000], detectionTime= 1591926649.611057
5	Timestamp Note Message	1591926649.930558 GOOSE::Poisoning Injection of a packet with expected numbers: stNum=1, sqNum=31, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, detectionTime= 1591926649.931023
6	Timestamp Note Message	1591926649.970577 GOOSE::Data_Manipulation False data injection attack: stNum=1, sqNum=31, srcMAC=00:09:8e:21:73:33, datSet=LIED10MEAS/LLN0$Measurement, allData=[310,310,360,38105,38105,38105,30000000,18500000,50.000000,0.850000], detectionTime= 1591926649.971107

We show examples of alerts generated by SEEZE in Tables 5.1 and 5.2. The information in alerts includes a timestamp; a note, i.e., a user-specified category for the alert; and a message, i.e., a user-specified description of the alert. Table 5.1 shows the alerts corresponding to the first data manipulation attack (DM.1) present in the dataset. DM.1 involved the injection of three valid GOOSE messages consisting of false current measurements. As we see in the table, SEEZE was able to identify each malicious packet, both as a poisoning attack and as a false data injection attack.

Table 5.2: Alerts generated by SEEZE for the composite attack, showing the detection of both GOOSE poisoning and data manipulation.

No.	Fields	Values
1	Timestamp Note Message	1591926846.730519 GOOSE::Poisoning Injection of a packet with high stNum: stNum=9999, sqNum=0, srcMAC=00:09:8e:21:73:32, datSet=LIED11CTRL/LLN0$Status, detectionTime=1591926846.733977
2	Timestamp Note Message	1591926846.914502 GOOSE::Poisoning Previous state recurred: stNum=1, sqNum=15, srcMAC=00:09:8e:21:73:32, datSet=LIED11CTRL/LLN0$Status, detectionTime=1591926846.915299
3	Timestamp Note Message	1591926846.946428 GOOSE::Data_Manipulation False data injection attack: stNum=2, sqNum=0, srcMAC=00:09:8e:21:73:32, datSet=LIED11CTRL/LLN0$Status, allData=[0,1,0,1,F], detectionTime=1591926846.947130

Table 5.2 shows the generated alerts corresponding to the composite attack, further demonstrating SEEZE's detection accuracy. That attack involved two malicious messages: a high-`StNum` GOOSE message and a message attempting to modify a circuit breaker's status. Accordingly, the first alert in the table corresponds to the detection of the first malicious message.

The second attack message consisted of the `StNum` and `SqNum` counters' expected values and a malicious payload. For this message, the (`StNum`,`SqNum`) values were (2,0). The benign message preceding to the malicious message had values of (1,14). Even though the malicious message was a valid transition for the counter values, one corresponding to a new state, it was detected by SEEZE when the next benign message arrived with (1,15) values. The detection happened because of the recurrence of a previous state (alert no. 2). Moreover, SEEZE also detected the malicious payload by identifying the inconsistency between the circuit breaker's status and corresponding measurement (alert no. 3). It is also important to note that although the detection of data manipulation happened after GOOSE poisoning,

the actual sequence of those two packets was the opposite. That order was because of the time parameters of the two detectors. The time window for the physical-behavior-based detector was longer than the GOOSE poisoning detector's staging state (i.e., the `TAL` of the previous benign message).

5.6.4 Performance Results

Our next objective was to evaluate SEEZE's overhead in analyzing network traffic. To achieve that, we measured the overall response time at the maximum traffic rate that SEEZE could handle without dropping packets. The response time accounted for the network latency, waiting time, processing time, and log-writing time.

In the evaluation dataset, about 87% of the traffic represents GOOSE communication, whereas the remaining 13% includes conventional protocols, such as ARP, ICMP, DNS, and DHCP. The traffic thus resembles a real substation scenario in which other types of traffic accompany IEC 61850 protocols. In our experiments, we performed Zeek's default analysis for non-GOOSE packets, whereas for all GOOSE packets, we applied the multilayer security analysis.

Figure 5.4 shows the per-packet response time, separated into the shares of all the detection layers. The "Base" layer at the bottom of the bar graph corresponds to the case in which SEEZE performed no security analysis but just the basic logging of GOOSE communication. The three shaded boxes stacked on Base represent the whitelisting-based, specification-based, and physical-behavior-based detectors, respectively. As shown in the figure, the per-packet response time for the Base case was 0.038 ms. When we added the successive layers of detectors, the response time increased, reaching up to 0.127 ms when all the layers were active.

The response time corresponds to the maximum throughput that SEEZE could realize without any packet drop, across about 100 experimental settings for each layer. We therefore do not report a confidence interval across those settings. Moreover, since we computed the average response time by dividing the completion time (corresponding to the maximum throughput) by the total number of packets, we could not compute the variance. However, we note that since the dataset has GOOSE as the majority of traffic, the average response time is reflective of what GOOSE messages incurred. Within the GOOSE traffic, the benign messages would have higher response times than the attacks, because benign messages are processed by all the layers. In contrast, attack packets are processed only until an attack is detected.

SEEZE thus achieved a response time of 0.127 ms per packet, even when deployed as a VM

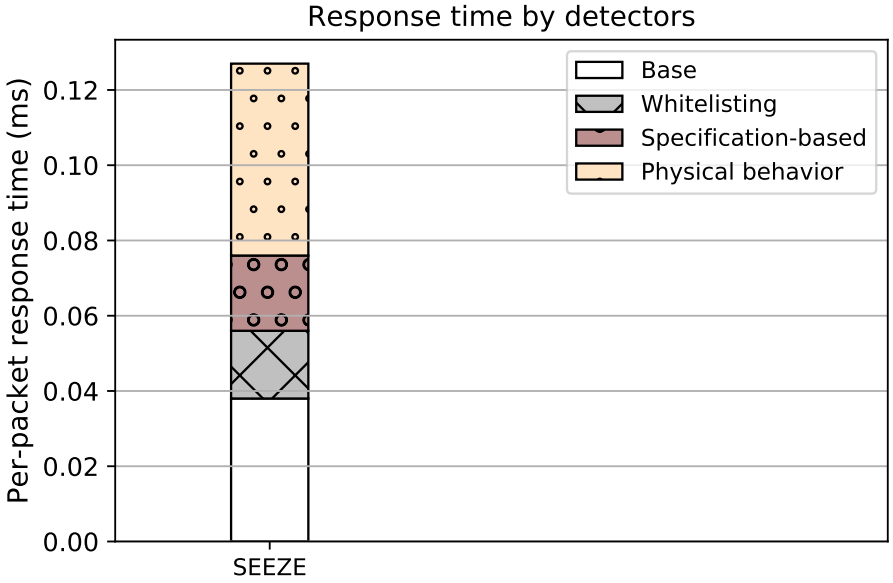


Figure 5.4: Packet analysis response time for SEEZE.

on a general-purpose workstation. That value is less than 5% of the most stringent GOOSE transfer time of 3 ms. Therefore, SEEZE is suitable for a concurrent security analysis of GOOSE traffic. We note here that the response time is likely to be smaller when SEEZE is deployed directly on the hardware at a substation edge server, as discussed in Section 5.4. Moreover, the performance optimization techniques presented in Chapter 4 can help further reduce the response time.

5.6.5 Comparison with Existing Similar Systems

We now compare our results with those of existing research efforts that have also adopted a multi-attribute approach. Those systems are (1) multidimensional IDS for IEC 61850-based SCADA networks (*SCADA-IDS*) [120, 130] and (2) security monitoring system that use the IEC 61850 network and system management (*NSM-IDS*) [118].

SCADA-IDS, like SEEZE, consists of multiple layers of detectors, including whitelisting, specification-based, and physical behavior-based detectors. It does detect attacks on MMS and SV protocols, in addition to those on GOOSE. SDADA-IDS is shown to achieve an execution time of less than or equal to 254 μs.

In comparison with SCADA-IDS, SEEZE focuses only on GOOSE communication and achieves a response time of 127 μs. Within that response time, SEEZE performs GOOSE

poisoning detection, which is not present in SCADA-IDS. Finally, SEEZE performs a highly stateful analysis of the traffic to separate benign disturbances from attacks. SCADA-IDS, on the other hand, would generate false positives for the disturbance scenarios [108].

NSM-IDS uses a forecasting-based anomaly detection approach to detect DoS conditions caused by malicious GOOSE header fields. Its threat model covers a broader class of attacks on GOOSE headers than the one of SEEZE. However, unlike SEEZE, which can detect harmful measurements, NSM-IDS does not handle attacks on the payload. Further, as NSM-IDS is an anomaly detection system, it might suffer from high false-positive rates.

NSM-IDS seems suitable for offline analysis of network health data to identify anomalies. In contrast, SEEZE can analyze traffic to detect attacks in real-time. Unfortunately, the authors of NSM-IDS do not provide results on the detection accuracy or the response time. Therefore, we are unable to compare them with SEEZE's.

5.7 RELATED WORK

5.7.1 Network-Level IDSes

Substation environments consist of resource-constrained IEDs and simple, insecure protocols. Therefore, the network-level detection of attacks on control traffic has been a popular choice among the related research efforts. Such efforts also include specification-based methods, e.g., [131, 132]. The authors of [106, 133] harnessed the capabilities of software-defined networking to develop network-based IDSes. Another such system, HAMIDS [126], uses a distributed architecture of protocol analyzers for conventional protocols such as TCP, UDP, ARP, CIP, and EtherNet/IP.

In particular, for IEC 61850 substations, the network security monitor proposed in [118] relies on SNMP to collect usage and statistical data from IEDs. Such data collection is then used to train a machine learning model and detect cyber attacks. The authors of [116] describe a substation network IDS focused on detecting attacks on conventional protocols, such as ARP, ICMP, FTP, and HTTP.

