A QUANTITATIVE METHODOLOGY FOR EVALUATING AND DEPLOYING SECURITY MONITORS

BY

UTTAM THAKORE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor William H. Sanders

ABSTRACT

Despite advances in intrusion detection and prevention systems, attacks on networked computer systems continue to succeed. Intrusion tolerance and forensic analysis are required to adequately detect and defend against attacks that succeed. Intrusion tolerance and forensic analysis techniques depend on monitors to collect information about possible attacks. Since monitoring can be expensive, however, monitors must be selectively deployed to maximize their overall utility. We identify a need for a methodology for evaluating monitor deployment to determine a placement of monitors that meets both security goals and cost constraints.

In this thesis, we introduce a methodology both to quantitatively evaluate monitor deployments in terms of security goals and to deploy monitors optimally based on cost constraints. First, we define a system and data model that describes the system we aim to protect, the monitors that can be deployed, and the relationship between intrusions and data generated by monitors. Second, we define a set of quantitative metrics that both quantify the utility and richness of monitor data with respect to intrusion detection, and quantify the cost associated with monitor deployment. We describe how a practitioner could characterize intrusion detection requirements in terms of target values of our metrics. Finally, we use our data model and metrics to formulate a method to determine the cost-optimal, maximum-utility placement of monitors. We illustrate our approach throughout the thesis with a working example, and demonstrate its practicality and expressiveness with a case study based on an enterprise Web service architecture. The value of our approach comes from its ability to determine optimal monitor placements, which can be counterintuitive or difficult to find, for nearly any set of cost and intrusion detection parameters.

To my family and late grandmother, for their guidance, support, and limitless love.

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. William H. Sanders, for his technical advice, encouragement, and support. His guidance ensured that I did not get off-track with my work. I would also like to thank Dr. Gabriel Weaver for his countless hours of technical discussions, guidance with my research, advice about graduate school, and revisions. He consistently challenged me to produce my best work. Without his help, this work would never have come to fruition.

Additionally, I would like to thank Dr. Brett Feddersen, who helped immensely with the implementation of the experiment and with whom I had many interesting discussions about software development practices. A great deal of thanks also goes to Jenny Applequist, whose sharp eye caught many a poorly worded sentence or grammatical mistake. Every paper she returned full of red marks was a learning experience for me. Jenny never complained about the ridiculous, last-minute deadlines I imposed on her over the years, and has been a great help.

I thank the members of the PERFORM group, who enriched my work and personal lives in so many ways. Their feedback has helped improve the quality of my work, and their company in the office has made graduate school life anything but mundane. In particular, I thank Ahmed Fawaz, Carmen Cheh, Atul Bohara, Ben Ujcich, Varun Badrinath Krishna, Ken Keefe, Sobir Bazarbayev, Craig Buchanan, and Ronald Wright. I also thank my friends Arjun Athreya and Subho Banerjee for always being there to help me escape the office when I needed it.

Most importantly, I thank my loving family for being a source of emotional support and motivation throughout my life. I have received unconditional love and support from my parents, Arun and Rina, and my sister, Juhi. When I felt that I had hit a wall with my work, my father's words of wisdom helped me push through. Finally, I thank my late grandmother, Maya, who always put a smile on my face and was supportive of all of my endeavors. Seeing her was always the highlight of my visits home, and I will miss her immensely.

This material is based 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 U.S.

Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and the Air Force Office of Scientific Research, or the U.S. Government.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Intrusion detection and prevention systems have become increasingly sophisticated and capable of detecting complex, coordinated attacks. Despite such advances, attackers continue to successfully compromise networked computer systems [1, 2, 3]. Attackers and defenders remain caught in the cat-and-mouse game of modern computer security. As existing systems are patched and updated and new systems are developed, the space of exploitable vulnerabilities changes. Attackers find new vulnerabilities in new and updated systems and exploit vulnerabilities that are not yet known to defenders (otherwise known as *zero-days*) in existing systems. At the same time, defenders detect attacks, often well after the fact, and implement patches to close existing vulnerabilities. Security researchers also devise new mechanisms for protecting systems that eliminate entire classes of attacks. However, attackers and often *other security researchers* invent clever workarounds to those mechanisms. For example, when operating systems began implementing executable space protection to combat buffer overflow attacks, attackers eventually devised Return-Oriented Programming (ROP) to circumvent the protections [4]. It suffices to say that intrusion prevention and detection are not on their own enough to secure computer systems; mechanisms for both further information-gathering and online response are required to adequately deal with attacks that succeed.

Intrusion tolerance, which emerged from the field of fault tolerance, recognizes that attackers will be able to infiltrate computer systems [5]. Instead of attempting to prevent all possible intrusions, intrusion-tolerant systems anticipate that some intrusions will succeed and instead *tolerate* the intrusions, triggering response and recovery mechanisms that prevent the intruder from causing a system failure or a critical security breach. We believe that the correct approach to securing large, distributed systems is to employ elements of intrusion tolerance, not only preventing as many attacks as possible, but also detecting successful attacks as early as possible and taking response actions that can provide more information about intruders and prevent them from causing catastrophic damage to the system. Even with the absolute best intrusion tolerance system, however, it is impossible to ensure that attackers are never able to break into the system undetected. Security administrators must also perform forensic analysis on logs to detect intrusions after they have occurred and identify the exploited attack vectors, which can then be mitigated, more closely monitored, or closed.

We identify the following components of intrusion tolerance: collection of data about the system that might provide evidence of intrusions, fusion of the data to determine what attacks are taking place in the system, and selection and execution of a response action based on the observed state of the system and perceived attack actions. Data collection is foundational among these components, as fusion and response both depend on proper information about the system.

Security data collection in computer systems is performed by means of various types of sensors, which we refer to as *security monitors* or simply *monitors*. Examples of such monitors include host-level intrusion detection systems, such as antivirus suites; network-level intrusion detection systems; firewalls; application logs; system event logs; and network flow logs. It is important to note that different monitors generate different types of information, as a result of which, the utility of the logs may depend on the algorithms that are used to analyze the data. Ultimately, however, for intrusion tolerance and forensics analysis techniques to be effective, it is essential that logs be correct and contain adequate information about events that may take place in the system [6, 7].

In practice, monitor deployment is generally conducted by a system administrator, who either deploys monitors based on domain knowledge and expected monitoring needs or uses existing commercial coordinated monitoring and analysis tools that in turn deploy monitors. The simplest approach monitor deployment is to deploy as many monitors as possible and use all generated data during analysis. However, even for a relatively restricted domain such as Linux system resource monitoring, the list of available monitors is overwhelming. For example, ServerDensity provided a list of over 80 Linux monitoring tools [8], which represents just a sampling of tools available. Many of the monitors provide duplicate functionality and differ just in presentation. It is unclear what minimal set of tools might provide adequate information about system resources, or in the event of compromise (by a rootkit, for example), which tools, if any, would be trustworthy. Furthermore, storing all of the data for future forensic analysis or large-timescale intrusion detection would be impractical. Extending the aforementioned problems to large, diverse, and complex systems such as enterprises makes deploying monitors in such a manner impractical and inefficient.

It might seem appropriate, then, to rely on commercial off-the-shelf (CotS) packaged monitoring and analysis solutions. Such tools are often referred to as *Security Information and Event Management* (*SIEM*) systems, and for domains such as enterprise systems or cloud data centers, many such tools exist [9, 10, 11, 12].

SIEM systems offer a wide list of features, including compliance with federal regulations, consolidated management of monitoring, and automated monitor reconfiguration. Such tools are attractive because they ensure compliance and offload all monitoring responsibilities to the solution provider. However, it is again unclear how the tools differ in functionality, or what kinds of weaknesses remain in the monitoring and fusion performed by the tools (in the case of compromise of monitors, for example). The algorithms used by the tools to deploy monitors, perform detection, and generate alerts are proprietary, and important information could be hidden by the filtering performed by said algorithms. Overall, there is no widely applicable, quantitative methodology for evaluating the effectiveness of monitoring tools.

In addition to the issues just mentioned, one must take into account the *cost* of monitoring. Monitoring infrastructure can be expensive to purchase and maintain, and organizations generally have limited log storage capacity. It is not possible to instrument one's entire system or store all data indefinitely; monitors must be *selectively* deployed based on cost and utility tradeoffs.

Constraints on monitor placement may come from many sources. Budgetary limits, for example, may be at odds with the operational goals of detecting as many intrusions as possible or enabling proper a posteriori analysis of logs. There is further a tradeoff between the cost of data collection and the richness and quality of the data collected. Hence, there is a need for a methodology for evaluating monitor deployment to determine an optimal placement of monitors that meets security goals and requirements.

It is often said that "you can't improve what you don't measure." Security metrics provide a means for measuring properties of computer security systems. According to the National Institute of Standards and Technology (NIST) [13] and the SANS Institute [14], security metrics are seen as an important factor in evaluating the effectiveness and efficiency of security architectures. We believe that to holistically evaluate monitor placement, it is necessary to define a set of quantitative monitor deployment metrics.

1.1 Contributions and Organization

This thesis makes several contributions, as described below.

- To begin to solve the problem of monitor placement, we first define a *system and data model* that describes the system we aim to protect and the monitors that can be deployed in the system. We represent the system as a graph of networked nodes that contain both system resources on which monitors can be deployed and standalone monitoring resources. Since the ultimate goal of intrusion detection is to detect attacks on a system, we relate our metrics and deployment methodology to the attacks that we would like to detect. We represent monitors as sources of evidence of attacks in the system. Our data model provides a mapping of monitor information to attacks that is motivated by actual algorithms and techniques that would perform intrusion detection using the collected data.

- Second, we define a set of utility and cost *metrics* that can be used to characterize the ability of a deployment of monitors to meet intrusion detection goals and to capture the cost associated with monitor deployment. The goal of the metrics is to provide quantitative means for evaluating the information provided by monitors relative to the attacks to be detected in the system. We provide the intuition and motivation behind each of the metrics and define how they would be computed. While our set of metrics is by no means complete, it provides a baseline for quantitative evaluation of monitor placement.

- Third, we formulate a *method to determine the optimal placement of monitors* using our metrics and data model that is based on the tradeoff between intrusion detection requirements and cost constraints. We first introduce a set of weights and constraints, and show how a practitioner can use them to specify intrusion detection goals and requirements. We then consider two different types of cost constraints, and devise two integer programs that can be solved to find an optimal deployment of monitors for a given set of requirements.

- Finally, we demonstrate the use of our optimal monitor deployment methodology through a *case study* in which we build a mock enterprise Web service. We implement all components of our model and describe a method to programmatically extract indicators from logs collected from monitors, which we implement for the monitors in our case study. To evaluate our approach, we construct a series of intrusion detection scenarios and show how a practitioner could represent the intrusion detection requirements and goals for the scenarios within our model. We show that for each of the scenarios, our methodology is capable of determining the optimal monitor placement, some of which are not intuitive.

The rest of this thesis is organized as follows. Chapter 2 provides a survey of research that has been done in determining optimal monitor or IDS placement in networked systems to maximize intrusion detection ability or minimize attacker capability. In Chapter 3, we describe our system and data model, which we use to represent intrusions, monitors, and the evidence monitors provide of intrusions, and the fundamental assumptions that we make in our model. We then define our core set of monitor deployment metrics in Chapter 4. Subsequently, in Chapter 5, we define the representation of intrusion detection goals within our model, and we present our optimization equations that can optimize monitor deployment based on the intrusion detection requirements. Putting all of the pieces together, we illustrate how our overall methodology can be used to deploy monitors with the case study experiment in Chapter 6. Finally, we conclude and identify areas of ongoing and future research in Chapter 7.

CHAPTER 2

RELATED WORK

2.1 Industry Approaches

In practice, monitor deployment is generally done by a system administrator, who either deploys monitors based on domain knowledge and expected monitoring needs or uses existing commercial coordinated monitoring and analysis tools that in turn deploy monitors.

Some organizations, such as Cisco and the SANS Institute, publish strategies for the deployment of network IDSes [15, 16, 17]. The strategies, however, provide high-level guidelines and suggestions rather than formal approaches to IDS placement, overlooking the theoretical soundness of the deployment. For example, in [15], Pappas suggests not to deploy inline IPSes on high-throughput links, to deploy IDSes and IPSes in pairs, and to use penetration testing to evaluate one's IDS deployment. In [17], Carter makes recommendations such as placing an IDS outside the firewall and placing IDSes between internal groups on the network. Neither Pappas nor Carter provides a sound, quantitative mechanism by which to evaluate IDS placements in terms of their ability to detect intrusions.

Many commercial off-the-shelf (CotS) packaged monitoring and analysis solutions exist for domains such as enterprise systems or cloud data centers. Prominent vendors include NetIQ [9], IBM [10, 18], Symantec [11], Tripwire [19], and Splunk [20]. Such tools control monitor deployment, monitor configuration, and log management behind the scenes, and perform monitor data fusion and intrusion detection to provide a security administrator with high-level alerts about potential threats to the system. Other security information and event management (SIEM) and incident forensics tools offered by the same vendors give an administrator the ability to perform forensic analysis by retracing the actions of an attacker through collected logs. SIEM tools and solutions provide a long list of features, and are particularly attractive because they ensure compliance with data privacy and security regulations and offload all monitoring responsibilities to the tool, alleviating the burden for a security administrator.

As an example of the features provided by SIEM tools, consider the IBM QRadar Security

Intelligence Platform [18]. The suite offers a Log Manager tool that collects, analyzes, and archives log data; a SIEM tool that consolidates and normalizes log data across the entire network and correlates the data to detect intrusions; a Vulnerability Manager tool that discovers new vulnerabilities and prioritizes their mitigation; and an Incident Forensics tool to trace the step-by-step actions of an attacker and dig into logs to conduct forensic investigation of potential malicious network activity. All of the tools can work in tandem to perform coordinated monitoring, intrusion detection, and response for a security administrator.

However, a significant issue with industry SIEM tools is that the algorithms behind their operation are closed-source and proprietary, so it is not possible to understand how the tools differ in functionality or what intrusion detection guarantees the tools provide. Since the monitoring performed by the tools is also hidden from the administrator, it is not possible to evaluate the cost-effectiveness of the monitoring or efficiency of the algorithms performing intrusion detection. As a further consequence, it is not possible to integrate data from additional monitors or different SIEM tools together to improve the intrusion detection capabilities of the tools. We believe there is a need for a widely applicable, quantitative methodology for evaluating the effectiveness of monitoring tools and determining optimal monitor deployments based on a security administrator's unique constraints and intrusion detection requirements.