5.7.2 Multilayer Detectors

Like our work, the following systems have adopted multilayer analysis for intrusion detection. The multi-attribute IDS described in [130] uses protocol-based whitelists and several behavior-based detectors to detect SCADA-specific cyber threats. That system is further extended and applied to IEC 61850 substations in [120]. EDMAND [105] extracts three

components from network traffic—transport, operation, and content—and performs statistical analysis to detect network anomalies associated with DNP3 and Modbus traffic. Work presented in [118] uses a combination of machine-learning-based and rule-based methods on host-level data to detect DoS conditions in IEC 61850 traffic. In comparison, SEEZE focuses on a broad class of false data injection attacks, including sophisticated GOOSE poisoning and data manipulations, and detects attacks concurrently with the data transmission.

5.7.3 Zeek for Smart Grid Security

Zeek is an open-source and extensible platform that allows integration of protocol analyzers and custom scripts to define events of interest and detect violations. Our decision to use Zeek was based upon existing related research. A proof-of-concept on the utility of Zeek in a smart grid control network is provided in [121]. In that work, the authors describe how they developed a DNP3 protocol analyzer and validation policy to detect violations. Later use of such a protocol parser for DNP3 is discussed in [131, 132] in the context of developing specification-based intrusion detection systems for SCADA networks.

In addition, recent efforts have also used Zeek as a network traffic extractor [105] and a deep packet inspector [106]. HAMIDS, presented in [126], uses multiple instances of Zeek, each sending logs to a central location. Each instance involves a variety of protocol parsers (e.g., TCP, UDP, ARP, CIP, and EtherNet/IP), an event engine, and a policy script interpreter. Zeek, however, does not offer interfaces for the integration of data-link-layer protocol analyzers. Therefore, the GOOSE analyzer proposed in [107] has not been integrated with Zeek at the time of this writing.

5.8 CONCLUSION

False data injection is a significant challenge for the security of substation networks because of the limited processing power of end devices and insecure and simple protocols. Adversaries who can insert themselves behind the firewall can issue commands and control devices to cause harmful results. In this chapter, we presented a system, SEEZE, that relies on improved network monitoring for the effective detection of false data injection. SEEZE draws upon the successful application of Zeek in smart grid network monitoring and security. Building on Zeek's real-time traffic analysis capabilities, SEEZE performs a highly stateful analysis without sacrificing performance. SEEZE has adopted a multilayer detection scheme involving the following layers: (1) whitelisting, (2) a protocol-semantic-based layer, (3) a protocol-specification-based layer, and (4) a physical-behavior-based layer. The layers allow

for fusing of the protocol specifications, system configuration knowledge, and the substation's cyber-physical state. Through an extensive evaluation using substation network traffic, we demonstrated that SEEZE can detect various attack scenarios. Those scenarios include, (1) GOOSE poisoning, (2) injection of GOOSE packets with a manipulated header or payload, and (3) modification of the payload on packets in transit. SEEZE achieved a fast per-packet response time, which was less than 5% of the most stringent GOOSE transfer time. Thus, SEEZE is suitable for real-time security analysis of GOOSE communication.

CHAPTER 6: CONCLUSION

Attacks on networked systems continue to happen and remain undetected for long periods. Through persistence, the attacks gain capabilities to carry out malicious actions. Traditionally, intrusion detection has involved looking at individual monitors separately, which leads to missing of important alerts among large volumes of logs. In addition, some patterns might not reveal themselves in single monitors. Thus, to improve detection, we need to analyze security information in a collaborative manner. We refer to the methodology that enables such collaborative analysis as *information-fusion-based intrusion detection*.

In this dissertation, we presented theoretically sound and practically useful techniques for improving the detection of advanced cyber threats. The advanced threats are based on persistent and targeted tactics, often launched in a reusable and layered manner, in the compromise of large organizations. Accordingly, we utilized the fusion of diverse monitoring information to detect malicious events accurately and provide alerts to enable better visibility into the current state of the system. More specifically, we introduced a set of unsupervised anomaly detection methods to address initial compromise and lateral movement in enterprise networks. We then presented a specification-based system and combined both cyber and physical attributes for efficient detection of false data injection attacks on power grid substation networks. Finally, to accommodate the strict timing requirements of such networks, we devised systematic methods to overcome bottlenecks, increase throughput, and evaluate the coverage of the false data injection detection system. Thus, this thesis provides a well-formed basis for improving the detection of advanced cyber threats.

In this chapter, we briefly review the work we have presented in this dissertation and describe potential avenues for the expansion of our work.

6.1 REVIEW OF CONTRIBUTIONS

6.1.1 Initial Compromise Detection

First, we presented a novel intrusion detection technique that uses unsupervised clustering algorithms to identify malicious behavior within large volumes of monitor data. We formulated the problem as identification of hosts that exhibit behavior that deviates significantly from normal usage patterns in order to detect compromises. We did not wish, however, to explicitly profile normal usage; instead, we wanted to learn the profiles in an unsupervised manner from the data.

To construct the empirical models, we used the dataset published by the Visual Analytics Community for the VAST 2011 Challenge [48]. From that dataset, we utilized the firewall logs, Snort intrusion detection system logs, and Windows operating system security event logs. The dataset also includes traces of attacks, of which we considered two categories that are typically used to prepare for the later stages of an advanced threat. The first category includes network scans and flooding-based network attacks, such as port scans, port sweeps, and network-layer Denial-of-Service (DoS) or Distributed DoS attacks. The second category of attacks that we aim to detect is the presence of malware on a host.

We then extracted features from the firewall and system logs and used Snort IDS logs only to compare and validate the results. We chose features based on both the attacks discussed in our threat model and domain knowledge of the enterprise network on which the dataset was based. We identified four categories of features, namely: (1) identification features (IP address and timestamp), (2) traffic-based features (statistics on connections and ports), (3) service-based features (counts of accesses to workstations and important servers), and (4) authentication-based features (statistics on logon/logoff and authentication protocols). We performed several analyses to refine the set of features, including the examination of empirical CDFs, analysis of linear dependence between each pair of feature vectors, and pruning of redundant features.

We applied the DBSCAN clustering algorithm to discover clusters of similar data points and analyzed the joint distributions of features in each cluster to identify the types of behavior associated with each cluster. In this context, we computed the *normalized average feature value vector*. Using those vectors, we then defined a *cluster difference* metric and a *cluster normalcy* metric. With such metrics, we generated a prioritized list of the malicious clusters, ordered by decreasing value of cluster normalcy. Thus, by separating the clusters by behavior, reducing the clusters to those that are anomalous, and prioritizing the clusters by likely maliciousness, we significantly simplified the work that must be performed by an administrator.

We evaluated the ability of our approach to identify the attacks actually present in the dataset. Our approach performed very well, separating out the attacks and providing those clusters as the ones most likely to be malicious. As a sanity check, we compared the performance of our completely unsupervised approach to that of Snort. We found that Snort detected the network-based attacks, but it missed the malicious internal host. Detection using Snort also required an administrator to sift through many lines of logs. Hence, by using the joined network-level and host-level data, our approach could detect more attacks than a state-of-the-art network IDS could.

The main contributions of this work include (1) unsupervised cluster analysis to automat-

ically learn usage behavior and (2) a cluster prioritization approach to identify malicious behavior. The key innovation was to use joined network-level and host-level data to detect attacks that were not detectable with either data source alone.

6.1.2 Lateral Movement Detection

Next, we addressed attacks that would need to have several interactions (e.g., internal reconnaissance, credential access, vulnerability exploits, and communication with malicious external domains) with the victim network to acquire persistence. We presented an approach to detect such attacks with high accuracy and a low false alarm rate in order to thwart an attack campaign and prevent actual damage.

Our approach relies on the information collected by three types of monitors: (1) a Network Flow Monitor that processes internal network traffic and generates flow records, (2) a C&C Monitor that analyzes the network communication of the hosts and assigns a relative suspiciousness score to each host, and (3) an LM Monitor that creates traces of internal connections that are chained together based on their dependence on each other.

We represented the information recorded by the monitors in a graph-based model, called a *host communication graph* (*HC graph*). Using the HC graph, we identified characteristic patterns of malicious LM and C&C, and represented them as statistical features. Those features include the `c_score` (which was provided by the C&C Monitor) and five statistics on lateral movement graphs (that were generated by the LM Monitor). The LM-based features included the number of LM graphs and statistics on the size of these graphs (mean, range, interquartile range, and median absolute deviation). The HC graph thus allowed us to combine diverse indicators and construct a system-wide state. We then analyzed the empirical probability distributions of the features to identify their relative importance and potential effects in finding anomalies. We found that some of the features were correlated. Therefore, we applied PCA to analyze the correlation among the features and reduce the dimensionality of the dataset before anomaly detection was performed.

The traditional anomaly detection methods that assume that the size of the anomaly (e.g., the number of compromised hosts) is significantly smaller than the size of normal activity did not fit directly in our case. Therefore, we proposed a combination of multiple anomaly detection algorithms to achieve high accuracy and robustness while facing variations in attacker behavior. Intuitively, the approach is first to find the hosts that exhibit at least one of the two types of suspicious indicators—abnormal `c_score` or high disparity in lateral movement graphs—and then use the relative similarity of hosts to identify compromised hosts that do not directly show any of the suspicious indicators. More specifically, to detect

the two types of suspicious indicators, we use MAD-based outlier analysis and PCA-based extreme vale analysis, respectively. Groups of similar hosts were obtained using k-means clustering.

For our experiments, we used a dataset of normal network operations from the Los Alamos National Lab's enterprise network. To inject attack traces in the dataset, we developed a model for C&C and LM activities that uses the N-intertwined susceptible-infected-susceptible (SIS) virus spread model. That model was selected after a thorough study of the graph structures generated by adversarial LM tactics in past APT incidents.

The trace injection method implements a small set of tunable parameters that allowed us to simulate different types of LM activities and evaluate the detection method's sensitivity to changes in the attacker's behavior. In particular, we varied the number of hosts that were part of the malicious LM and the number of hosts that were infected by C&C malware and then computed the true and false positive rates (TPR and FPR) of detection in each setting. Overall, on average (computed across all experiment runs), we achieved a TPR of 88.7% (confidence interval 1.9%) and an FPR of 14.1% (confidence interval 3.6%). In one of the experimental configurations, for example, a larger number of hosts were part of malicious chains; the average FPR was only 13.13%, while an average TPR of 91.08% was still achieved.

In summary, the main contributions of this work include (1) a novel ensemble of anomaly detectors to identify with high accuracy malicious lateral movement, and (2) a general LM attack-injection approach that allows one to generate traces with a variety of attack models and test detection methods.

6.1.3 GOOSE Poisoning Detection

Next, we addressed the final phase of an advanced threat, in which it can carry out malicious actions to disrupt the service. For that, we studied IEC 61850-compliant substation networks. IEC 61850 is an international standard that has been adopted in substations around the world. It enables cost reduction and ease of maintenance in implementing digital substations. However, cyber security is a big challenge in such systems because of unencrypted channels and Ethernet-based protocols.

In particular, we addressed GOOSE poisoning, a common class of false data injection attacks affecting system availability. Such attacks work by injecting valid GOOSE messages with modified headers to cause IEDs to stop processing legitimate traffic.

We introduced a network-level system, ED4GAP, dedicated to detecting GOOSE poisoning attacks. ED4GAP relies on the extraction of features from substation traffic, analysis

of protocol specifications, and a comprehensive examination of communication properties to detect violations and generate alerts. The detection uses a state machine model for the counters contained in the GOOSE header. With the model, ED4GAP performs a stateful analysis of traffic and stores traffic metadata in a staging state for a limited time. We showed that ED4GAP could detect all variants of GOOSE poisoning by storing in the staging state information on all the packets arriving within the `TAL` of the last benign message.

We used the Zeek network security monitor to implement a prototype of ED4GAP. We further devised a systematic approach to identifying bottlenecks and speeding up the traffic processing throughput by using the exclusive pinning of CPU cores. Using traces of substation communication, we showed that the presented system could analyze traffic and detect all the variants of a GOOSE poisoning attack with very low overhead. Thus, it is suitable for the context of high-speed communication within a substation.

This work's main contribution was to improve the accuracy of the detection of GOOSE poisoning attacks without sacrificing performance. The key innovation was the highly stateful analysis of traffic through use of the staging state, whose properties and size rely on protocol specifications.

6.1.4 False Data Injection Detection

Next, we extended coverage of the detection of false data injection beyond GOOSE poisoning. By using Zeek as the underlying platform and guided by our information fusion philosophy, we developed several layers of detectors that together cover broader classes of false data injection on GOOSE communication.

The detection layers include whitelisting, GOOSE semantic analysis, GOOSE poisoning detection, and physical behavior-based detection. The first two layers address the detection of early indicators of an attack, including malformed and semantically invalid messages. Next, the GOOSE poisoning layer covers the detection of GOOSE header manipulation attacks. Finally, physical behavior-based detection describes rules based on the physical architecture of the substation and safe operating ranges. That last layer is intended to cover attacks involving harmful payload, either through injection of packets or by manipulation of in-transit packets.

We implemented a system, SEEZE, based on the multilayer detection approach. We then evaluated its detection accuracy and performance overhead. For the evaluations, we used a substation network dataset with injected attacks. Those attacks included instances of GOOSE poisoning, bad measurements, incorrect GOOSE semantics, and access-control violations. Our results showed that SEEZE could detect all the attacks with a small response

time of less than 5% of the most stringent GOOSE transfer delay. Our results showed an improvement over the existing state-of-the-art multilayer detection systems.

In conclusion, we have proven our thesis statement: By utilizing information from multiple sources, data-driven methods can improve the ability of a network to detect sophisticated attacks.

6.2 FUTURE DIRECTIONS

6.2.1 Generalization of Approaches

In Chapters 2 and 3, we presented unsupervised anomaly detection methods that learn normal and abnormal behaviors without access to labeled examples. Thus, our methods have overcome one of the biggest challenges in cyber-security research: the lack of labeled datasets. However, the last decade has seen tremendous growth in the capabilities of supervised machine learning approaches, in particular the deep learning methods. Such growth has been driven by both new algorithms and better hardware capabilities. Our methods can be extended to benefit from such flourishing of deep learning. In particular, the results of the unsupervised detection methods can be used by analysts to train supervised learning models, e.g., a neural network or a random forest. Those trained models would then serve two purposes: (1) to reduce human effort by automatically reasoning about the current security posture, and (2) to improve the detection methods continuously by providing feedback to the previous stages of feature selection and clustering. To support such an architecture for combination and continuous improvement, we can adopt data stream clustering algorithms that can partition observations continuously while taking into account restrictions of memory and time.

The multiple detectors presented in this dissertation can be used together or separately to thwart the various stages of an advanced attack. Our focus has been mainly on using them separately because intrusion detection cannot be perfect, and some attacks would evade a detector and reach the next stage. However, once an attack has been detected, one would like to react as soon as possible to restore the system's safety. Waiting for a detected attack to reach its later stages is never a preferred option. Therefore, we did not study the correlation of detectors across different stages. Our approaches, however, can support alert correlation to provide higher-level awareness of attacks and enable root-cause analysis to identify attack strategies. To achieve those goals, we can incorporate frameworks proposed in the literature, such as CAPTAR [134] and DATES [135].

6.2.2 Lateral Movement Detection and Response Framework

In Chapter 3, we used a graph-based representation to guide the lateral movement detection approach. In that work, we addressed attacks that need to have several interactions with the victim network to acquire persistence. An attacker who can spread laterally without such interactions might evade detection by our approach. For example, a malicious insider or outsider with access to stolen credentials of several authorized users can laterally spread without generating anomalies.

We note that our modeling and detection approach can form the basis for a comprehensive framework for detection of and response to lateral movement attacks, including those that use the techniques mentioned above to be more targeted. In the following, we discuss a concrete example of such a framework, using credential-based lateral movement as the use case.

We can define a network authentication graph (NAG) to represent the authentications in an enterprise that typically utilizes Single Sign-On (SSO) systems. Formally, NAG is a directed graph $G(V, E)$. The set of vertices $V = \{1, 2, \ldots, n\}$ represents the hosts in the system. The set of edges $E \subseteq V \times V$ represents the network authentications between the hosts. A directed edge $(u, v) \in E$ represents an authentication from computer u to computer v. In a typical enterprise network, the authentications could be initiated either by human users or by automated services. In this model, we can capture all such authentications.

The NAG can be used to compute the risk associated with lateral movement attacks proactively. To do that, we can assign a risk score to each vertex. For the risk score, we define a vertex labeling function as $f_{\text{rs}} : V \rightarrow [0, 1]$, which labels every vertex of the graph G with a risk score between 0 and 1. The score quantifies the vulnerability of a computer with respect to lateral movement attacks, relative to other computers in the network. The risk score, essentially, is a function of the users and services present on a given computer. Finally, we also label each edge with the set of accounts (denoted by $\mathcal{A}$) that can move over the edge. Therefore, we define an edge-labeling function as $f_{\text{u}} : E \rightarrow \mathcal{A}; \mathcal{A} \subseteq \mathcal{U}$, where $\mathcal{U}$ is the set of all accounts in the network. Thus, the NAG would allow us to represent the information that is crucial in responding to credential-based lateral movements.

We call the critical servers in a given network as *high-value assets* (*HVAs*). As the NAG provides risk values for each vertex, paths reaching from hosts to HVAs that have high risk can be identified. Equipped with those paths, security administrators can take actions to lower the risk, for example, by increasing the monitoring on the paths or downgrading the hosts on the paths to have fewer services.

Next, the NAG can also be used to identify the regions that should be considered com-

promised, given a set of infected machines. With that information, the following concrete responses can be carried out: (1) learning the target of an attack by actively disturbing the lateral movement chains and evaluating the changes in the attacker's path, (2) blocking or throttling a suspected remote address, and (3) isolating from the rest of the network a subset of hosts that are part of malicious chains.

6.2.3 Supporting Future Architectures

Of the multiple avenues of future research opened up by the work presented in Chapters 4 and 5, we list the following.

At the time of this writing, cryptography-based systems are not yet standard in real-world substations because of the limited processing power of IEDs. Bump-in-the-wire (BITW) approaches have been proposed that rely on computing signatures and codes that use pre-shared symmetric keys. Our multilayer edge-based system will be able to incorporate detection of attacks on the BITW approaches when such systems are applied in real substations. For instance, we can include the detection of invalid signatures or violation of message authentication codes.

Another field of recent research is the adoption of cloud computing in control networks and supervisory control and data acquisition (SCADA) applications.. Such deployments are often referred to as *SCADA-as-a-Service* (*SCaaS*) or *SCADA-Cloud architectures*. The proliferation of distributed energy resources and microgrids is the primary driver of those advancements. Our edge security architecture is compatible with the adoption of cloud computing technologies. The substation logs and alerts generated by our system can be securely transmitted to cloud systems for analysis over a longer timescale and broader system boundaries. Since our system resides on an edge server, its communication with the cloud system will not affect the substation's end devices and control communication.

In conclusion, we believe that several immediate and future extensions to our methods are possible, including those discussed above. With those additional capabilities, our methods will be able to support novel analyses and future deployments.

APPENDIX A: SOFTWARE TESTBED TO FACILITATE EVALUATION ON REALISTIC NETWORKS

Validation and evaluation of intrusion detection solutions for large networked systems remain challenging problems. Complex cyber-physical systems like substation networks pose additional restrictions. In that regard, we present a network testbed to provide an evaluation platform for quick validation and rapid prototyping of various networking applications, including network traffic engineering and cyber security. We call the testbed MNEST, which stands for Mininet-based NEtworking and Security Testbed. MNEST relies on Mininet and software-defined networking (SDN) technologies to emulate arbitrary network architectures, generate real traffic, and collect benchmarks. To demonstrate MNEST's utility, we describe a case study supporting the empirical evaluation of the G2 network optimization framework [136]. Further, we discuss specific extensions to it that will support the evaluation of the methods presented in Chapters 4 and 5.