2.2 Monitor Deployment Literature

Interest in the topic of resource-limited security monitor and intrusion detection system (IDS) placement and evaluation has increased in recent years. The research literature in the area largely follows three tracks: placement based on maximizing detection using attack graphs, placement based on game-theoretic considerations, and placement based on ensuring security policy adherence of information flow graphs.

2.2.1 Attack Graph-Based Monitor Deployment

Attack graphs have been used extensively in the literature to describe attacks and their relation to monitors [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]. Attack graphs model attacks in terms of the component attack stages that can be detected by monitors and IDSes. The attacker starts in one of multiple initial states in the graph, where the intial states represent the attacker's starting capabilities and level of access to the system. Through actions in the system, such as exploitation of vulnerabilities or reconnaissance actions, the attacker traverses through connected states in the graph to increase its capabilities and access to the system. The attacker's goals are represented as a set of target states in the graph. A single successful attack in an attack graph would be represented as a single path in the attack graph leading from an initial state to a target state. As attack graphs describe attacks in stages of execution that can be detected incrementally, they provide an intuitive method for modeling intrusions in a distributed system.

Attack graph formalisms can largely be categorized into two types. In the first, the nodes of the graph represent the state of the entire system, and edges represent actions the attacker could take to change system state. Examples include the model-checking-based attack scenario graph presented by Sheyner et al. [21], the network attack graph used by Jajodia et al. in their Topological Vulnerability Analysis tool [27], and the attack graph generated by the tool developed by Swiler et al. [29]. In the second, the vertices of the graph represent logical attack stages or attack conditions, and the edges represent causal or precondition-postcondition relationships between the nodes. Examples include the exploit dependency graph defined by Noel et al. [22], in which vertices are exploits and edges are dependencies between exploits; the Correlated Attack Modeling Language (CAML) defined by Cheung et al. [24], in which vertices are attack steps and edges are inferences that can be made given attack steps and monitor information to detect subsequent attack steps; and the attack execution graph used by LeMay et al. in ADVISE [23], in which nodes represent many properties of an attacker's behavior, including attack steps, attack goals, attacker knowledge and skill, and attacker access to the system, and edges represent dependencies between nodes.

Many researchers have proposed methods for determining monitoring requirements or hardening measures from attack graphs. Sheyner et al. [21] define an attack scenario graph in which paths in the graph represent exploits that can be performed between nodes in the system network graph. Within their formalism, preventing exploits from being performed removes edges from the graph, thus making some attack goals impossible to achieve. Therefore, the authors attempt to find the minimal and minimum sets of attack steps that must be blocked to prevent an attacker from reaching his or her objectives.

In CAML [24], Cheung et al. consider how information from different monitors and IDSes must be correlated in order to detect individual attack steps. They separate attack steps that can be detected directly from sensors from those that require inference on observed attack steps, and specify the temporal and prerequisite relationships between different attack steps. Using their attack scenario graph, the authors construct a scenario recognition engine that can detect attacks using information arriving from sensors in real time. The authors suggest the use of their approach to actively tune monitors based on the global view of system security provided by the scenario recognition engine, but they do not actually provide such a method.

Jajodia et al. [28] use their Topological Vulnerability Analysis (TVA) tool to perform a host of analyses about how changes to a networked system impact its overall security. In TVA, attacks are represented as paths through an attack graph, where vertices in the graph represent vulnerabilities and edges represent their interdependencies. Traversal of the attack graph corresponds to an attacker's taking attack actions and progressively compromising components in the system. The attack graph contains topological information, in that each vertex corresponds to a vulnerability on a given asset, and an edge only exists between two vertices if an attacker with access to the originating vertex has the capability to perform the exploit associated with the destination vertex. The authors illustrate their approach and suggest how it could be used to decide where to invest resources to have the most impact on system security, or analyze how changes to the system could impact overall risk, but again do not provide details on how this would work.

Noel et al. [33] use a probabilistic attack graph model to study the problem of resource allocation to maximize security benefit. The authors use live network scans and vulnerability database information to populate their topological attack graph model, which is similar in construction to the one used by TVA, and assign exploit success probabilities to each of the edges based on publicly available vulnerability scores. The authors then use simulations to evaluate the return on investment from making different changes to the monitors to determine which set of changes is cost-optimal. While the authors' approach makes decisions at a finer grain than our approach, such as deciding whether or not to block traffic of a specific protocol, it considers monitoring only at the level of IDSes, when in fact in practice, monitors from various levels are used to detect intrusions. Furthermore, it does not consider the addition or replacement of monitors.

Albanese et al. [26] focus on the issue of *real-time* intrusion detection. The authors use *temporal* attack graphs, which incorporate time delays and time windows into the definition of an attack graph. The authors merge multiple temporal attack graphs, each for a different attack, into a single *temporal multi-attack graph* and use an indexing approach to efficiently update information about which attacks are occurring as IDS alerts and other monitor data are observed. The authors claim that by doing so, they are able to detect attacks in real-time from heterogeneous sensors.

The approaches discussed above provide a sense of how a system should be changed in order to detect or prevent a given set of attacks. They do not, however, consider how placement of monitors or the types of monitors being deployed affect the data that are being fed to the intrusion detection algorithms to determine system security state. Considering that the intrusion detection algorithms rely solely on the logs generated by monitors to perform detection, the monitor placement problem is important.

Some researchers have used attack graphs to predictively determine optimal monitor deployments. For example, Noel et al. [31] use TVA attack graphs to model the system and determine where to place NIDSes to minimize monitor cost and ensure detection ability for all attacks represented by the attack graph. The authors take advantage of the fact that the TVA attack graph captures topological information about the progression of an attack to identify the critical paths through the network an attacker would need to take to accomplish an attack. The authors then determine which of a set of available NIDSes should be enabled to ensure coverage of the attack graph. The authors' approach neglects the goal of preemptive or rapid detection, which may be aided by monitor redundancy, and fails to consider monitors other than NIDSes.

Almohri et al. [32] also attempt to minimize the success probability of attackers, but they take mobile devices into account. The authors define a logical attack graph that acts much like a fault tree. Vertices represent system state, attack preconditions, and attacker goals. An edge between two vertices represents the conditional expected probability of attack success for one vertex given the probability of attack success of the other vertex. Given starting vertex probabilities, the overall attack success probability can be determined using conditional probability. The authors determine monitor cost based on monitor resource utilization, purchase cost, and code library utilization. They use their model to evaluate the effects of different security hardening devices, such as firewalls and IDSes, on the overall attack success probability, and devise a linear program that yields a cost-optimal deployment of security resources that minimizes attack success probability. Because of the probabilistic nature of the authors' approach, they are able to quantify the uncertainty of detection and consider the effect of unknown attacks, such as zero-days. However, the authors do not provide a mechanism to represent the cost of a successful attack in the objectives of their optimization problem. It may be the case that one attack with a low detection probability has devastating consequences and must be prevented at the expense of detecting other attacks.

The predictive deployment approaches described above do not take into account the effect of monitor compromise on intrusion detection. Indeed, Noel et al. attempt to minimize monitor redundancy to achieve a cost-optimal monitor deployment, but they do not consider how compromise of one of the IDSes on a critical path could eliminate a data source needed for detection of many attacks. We believe that since intrusions in the system may compromise the data provided by monitors, it is important to consider compromise in an evaluation of monitor placement.

2.2.2 Game-Theoretic Monitor Deployment

Game theory, too, lends itself to modeling of attacker and defender behavior when a system is under attack. Schmidt et al. [34] attempt to determine an optimal IDS placement strategy for a defender, assuming a rational attacker. They model the behavior of an attacker and defender using a two-person, zero-sum, non-cooperative Markov game over a topological attack graph. Attacker cost is represented as the cost of detected malware, and defender cost is represented as the cost of missing a malware packet. At any step, the defender is capable of deploying a packet sniffer anywhere in the network that feeds into an IDS with perfect detection capability, and the attacker is capable of advancing an attack through the network. The authors use simulation to show that a defender using their placement algorithm can drastically reduce the number of malware packets that go undetected. In a realistic system, however, the defender may be playing against many attackers, each with different goals, so any placement decisions should optimize over all attacks to detect. Furthermore, in most cases, because of intrusion detection delay or false negatives, the defender may not learn of the attacker's behavior immediately and take response actions before the attacker's next move, so this model of attacker and defender behavior is not viable. The authors do consider the effect of compromise of the IDS, noting that under compromise, the optimal deployment provided by their algorithm does little to ensure detection of attacks.

Zhu et al. [25] likewise apply game-theoretic principles to attack graphs, modeling attacker and defender behaviors in a zero-sum stochastic game to determine how to optimally *configure* IDSes to minimize the utility of an intelligent attacker. Unlike Schmidt et al., the authors model the game as an N-person cooperative game in which utility for each player is defined as a binary value. The authors assume that the IDS takes prior attack graph knowledge as input and will use detection of early attack stages to attempt to prevent future attack stages. The authors use game theory to find an initial configuration for the IDSes that is optimal in terms of defender attack detection ability and detection efficiency.

Like many of the attack-graph-based deployment approaches, both of the above approaches focus on deployment or configuration of network IDSes, neglecting the deployment of lower-level monitors, such as access logs, that can yield valuable information for intrusion detection and forensic analysis.

2.2.3 Information Flow Graph-Based Monitor Deployment

Other researchers have proposed the use of network information flow graphs to determine where to place monitors. Information flow graphs represent how network traffic can flow between hosts in a network, taking into account network topology; the configuration of network hardware, such as routers and switches; the configuration of security hardware, such as IPSes and firewalls; and the flows established by hosts.

Talele et al. [35, 36] model the flow of security information among hosts and network IDSes in terms of information flow graphs and attempt to determine minimal-cost traffic mediator placements that can prevent all information flows disallowed by security policies. The authors use graph theoretic algorithms to determine where in the network to place the mediators. They argue that by preventing network flows that violate security policies, their approach can also restrict attack paths. The authors make the strong assumption, however, that all devices, operating systems, and programs that enforce information flow policies do so correctly, which may not be the case if the systems are compromised.

Bloem et al. [37] model general network traffic using information flow graphs and determine optimal network filtering policies that ensure detection of malware in the network. adherence to security policies. The authors aim to minimize the total amount of network packet sampling that must be conducted by monitors across the network while ensuring a minimum amount of malware detection. They define the optimal deployment problem as a mixed-integer nonlinear program, and solve the program to obtain the optimal filtering policies. The authors' approach focuses very specifically on malware detection through network filtering, so it does not generalize well to overall network intrusion detection.

Again, these two approaches focus only on the deployment or configuration of network IDSes. Furthermore, examining only information flow policies ignores the ability of an IDS or forensic analyst with access to *all* of the data in detecting intrusions.

2.2.4 Summary

Monitor deployment in industry currently relies on proprietary tools for which it is not possible to evaluate the cost-effectiveness or efficacy of monitor deployment, or on best practices that neglect monitor compromisability or monitor cost. Approaches in the research literature use attack graphs, game theory, and information flow graphs to model the system, and reason about monitor deployment by taking intrusion detection ability, attacker capabilities, monitor cost, and probability of attack steps into account. However, most of the approaches focus primarily on NIDSes, and almost none of them consider the effect of compromise on the deployment of monitors. We see the need to model the system while taking cost and compromise considerations into account, and to determine optimal monitor deployments in terms of both.

CHAPTER 3

SYSTEM AND DATA MODEL

3.1 Guiding Observations and Assumptions

3.1.1 Observations

In order to guide our research in monitor deployment and development of monitoring metrics, we make two observations about monitors, monitor data, and intrusion detection.

3.1.1.1 Monitor Faultiness and Compromise

We observe that monitors, once deployed, become part of a system's attack surface. The monitoring system is made up of software and hardware components, just like the systems that we aim to protect. Monitoring components are deployed upon system assets, which we know can be attacked and compromised. Since any computer system is susceptible to hardware and software faults, the monitoring system itself could be faulty or compromised, and, as a result, produce false alerts or fail to produce alerts at all. For example, a monitor that observes the resource consumption behavior of virtual machines must have some component that resides on the same physical machine as the VMs. If the physical machine is subject to hardware faults that cause CPU operations to return incorrect results, the VM resource monitor will also be subject to faults, which would make the information provided by the monitor unreliable.

Understanding that monitors may be unreliable or be the target of attack motivates our research in two ways. First, we take the effect of compromise into account when developing our set of metrics, as we cannot assume that the presence of a monitor or the generation of an alert implies that the monitor will always generate indicators or that the indicators are necessarily correct. For example, in [38], Pham et al. showed that on Windows systems, some rootkits can hide themselves from OS process enumeration methods, thus rendering themselves undetectable from intrusion detection mechanisms that run within the OS or rely on user space monitor data. If such a rootkit were installed on a Windows VM, any activities taken by the rootkit would not be captured by monitors deployed within the VM, even if the same activity were detectable if performed by a normal process. Second, our models and metrics quantify 1) the ability to detect intrusions when all monitors are functioning correctly as well as 2) the redundancy of monitors. We believe that increased redundancy in data collection can increase the ability to detect intrusions and perform forensic analysis when some monitors are unavailable or are providing erroneous data. For example, in [38], though it is not possible to detect hidden rootkits from OS-level monitors such as Ninja, Pham et al. are able to use their hypervisor-based hidden rootkit detection monitor to identify hidden processes and detect the rootkit.

The problem of monitoring on systems susceptible to compromise has been studied in the past [7], but we consider the problems of securing the logs or ensuring that it is possible to detect tampering to be outside the scope of this thesis. Instead, we focus on capturing the *effect* of compromise on the logs generated by monitors, and, in turn, on our ability to detect intrusions.

3.1.1.2 Monitor Heterogeneity

We observe that different monitors produce information in heterogeneous formats. As a result, our model should be able to take heterogeneous information into account when representing the information produced by monitors and consumed by intrusion detection systems and forensic analysts.

3.1.2 Guiding Assumptions

We also make the following two simplifying assumptions that make the problems of modeling monitors and intrusion detection approaches tractable.