The case study presented in this appendix was published in [136] and [137]. This appendix presents the author's contribution, which was the testbed's development and application in evaluating the G2 framework. The framework itself was developed by Dr. Jordi Ros-Giralt, and it is briefly described to provide context for the author's work. Although the testbed is relevant for evaluating the advanced threat detection methods presented in this dissertation, the current case study does not relate to that theme. Therefore, the work was not included in the main body of the dissertation.

A.1 SUMMARY OF CONTRIBUTIONS

Empirical evaluation of new methods on real systems is costly, sometimes prohibitively so. Simulation-based and trace-based tools provide alternative evaluation methodologies. However, without access to real network components and traffic, the simulation-based systems fall short in attempting to realize the characteristics of real environments.

Emulation-based platforms, e.g., Mininet [138], can provide the benefits of both classes of systems mentioned above. Such platforms provide real networking components, typically implemented with software. At the same time, they allow rapid and cost-effective experimentation. Therefore, new methods evaluated on emulated platforms can transition to real environments with minimal changes.

Motivated by those observations, we introduce MNEST, a Mininet-based NEtworking and Security Testbed. We present the testbed's components and illustrate its use in supporting a network optimization framework. Specifically, we have used the testbed to generate real-

world network architectures and collect performance results for the G2 network optimization framework [136]. G2 uses those network architectures to provide insights into bottleneck structures and suggest performance improvements. We describe in detail the emulation of one such network architecture, i.e., Google's B4 network [139], along with the benchmarks collected. Last, we discuss an extension to MNEST to support the evaluation of cyber-physical threat detection in substation networks.

Our main contributions can be summarized as follows.

Design and implementation of MNEST. We present the design and implementation of MNEST, a Mininet-based Networking and Security Testbed. MNEST uses Mininet, SDN, and various UNIX/Linux system tools to create realistic network architectures and conduct experiments.

Support for empirical evaluation of G2. We have extended MNEST by implementing modules to support experiments for the G2 network optimization framework. We have bundled the resulting platform as a sandbox, which we have made available to the research community [140].

The rest of the appendix is organized as follows. In Section A.2, we present the design of MNEST, including the details of the technologies used. Next, in Section A.3, we describe our case study on supporting the evaluation of the G2 network optimization framework. In Section A.4, we mention specific extensions to MNEST to enable the evaluation of our approach for detecting false data injection attacks in substation networks. Finally, we discuss the related work in Section A.5 and conclude in Section A.6.

A.2 THE MNEST TESTBED

MNEST provides a flexible platform for emulating arbitrary network topologies, generating network traffic, creating flows, and writing custom experiment scripts. It uses the Mininet network emulator [138] with the POX SDN controller [141] to instantiate realistic networks. It allows the generation and monitoring of network traffic through the use of Linux tools and other third-party systems. In this section, we describe the testbed's architecture, followed by details on the technologies used.

A.2.1 Architecture

Figure A.1 shows a high-level system diagram of MNEST. To generate a network and perform experiments, an operator provides three types of inputs to MNEST:

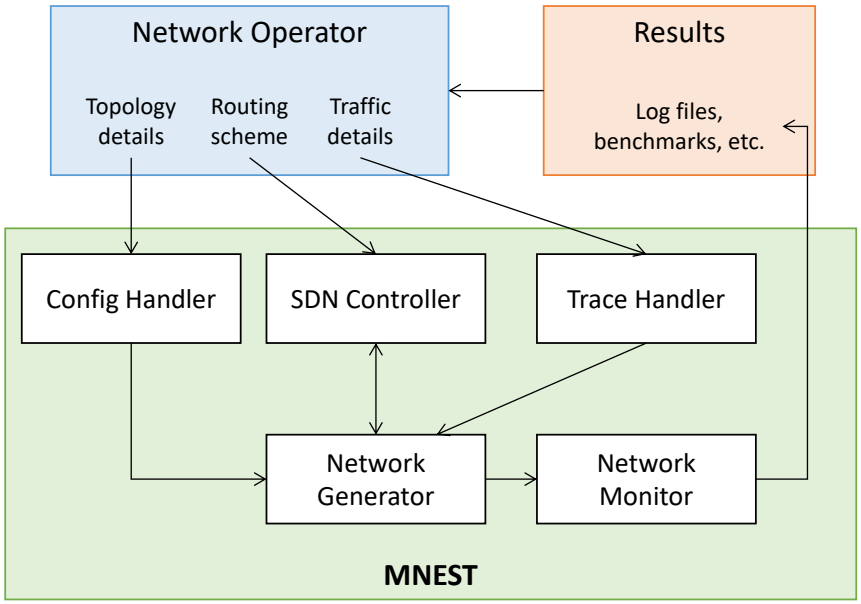


Figure A.1: MNEST architecture.

- Topology details consisting of important configuration settings related to topology (e.g., details of hosts, switches, and links), benchmarking (i.e., experiment and monitoring settings), and input/output paths.
- A routing scheme that indicates the mechanism that will be used to route traffic over the network, e.g., the shortest path. Further, any custom scheme can be specified by providing specific path information in the JSON format.
- Traffic details, including the specifications of network flows to be run on the emulated network. Such details include flow endpoints, data sizes, and timestamps.

With those inputs, MNEST emulates a Mininet network, launches traffic, and runs specified experiments to generate logs and benchmarks. The following components form the core of MNEST:

Config Handler. This component is responsible for parsing the topology-related configurations and passing them to the Network Simulator module.

Network Generator. The Network Generator creates a Mininet topology and launches a Mininet network by using the topology-related configurations supplied by the Config Handler. This module essentially enables MNEST to utilize the Mininet platform and related capabilities.

SDN Controller. MNEST uses the POX SDN framework to manage the control plane of the network. In particular, the controller installs flow rules on the switches for both layer-two and layer-three packets, according to the specified routing configurations. It also monitors the switches to identify any topology-related changes and failures.

Trace Handler. This module launches traffic flows on the generated network.

Network Monitor. The Network Monitor implements monitoring and data collection modules to observe network traffic and compute desired statistics, including throughput, flow completion times, flow convergence times, and switch statistics, among others. It can also support custom monitoring platforms, such as Zeek [39], to extend the monitoring to security applications.

A.2.2 Overview of the Technologies Used

Mininet. MNEST uses Mininet to emulate arbitrary network architectures. Mininet provides an ideal way to create inexpensive and flexible network testbed. Using process-based virtualization, Mininet runs a collection of hosts, switches, routers, and links on a single Linux kernel, turning it into a complete network.

More specifically, each Mininet host is a container attached to a *network namespace*. A network namespace holds a virtual network interface, along with its associated data. Virtual interfaces connect to software switches, e.g., Open vSwitch [142], through virtual Ethernet links.

Networks emulated using Mininet run a real kernel, network stack, and application code. A user can interact with the hosts and switches and run arbitrary applications on them as if they were real physical components. The supported ways for the user to interact with Mininet components include CLI and Python API. Therefore, the code developed for a Mininet network can move to a real system with minimal changes

In comparison with actual hardware testbeds, Mininet offers an inexpensive and always-available solution. In contrast with simulators, such as ns-3 [143], Mininet runs real code (i.e., OS kernel, applications, and network control plane) and allows integration with real networks.

SDN Technologies. MNEST uses the POX SDN framework [141] to implement a routing scheme, specific to a given experiment. In such an implementation, a custom forwarding module at the POX controller uses the OpenFlow protocol [144] to interact with switches

and remotely control their forwarding tables. Mininet inherently supports both POX and OpenFlow, making it easy to integrate the forwarding modules with emulated networks.

Linux Tools. Finally, MNEST supports several tools that are either native or installed on UNIX/Linux platforms. For instance, the use of `iPerf` [145] allows MNEST to generate TCP and UDP traffic and measure network throughput and related metrics. Similarly, `tc` utility enables MNEST to control link capacity using hierarchical token bucket (HTB). It also facilitates the collection of statistics, such as the number of queued packets and the number of dropped packets, from specific switch interfaces.

A.3 CASE STUDY: SUPPORTING EVALUATION OF THE G2 FRAMEWORK

We now present a case study wherein we utilized the MNEST testbed to support the evaluation of the G2 network optimization framework [136]. To provide the necessary context for our work, we will give a brief overview of the G2 framework. We will then describe the encapsulation of MNEST as a sandbox, *G2-Mininet* [140], to facilitate emulation of the networks used in the evaluation of the G2 framework. Finally, we will present the benchmarking results for one such network architecture, i.e., Google's B4 network [139].

A.3.1 Summary of the G2 Framework

G2 is a network optimization framework for characterizing the interactions of bottlenecks and the performance of flows. It informs traffic engineering decisions and performance improvement strategies by providing insights into the effects of perturbations in links and flows.

Bottleneck links play a crucial role in determining the performance of computer networks. However, in real-world networks, it is challenging to identify the hidden relationships among bottlenecks, leading to limited visibility into the precise effects of link and flow changes on the performance.

G2 addresses that problem by utilizing the theory of bottleneck ordering. The theory shows that the relationships of bottlenecks in a network follow a specific digraph structure called the *bottleneck structure*. The bottleneck structure reveals the inherent topological properties of a network and identifies the region of influence for every link. With such insights, the theory enables a high-precision analysis of bottlenecks and flows and allows operators to optimize network performance.

A.3.2 The G2-Mininet Sandbox

To demonstrate the importance of bottleneck structure analysis and related insights in improving network performance, G2 needs to experiment with real-world network architectures. To support such experimentation, we customized MNEST by implementing G2-specific modules and experiment scripts. We bundled that customization as a sandbox, which we call *G2-Mininet*.

G2-Mininet essentially emulates the desired Mininet network, given the topology, routing, and traffic configurations. The G2 framework then monitors those networks to obtain real-time traffic flow information, compute bottleneck structure, and facilitate optimization.

In G2-Mininet, we implemented a new forwarding module for POX to allow static routing on emulated network architectures. We then provided the ability to select a specific TCP congestion control algorithm (e.g., BBR [146] or Cubic [147]) and scale to larger network configurations on multi-core machines. Finally, we used `iPerf` to generate network traffic.

By varying the topology, the congestion control algorithm, and the properties of traffic-flows, we can generate a variety of real-world network scenarios. G2-Mininet can then create a network corresponding to each scenario and allow G2 to analyze the network performance. To achieve that goal, we implemented modules for benchmark collection as part of the G2-Mininet sandbox. The statistics that we collected during the experimentation include instantaneous network throughput, flow completion times, flow convergence times, and Jain's fairness indexes [148].

We have open-sourced G2-Mininet [140] and included all other network configurations not discussed in this chapter for the use of the research community.

A.3.3 Emulating Google's B4 Network

To demonstrate that G2-Mininet can faithfully emulate a real-world architecture, we defined an experiment involving Google's B4 network topology. Specifically, our objective was to emulate the B4 network, launch real traffic, compute useful statistics, and validate those statistics against theoretical values.

B4 [139] is a private WAN connecting Google's data centers across the planet. It uses SDN and OpenFlow to provide centralized traffic engineering and network management. B4 essentially removes "protocol speakers" from the switches and puts them on high-performance servers to leverage Google's compute to do smart pathing. B4 thus allows Google to achieve better performance, scalability, and availability in their data center network.

Using G2-Mininet, we created B4's global deployment architecture, which we show in

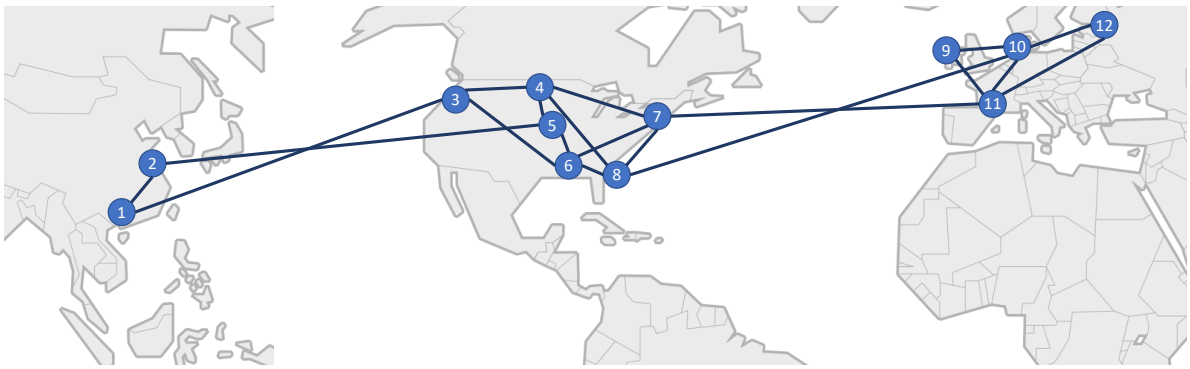


Figure A.2: Google's B4 network as described in [139]. The numbers are the IDs that we used to refer to each site. We emulated each site with single host-switch pair.

Table A.1: Traffic flows launched on the emulated B4 network.

Flow ID	Start time (s)	Source	Destination
1	0	$h1$	$h12$
2	2	$h2$	$h10$
3	4	$h3$	$h10$
4	6	$h5$	$h9$
5	8	$h6$	$h12$

Figure A.2. In our emulation, we represented each site with a single network switch and associated one host with each such switch. The numbers in the figure are the switch (as well as the host) IDs, as utilized in our experiments. Thus we called the switches $s1$ through $s12$ and the hosts $h1$ through $h12$. We configured the network with the BBR congestion-control algorithm and the shortest-path routing scheme. We set the following properties for the network links: bandwidth = 1000 Mbps, delay = 2 ms, and loss = 0%.

We then generated five flows of network traffic, each of size 256 MB, as shown in Table A.1. The flows were designed such that each of them passed through the link between switches $s6$ and $s8$. We call that link our *bottleneck link*. We configured the bottleneck link with bandwidth = 100 Mbps, delay = 10 ms, and loss = 1%.

Theoretically, each flow's effective throughput should be 20 Mbps (because five flows share the bottleneck link of 100 Mbps capacity). Thus, it should take roughly 102.4 s for each flow to complete. However, given the link properties, i.e., the delay and loss on the bottleneck link and the different start times of the flows, we expected the throughputs (and the completion times) to be different from their theoretical values.

We then launched the traffic flows on the emulated B4 network and computed network

throughput and flow completion times. We present those results in Table A.2.

Table A.2: Statistics obtained from the experiment with the B4 network.

Flow ID	Sender Avg. Mbps	Receiver Avg. Mbps	Flow completion time (s)
1	16.30	16.07	126.5
2	15.95	15.53	128.0
3	17.48	17.16	119.2
4	21.63	20.38	100.6
5	19.39	18.63	110.0

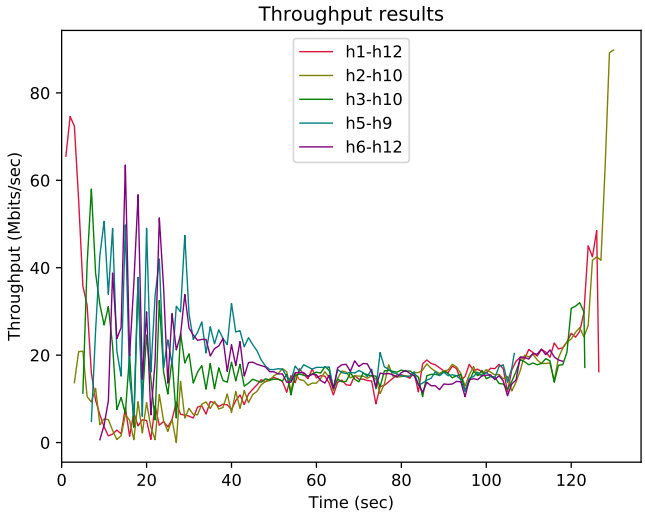


Figure A.3: Time series of receiver throughputs. The notation "hi-hj" in the legend denotes a flow from source "hi" to destination "hj".

In addition to determining the average statistics, we analyzed the instantaneous throughput for each flow on both the sender and receiver, as well as switch statistics related to packets that were queued and dropped. We present some of those results in the following figures.

Figure A.3 shows the time series of throughput on the receiver side for the five flows. In the beginning, only flow 1 was present, and it took the maximum available bandwidth. As the subsequent flows kept coming in, TCP tried to adjust the individual bandwidths to match with fair shares. That adjustment took some time, and flows started to converge roughly around 50 s. Those results are consistent with TCP's expected behavior with the BBR congestion control algorithm (ref. Figure 6 in the original BBR paper [146]).

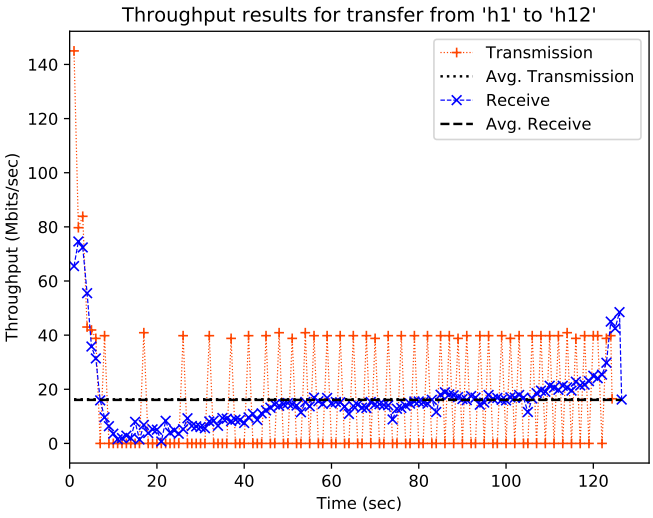


Figure A.4: Timeseries of throughput for flow 1.

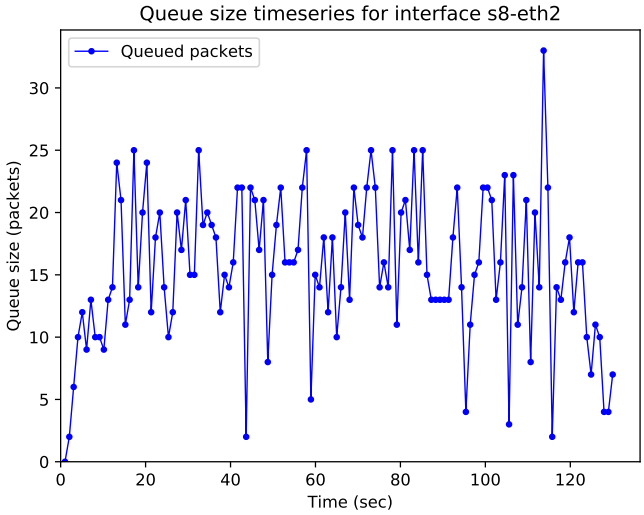


Figure A.5: Time series of queued packets on $s8$'s interface (eth2) to the bottleneck link ($s6$-$s8$).

Next, Figure A.4 shows the instantaneous throughput for one of the flows, including both the sender and receiver. The sender's behavior matches with that of a typical TCP source, as it ramps up the sending rate until congestion happens and then restarts from zero. Similarly, the receiver time series shows that the BBR congestion control algorithm takes some time to converge to the available fair share.

Finally, Figure A.5 shows the time series of queued packets on switch $s8$'s interface to the bottleneck link. The time series has a periodic nature, mostly due to the TCP sender's

behavior and consistent with Figure A.4.