3.1.2.1 Independence of Monitors

We assume that monitors generate information independently given that the assets on which they run do not in some way depend on a common component. That is, different monitors use independent underlying mechanisms to generate their alerts, even if they produce the same types of alerts. That may not always be the case. For example, consider two host-level network traffic monitors, Snort and tcpdump. Snort [39] is a network intrusion detection and prevention system that can perform real-time network traffic analysis on IP traffic. Here, we consider the case where Snort is being run on a host, monitoring all traffic on the network interfaces connected to the host. Tcpdump [40] is a command-line packet analyzer that can display all IP traffic over the network to which a host is attached. To perform full packet capture, Snort and tcpdump both rely on a packet capture library (libpcap on Unix-like systems and WinPcap on Windows) to obtain copies of full packets from the operating system. Since Snort and tcpdump run as separate processes, the compromise of one does not necessarily result in compromise of the other. If the underlying packet capture library on which they both depend were replaced with a malicious copy, however, both Snort and tcpdump would no longer be trustworthy.

While understanding the dependencies between monitors is important, to do so would require static source-code or binary analysis of each monitor to determine which shared libraries and operating system utilities are used by each. Constructing such a model would allow us to gain a more nuanced understanding of how compromise of system components would affect intrusion detection ability, but we defer this to future work.

3.1.2.2 Intrusion Detection Ability

We separate the problem of monitor deployment from the problem of intrusion detection using information provided by monitors. Through our model, we aim to be able to accommodate arbitrary fusion or analysis techniques and determine a deployment of monitors that can provide enough information for detection.

We approach the problem of monitor deployment from a *proactive* perspective. That is, unlike the problem of intrusion detection, in which the placement of monitors is fixed and the goal is to use data from already-deployed monitors to detect intrusions, we consider the problem of deploying monitors *before* the system is running in order to support intrusion detection. Thus, we develop a model that supports evaluation of the ability of monitors to aid in intrusion detection before any traffic passes through the system and monitors begin generating data.

For a realistically large set of events, most intrusion detection approaches in the literature and those in use in industry have nonzero *false positive* and *false negative rates* [41, 42]. That is, the intrusion detection approaches may claim to detect an event when the event is not actually taking place (false positive), or may fail to detect an event that is taking place (false negative). Because our analysis of monitor deployment is performed prior to monitor data generation or intrusion detection, we cannot guarantee that events will always be correctly detected by an intrusion detection system or forensic analyst, or quantify the probability that they will be detected.

Thus, to simplify our characterization of the fusion approach taken, we make the assumption of *perfect forensic ability* of the intrusion detection system or forensic analyst. Under this assumption, if an attack occurs in the system, the forensic analyst or IDS will be able to detect the attack if given access to a set of monitors that generates some minimal amount of information about the intrusion, where we define the minimal amount of information needed for detection in our model. Put in intrusion detection terminology, we assume that the IDS or forensic analyst has a zero false negative rate.

While that assumption would be unrealistic, it allows us to be agnostic with respect to fusion and forensic analysis approaches, assuming a best-case scenario for said processes. For online intrusion detection, the assumption corresponds to an ideal fusion algorithm that can consider all monitor information simultaneously in real time and detect intrusions with a 0% false negative rate given that enough information exists to detect the intrusion. For after-the-fact forensic analysis, the assumption corresponds to an ideal forensic analyst, who, given enough time, money, and resources, could detect any attack that could possibly be detected given the information provided. We consider the problem of incorporating the abilities of specific fusion or forensic analysis approaches into monitor deployment as future work that is outside the scope of this thesis.

Our proactive perspective does not preclude use of our monitors to evaluate the effectiveness of monitor placement after the system has started running. In fact, we aim for our monitor deployment metrics to support redeployment of monitors as a response action to increase the focus of monitoring as the understanding of system state changes. Ultimately, we envision that our monitor deployment metrics and deployment methodology can be used as part of a larger intrusion tolerance system in which the decisions on where to deploy monitors are driven by monitor data fusion and response actions.

3.2 Definitions

In order to quantitatively evaluate the usefulness of a set of security monitors for intrusion detection, it is first necessary to define a model on top of which the quantitative metrics can be defined. Such a model must be able to accurately represent both the capability of monitors to generate information pertaining to the state of the system and the techniques or algorithms used to aggregate, correlate, and analyze the data to detect intrusions in the system.

We first define the following components of the monitoring system:

Definition 1. Monitors *are the sensors that collect and report information about the state of the system.*

Monitors observe heterogeneous system components and generate information about the system that can be used in intrusion detection and forensic analysis. While we do not adopt the formalism for the system and state provided by Bishop in his formal model of security monitoring [43], monitors under our framework perform the *logging* functionality defined in Bishop's model, and may optionally perform some *auditing* functionality.

In security and intrusion detection literature, the primary focus is on monitors that fuse raw system information, such as raw network packets or raw syslogs, to produce alerts about possible malicious activity in the system. Such monitors necessarily perform *auditing* as defined by Bishop in his formal model of security monitoring [43], and are often called "intrusion detection systems" (IDS). While IDSes produce a consolidated view of the security of a system, we believe that it is important to support the representation of all types of monitors in our model, as new approaches to coordinated intrusion detection may fuse information from different sets of monitors in novel ways not captured by existing IDSes. We desire to be able to use our model to reason about monitor deployment for a wide set of fusion approaches. Thus, we do not restrict our definition of monitors to those that provide security-related alerts; instead, we allow for the inclusion of any monitor that could provide information about the system that might be used in intrusion detection.

Some examples of monitors include the Apache server access log, which logs all requests processed by an Apache HTTP server; Snort, which logs header information for network traffic observed over a network interface and can perform traffic analysis and packet inspection on the traffic to generate alerts; and nmap, a network topology mapper that performs a network scan to detect the topology and reachable endpoints in a network and could be configured to perform periodic scans of the network. As a result of the heterogeneity of the information provided by different monitors, the format and informational content of the alerts generated can vary considerably between different monitors. Our model represents monitors in a general manner by abstracting the information generated by monitors, as discussed later in this chapter.

Definition 2. Events *are (1) attacks or intrusions in the system or (2) actions taken in the system that could be symptomatic of an attack or intrusion.*

Events represent the malicious or suspicious actions that an attacker might perform in the system that a security administrator would want to detect. Events could be explicit violations of security policy, or actions that an attacker might take in the process of conducting an attack. While we acknowledge the importance of other types of events, such as quality-of-service-related events, we focus our attention in this thesis on intrusion events. Since our analysis takes place offline, we elide the notion of time in our definition of an event. We intend for our definition of events to be general, so as to allow practitioners to enumerate events using any methodology of their choice.

Event definitions may include source and/or target information for an intrusion, or may represent any manifestation of the intrusion, depending on intrusion detection goals. For example, for a TCP SYN flood denial-of-service attack, a practitioner could represent the attack within our model as a set of events "TCP SYN flood DoS attack on target t_i" (for some set of targets t_i) or as "any instance of a TCP SYN flood DoS attack in the system" (generalizing the set of targets).

Events are not concrete pieces of data that are directly generated by monitors. Rather, detection of an event potentially requires the correlation of information from multiple monitors. Within our model, events are completely described by the relationship between the event and the information provided by monitors, which serves as an abstraction of fusion and forensic analysis. We discuss that relationship later in this chapter.

Definition 3. Indicators *are the primitives that represent information provided by monitors about the events taking place in the system.*

Indicators represent the semantic meaning of the data generated by the monitors, not the actual data that are produced. A single indicator is an observation that can be completely determined by some logical predicate evaluated over information generated by a single monitor and can be used to define the conditions necessary to detect an intrusion.

As an example, consider a system call monitor for an operating system. The monitor will generate log entries in an operating-system-specific format for each system call made by a process. The entries might contain information such as process ID, timestamp, and function name. From the perspective of intrusion detection, however, if we are interested in whether a specific process has executed a specific function call, we could represent such a query as a logical predicate and its instantiation within the logs as an indicator, and therefore consider it observable by the system call monitor. We leave the exact specification of the indicator logic to future work, but some examples of related work exist in the literature [44, 45, 46].

By taking the approach described above, we eliminate the need to consider the heterogeneous formats of the data generated by individual monitors or the need to define a common format and technique for conversion. The approach also facilitates enumeration of indicators by security domain experts, who may not be able to specify the exact format for the logs gen-

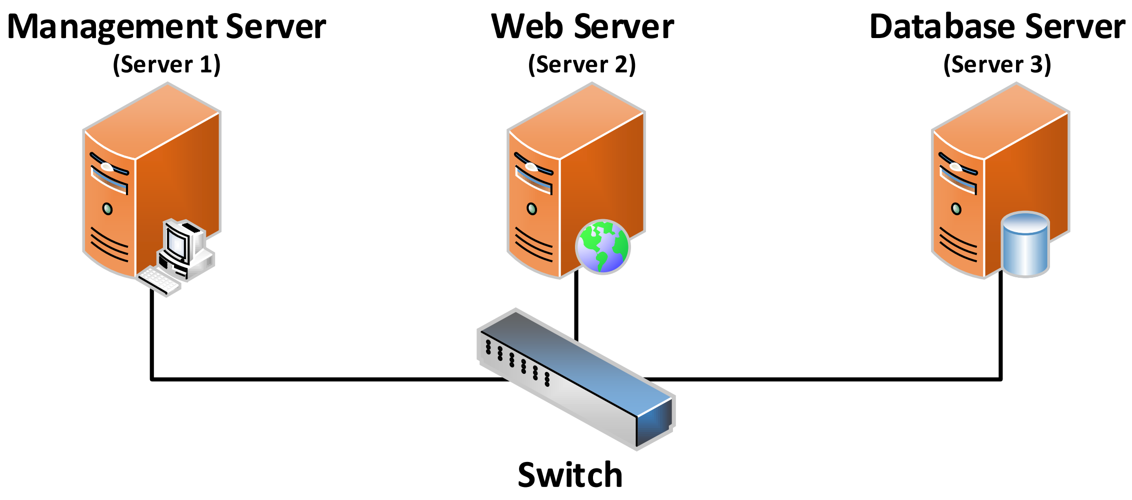

Figure 3.1: Architectural diagram for the working example. Server 1 runs SSH; Server 2 runs the Web application; and Server 3 runs the MySQL database for the Web application.

erated by each of the monitors they can deploy, but understand what semantic information is provided by each log. Furthermore, our approach presents the potential for programmatic enumeration of indicators, which we discuss and illustrate in our case study in Chapter 6.

Definition 4. Assets *are the system's computing components that we wish to be able to protect from intrusions or that may have monitors deployed on them.*

Assets can be defined at any granularity in the system that is considered relevant by a security administrator. Our definition of assets is the same as that introduced by Thakore et al. in [47]. Assets of value may include host machines, servers, virtual machines, applications, and network hardware. While most monitors will likely be deployed on the systems they protect, there are assets, such as firewalls, that serve only as monitoring components.

3.3 Working Example

To illustrate the concepts in this and the following two chapters, we use a small running example that contains commonly used components and is vulnerable to attacks that are representative of those experienced by the larger systems one would see in enterprises.

Consider the system illustrated in Figure 3.1. The system provides a Web service to consumers, who access the service over the Internet. The system is composed of three Linux servers connected by a single network switch. Server 1 runs an SSH daemon service on top of Linux, with its firewall rules set to allow SSH connections from the outside Internet for management purposes. Server 2 runs an Apache HTTP server on top of Linux, and hosts a website that uses PHP. Server 3 runs a MySQL database that is used by the website hosted on the Apache HTTP server. The network switch acts as a firewall and an intranet switch, preventing all access to Servers 1, 2, and 3 from the outside Internet except for port 22 access to Server 1 for SSH and port 80 access to Server 2 for HTTP. Within the internal network of the system, there is no access restriction between servers, so each of the servers is capable of accessing any open ports on any of the other servers.

In our working example, monitors can be deployed to monitor each of the application assets within the system. On Server 1, the SSH daemon service can be configured to generate SSHd logs, which contain information about all SSH access attempts to the system and unexpected connections to port 22. On Server 2, the Apache HTTP server can be configured to generate Apache request logs, which create an entry for each HTTP request sent to the Apache server. PHPIDS [48], a PHP Web application IDS, can also be deployed to Server 2 to detect PHP and other attacks on the website. On Server 3, the MySQL server can be configured to generate MySQL query logs, which log every query submitted to the MySQL server. Finally, as all of the network traffic to and from Servers 1, 2, and 3 passes through the network switch, the switch is able to observe and monitor all traffic in the system. It can be configured to mirror all of its traffic to Server 1 for analysis by Snort.

3.4 System Model

Before describing how we represent monitor information, we must first describe the formalization we use to represent the set of system components.

First, we define the base system components:

$$\begin{aligned} V_S &= \{v \mid v = \text{a system component to protect}\} \\ E_S &= \{e \mid e = (v_i, v_j)\,,\ v_i, v_j \in V_S,\ v_i \text{ depends on } v_j \text{ for proper operation}\} \end{aligned}$$

V_S represents the complete set of computing assets in the system that are either of value or may have monitors deployed on them, as defined above.

E_S represents the set of asset dependence relationships in the system. Note that these are not the same as network connections between the assets; instead, they represent dependencies between different assets v_i and v_j that we can use to understand the effects of compromise.

An example of such a relationship is the dependence between SSHd and the server on which it runs. If the server were to be compromised, it could tamper with SSH traffic, causing any actions taken by SSHd and any data in the SSHd logs to be untrustworthy. In that case, we would represent the dependence relationship as an edge e in E_S between SSHd and Server 1.

To represent the monitoring infrastructure, we augment the system graph with the set of monitors that are deployed. The monitors are defined as additional vertices and edges in the graph, which we define as follows:

$$M = \{m \mid m = \text{a monitor that can be deployed in the system}\}$$
$$E_M = \{e_m \mid e_m = (m, v), m \in M, v \in V_S, m \text{ can be deployed on } v\}$$

M represents the set of monitors that can be deployed in the system. Each of the elements in M can be deployed by the system administrator to one of the assets in V_S in order to increase the amount of system information collected. For example, in our working example, M would include the MySQL query logs, irrespective of whether query logging is enabled.