In summary, our results from emulation of Google's B4 network show that G2-Mininet can create real-world network architectures and collect performance statistics from live traffic. Those statistics were found to be consistent with the expected behaviors and theoretical limits, and thus validated the correctness of the emulated networks.

The G2 framework uses such results to analyze a network's bottleneck structure and provide insights into the causes and effects of specific traffic policing. Moreover, B4 is one example of several network architectures that we developed by using G2-Mininet to support the evaluation of the G2 framework. The discussion of those additional network architectures and related insights is out of the scope of this dissertation. We refer the reader to [136, 137] for further information on those topics.

A.4 APPLICATION OF MNEST TO SUBSTATION NETWORKS

MNEST offers a flexible platform that can be extended and specialized for various application domains. In the previous section, we showed one such application, i.e., support for the empirical evaluation of a network optimization framework. In this section, we discuss some specific extensions to MNEST that will allow its incorporation in the evaluation of systems developed for IEC 61850-based substation networks. In particular, the trace-based evaluation of the systems presented in Chapters 4 and 5 can be further supported by experiments conducted using MNEST. To achieve that goal, MNEST needs to work with GOOSE protocol traffic, which it can do if the following technologies are incorporated:

libiec61850. libiec61850 [149] is an open-source library that offers the full ISO stack for an IEC 61850-compliant substation network. In essence, by using this library, we can convert hosts into substation IEDs and implement GOOSE publishers and subscribers.

Ethernet traffic. MNEST can transmit GOOSE traffic encapsulated in Ethernet frames over a Mininet network. In the generation of such traffic, MNEST can support two modes: (1) reading and replaying of existing packet traces, and (2) production of new traffic, given configurations, through the use of traffic generators such as GEESE [150].

Zeek. We can attach Zeek to monitor target interfaces on the Mininet switches. Zeek can then parse GOOSE protocol traffic to provide input for security analysis.

Using the above extensions, MNEST can provide a realistic substation environment for evaluating the SEEZE system (Chapter 5). We defer the implementation of such extensions to our future work.

A.5 RELATED WORK

MNEST draws upon the successful use of Mininet in emulating realistic architectures for network research and development. Some examples of such applications include a collection of Mininet-based projects that reproduce research results [151], a testbed for distributed SDN development [152], and a high-fidelity network emulation [153].

In the context of smart grid control networks, Dong et al. [133] propose to use a Mininet-based testbed to evaluate the opportunities and downsides of the application of SDN. MNEST provides a Mininet-based platform to enable experiments for various networking applications. With some easy extensions, as mentioned in Section A.4, MNEST can be tailored to IEC 61850-compliant substation networks.

Several other related research efforts propose frameworks for generating datasets and evaluating new security measures in close-to-real environments. Examples of such systems, focused on IEC 61850 substations, include (1) hardware-in-the-loop simulation [154] that uses high-fidelity digital simulators for power systems such as RTDS, and (2) software simulations, e.g., [155] and [156] that use frameworks such as OMNeT++ and MATLAB/Simulink. MNEST can in fact complement those frameworks by integrating their power-system simulations. It can support both software and hardware-in-the-loop systems.

A.6 CONCLUSION

In this appendix, we presented MNEST, a testbed to facilitate the emulation of real-world network architectures. MNEST uses Mininet and SDN-related technologies to enable experimentation with a real network stack and traffic and to collect benchmarks. The methods evaluated using MNEST can thus be deployed to real networks with minimal changes. We described a case study on supporting the evaluation of the G2 network optimization framework. In that study, we demonstrated the emulation of Google's B4 network topology and the collection of benchmarks useful for G2's evaluation. We also discussed specific possible extensions to MNEST for supporting the evaluation of methods presented in this dissertation and reducing the deployment gaps. We leave the implementation of such extensions for our future work.

REFERENCES

[1] N. E. Weiss and R. S. Miller, "The Target and other financial data breaches: Frequently asked questions," Congressional Research Service, Tech. Rep. R43496, 2015.

[2] A. W. Mathews and D. Yadron, "Health insurer Anthem hit by hackers," 2015, The Wall Street Journal. [Online]. Available: https://www.wsj.com/articles/health-insurer-anthem-hit-by-hackers-1423103720

[3] P. Zengerle and M. Cassella, "Millions more Americans hit by government personnel data hack," 2015, Reuters. [Online]. Available: http://reut.rs/1frLcqq

[4] D. E. Whitehead, K. Owens, D. Gammel, and J. Smith, "Ukraine cyber-induced power outage: Analysis and practical mitigation strategies," in *Proc. 2017 70th Annual Conference for Protective Relay Engineers (CPRE)*, 2017, pp. 1–8.

[5] R. M. Lee, "CrashOverride: Analyzing the threat to electric grid operation," 2017, Dragos, Inc. [Online]. Available: https://dragos.com/resource/crashoverride-analyzing-the-malware-that-attacks-power-grids/

[6] "ATT&CK: Adversarial Tactics, Techniques & Common Knowledge," 2020, MITRE. [Online]. Available: https://attack.mitre.org/tactics/TA0008/

[7] "Lateral movement: How do threat actors move deeper into your network?" TrendMicro, Inc., Tech. Rep., 2013.

[8] D. McWhorter, "APT1: Exposing one of China's cyber espionage units," Mandiant, Tech. Rep., 2013.

[9] C. Wueest, "Targeted attacks against the energy sector," Symantec Corporation, Security Response, 2014.

[10] B. Pierson, "Anthem to pay record $115 million to settle U.S. lawsuits over data breach," 2017, Reuters. [Online]. Available: http://reut.rs/2sLmF76

[11] "Microsoft advanced threat analytics," 2016, Microsoft. [Online]. Available: https://www.microsoft.com/en-us/cloud-platform/advanced-threat-analytics

[12] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare and Security Research*, vol. 1, pp. 80–106, 2011.

[13] R. Brewer, "Advanced persistent threats: Minimizing the damage," *Network Security*, vol. 2014, no. 4, pp. 5–9, 2014.

[14] B. Bencsáth, G. Pék, L. Buttyán, and M. Felegyhazi, "The cousins of Stuxnet: Duqu, Flame, and Gauss," *Future Internet*, vol. 4, no. 4, pp. 971–1003, 2012.

[15] S. Yin and O. Kaynak, "Big data for modern industry: Challenges and trends [point of view]," *Proc. IEEE*, vol. 103, no. 2, pp. 143–146, 2015.

[16] "Big data analytics for security (big data working group)," 2013, Cloud Security Alliance. [Online]. Available: https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf

[17] A. Juels and T.-F. Yen, "Sherlock Holmes and the case of the advanced persistent threat," in *Proc. 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2012, pp. 1–4.

[18] F. Castanedo, "A review of data fusion techniques," *The Scientific World Journal*, vol. 2013, pp. 1–19, 2013.

[19] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proc. IEEE*, vol. 85, pp. 6–23, 1997.

[20] E. Waltz, J. Llinas et al., *Multisensor data fusion*. Artech House, 1990, vol. 685.

[21] D. L. Hall and S. A. McMullen, *Mathematical techniques in multisensor data fusion*. Artech House, 2004.

[22] J. R. Boyd, "A discourse on winning and losing," *Air University Press*, 2018. [Online]. Available: https://www.airuniversity.af.edu/Portals/10/AUPress/Books/B_0151_Boyd_Discourse_Winning_Losing.pdf

[23] A. N. Steinberg, C. L. Bowman, and F. E. White, "Revisions to the JDL data fusion model," in *Proc. SPIE 3719, Sensor Fusion: Architectures, Algorithms, and Applications III*, 1999, pp. 430–441.

[24] J. Llinas, C. L. Bowman, G. L. Rogova, A. N. Steinberg, E. L. Waltz, and F. E. White, "Revisions and extensions to the JDL data fusion model II," in *Proc. Seventh International Conference on Information Fusion*, 2004, pp. 1218–1230.

[25] T. Bass, "Intrusion detection systems and multisensor data fusion," *Communications of the ACM*, vol. 43, pp. 99–105, 2000.

[26] A. Stotz and M. Sudit, "Information fusion engine for real-time decision-making (INFERD): A perceptual system for cyber attack tracking," in *Proc. 2007 10th International Conference on Information Fusion*, 2007, pp. 1–8.

[27] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proc. 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001, pp. 69–73.

[28] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Network anomaly detection and classification via opportunistic sampling," *IEEE Network*, vol. 23, no. 1, pp. 6–12, 2009.

[29] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security (TiSSEC)*, vol. 3, no. 4, pp. 227–261, 2000.

[30] G. Giacinto, F. Roli, and L. Didaci, "Fusion of multiple classifiers for intrusion detection in computer networks," *Pattern recognition letters*, vol. 24, no. 12, pp. 1795–1803, 2003.

[31] C. Siaterlis and B. Maglaris, "Towards multisensor data fusion for DoS detection," in *Proc. 2004 ACM symposium on Applied computing*, 2004, pp. 439–446.

[32] D. Yu and D. Frincke, "Alert confidence fusion in intrusion detection systems with extended Dempster-Shafer theory," in *Proc. 43rd annual Southeast regional conference*, 2005, pp. 142–147.

[33] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proc. 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2001, pp. 85–103.

[34] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proc. 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2001, pp. 54–68.

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

[36] E. Totel, B. Vivinis, and L. Mé, "A language driven intrusion detection system for event and alert correlation," in *Proc. 19th IFIP International Information Security Conference (IFIP SEC)*, 2004, pp. 209–224.

[37] H. Dreger, C. Kreibich, V. Paxson, and R. Sommer, "Enhancing the accuracy of network-based intrusion detection with host-based context," in *Proc. Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2005, pp. 206–221.

[38] A. M. Fawaz and W. H. Sanders, "Learning process behavioral baselines for anomaly detection," in *Proc. 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2017, pp. 145–154.