It is possible to include (or exclude as necessary) any set of monitors in the set over which the monitor deployment optimization is to be performed. If there is a monitor that could be deployed with multiple possible configurations, each configuration could be treated as a different monitor, and the optimization could be run multiple times, where the set M would contain a different configuration of the monitor for each run.

E_M represents the deployment relationships between the monitors in M and the system assets in V_S. Specifically, an edge e_m exists in E_M if the correct operation of monitor m depends directly on the correct operation of asset v. Since the asset dependence relationships are captured by the edges in E_S, edges in E_M are directed to the highest-level assets on which the monitor depends. For example, consider the case of the MySQL query log given above. MySQL runs on the Server 3 Linux instance, which is connected to the other servers and to the outside Internet by the network switch. If the switch, Server 3, or the MySQL server instance were compromised, the integrity of the query log would be suspect. We would, however, map the query log directly to the MySQL server vertex in V_S, and not to the other assets.

The significance of the monitor deployment mappings in E_M is that, as is the case with dependence relationships in E_S, if an asset v is compromised or taken down, m may also potentially be compromised or taken down.

Definition 5. *We define the* system graph, *S, as a dependency graph constructed from the*

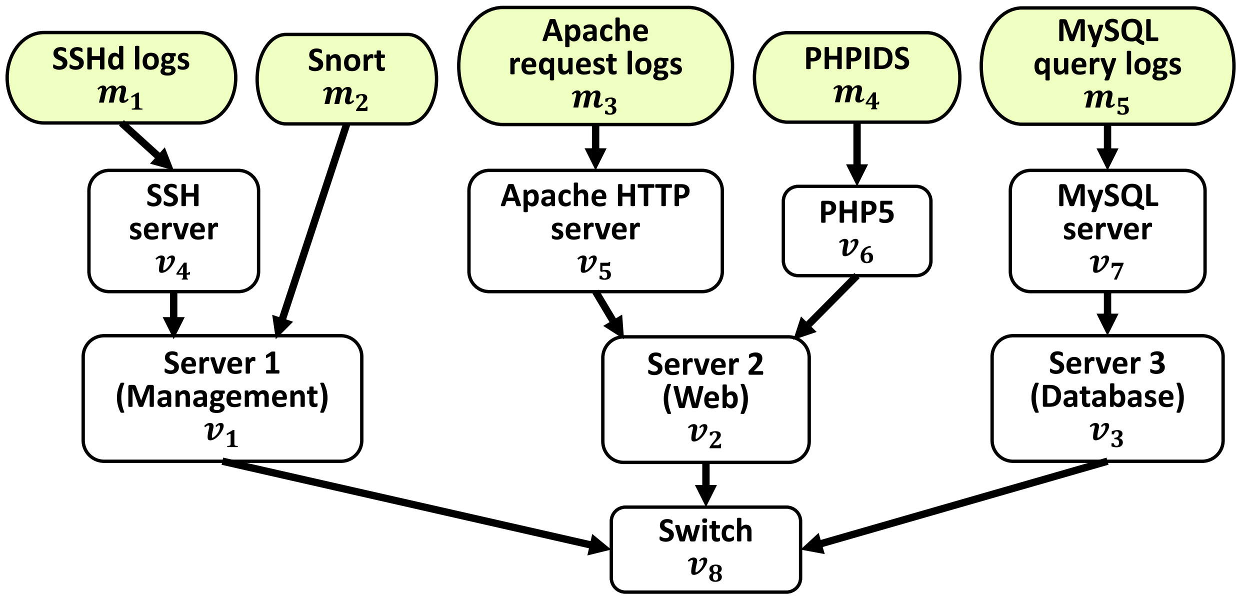

Figure 3.2: System model for the working example. White shapes represent assets, shaded green shapes represent monitors, and arrows represent dependence relationships.

components in the system and their dependence relationships:

$$S = (V, E)$$

where $V = V_S \cup M$ *and* $E = E_S \cup E_M$.

This representation of the system graph in terms of assets and their dependencies is driven by the guiding principles of monitor and asset compromisability we discuss in Section 3.1.1.1. Our aim is to capture how monitors can be used to gain information about the system and how components in the system depend on one another.

Our definition of system assets is motivated by the definitions of assets and layers proposed by Thakore et al. in [47] and [49]. The use of a graph to represent the system is motivated by the cyber-physical topology language proposed by Weaver et al. in [50]. As suggested by Weaver et al., such a representation allows us to perform interesting operations like contraction to higher-level views, which we could use to analyze monitor deployment based on the compromise of different sets of system components.

In our working example, we treat all of the hardware and software components as assets. Where a software component is deployed on a hardware component, the software component *depends* on the hardware component, so an edge in E_S exists between the two. The edges in E_M are defined by the deployment of the monitors to the software and hardware components. The equations and figure describing the system model for the working example are given by

$$\begin{aligned}
v_1 &= \text{Server 1} & e_1 &= (v_1, v_8)\\
v_2 &= \text{Server 2} & e_2 &= (v_2, v_8)\\
v_3 &= \text{Server 3} & e_3 &= (v_3, v_8)\\
v_4 &= \text{SSH server} & e_4 &= (v_4, v_1)\\
v_5 &= \text{Apache HTTP server} & e_5 &= (v_5, v_2)\\
v_6 &= \text{PHP5} & e_6 &= (v_6, v_2)\\
v_7 &= \text{MySQL server} & e_7 &= (v_7, v_3)\\
v_8 &= \text{Switch} & &
\end{aligned}$$

$$\begin{aligned}
V_S &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}\\
E_S &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}
\end{aligned}$$

$$\begin{aligned}
m_1 &= \text{SSHd logs} & e_{m_1} &= (m_1, v_4)\\
m_2 &= \text{Snort} & e_{m_2} &= (m_2, v_1)\\
m_3 &= \text{Apache request logs} & e_{m_3} &= (m_3, v_6)\\
m_4 &= \text{PHPIDS} & e_{m_4} &= (m_4, v_7)\\
m_5 &= \text{MySQL query log} & e_{m_5} &= (m_5, v_8)
\end{aligned}$$

$$\begin{aligned}
M &= \{m_1, m_2, m_3, m_4, m_5\}\\
E_M &= \{e_{m_1}, e_{m_2}, e_{m_3}, e_{m_4}, e_{m_5}\}
\end{aligned}$$

Figure 3.3: Equations describing the working example system model illustrated in Figure 3.2.

Equation 3.3 and Figure 3.2.

3.5 Event and Indicator Model

We have defined events and indicators above. We now define the sets of each that we can use in our evaluation of the efficacy and cost-effectiveness of monitor deployment.

Definition 6. *The* **set of events of interest***, Φ, is the set of all events that may take place in the system that we would like to be able to detect.*

Currently, events of interest (that is, those worth dedicating monitoring resources to aim to detect) must be specified by a security administrator with domain expertise based on attacks of importance. A systematic method by which the set Φ can be obtained is left to future work and is discussed in Section 7.2.

For our working example, we are interested in detecting the following events of interest:

$$\begin{aligned}
\phi_1 &= \text{SSH brute-force} \\
\phi_2 &= \text{phpMyAdmin attack} \\
\phi_3 &= \text{Incidence of botnet running on servers} \\
\phi_4 &= \text{SQL injection}
\end{aligned}$$

Definition 7. *The* **set of observable indicators***, I, is the set of indicators that can be generated by at least one of the monitors, $m \in M$.*

It may be possible to observe other indicators given the presence of other monitors, but we restrict our view of the universe of all possible indicators to those that we can observe given the monitors that we can deploy, since our control over monitor deployment only extends to such monitors.

For our working example, we define the following observable indicators:

$$\begin{aligned}
\iota_1 &= \text{failed SSH attempts} > \mathit{threshold}_{\text{SSH}} \\
\iota_2 &= \text{SSH initiation attempts} > \mathit{threshold}_{\text{SSH}} \\
\iota_3 &= \text{HTTP request to phpMyAdmin from IP not in whitelist} \\
\iota_4 &= \text{IRC } \texttt{NICK CHANGE} \text{ message on non-IRC port} \\
\iota_5 &= \text{traffic on IRC ports} \\
\iota_6 &= \text{IRC botnet control packet matches} \\
\iota_7 &= \text{MySQL version string in packet} \\
\iota_8 &= \text{requests to phpMyAdmin pages} > \mathit{threshold}_{\text{phpMyAdmin}} \\
\iota_9 &= \text{HTTP } \texttt{POST} \text{ of PHP file} \\
\iota_{10} &= \text{MySQL injection } \textbf{@@version} \text{ HTTP } \texttt{GET} \text{ attempt} \\
\iota_{11} &= \text{any phpMyAdmin PHPIDS alert} \\
\iota_{12} &= \text{any MySQL injection PHPIDS alert} \\
\iota_{13} &= \text{MySQL injection-type query}
\end{aligned}$$

3.5.1 Relationship Between Monitors and Indicators

We can now define a representation for the information produced by monitors. *The monitor-indicator generation relationship* describes how a monitor contributes to the goals of intrusion detection and forensic analysis by generating indicators, corresponding to the relationship mentioned in Section 3.2.

Definition 8. *The* **monitor-indicator generation relationship** *is given by a function* $\alpha : M \to \mathcal{P}(I)$ *such that*

$$\alpha(m) = \{\iota \in I \mid \textit{monitor } m \textit{ can generate } \iota\} \tag{3.1}$$

where $\mathcal{P}(I)$ *represents the power set of* I*.*

We make a few points about the mapping α. First, the mapping is one-to-many from monitors to indicators, and represents the complete set of indicators that can be generated by a given monitor. The mapping is deterministic, in that we assume that if an event occurs, the indicator will always be generated.

In our working example, we can use the SSH logs to identify failed SSH attempts. We can use Snort to observe a number of activities, including SSH initiation attempts, HTTP traffic, IRC traffic, and MySQL queries. We can use the Apache request logs to detect phpMyAdmin attack attempts and some SQL injection attempts. We can use PHPIDS to observe phpMyAdmin attack attempts and SQL injection attempts. Finally, we can use the MySQL query logs to conclusively detect SQL injection attempts. The following equations describe the monitor-indicator generation relationship for our working example:

$$\begin{aligned} \alpha(m_1) &= \{\iota_1, \iota_2\} \\ \alpha(m_2) &= \{\iota_2, \iota_3, \iota_4, \iota_5, \iota_6 \iota_7\} \\ \alpha(m_3) &= \{\iota_3, \iota_8, \iota_9, \iota_{10}\} \end{aligned} \qquad \begin{aligned} \alpha(m_4) &= \{\iota_{11}, \iota_{12}\} \\ \alpha(m_5) &= \{\iota_{13}\} \end{aligned}$$

3.5.2 Evidence Required to Detect Events

Next, we define a representation for *the evidence required to detect an event.* As mentioned in Section 3.1.2.2, this relationship stands in for the actual methods by which intrusion detection algorithms and forensic analysis techniques relate monitor information to attacks, corresponding to the relationship between events and indicators mentioned in Section 3.2. Therefore, it should be general enough to represent many different fusion algorithms.

First, we consider the minimal sets of indicators that must be observed to detect an event. For any given event, ϕ, there may be multiple methods by which the event can be detected. For example, it would be possible to detect an SSH brute-force attack by examining the SSHd log entries for a host with a high number of failed login attempts within a time window or by observing from a Snort alert that a host exceeded a predefined threshold of SSH login attempts within a time window. Each of the indicators on its own would provide enough information to enable detection of the attack. Thus, we define the minimal sets of indicators as follows.

Definition 9. *A* **minimal indicator set** *for an event ϕ is a set of indicators, σ, such that the generation of all indicators in σ is sufficient to detect ϕ.*

Now, we define the evidence required to detect an event.

Definition 10. *The* **evidence required to detect an event** *is given by a function $\beta : \Phi \to \mathcal{P}(\mathcal{P}(I))$, where*

$$\beta(\phi) = \{\sigma \mid \sigma \text{ can be used to detect } \phi\}$$

β is a one-to-many map from events to sets of indicators, where only one of the sets σ needs to be observed for event detection to be possible. As is the case with α, β is deterministic, which is related to the assumption of perfect forensic ability.

In our working example, the following equations describe the evidence required to detect the events of interest:

$$\begin{aligned} \beta(\phi_1) &= \{\{\iota_1\}, \{\iota_2\}\} & \beta(\phi_3) &= \{\{\iota_4\}, \{\iota_5\}, \{\iota_6\}\} \\ \beta(\phi_2) &= \{\{\iota_3\}, \{\iota_8\}, \{\iota_{11}\}\} & \beta(\phi_4) &= \{\{\iota_7, \iota_{10}\}, \{\iota_{12}\}, \{\iota_{13}\}\} \end{aligned}$$

Essentially, for the SSH, phpMyAdmin, and botnet events, each of the indicators related to the event can on its own be used to detect the event. For the SQL injection event, the detection of a successful injection attack requires evaluation of either PHPIDS alerts (ι_{12}) or the MySQL query logs (ι_{13}) for evidence of SQL injection, or the combination of an injection HTTP request (ι_{10}) and its subsequent response (ι_7).

Our model of event-to-indicator mappings supports a wide range of attack definitions and fusion algorithms. Most directly, our approach supports signature-based approaches, such as LAMBDA [44]. In such approaches, an alert is generated simply if a set of signatures is satisfied. Under our model, we could represent the components of the signatures as indicators from each monitor, and the signature itself as a minimal indicator set.

One limitation of our approach to mapping events to indicators is that we do not consider probabilistic relationships, which means we cannot accurately capture probabilistic fusion techniques. For example, Almgren et al. use Bayesian networks to detect network security violations [51] by constructing a graph between events and observations in the system and assigning probabilities to the edges that represent the likelihood of an observation given an event's occurrence. At any point, the authors assume that the most probable event, determined using Bayesian inference over the actual observations in the system, is actually taking place. In such a model, the occurrence of many events is inferred from the same set of indicators, so a straightforward application of our approach of using minimal indicator sets would require that all of the indicators in the network be correlated to all of the events, resulting in a relatively uninformative model. We leave consideration of probabilistic fusion and forensic analysis approaches to future work.

3.6 Detectability

For the purpose of representing intrusion detection goals within our model, we must define the conditions within our model by which we consider a monitor deployment capable of supporting detection of an event.