[39] "The Zeek network security monitor," 2020. [Online]. Available: https://www.zeek.org/

[40] "IEC 61850-8-1:2011+AMD1:2020 CSV communication networks and systems for power utility automation: Part 8-1," 2020. [Online]. Available: https://webstore.iec.ch/publication/66585

[41] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the ukrainian power grid," Electricity Information Sharing and Analysis Center (E-ISAC), Defense use case, 2016.

[42] "2014 JPMorgan Chase data breach," 2014, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/2014_JPMorgan_Chase_data_breach

[43] "Man in the cloud (MITC) attacks," Imperva, Inc., Tech. Rep., 2015. [Online]. Available: https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf

[44] "Sony Pictures Entertainment. System disruption notice letter," 2104. [Online]. Available: https://oag.ca.gov/system/files/120814letter_0.pdf

[45] "Internet security threat report," 2015, Symantec. [Online]. Available: http://www.symantec.com/security_response/publications/threatreport.jsp

[46] S. Katti, B. Krishnamurthy, and D. Katabi, "Collaborating against common enemies," in *Proc. 5th ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2005, pp. 365–378.

[47] P. Cao, E. Badger, Z. Kalbarczyk, R. Iyer, and A. Slagell, "Preemptive intrusion detection: Theoretical framework and real-world measurements," in *Proc. 2015 Symposium and Bootcamp on the Science of Security (HoTSoS)*, 2015, pp. 5:1–5:12.

[48] "Visual analytics community. VAST challenge, 2011," 2011. [Online]. Available: http://www.vacommunity.org/2011+-+2013+VAST+Cyber+Challenges

[49] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 2006.

[50] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. Second International Conference on Knowledge Discovery and Data Mining*, vol. 96, 1996, pp. 226–231.

[51] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, no. 1, pp. 53–65, 1987.

[52] M. Zamani and M. Movahedi, "Machine learning techniques for intrusion detection," *arXiv:1312.2177 [cs.CR]*, pp. 1–11, 2013.

[53] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proc. 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2004, pp. 219–230.

[54] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proc. 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005, pp. 217–228.

[55] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proc. 29th Annual Computer Security Applications Conference (ACSAC)*, 2013, pp. 199–208.

[56] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *Proc. ACM CSS Workshop on Data Mining Applied to Security (DMSA)*, 2001, pp. 5–8.

[57] C. Hood and C. Ji, "Proactive network-fault detection [Telecommunications]," *IEEE Transactions on Reliability*, vol. 46, no. 3, pp. 333–341, 1997.

[58] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. 3rd ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2003, pp. 234–247.

[59] N. Ye, "A Markov Chain model of temporal behavior for anomaly detection," in *Proc. 2000 IEEE Workshop on Information Assurance and Security*, 2000, pp. 171–174.

[60] W. Ma, D. Tran, and D. Sharma, "A study on the feature selection of network traffic for intrusion detection purpose," in *Proc. IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2008, pp. 245–247.

[61] M. Almgren, U. Lindqvist, and E. Jonsson, "A multi-sensor model to improve automated attack detection," in *Proc. 11th International Symposium on Recent Advances in Intrusion Detection*, 2008, pp. 291–310.

[62] A. Bohara, U. Thakore, and W. H. Sanders, "Intrusion detection in enterprise systems by combining and clustering diverse monitor data," in *Proc. Symposium and Bootcamp on the Science of Security (HoTSoS)*, 2016. [Online]. Available: https://doi.org/10.1145/2898375.2898400 pp. 7–17.

[63] M. Cruz, "APT myths and challenges," 2012, TrendMicro. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/infographic-apt-myths-and-challenges/

[64] "2016 data breach investigation report," 2016, Verizon. [Online]. Available: http://www.verizonenterprise.com/verizon-insights-lab/dbir/2016/

[65] A. D. Kent, "Cyber security data sources for dynamic network research," in *Dynamic Networks in Cyber-Security*, vol. 1, 2015, pp. 37–65.

[66] P. Van Mieghem, "The N-intertwined SIS epidemic network model," *Computing*, vol. 93, no. 2, pp. 147–169, 2011.

[67] "Visual analytics community. VAST challenge, 2013," 2013. [Online]. Available: http://vacommunity.org/VAST+Challenge+2013

[68] "GT malware passive DNS dataset," 2017, Impact Cybertrust. [Online]. Available: https://www.impactcybertrust.org/

[69] D. Dash, B. Kveton, J. M. Agosta, E. Schooler, J. Chandrashekar, A. Bachrach, and A. Newman, "When gossip is good: Distributed probabilistic inference for detection of slow network intrusions," in *Proc. national conference on Artificial Intelligence (AAAI)*, vol. 21, no. 2, 2006, pp. 1115–1122.

[70] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proc. 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 45–56.

[71] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "Is host-based anomaly detection + temporal correlation = worm causality?" Carnegie Mellon University, Tech. Rep. CMU-CS-07-112, 2007.

[72] A. Fawaz, A. Bohara, C. Cheh, and W. H. Sanders, "Lateral movement detection using distributed data fusion," in *Proc. 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, 2016, pp. 21–30.

[73] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proc. 2000 ACM SIGMOD international conference on Management of data (SIGMOD)*, vol. 29, no. 2, 2000, pp. 427–438.

[74] V. Badrinath Krishna, G. A. Weaver, and W. H. Sanders, "PCA-based method for detecting integrity attacks on advanced metering infrastructure," in *Proc. 12th International Conference on Quantitative Evaluation of Systems (QEST)*, 2015, pp. 70–85.

[75] P. J. Rousseeuw and C. Croux, "Alternatives to the median absolute deviation," *Journal of the American Statistical Association*, vol. 88, no. 424, pp. 1273–1283, 1993.

[76] A. Hagberg, N. Lemons, A. Kent, and J. Neil, "Connected components and credential hopping in authentication graphs," in *Proc. International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, 2014, pp. 416–223.

[77] J. Lambert, "Defenders think in lists. attackers think in graphs." 2015. [Online]. Available: https://blogs.technet.microsoft.com/johnla

[78] C. C. Aggarwal, "Outlier ensembles: Position paper," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 49–58, 2013.

[79] P. V. Mieghem, J. Omic, and R. Kooij, "Virus spread in networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 1–14, 2008.

[80] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, 1976.

[81] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[82] R. Amin, J. Ryan, and J. van Dorp, "Detecting targeted malicious email," *IEEE Security & Privacy*, vol. 10, no. 3, pp. 64–71, 2012.

[83] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "BotHunter: Detecting malware infection through IDS-driven dialog correlation," in *Proc. USENIX Security Symposium*, vol. 7, 2007, pp. 1–16.

[84] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proc. 28th Annual Computer Security Applications Conference (ACSAC)*, 2012, pp. 239–248.

[85] J. R. Johnson and E. A. Hogan, "A graph analytic metric for mitigating advanced persistent threat," in *Proc. 2013 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2013, pp. 129–133.

[86] P.-Y. Chen, S. Choudhury, and A. O. Hero, "Multi-centrality graph spectral decompositions and their application to cyber intrusion detection," in *Proc. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4553–4557.

[87] G. Gu, R. Perdisci, and J. Zhang et al., "BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *Proc. USENIX Security Symposium*, vol. 5, no. 2, 2008, pp. 139–154.

[88] M. Tylutki and K. Levitt, "A network-based response framework and implementation," in *Proc. IFIP International Working Conference on Active Networks*, 2005, pp. 65–82.

[89] M. A. Noureddine, A. Fawaz, W. H. Sanders, and T. Başar, "A game-theoretic approach to respond to attacker lateral movement," in *Proc. 7th International Conference on Decision and Game Theory for Security (GameSec)*, 2016, pp. 294–313.

[90] A. Bohara, M. A. Noureddine, A. Fawaz, and W. H. Sanders, "An unsupervised multi-detector approach for identifying malicious lateral movement," in *Proc. 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, 2017. [Online]. Available: https://doi.org/10.1109/SRDS.2017.31 pp. 224–233.

[91] "IEC TS 62351-6:2007 data and communications security: Part 6: Security for IEC 61850," 2007. [Online]. Available: https://webstore.iec.ch/publication/6909

[92] F. Hohlbaum, M. Braendle, and F. Alvarez, "Cyber security practical considerations for implementing IEC 62351," in *Proc. PAC World Conference*, 2010, pp. 1–8.

[93] M. G. da Silveira and P. H. Franco, "IEC 61850 network cybersecurity: Mitigating GOOSE message vulnerabilities," in *Proc. 6th Annual PAC World Americas Conference*, 2019, pp. 1–9.

[94] M. El Hariri, T. A. Youssef, and O. A. Mohammed, "On the implementation of the IEC 61850 standard: Will different manufacturer devices behave similarly under identical conditions?" *Electronics*, vol. 5, no. 4:85, pp. 1–13, 2016.

[95] W. Fangfang, W. Huazhong, C. Dongqing, and P. Yong, "Substation communication security research based on hybrid encryption of DES and RSA," in *Proc. 2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP)*, 2013, pp. 437–441.

[96] A. P. Premnath, J. Jo, and Y. Kim, "Application of NTRU cryptographic algorithm for SCADA security," in *Proc. 2014 11th International Conference on Information Technology: New Generations (ITNG)*, 2014, pp. 341–346.

[97] S. M. Farooq, S. M. S. Hussain, and T. S. Ustun, "Performance evaluation and analysis of IEC 62351-6 probabilistic signature scheme for securing GOOSE messages," *IEEE Access*, vol. 7, pp. 32 343–32 351, 2019.

[98] K. Boakye-Boateng and A. H. Lashkari, "Securing GOOSE: The return of one-time pads," in *Proc. 2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.

[99] G. Elbez, H. B. Keller, and V. Hagenmeyer, "Authentication of GOOSE messages under timing constraints in IEC 61850 substations," in *Proc. 6th International Symposium for ICS & SCADA Cyber Security Research (ICS-CSR)*, 2019, pp. 137–143.