Definition 11. *An event, ϕ, is* **detectable** *given a monitor deployment if it is possible to observe at least one of the minimal indicator sets in $\beta(\phi)$ using the indicators generated by the monitors that are deployed in the system. That is, detectability is a function $\delta : \mathit{\Phi} \times \mathcal{P}(M_d) \to 0, 1$ such that*

$$\delta(\phi, M_d) = 1 \leftrightarrow \exists \sigma \in \beta(\phi) \wedge \sigma \subseteq \bigcup_{m:\, m \in M_d} \alpha(m) \tag{3.2}$$

where M_d is the set of monitors deployed in the system, $M_d \subseteq M$.

By our assumption of perfect forensic ability, this definition of detectability implies that so long as at least one of the minimal indicator sets of ϕ is generated by the monitors in M_d, an intrusion detection system or forensic analyst will always be able to detect ϕ.

That definition of detectability drives the remainder of our work. Our goal is to ensure that the most important events are at least detectable and at best even detectable under compromise or failure. The choice and definitions of the metrics we present in Chapter 4 are based on the qualities of the monitor deployment that would most affect or reflect the ability of the system to support intrusion detection or forensic analysis under compromise or failure.

CHAPTER 4

MONITOR DEPLOYMENT METRICS

As described in Section 3.1.1.1, we recognize that a monitoring system may be faulty, under attack, or compromised. Therefore, we believe intrusion detection and forensic analysis benefit from validation and redundancy, since a redundant monitoring system would allow for intrusion detection even if some of the monitors were compromised or faulty. Our goals in developing monitor deployment metrics are to provide an understanding of what events an intrusion detection system is able to detect in an uncompromised state given a set of monitors, how much redundancy the monitors provide to support intrusion detection, and how compromise of monitors might affect the ability of an intrusion detection system to detect events.

We now define the four metrics we have formulated to encapsulate the capability of monitors to meet intrusion detection goals. We define three utility metrics, namely coverage, redundancy, and confidence, and one cost metric.

4.1 Coverage

We define *coverage* as the overall fraction of the events of interest that are detectable given a monitor deployment.

Definition 12. Coverage *is formally defined as follows:*

$$\textbf{Coverage}\left(\varPhi, M_d\right) = \frac{\left|\left\{\phi \mid \delta\left(\phi, M_d\right) \wedge \phi \in \varPhi\right\}\right|}{\left|\varPhi\right|} \tag{4.1}$$

where M_d is the set of monitors deployed, and $\varPhi$ is the set of events of interest to detect in the system.

Coverage provides a measure of how much of what a system administrator wants to detect can be detected using a set of monitors. If a small set of events must all be detected, it is possible to require 100% coverage of those events. If detection requirements are more relaxed and best-effort detection is acceptable, maximizing coverage can maximize the ability of an intrusion detection or forensic analysis system to detect a wide range of events when the system is not compromised.

It is important to note that we do not consider the "coverage" of system assets with our metrics (i.e., how many of the assets are being protected). The reason is that we do not believe that a metric that measures how many assets are protected necessarily provides valuable information about the security of the system. Ultimately, for an intrusion detection system to be considered effective, it must be able to detect as many different *intrusions* as possible and be able to detect high-impact intrusions with high probability. Furthermore, we believe that by carefully defining the set of events to detect, it is possible to ensure that the coverage metric we define above captures the protection of the assets. We assume that the security administrator generating the set of important events understands what parts of the system must be most protected, and can define the events to adequately cover those parts. Thus, we define our coverage metric from the perspective of event detection.

We now illustrate the use of the coverage metric using our working example. Let the set of events of interest be $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$. The coverage of Φ changes as the set of monitors deployed changes. For example, when SSHd logging and PHPIDS are deployed, it is possible to detect ϕ_1, ϕ_2, and ϕ_4, so $\textbf{Coverage}\left(\Phi, \{m_1, m_4\}\right) = 0.75$. However, when Snort and Apache request logging are deployed, it is possible to detect all of the events, so $\textbf{Coverage}\left(\Phi, \{m_2, m_4\}\right) = 1$.

4.2 Redundancy

We define *redundancy* for an event, ϕ, as the amount of evidence a set of monitors provides that supports the detection of ϕ.

Redundancy increases the ability to detect intrusions by increasing the number of alerts generated by monitors for a given event. As a corollary, it can also increase the confidence in any alerts generated by monitor data fusion algorithms.

To quantify redundancy, we consider two different measures for redundancy in the context of our data model. At the lowest level, we can consider sets of monitors to provide redundant information if they produce identical indicators. In such a case, the information from either monitor can be used to detect the same intrusions. In terms of detecting events, we can consider sets of monitors redundant if they provide different ways to detect the same event. In our model, redundancy of sets of monitors is represented as generation of the indicators for different minimal indicator sets.

To determine how many different ways we can detect the same event, we must determine the set of minimal indicator sets for an event that are detectable given a set of monitors.

Definition 13. *The set of* **detectable minimal indicator sets***, ς, for event ϕ, is defined as follows:*

$$\varsigma\left(\phi, M_d\right)=\left\{\sigma \mid \sigma \in \beta\left(\phi\right)\ \wedge\ \delta\left(\sigma, M_d\right)\right\}$$

Definition 14. Redundancy *is a composition of the two measures of redundancy mentioned above. Let M_d and ϕ be defined as above. Then, formally,*

$$\mathbf{Redundancy}\left(\phi, M_d\right)=\sum_{\sigma \in \varsigma\left(\phi, M_d\right)} \min_{\iota \in \sigma}\left|\left\{m \mid m \in M_d,\ \iota \in \alpha\left(m\right)\right\}\right| \tag{4.2}$$

In other words, redundancy for ϕ is the total number of different ways that the minimal indicator sets for ϕ can be detected using the monitors in M_d, where a minimal indicator set σ can be detected in k ways if each indicator $\iota \in \sigma$ is generated by k different monitors in M_d.

Redundancy is complementary to coverage, as one might attempt to maximize the redundancy for critical events and attempt to ensure at least a minimal threshold of redundancy for other high importance events, using coverage as a catchall to maximize the overall detection of events in the system.

For example, let us examine the same two sets of monitor deployments that we examined for coverage. When SSHd logs and PHPIDS are deployed, an SSH brute-force attack can be detected using either ι_1 or ι_2, so $\mathbf{Redundancy}\left(\phi_1, \{m_1, m_4\}\right) = 2$. Similarly, PHPIDS allows us to detect each of ϕ_2 and ϕ_4 in one way, so $\mathbf{Redundancy}\left(\phi_2, \{m_1, m_4\}\right) = \mathbf{Redundancy}\left(\phi_4, \{m_1, m_4\}\right) = 1$. Since ϕ_3 is not detectable using m_1 and m_4, $\mathbf{Redundancy}\left(\phi_1, \{m_1, m_4\}\right) = 0$.

When the set of monitors deployed is changed to Snort and Apache request logs, the redundancy for ϕ_1 decreases to 1, because only ι_2 can be used to detect the event. However, ϕ_2 can be detected using ι_3 as generated by either m_2 or m_3, or using ι_8, as generated by m_3. Thus, $\mathbf{Redundancy}\left(\phi_2, \{m_2, m_3\}\right) = 3$. Similarly, $\mathbf{Redundancy}\left(\phi_3, \{m_2, m_3\}\right) = 3$ and $\mathbf{Redundancy}\left(\phi_4, \{m_2, m_3\}\right) = 1$.

4.3 Confidence

We define *confidence* as the belief in the ability to detect an event using a set of monitors given that monitors may be compromised or faulty. This metric is closely related to the reliability or truthfulness of monitors and captures the expected reliability of the overall monitoring system in detecting intrusions.

To reason about the overall confidence in the ability of a deployment of monitors to support detection of events, we must first assess the truthfulness of the monitors.

Definition 15. *The* **truthfulness** *of a monitor is the belief that the indicators produced by the monitor are correct.*

In assessing the truthfulness of a monitor, we apply fuzzy logic, observing that the truthfulness of a monitor is not binary; that is, a monitor might not transition directly from producing only truthful indicators to producing false indicators upon compromise, but may instead continue to produce correct indicators for all but a handful of events. Monitors are not probabilistically truthful, but *variably* truthful. As argued by Hosmer in [52], fuzzy logic lends itself better to such situations than probability theory does. Therefore, we define a function γ_M that maps monitors to their truthfulness in generating indicators as a fuzzy logic degree of truth that represents *how many* (as a proportion of the aggregate) of the indicators generated by a monitor are truthful.

$$\gamma_M : M \to \{n \in \mathbb{R} \mid 0 \leq n \leq 1\}$$

We project the truthfulness of a monitor to the truthfulness of all indicators produced by the monitor. Given a set of monitors M_d, we consider the overall truthfulness γ_I of an indicator to be the disjunction in monoidal-t-norm-based logic (MTL) of the truthfulness values of the indicator for all monitors that generate the indicator. If no monitors generate the indicator, its truthfulness is 0 by default.

$$\begin{aligned}\gamma_I(\iota, M_d) &= \begin{cases} \displaystyle\bigvee_{m \in M_d \mid \iota \in \alpha(m)} \gamma_M(m), & \delta(\phi, M_d) \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} \displaystyle\max_{m \in M_d \mid \iota \in \alpha(m)} \gamma_M(m), & \delta(\phi, M_d) \\ 0, & \text{otherwise} \end{cases}\end{aligned}$$

In order to detect an event, we need to be able to detect all of the indicators from at least one of the minimal indicator sets for the event. Therefore, for a minimal indicator set σ, we can define the confidence in detecting σ as the MTL conjunction over the truthfulness values for all indicators in σ. Finally, we can define the confidence in detecting event ϕ to be the MTL disjunction of the confidence values for all of the minimal indicator sets for ϕ.

Put simply, the confidence metric is a transformation of the Boolean logic tree representing what combinations of indicators are required to detect an event into a fuzzy logic equation.

Definition 16. Confidence *is formally defined as follows:*

$$\textbf{Confidence}\left(\phi, M_d\right) = \bigvee_{\sigma \in \beta(\phi)} \bigwedge_{\iota \in \sigma} \gamma_I\left(\iota, M_d\right) \tag{4.3}$$

$$= \max_{\sigma \in \beta(\phi)} \min_{\iota \in \sigma} \gamma_I\left(\iota, M_d\right) \tag{4.4}$$

Within our working example, let us assume that SSHd logging and Snort are more susceptible to compromise because they reside on the management VM (Server 1); that the Apache request log and PHPIDS are moderately susceptible; and that the MySQL query log is relatively insusceptible. Let those assumptions be represented as $\gamma_M\left(m_1\right) = \gamma_M\left(m_2\right) = 0.3$, $\gamma_M\left(m_3\right) = \gamma_M\left(m_4\right) = 0.5$, and $\gamma_M\left(m_5\right) = 0.9$.

Now, if we deploy SSHd logging and Snort, we can detect ϕ_1, ϕ_2, and ϕ_3. Since all monitors in this deployment have a γ_M value of 0.3, all of the indicators generated also have a γ_I value of 0.3, so $\textbf{Confidence}\left(\phi_i, \{m_1, m_2\}\right) = 0.3$ for all $1 \leq i \leq 3$. If we instead deploy Snort and Apache request logs, $\gamma_I\left(\iota_3\right)$ increases to 0.5, because ι_3 can be generated by m_3 as well as by m_2. Thus, $\textbf{Confidence}\left(\phi_2, \{m_2, m_3\}\right)$ increases to 0.5. Since detection of ϕ_4 requires the generation of both ι_7 from m_2 and ι_{10} from m_3, the confidence in detecting the set ι_7, ι_{10} is the minimum of $\gamma_I\left(\iota_7\right) = 0.3$ and $\gamma_I\left(\iota_{10}\right) = 0.5$. Thus, $\textbf{Confidence}\left(\phi_2, \{m_2, m_3\}\right) = 0.3$.

4.4 Cost

We define *cost* as the overall value of the resources consumed by monitors that are deployed in a system. Our cost metric captures the cost (for our purposes, monetary cost) of the deployment, operation, and management of monitors and collection and storage of the data generated by monitors.

Our cost model is composed of two types of costs: resource utilization costs and purchase and management costs. Resource utilization costs represent the operational costs of running a monitor on a system asset per unit of time. We assume that the modeler already has pricing information for the different compute and storage resources used by monitors. For example, for systems such as public clouds, prices could be determined based on the public pricing model for compute resources used to bill customers. For other systems, prices could represent opportunity costs of using the resources for monitoring instead of for other business purposes, or data center operating costs amortized per unit of resource utilization.

Purchase and management costs represent the fixed cost of purchasing and deploying a monitor on the system and the recurring costs of maintaining and managing the monitors during operation of the system. Because we consider cost per unit time, the fixed purchase cost must be amortized over the expected lifetime of the monitor. The recurring management cost primarily represents the human cost of managing the monitoring infrastructure; we assume that the computing costs of the management software are negligible.

Definition 17. *The* **amortized purchase and management costs** *are defined by a function $P : M \to \mathbb{R}$, where $P(m)$ is the fixed and variable costs of monitor purchase, deployment, and management described above for monitor m, amortized over the lifetime of the monitor, into a cost per unit time.*

In order to calculate the overall resource cost of a set of monitors, we first define the costs of using resources on assets. Within our model, monitors consume the following resources on system assets (units of utilization are in parentheses):

- CPU utilization (in CPU cores)
- Memory utilization (in GB)
- Disk storage (in GB)
- Network communication (in GB per hour)

Definition 18. *For each asset in the system, the* **asset resource costs** *represent the cost per unit of utilization per unit time for each of the four resources listed above. That is, the resource costs for an asset $v \in V_S$ are defined by a function $ResourceCost : V_S \to \mathbb{R}^4$, where*

$$\mathbf{ResourceCost}(v) = \begin{bmatrix} \mathbf{CPUCost}(v) \\ \mathbf{MemoryCost}(v) \\ \mathbf{DiskCost}(v) \\ \mathbf{NetworkCost}(v) \end{bmatrix} \tag{4.5}$$

The costs may be dynamic; that is, the value of the resources being used by monitoring may change depending on system load, and thereby change the cost of deployment of a monitor. Our model does not disallow that, but for the sake of computation, we fix the costs at the time of evaluation of the metrics.

For assets at multiple levels, the user the options of either propagating resource costs for the lowest-level asset (one that is not dependent on any other assets) upwards to its dependent, higher-level assets, or assigning different resource costs based on additional considerations, such as application-specific costs.

Next, we define the resource utilization of the monitors. As stated earlier, each monitor consumes resources on the asset on which it is deployed, with the exception of storage.

Definition 19. *For each monitor, we define the* **long-term average monitor resource utilization** *for a monitor m as the expected amount of resources utilized by the monitor during a unit of time. It is given by a function* $\mathbf{ResourceUtilization} : M \to \mathbb{R}^4$, *where*

$$\mathbf{ResourceUtilization}\left(m\right) = \begin{bmatrix} \mathbf{CPUUtilization}\left(m\right) \\ \mathbf{MemoryUtilization}\left(m\right) \\ \mathbf{DiskUtilization}\left(m\right) \\ \mathbf{NetworkUtilization}\left(m\right) \end{bmatrix} \tag{4.6}$$

As it is possible for a monitor to log remotely and thus consume storage on a different asset, we define the relation $\mathbf{LogsTo} : M \to V_S$ to define where a monitor stores its logs.

Definition 20. *The* **cost** *of a set of monitors is the product sum of the monitor resource utilization with the asset resource costs for the asset on which the monitor is deployed. That is,*

$$\begin{aligned} \mathbf{Cost}\left(M_d\right) = \sum_{m \in M_d} (P\left(m\right) &+ \mathbf{CPUCost}\left(v\right) \mathbf{CPUUtilization}\left(m\right) \\ &+ \mathbf{MemoryCost}\left(v\right) \mathbf{MemoryUtilization}\left(m\right) \\ &+ \mathbf{NetworkCost}\left(v\right) \mathbf{NetworkUtilization}\left(m\right) \\ &+ \mathbf{DiskCost}\left(\mathbf{LogsTo}\left(m\right)\right) \mathbf{DiskUtilization}\left(m\right)) \end{aligned} \tag{4.7}$$

where $(m, v) \in E_M$. As cost is an expected rate of expenditure for the set of monitors, the units of cost are currency units per unit time.

For simplicity's sake, within our working example, let us set the resource costs for all servers to be the same, and let the resource costs for each of the applications be the same as those for the servers. That is, we let $\mathbf{ResourceCost}\left(v_i\right) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ for all i. Let us also set $\mathbf{ResourceUtil}\left(m_i\right) = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}^T$ for $i \in \{1, 3, 4, 5\}$ and $\mathbf{ResourceUtil}\left(m_2\right) = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix}^T$, as Snort requires that packets be transmitted to Server 1 for processing, but all other monitors log locally. In the example, the amortized purchase and management costs for each monitors is 0.50.

Given those parameters, we can calculate the cost for each monitor independently. Since all assets share the same resource cost, $\mathbf{Cost}\left(\{m_i\}\right) = 0.50 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = \$3.50$

for $i \in \{1, 3, 4, 5\}$, and $\textbf{Cost}(\{m_2\}) = 0.50 + 1 \cdot 2 + 1 \cdot 2 + 1 \cdot 2 + 1 \cdot 2 = \$8.50$. Monitor costs are additive, so the cost of a monitor deployment is simply the sum of the costs of all deployed monitors.

In this chapter, we define a set of quantitative metrics for monitor deployment that characterize the utility provided by the monitors to support intrusion detection and the cost of deploying the monitors. In order to use these metrics to perform optimal monitor deployment, we must define a methodology by which a practitioner can specify the requirements of monitor deployment in terms of values of the metrics and then find the best deployment of monitors that meets the requirements. We introduce such a methodology in the next chapter.

CHAPTER 5

OPTIMAL MONITOR DEPLOYMENT

Our goal is to provide a methodology to deploy monitors effectively in a system. A practitioner should be able to specify intrusion detection requirements using our methodology and be able to determine the optimal placement of monitors for the intrusion detection requirements.

5.1 Intrusion Detection Requirements

We define intrusion detection requirements in terms of the events the practitioner wishes to detect and target values of the metrics defined in Chapter 4. To support that, we allow a practitioner to specify *weights and constraints* on the values of the monitoring utility metrics for the optimal monitor deployment algorithm. That is, we define the following constants:

$$\mathbf{w}_{\text{Coverage}} \in [0,1] : \text{The weight for the coverage metric for the set of events } \mathit{\Phi} \tag{5.1}$$

$$\mathbf{w}_{\text{Redundancy}_\phi}, \mathbf{w}_{\text{Confidence}_\phi} \in [0,1] \ \ \forall \phi \in \mathit{\Phi} : \text{The weights for the redundancy and confidence metrics for each of the events in } \mathit{\Phi} \tag{5.2}$$

$$\mathbf{min}_{\text{Coverage}} \in [0,1] : \text{The minimum value for the coverage metric for the set of events } \mathit{\Phi} \tag{5.3}$$

$$\mathbf{min}_{\text{Redundancy}_\phi} \in \mathbb{Z} \ \forall \phi \in \mathit{\Phi} : \text{The minimum values for the redundancy metric for each of the events in } \mathit{\Phi} \tag{5.4}$$

$$\mathbf{min}_{\text{Confidence}_\phi} \in [0,1] \ \ \forall \phi \in \mathit{\Phi} : \text{The minimum values for the confidence metric for each of the events in } \mathit{\Phi} \tag{5.5}$$

The weights and constraints can be used to specify the importance of each of the events and each of the metrics in overall deployment. For example, if $\mathit{\Phi}$ contains two events, ϕ_1 and ϕ_2, and a practitioner wants to be able to detect ϕ_1 under attack and wants to be able to detect ϕ_2 in as many ways as possible, the practitioner could set $\mathbf{w}_{\text{Confidence}_{\phi_1}}$ and

$\mathbf{w}_{\text{Redundancy}_{\boldsymbol{\phi_2}}}$ to large values and all other weights to small or zero values. Then, the utility of a monitor deployment would depend most heavily on the confidence in detecting ϕ_1 and the number of redundant ways to detect ϕ_2, so the optimal monitor deployment chosen would prioritize these two metrics' values.

Given weights for each of the metrics' values, we define intrusion detection requirements as conditions on detectability of the events of interest. The following are three types of requirements that practitioners may have that we support in our optimal monitor deployment methodology:

- A practitioner may desire *best effort detection* of an event ϕ, which means that ϕ does not necessarily need to be detectable by the optimal monitor deployment, but detectability of the event is preferred to lack of detectability. That might be the case for most noncritical events that, while important, may not warrant extensive, expensive monitoring. Best-effort detection may be specified in terms of any of the metrics (i.e., maximum coverage, maximum redundancy, or maximum confidence) and can be represented by nonzero weights $\mathbf{w}_{\text{Coverage}}$, $\mathbf{w}_{\text{Redundancy}_\phi}$, and $\mathbf{w}_{\text{Confidence}_\phi}$.

- A practitioner may desire *mandatory detection* of an event ϕ, which means that ϕ must be detectable by whatever set of monitors is determined by the optimal deployment algorithm. That might be required for important events that would cause financial loss or breach of sensitive data. Mandatory detection can be represented by requiring $\text{min}_{\text{Redundancy}_\phi} = 1$. If the entire set of events must be detected, that can be represented by requiring $\mathbf{min}_{\text{Coverage}} = 1$.

- A practitioner may desire *redundant detection* of an event ϕ, which means that ϕ should be detectable in more than one way. That might be required for critical events that would cause catastrophic system failure, major financial loss, or legal ramifications. If a specific redundancy level k is desired, redundant detection can be represented by requiring $\text{min}_{\text{Redundancy}_\phi} = k$. Otherwise, opportunistic redundant detection can be represented by a high nonzero weight $\mathbf{w}_{\text{Redundancy}_\phi}$.

Additionally, practitioners are often bound by business constraints on system security. We represent such constraints using the constant **maxCost**, an upper limit on the value of the cost metric. If the budget for monitoring is restricted, a practitioner could represent that within our methodology by setting the value of **maxCost**.

5.2 Constrained-Cost Optimal Deployment

To solve for optimal monitor deployment under a fixed cost constraint, we must be able to optimize over the monitoring utility metrics by determining the set of monitors that provides the maximum utility while satisfying the constraints specified by the intrusion detection requirements. Given the types of inequality constraints we have defined above as intrusion detection requirements, mathematical programming lends itself well to the optimization problem.

In our model, we assume that monitors can either be deployed or not deployed. Therefore, their deployment either contributes to the values of the metrics (i.e., the deployment takes a value of 1) or does not (i.e., the deployment takes a value of 0). Therefore, we represent our optimal deployment program as a 0-1 integer program with inequality constraints. The program is given by Equation 5.6. Unless otherwise specified, by default, the **min** and **w** constants are set to 0, and **maxCost** is set to ∞.

Definition 21. *We define the* **cost-constrained monitor deployment program** *as the following integer program:*

$$\begin{array}{rl}
\text{given} & S=(V,E)\,,\ \mathit{\Phi},\ I,\ \alpha,\ \beta,\ \gamma_M \\
& P,\ \mathbf{ResourceCost},\ \mathbf{ResourceUtil},\ \mathbf{LogsTo} \\
& \mathbf{w}_{\text{Coverage}},\mathbf{min}_{\text{Coverage}} \\
& \mathbf{w}_{\text{Redundancy}_\phi},\mathbf{min}_{\text{Redundancy}_\phi} \quad \forall\phi\in\mathit{\Phi} \\
& \mathbf{w}_{\text{Confidence}_\phi},\mathbf{min}_{\text{Confidence}_\phi} \quad \forall\phi\in\mathit{\Phi} \\
& \mathbf{maxCost} \\
\underset{M_d}{\text{maximize}} & \mathbf{w}_{\text{Coverage}}\mathbf{Coverage}\left(\mathit{\Phi},M_d\right)+ \\
& \sum_{\phi\in\mathit{\Phi}}\Big(\mathbf{w}_{\text{Redundancy}_\phi}\mathbf{Redundancy}\left(\phi,M_d\right)+ \\
& \qquad \mathbf{w}_{\text{Confidence}_\phi}\mathbf{Confidence}\left(\phi,M_d\right)\Big) \\
\text{subject to} & \mathbf{Cost}(M_d)\le\text{maxCost} \\
& \mathbf{Coverage}\left(\mathit{\Phi},M_d\right)\ge\mathbf{min}_{\text{Coverage}} \\
& \mathbf{Redundancy}\left(\phi,M_d\right)\ge\mathbf{min}_{\text{Redundancy}_\phi} \quad \forall\phi\in\mathit{\Phi} \\
& \mathbf{Confidence}\left(\phi,M_d\right)\ge\mathbf{min}_{\text{Confidence}_\phi} \quad \forall\phi\in\mathit{\Phi} \\
& M_d\in\{0,1\}^{|M|}
\end{array} \tag{5.6}$$

The cost-constrained optimal monitor deployment program defined by Equation 5.6 takes as input the system and data models, metric value weights, minimum value constraints on the metrics, and cost constraints. The program searches over the space of all possible deployments of monitors. All monitors in the deployments are either enabled or disabled. The objective function of the program maximizes the utility of the monitors in detecting intrusions, where utility is defined as a weighted sum of all the metric values. The program is subject to a maximum cost constraint specified by the practitioner and minimum value constraints on the utility metric values, which ensure that the monitor deployment obtained from the solution to the program abides by the minimum values of the metrics that are specified. Rather than find the lowest-possible-cost monitoring (which, logically speaking, would be deployment of no monitors at all), the program will find the highest-utility monitor deployment whose monitoring cost is lower than the maximum cost specified.

The program can be solved using the branch and bound algorithm to find an optimal monitor deployment for the detection requirements described in Section 5.1. A practitioner could use the metric value weights and minimum value constraints defined in Equations (5.1) to (5.5) to define a utility function that captures the requirements.

The program is nonlinear in terms of the monitor deployment vector, M_d, because the objective function is a weighted sum of the utility metrics' values, which are themselves nonlinear. We now prove nonlinearity for each of the utility metrics.

Theorem 1. *The coverage metric defined by Equation 4.1 is nonlinear in terms of monitor deployment, M_d. That is, $\exists$ $\mathit{\Phi}$, α, β, M_d such that*
$\mathbf{Coverage}\left(\mathit{\Phi}, M_d\right) \neq \sum_{m \in M_d} \mathbf{Coverage}\left(\mathit{\Phi}, \{m\}\right)$.

Proof. Assume by way of contradiction that the coverage metric is linear. Then, by definition, for any values of $\mathit{\Phi}$, α, and β, for two disjoint sets of monitors M_1 and M_2,

$$\mathbf{Coverage}\left(\mathit{\Phi}, M_1 \cup M_2\right) = \mathbf{Coverage}\left(\mathit{\Phi}, M_1\right) + \mathbf{Coverage}\left(\mathit{\Phi}, M_2\right)$$

Let $M = \{m_1, m_2\}$, where m_1 and m_2 are unique monitors that produce identical indicators. Specifically, let $\alpha(m_1) = \alpha(m_2) = \{\iota_1\}$. Further, let $\mathit{\Phi} = \{\phi_1\}$, and let $\beta(\phi_1) = \{\{\iota_1\}\}$. Let $M_1 = \{m_1\}$ and let $M_2 = \{m_2\}$. By construction, $M_1 \cap M_2 = \varnothing$. Then, since m_1 and m_2 produce ι_1,

$$\mathbf{Coverage}\left(\mathit{\Phi}, M_1\right) = 1$$
$$\mathbf{Coverage}\left(\mathit{\Phi}, M_2\right) = 1$$

Since both monitors independently cover $\mathit{\Phi}$, both monitors together will also cover $\mathit{\Phi}$. In other words,

$$\begin{aligned}\textbf{Coverage}\left(\mathit{\Phi}, M_1 \cup M_2\right) &= 1 \\ &\neq \textbf{Coverage}\left(\mathit{\Phi}, M_1\right) + \textbf{Coverage}\left(\mathit{\Phi}, M_2\right)\end{aligned}$$

That is a contradiction! Thus, the coverage metric is not linear. □

Theorem 2. *The redundancy metric defined by Equation 4.2 is nonlinear in terms of monitor deployment, M_d. That is, $\exists\ \phi,\ \alpha,\ \beta,\ M_d$ such that* $\textbf{Redundancy}\left(\phi, M_d\right) \neq \sum_{m \in M_d} \textbf{Redundancy}\left(\phi, \{m\}\right)$.