[100] D. Ishchenko and R. Nuqui, "Secure communication of intelligent electronic devices in digital substations," in *Proc. 2018 IEEE/PES Transmission and Distribution Conference and Exposition*, 2018, pp. 1–5.

[101] E. Esiner, D. Mashima, B. Chen, Z. Kalbarczyk, and D. Nicol, "F-Pro: a fast and flexible provenance-aware message authentication scheme for smart grid," in *Proc. 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1–7.

[102] J. Hoyos, M. Dehus, and T. X. Brown, "Exploiting the GOOSE protocol: A practical attack on cyber-infrastructure," in *Proc. 2012 IEEE Globecom Workshops*, 2012, pp. 1508–1513.

[103] N. Kush, E. Ahmed, M. Branagan, and E. Foo, "Poisoned GOOSE: Exploiting the GOOSE protocol," in *Proc. Twelfth Australasian Information Security Conference (AISC)*, 2014, pp. 17–22.

[104] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23, p. 2435 2463, 1999.

[105] W. Ren, T. Yardley, and K. Nahrstedt, "EDMAND: Edge-based multi-level anomaly detection for SCADA networks," in *Proc. 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2018, pp. 1–7.

[106] V. Coughlin, C. Rubio-Medrano, Z. Zhao, and G. Ahn, "EDSGuard: Enforcing network security requirements for energy delivery systems," in *Proc. 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2018, pp. 1–6.

[107] M. Kabir-Querrec, "Cyber security of smart-grid control systems: Intrusion detection in IEC 61850 communication networks," Ph.D. dissertation, Université Grenoble Alpes, 2017. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01609230v2

[108] P. P. Biswas, H. C. Tan, Q. Zhu, Y. Li, D. Mashima, and B. Chen, "A synthesized dataset for cybersecurity study of IEC 61850 based substation," in *Proc. 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1–7.

[109] "2018 internet security threat report," 2018, Symantec. [Online]. Available: https://www.symantec.com/security-center/threat-report

[110] A. Greenberg, "A fresh look at the 2016 blackout in Ukraine," 2019, Wired. [Online]. Available: https://www.wired.com/story/russia-ukraine-cyberattack-power-grid-blackout-destruction/

[111] "IEC 61850: Power utility automation," 2020. [Online]. Available: https://www.iec.ch/smartgrid/standards/

[112] A. Valdes, C. Hang, P. Panumpabi, N. Vaidya, C. Drew, and D. Ischenko, "Design and simulation of fast substation protection in IEC 61850 environments," in *Proc. 2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, 2015, pp. 1–6.

[113] "Tcpreplay - pcap editing and replaying utilities," 2020. [Online]. Available: https://tcpreplay.appneta.com/

[114] "Python Scapy," 2020. [Online]. Available: https://scapy.net/

[115] J. Ros-Giralt, A. Commike, P. Cullen, and R. Lethin, "Algorithms and data structures to accelerate network analysis," *Future Generation Computer Systems*, vol. 86, pp. 535–545, 2018.

[116] U. K. Premaratne, J. Samarabandu, T. S. Sidhu, R. Beresh, and J. Tan, "An intrusion detection system for IEC61850 automated substations," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2376–2383, 2010.

[117] Q. Yang, W. Hao, L. Ge, W. Ruan, and F. Chi, "Farima model-based communication traffic anomaly detection in intelligent electric power substations," *IET Cyber-Physical Systems: Theory Applications*, vol. 4, no. 1, pp. 22–29, 2019.

[118] A. Albarakati, C. Robillard, M. Karanfil, M. Kassouf, R. Hadjidj, M. Debbabi, and A. Youssef, "Security monitoring of IEC 61850 substations using IEC 62351-7 network and system management," in *Proc. 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids*, 2019, pp. 1–7.

[119] J. Hong, C. Liu, and M. Govindarasu, "Detection of cyber intrusions using network-based multicast messages for substation automation," in *Proc. Innovative Smart Grid Technologies (ISGT)*, 2014, pp. 1–5.

[120] Y. Yang, H. Xu, L. Gao, Y. Yuan, K. McLaughlin, and S. Sezer, "Multidimensional intrusion detection system for IEC 61850-based SCADA networks," *IEEE Transactions on Power Delivery*, vol. 32, no. 2, pp. 1068–1078, 2017.

[121] H. Lin, A. Slagell, C. D. Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting Bro into SCADA: Building a specification-based iintrusion detection system for the DNP3 protocol," in *Proc. Eighth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)*, 2013, pp. 1–4.

[122] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 1–33, 2011.

[123] R. B. Bobba, K. M. Rogers, Q. Wang, H. Khurana, K. Nahrstedt, and T. J. Overbye, "Detecting false data injection attacks on DC state estimation," in *Proc. 1st Workshop on Secure Control Systems (SCS)*, 2010.

[124] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[125] S. Findlay and E. White, "India confirms cyber attack on nuclear power plant," 2019, Financial Times. [Online]. Available: https://www.ft.com/content/e43a5084-fbbb-11e9-a354-36acbbb0d9b6

[126] H. R. Ghaeini and N. O. Tippenhauer, "HAMIDS: hierarchical monitoring intrusion detection system for industrial control systems," in *Proc. 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, 2016, pp. 103–111.

[127] "Snort - network intrusion detection & prevention system," 2020. [Online]. Available: https://www.snort.org/

[128] "Suricata - open source IDS / IPS / NSM engine," 2020. [Online]. Available: https://suricata-ids.org/

[129] "Zeek input framework," 2020. [Online]. Available: https://docs.zeek.org/en/current/frameworks/input.html

[130] Y. Yang, K. McLaughlin, S. Sezer, T. Littler, E. G. Im, B. Pranggono, and H. F. Wang, "Multiattribute SCADA-specific intrusion detection system for power networks," *IEEE Transactions on Power Delivery*, vol. 29, no. 3, pp. 1092–1102, 2014.

[131] C. McParland, S. Peisert, and A. Scaglione, "Monitoring security of networked control systems: It's the physics," *IEEE Security & Privacy*, vol. 12, no. 6, pp. 32–29, 2014.

[132] H. Lin, A. Slagell, Z. Kalbarczyk, P. Sauer, and R. K. Iyer, "Semantic security analysis of SCADA networks to detect malicious control commands in power grids," in *Proc. First ACM Workshop on Smart Energy Grid Security (SEGS)*, 2013, pp. 29–34.

[133] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk, "Software-defined networking for smart grid resilience: Opportunities and challenges," in *Proc. 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 61–68.

[134] W. Ren, T. Yu, T. Yardley, and K. Nahrstedt, "CAPTAR: Causal-polytree-based anomaly reasoning for SCADA networks," in *Proc. 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1–7.

[135] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes, "Detection, correlation, and visualization of attacks against critical infrastructure systems," in *Proc. 2010 Eighth International Conference on Privacy, Security and Trust*, 2010, pp. 15–22.

[136] J. Ros-Giralt, S. Yellamraju, A. Bohara, R. Lethin, J. Li, Y. Lin, Y. Tan, M. Veeraraghavan, Y. Jiang, and L. Tassiulas, "G2: A network optimization framework for high-precision analysis of bottleneck and flow performance," in *Proc. 2019 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, 2019, pp. 48–60.

[137] J. Ros-Giralt, A. Bohara, S. Yellamraju, M. H. Langston, R. Lethin, Y. Jiang, L. Tassiulas., J. Li, Y. Tan, and M. Veeraraghavan, "On the bottleneck structure of congestion-controlled networks," *Proc. the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 3, no. 3, pp. 1–31, 2019.

[138] "Mininet: An instant virtual network on your laptop (or other PC)," 2020. [Online]. Available: mininet.org/

[139] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu et al., "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[140] "G2-Mininet sandbox," 2019. [Online]. Available: https://github.com/reservoirlabs/g2-mininet

[141] "The POX network software platform," 2020. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[142] "Open vSwitch: An open virtual switch," 2020. [Online]. Available: https://www.openvswitch.org/

[143] "The ns-3 network simulator," 2020. [Online]. Available: https://www.nsnam.org/

[144] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[145] "iperf: The TCP, UDP, and SCTP bandwidth measurement tool," 2020. [Online]. Available: https://iperf.fr/

[146] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[147] S. Ha, I. Rhee, and L. Xu, "Cubic: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[148] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *arXiv preprint cs/9809099*, 1998.

[149] "libIEC61850: open source libraries for IEC 61850," 2020. [Online]. Available: https://libiec61850.com/libiec61850/

[150] Y. Lopes, D. C. Muchaluat-Saade, N. C. Fernandes, and M. Z. Fortes, "Geese: A traffic generator for performance and security evaluation of IEC 61850 networks," in *Proc. 2015 IEEE 24th International Symposium on Industrial Electronics (ISIE)*, 2015, pp. 687–692.

[151] L. Yan and N. McKeown, "Learning networking by reproducing research results," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 2, p. 19–26, 2017.

[152] B. Lantz and B. O'Connor, "A Mininet-based virtual testbed for distributed SDN development," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 365–366, 2015.

[153] B. Heller, "Reproducible network research with high-fidelity emulation," Ph.D. dissertation, Stanford University, 2013.

[154] Y. Yang, H. Jiang, K. McLaughlin, L. Gao, Y. Yuan, W. Huang, and S. Sezer, "Cybersecurity test-bed for IEC 61850 based smart substations," in *Proc. 2015 IEEE Power & Energy Society General Meeting*, 2015, pp. 1–5.

[155] G. Elbez, H. B. Keller, and V. Hagenmeyer, "A Cost-efficient Software Testbed for Cyber-Physical Security in IEC 61850-based Substations," in *Proc. 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2018, pp. 1–6.

[156] M. Golshani, G. A. Taylor, and I. Pisica, "Simulation of power system substation communications architecture based on IEC 61850 standard," in *Proc. 2014 49th International Universities Power Engineering Conference (UPEC)*, 2014, pp. 1–6.