Proof. Assume by way of contradiction that the redundancy metric is linear. Then, by definition, for any values of ϕ, α, and β, for two disjoint sets of monitors M_1 and M_2,

$$\textbf{Redundancy}\left(\phi, M_1 \cup M_2\right) = \textbf{Redundancy}\left(\phi, M_1\right) + \textbf{Redundancy}\left(\phi, M_2\right)$$

Let $M = \{m_1, m_2\}$, and let $\alpha(m_1) = \{\iota_1\}$ and $\alpha(m_2) = \{\iota_2\}$. Further, let $\mathit{\Phi} = \{\phi_1\}$, and let $\beta(\phi_1) = \{\{\iota_1, \iota_2\}\}$. Let $M_1 = \{m_1\}$ and let $M_2 = \{m_2\}$. By construction, $M_1 \cap M_2 = \varnothing$. Then,

$$\begin{aligned}\textbf{Redundancy}\left(\phi_1, M_1\right) &= 0 \\ \textbf{Redundancy}\left(\phi_1, M_2\right) &= 0\end{aligned}$$

Alone, neither monitor allows for detection of $\mathit{\Phi}$. However, deploying both monitors together does cover ϕ_1. In other words,

$$\begin{aligned}\textbf{Redundancy}\left(\phi_1, M_1 \cup M_2\right) &= 1 \\ &\neq \textbf{Redundancy}\left(\phi_1, M_1\right) + \textbf{Redundancy}\left(\phi_1, M_2\right)\end{aligned}$$

That is a contradiction! Thus, the redundancy metric is not linear. □

Theorem 3. *The confidence metric defined by Equation 4.3 is nonlinear in terms of monitor deployment, M_d. That is, $\exists\ \phi,\ \alpha,\ \beta,\ M_d$ such that* $\textbf{Confidence}\left(\phi, M_d\right) \neq \sum_{m \in M_d} \textbf{Confidence}\left(\phi, \{m\}\right)$.

Proof. Assume by way of contradiction that the confidence metric is linear. Then, for any values of S, ϕ, α, β, and γ_M, for two disjoint sets of monitors M_1 and M_2,

$$\textbf{Confidence}\left(\phi, M_1 \cup M_2\right) = \textbf{Confidence}\left(\phi, M_1\right) + \textbf{Confidence}\left(\mathit{\Phi}, M_2\right)$$

Let $M = \{m_1, m_2\}$, where m_1 and m_2 are unique monitors that produce identical indicators. Specifically, let $\alpha(m_1) = \alpha(m_2) = \{\iota_1\}$. Further, let $\varPhi = \{\phi_1\}$, and let $\beta(\phi_1) = \{\{\iota_1\}\}$. Let $M_1 = \{m_1\}$ and let $M_2 = \{m_2\}$. By construction, $M_1 \cap M_2 = \varnothing$.

Additionally, let $\gamma_M(m_1) = \gamma_M(m_2) = 0.5$. As a result, $\gamma_I(\iota_1, M_1) = \gamma_I(\iota_1, M_2) = \gamma_I(\iota_1, M_1 \cup M_2) = 0.5$. Then,

$$\begin{aligned}\textbf{Confidence}\,(\phi_1, M_1) &= 0.5\\ \textbf{Confidence}\,(\phi_1, M_2) &= 0.5\end{aligned}$$

However, since $\gamma_I(\iota_1, M_1 \cup M_2) = 0.5$,

$$\begin{aligned}\textbf{Confidence}\,(\phi_1, M_1 \cup M_2) &= 0.5\\ &\neq \textbf{Confidence}\,(\phi_1, M_1) + \textbf{Confidence}\,(\phi_1, M_2)\end{aligned}$$

That is a contradiction! Thus, the confidence metric is not linear. □

5.3 Unconstrained-Cost Optimal Deployment

The maximum utility program given by Equation 5.6 assumes optimization based on an upper bound on cost. In some cases, such as those where monitor cost is negligible or security is of the highest importance, it may instead be desirable to *minimize* the cost of monitoring while meeting hard intrusion detection requirements. In such cases, the nonlinear program specified by Equation 5.6 cannot solve for an optimal deployment. Instead, we must create an alternative formulation of the program in which the objective function is minimization of the cost metric.

When optimizing the value of the cost metric, the monitoring utility metric weights defined in Equations 5.1 and 5.2 no longer hold any significance, as they are not an element of the utility function of the optimization equation. Without minimality restrictions on the values of the utility metrics, the program will always return the trivial zero-cost solution of deploying no monitors. Therefore, we define the integer program solely in terms of the minimality requirements on the metric values. The program is given by Equation 5.7.

Definition 22. *We define the* **minimum cost monitor deployment program** *as the following integer program:*

$$\begin{array}{rl} \text{given} & S=(V,E)\,,\ \varPhi,\ I,\ \alpha,\ \beta,\ \gamma_M \\ & P,\ \mathbf{ResourceCost},\ \mathbf{ResourceUtil},\ \mathbf{LogsTo} \\ & \mathbf{min}_{\text{Coverage}} \\ & \mathbf{min}_{\text{Redundancy}_\phi}, \mathbf{min}_{\text{Confidence}_\phi} \quad \forall \phi \in \varPhi \\ \underset{M_d}{\text{minimize}} & \mathbf{Cost}\left(M_d\right) \\ \text{subject to} & \begin{array}{rl} \mathbf{Coverage}\left(\varPhi, M_d\right) \geq \mathbf{min}_{\text{Coverage}} & \\ \mathbf{Redundancy}\left(\phi, M_d\right) \geq \mathbf{min}_{\text{Redundancy}_\phi} & \quad \forall \phi \in \varPhi \\ \mathbf{Confidence}\left(\phi, M_d\right) \geq \mathbf{min}_{\text{Confidence}_\phi} & \quad \forall \phi \in \varPhi \\ M_d \in \{0,1\}^{|M|} & \end{array} \end{array} \tag{5.7}$$

The unconstrained-cost optimal monitor deployment program defined by Equation 5.7 takes as input just the system and data models and minimum value constraints on the metrics. The program searches over the space of all possible deployments of monitors for a monitor deployment subject to minimum value constraints on the utility metric values. The minimum value constraints ensure that the monitor deployment obtained from the solution to the program meets some minimum required level of monitoring. The objective function of the program minimizes the cost of the monitors.

The program defined by Equation 5.7 can be used to solve for an optimal monitor deployment for the mandatory and redundant detection requirements described in Section 5.1. A practitioner could use the minimum value constraints defined in Equations 5.1 and 5.2 to define a utility function that captures the requirements.

5.4 Application of Optimal Deployment Programs

We foresee both offline and online use of our optimal monitor deployment methodology. In an offline setting, a practitioner could use our methodology to determine which monitors to deploy before operation of the system to increase the efficacy of a forensic analyst or intrusion detection algorithm during operation. In an online setting, a forensic analyst or intrusion detection algorithm could use observed system state to update the system model and rerun the optimal deployment programs to determine how to redeploy monitors to continue meeting intrusion detection requirements. In this thesis, we focus on the offline analysis and deployment. Understanding how to use observations from monitors to update system state is outside the scope of this work, so we present our ideas on online use of our methodology in Section 7.2.

Using our optimal monitor deployment programs, a practitioner can determine what set of monitors would facilitate forensic analysis or intrusion detection. The practitioner could define intrusion detection requirements in terms of values of the weights and constraints on the monitoring metrics, and then use the appropriate optimal deployment program to determine which monitors to deploy. We now illustrate the use of our optimal monitor deployment methodology using our working example. Let us consider two cases.

Case 1: In the first case, our sole concern is with maximizing coverage of all events with a maximum cost of \$11. Since there is a maximum cost constraint, we use Equation 5.6.

Since we have no minimum detection requirements, we set $\mathbf{min}_{\text{Coverage}} = \mathbf{min}_{\text{Redundancy}_\phi} = \mathbf{min}_{\text{Confidence}_\phi} = 0$ for all $\phi \in \varPhi$. Furthermore, since we care only about maximizing coverage, we also set $\mathbf{w}_{\text{Coverage}} = 1$ and $\mathbf{w}_{\text{Redundancy}_\phi} = \mathbf{w}_{\text{Confidence}_\phi} = 0$ for all $\phi \in \varPhi$. To account for the cost constraint, we set $\mathbf{maxCost} = \$11$. Given those constraints and the model values we define in previous chapters, solving Equation 5.6 yields the deployment $M_d = \{m_1, m_4\}$, where m_1 is the SSHd log and m_4 is PHPIDS. Notice that $\mathbf{Coverage}\,(\varPhi, M_d) = 0.75$ and $\mathbf{Cost}\,(M_d) = \$7$. Because of the cost constraint, it is not possible to detect all of the events, but our approach yields a deployment that minimizes cost while yielding maximum coverage.

Case 2: In the second case, we wish to guarantee detection of the botnet (ϕ_3) and SQL injection (ϕ_4) attacks and maximize the redundancy and confidence for those two attacks, performing best-effort detection for the ϕ_1 and ϕ_2. Furthermore, the monitors must have a maximum cost of \$16.

Again, since we aim to maximize detection utility with a maximum cost constraint, we use Equation 5.6. Here, we set $\mathbf{min}_{\text{Redundancy}_{\phi_3}} = \mathbf{min}_{\text{Redundancy}_{\phi_4}} = 1$ to account for the detection guarantees, and set $\mathbf{w}_{\text{Redundancy}_{\phi_3}} = \mathbf{w}_{\text{Redundancy}_{\phi_4}} = 1$ to maximize redundancy and $\mathbf{w}_{\text{Confidence}_{\phi_3}} = \mathbf{w}_{\text{Confidence}_{\phi_4}} = 1$ to maximize confidence. To capture best-effort detection guarantees for ϕ_1 and ϕ_2, we can set $\mathbf{w}_{\text{Coverage}} = 2$. To account for the cost constraint, we set $\mathbf{maxCost} = \$16$. We set all other parameters to 0. Solving Equation 5.6 yields the deployment $M_d = \{m_2, m_3, m_5\}$, where m_2 is Snort, m_3 is the Apache request log, and m_5 is the MySQL query log. The monitor deployment M_d yields $\mathbf{Redundancy}\,(\phi_3, M_d) = 3$, $\mathbf{Redundancy}\,(\phi_4, M_d) = 2$, $\mathbf{Confidence}\,(\phi_3, M_d) = 0.3$, $\mathbf{Confidence}\,(\phi_4, M_d) = 0.9$, and $\mathbf{Cost}\,(M_d) = \$15.50$. Note that without the confidence constraints, the optimal deployment could contain m_4 instead of m_5, and with maxCost = \$15, the optimal deployment would instead be $M_d = \{m_2, m_5\}$.

These two cases illustrate the expressiveness of our methodology. By varying the weights and constraints that define intrusion detection goals, a practitioner can phrase a multitude of questions about monitor deployment and study the effect of deploying monitors on intrusion detection ability.

5.5 Complexity Analysis

We now consider the complexity of our approach in terms of the size of the model.

To compute the objective function and constraint functions for both the constrained-cost and unconstrained-cost optimal monitor deployment programs, it is necessary to compute the values of the following metrics for each of the possible monitor deployments, M_d:

- **Coverage** (Φ, M_d)
- **Redundancy** $(\phi, M_d) \quad \forall \phi \in \Phi$
- **Confidence** $(\phi, M_d) \quad \forall \phi \in \Phi$
- **Cost** (M_d)

There are $2\,|\Phi| + 2$ functions that must be computed for each monitor deployment (M_d) evaluated while solving the program. To determine the overall complexity of our implementation, we first evaluate the complexity of computing each of the metric functions.

5.5.1 Coverage

The algorithm we use to compute coverage is given by Algorithm 1. In the first for loop, we generate the set of all indicators that can be generated by the set of monitors M_d. We call that set $\mathcal{I}$. Generation of $\mathcal{I}$ is done by computing the union of all values of $\alpha(m)$ for all m in M_d. In the worst-case, each value of $\alpha(m)$ will consist of all observable indicators, I, so the complexity of computing the set is $O\left(|M|\,|I|\right)$. Then, we check for each event ϕ in Φ whether any minimal indicator set σ in $\beta(\phi)$ is a subset of $\mathcal{I}$. In the worst-case, that requires B checks, where $B = \sum_{\phi \in \Phi} |\beta(\phi)|$. Checking that a minimal indicator set σ is a subset of $\mathcal{I}$ requires $|\sigma|$ operations, where, in the worst-case, $\sigma = I$. Thus, the worst-case computational complexity of determining the coverage of a set of events Φ is $O\left(|M|\,|I| + B\,|I|\right)$.

5.5.2 Redundancy

The algorithm we use to compute redundancy is given by Algorithm 2. In the first set of nested for loops, we count the number of monitors that can generate each of the indicators.

Algorithm 1 Algorithm to compute the value of the coverage metric

```
function Coverage($\Phi$, $M_d$)
    $\mathcal{I} \leftarrow \varnothing$
    for all $m$ in $M_d$ do
        $\mathcal{I} \leftarrow \mathcal{I} \cup \alpha(m)$
    end for
    $coverage \leftarrow 0$
    for all $\phi$ in $\Phi$ do
        for all $\sigma$ in $\beta(\phi)$ do
            if $\sigma \subseteq \mathcal{I}$ then
                $coverage \leftarrow coverage + 1$
            else
                break
            end if
        end for
    end for
    return $coverage / |\Phi|$
end function
```

We represent that mapping as $\mathcal{C}$. In the worst-case, the outer for loop iterates $|M|$ times, and the inner for loop iterates $|I|$ times. Thus, the complexity of this step is $O(|M|\,|I|)$. In the second set of nested for loops, we check how many ways each minimal indicator set σ in $\beta(\phi)$ can be generated, and sum these values to compute the redundancy. In the worst-case, the outer loop iterates $|\beta(\phi)|$ times, and the inner loop iterates $|I|$ times. Thus, the worst-case computational complexity of computing redundancy for an event ϕ is $O(|M|\,|I| + |\beta(\phi)|\,|I|)$.

During the computation of **Redundancy** (ϕ, M_d) for all events ϕ in Φ, the value of $\mathcal{C}$ can be computed once, stored, and reused for all other events. Furthermore, the number of iterations of the second outer loop performed in total over all events is given by B. Therefore, the worst-case computational complexity of determining the redundancy for all events ϕ in Φ is $O(|M|\,|I| + B\,|I|)$.

5.5.3 Confidence

The algorithm we use to compute confidence is given by Algorithm 3. In the first set of nested for loops, we compute the values of γ_I for each of the indicators. In the worst-case, the outer for loop iterates $|M|$ times, and the inner for loop iterates $|I|$ times. Thus, the complexity of this step is $O(|M|\,|I|)$. In the second set of nested for loops, we compute the confidence metric value using the values of γ_I. In the worst-case, the outer loop iterates $|\beta(\phi)|$ times, and the inner loop iterates $|I|$ times. Thus, the worst-case computational

Algorithm 2 Algorithm to compute the value of the redundancy metric

```
1: function Redundancy($\phi$, $M_d$)
2:     $\mathcal{C} \leftarrow 0^{|\mathcal{I}|}$
3:     for all $m$ in $M_d$ do
4:         for all $\iota$ in $\alpha(m)$ do
5:             $\mathcal{C}[\iota] \leftarrow \mathcal{C}[\iota] + 1$
6:         end for
7:     end for
8:     $redundancy \leftarrow 0$
9:     for all $\sigma$ in $\beta(\phi)$ do
10:        $count \leftarrow 0$
11:        for all $\iota$ in $\sigma$ do
12:            $count \leftarrow \min(count, \mathcal{C}[\iota])$
13:        end for
14:        $redundancy \leftarrow redundancy + count$
15:    end for
16:    return $redundancy$
17: end function
```

complexity of computing confidence for an event ϕ is $O(|M|\,|I| + |\beta(\phi)|\,|I|)$.

As was the case for redundancy, during computation of **Confidence** (ϕ, M_d) for all events ϕ in Φ, the value of γ_I can be computed once, and the number of iterations of the second outer loop performed in total is B. Therefore, the worst-case computational complexity of determining the confidence for all events ϕ in Φ is $O(|M|\,|I| + B\,|I|)$.

5.5.4 Cost

The algorithm we use to compute cost is given by Algorithm 4. The algorithm iterates only once over the set of monitors M_d and computes a single arithmetic operation for each iteration. Thus, the worst-case computational complexity of computing cost for a set of monitors is simply $O(|M|)$.

5.5.5 Optimal Monitor Deployment Programs

In general, integer nonlinear programming (INLP) is NP-hard. Additionally, since our optimal deployment programs use the metrics' values as objective and constraint functions, the programs have nonlinear, non-convex objective and constraint functions. As a result, it is not possible to use convex optimization techniques (such as interior point methods) or linear

Algorithm 3 Algorithm to compute the value of the confidence metric

```
 1: function CONFIDENCE($\phi$, $M_d$)
 2:     $\gamma_I \leftarrow 0^{|\mathcal{I}|}$
 3:     for all $m$ in $M_d$ do
 4:         for all $\iota$ in $\alpha(m)$ do
 5:             $\gamma_I[\iota] \leftarrow \max(\gamma_I[\iota], \gamma_M(m))$
 6:         end for
 7:     end for
 8:     $confidence \leftarrow 0$
 9:     for all $\sigma$ in $\beta(\phi)$ do
10:         $conf_\sigma \leftarrow 1$
11:         for all $\iota$ in $\sigma$ do
12:             $conf_\sigma \leftarrow \min(conf_\sigma, \gamma_I[\iota])$
13:         end for
14:         $confidence \leftarrow \max(confidence, conf_\sigma)$
15:     end for
16:     return $confidence$
17: end function
```

relaxation techniques to solve the programs.

Furthermore, our programs have the additional constraint that the monitor deployment variables are binary, so mixed-integer nonlinear program solvers that use gradient descent approaches are also of limited use, since the metric functions are discontinuous over the search space. To solve the optimal deployment programs, we implement a version of the branch-and-bound algorithm that performs a search over the state space of all possible monitor deployments and prunes the set of possible deployments when the constraints are not met.

The algorithm we implement to solve the optimization equations is given by Algorithm 5.

Algorithm 4 Algorithm to compute monitor cost

```
1: function COST($M_d$)
2:     $cost \leftarrow 0$
3:     for all $m$ in $M_d$ do
4:         $a \leftarrow parentAsset(m)$
5:         $cost \leftarrow cost + P(m) + a.CPUCost \times m.CPUUtilization + a.MemCost \times$
           $m.MemUtilization + LogsTo(m).DiskCost \times m.DiskUtilization + a.NetworkCost \times$
           $m.NetworkUtilization$
6:     end for
7:     return $cost$
8: end function
```

First, we note that by branching based on the presence of a value for **maxCost**, we are able to use a single algorithm to solve both optimal deployment programs.

The size of the total state space is $2^{|M|}$, where M is the set of all monitors over which the optimization is being performed. In the worst-case, the algorithm will need to examine the entire state space. For each iteration of the search, in the worst-case, all of the metric values will need to be computed for all events in Φ. Thus, from the analysis performed above, the worst-case complexity of each iteration is given by $O\left(|M|\,|I| + B\,|I| + |M|\right) = O\left(|M|\,|I| + B\,|I|\right)$.

Thus, the overall worst-case computational complexity required to search over all possible solutions to find the optimal monitor deployment is $O\left(2^{|M|}\,|I|\left(|M| + B\right)\right)$.

If we assume that each event has a constant number of minimal indicator sets, B grows linearly with the number of events, $|\Phi|$. We can then rewrite the overall worst-case complexity of computing the optimal monitor deployment as $O\left(2^{|M|}\,|I|\left(|M| + |\Phi|\right)\right)$. From this analysis, we can see that the computational complexity of our optimal deployment algorithm grows linearly in the number of indicators, linearly in the number of events, and superexponentially in the number of monitors. In many cases, we anticipate that the pruning performed by our algorithm can significantly reduce the number of monitor deployments examined, thus significantly reducing the average case computational complexity.

We leave further investigation of possible optimizations and heuristic approaches to solving the optimal deployment programs to future work.

Algorithm 5 Algorithm to compute the optimal monitor deployment

```
function FindOptimalDeployment($\Phi$, $M$)
    $S \leftarrow \mathcal{P}(M)$
    $best \leftarrow \varnothing$
    while $S$ is not empty do
        $possible \leftarrow$ True
        choose $M_d$ randomly from $S$
        $cost \leftarrow$ Cost($M_d$)
        if (maxCost exists and $cost >$ maxCost) or (not maxCost exists and $cost >$ Cost($best$)) then
            remove all deployments from $S$ that are proper supersets of $M_d$
        else
            $coverage \leftarrow$ Coverage($\Phi$, $M_d$)
            if $coverage <$ $\mathbf{min}_{\text{Coverage}}$ then
                $possible \leftarrow$ False
            else
                for all $\phi$ in $\Phi$ do
                    if $possible$ then
                        $redundancy[\phi] \leftarrow$ Redundancy($\phi$, $M_d$)
                        if $redundancy[\phi] <$ $\mathbf{min}_{\text{Redundancy}_\phi}$ then
                            $possible \leftarrow$ False
                            break
                        end if
                    end if
                    if $possible$ then
                        $confidence[\phi] \leftarrow$ Confidence($\phi$, $M_d$)
                        if $confidence[\phi] <$ $\mathbf{min}_{\text{Confidence}_\phi}$ then
                            $possible \leftarrow$ False
                            break
                        end if
                    end if
                end for
            end if
                                                        ▷ continued on page 50
```

Algorithm 5 (continued)

```
            if not $possible$ then
                remove all deployments from $S$ that are proper subsets of $M_d$
            else
                if maxCost exists then
                    calculate utility function for $M_d$
                    if $M_d.utility > best.utility$ or ($M_d.utility = best.utility$ and $cost <$ Cost($best$)) then
                        $best \leftarrow M_d$
                        remove all deployments from $S$ that are proper subsets of $M_d$
                    end if
                else
                    $best \leftarrow M_d$
                    remove all deployments from $S$ that are supersets of $M_d$
                end if
            end if
        end if
        remove $M_d$ from $S$
    end while
    return best
end function
```

CHAPTER 6

CASE STUDY: ENTERPRISE WEB SERVICE

6.1 Experimental Setup

To evaluate our approach to modeling and evaluating the efficacy of monitor deployment, we use a use case that models an enterprise Web service. The model contains software platforms that are commonly used in Web service architectures: a firewall, an HTTP server platform, a database, and a server-side scripting language that generates served pages. For simplicity, we restrict the size of the enterprise system to one each of the aforementioned components and restrict the event space to a small set of attacks for each of the components in our sample system.

Through this case study, we observe how different intrusion detection goals affect the placement of monitors and illustrate the utility of our approach in determining optimal monitor placements. We hypothesize that the use of our metrics will provide unexpected optimal placements, which would support the value of our quantitative approach.

6.1.1 Goal

The system architecture for the case study is illustrated in Figure 6.1. Our goal in this case study is to protect a Web application running within the “intranet” from the attacker VM, which represents attackers on the “Internet.” We implemented our experiments using a series of virtual machines running atop QEMU-KVM, networked together using separate VLANs for the “Internet” and “intranet” to provide network isolation. The attacker VM runs Kali 64-bit GNU/Linux 1.1.0 and is on the same VLAN as one of the network interfaces to the firewall VM. The firewall, Web server, and database server VMs all run Metasploitable v2.0.0 [53] and are networked together on a separate VLAN from the attacker VM. Metasploitable is an Ubuntu 8.04 image that is configured with services that have known vulnerabilities and backdoors, which we use for ease of experimentation. The firewall VM acts as a firewall and management virtual machine, running an instance of the Uncomplicated Firewall (UFW)

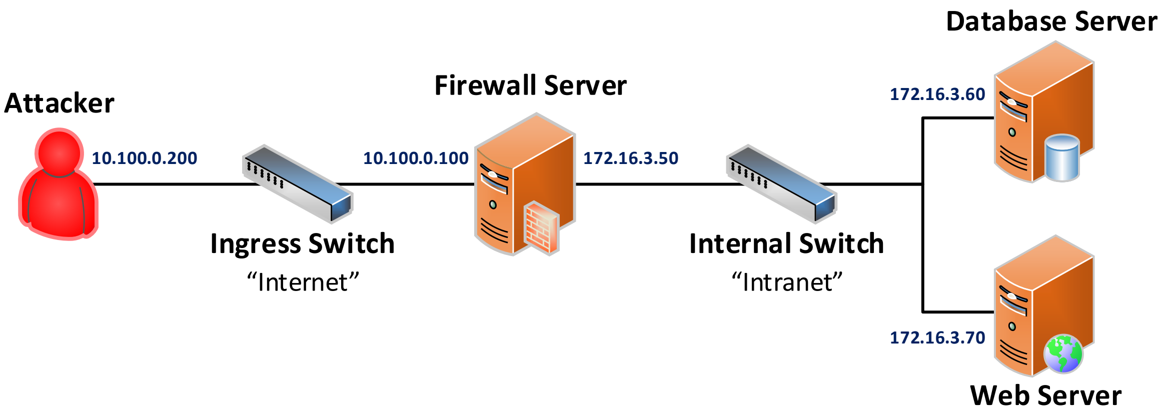

Figure 6.1: Architectural diagram for the case study experiment. The attacker can access the Web server through the ingress switch and firewall, and can use SSH to log in to the firewall. The subnet for the "Internet" is 10.100.0.0/24, and the subnet for the "intranet" is 172.16.3.0/24.

configured with firewall rules to block all external traffic to the "intranet" except Web traffic to and from the Web server and SSH traffic to and from the SSH server. The subnet for the "Internet" is 10.100.0.0/24, and the subnet for the "intranet" is 172.16.3.0/24.

We ran the Mutillidae Web application [54] for our case study. Mutillidae is a deliberately vulnerable Web application that implements vulnerabilities that correspond directly to the Open Web Application Security Project (OWASP) Top 10 list of Web application security weaknesses from 2013 [55]. Mutillidae uses the Apache 2.2.8 HTTP server, PHP 5.2.4, and MySQL server v14.12, distribution 5.0.51a. We have configured the Mutillidae instance running on the Web server to use Apache and PHP on the Web server and MySQL on the database server.

6.1.2 Monitors

To perform monitoring within the system, we used the built-in logging capabilities of each of the software packages installed. They include SSHd logging for the SSH server, UFW's logs for the firewall, Apache access logs for the HTTP server, and MySQL query logs for the SQL server. For this experiment, we restricted the set of assets to protect to those pertaining to the use of the Mutillidae Web application and management of the network. Specifically, they include all three servers and the Apache HTTP server, PHP, MySQL server, and SSH server instances. The system model for the case study and the associated component labels are illustrated in Figure 6.2. The equations describing the case study system model are given in Figure 6.3.

6.1.3 Security Events

We now describe the events of interest for our case study experiment. We used the OWASP Top 10 list [55] as the source for the events of interest. The OWASP Top 10 provides a set of the most common, important categories of vulnerabilities observed in attack datasets reflecting hundreds of organizations and thousands of Web applications. As Mutillidae is built to exhibit vulnerabilities in all OWASP Top 10 categories, we selected 5 attacks, one from each of the highest-ranked OWASP Top 10 categories, as the events of interest. They are as follows:

- ϕ_1: [Category A1: Injection] SQL injection through the `capture-data.php` page in Mutillidae
- ϕ_2: [Category A2: Broken Authentication] Authentication bypass using SQL injection on the `login.php` page
- ϕ_3: [Category A3: Cross-site Scripting (XSS)] XSS on the `capture-data.php` page
- ϕ_4: [Category A4: Insecure Direct Object References] Access to the PHP configuration page through brute-force directory traversal
- ϕ_5: [Category A5: Security Misconfiguration] SSH brute-force attack on the firewall machine

6.2 Implementation

The process by which we constructed our model of the system from the raw logs is illustrated in Figure 6.4. First, we ran a given attack to generate the logs for the events associated with the attack. Then, we parsed the raw log entries, converting them into a series of data fields. Next, we converted the parsed log entries into indicators based on logical and temporal predicates on the fields of the logs. Using the indicators extracted from the logs, we constructed minimal indicator sets and the event-indicator mappings for each event by hand. Finally, we used the experiment topology and the Mutillidae and OWASP resources to define the remaining components of our model.

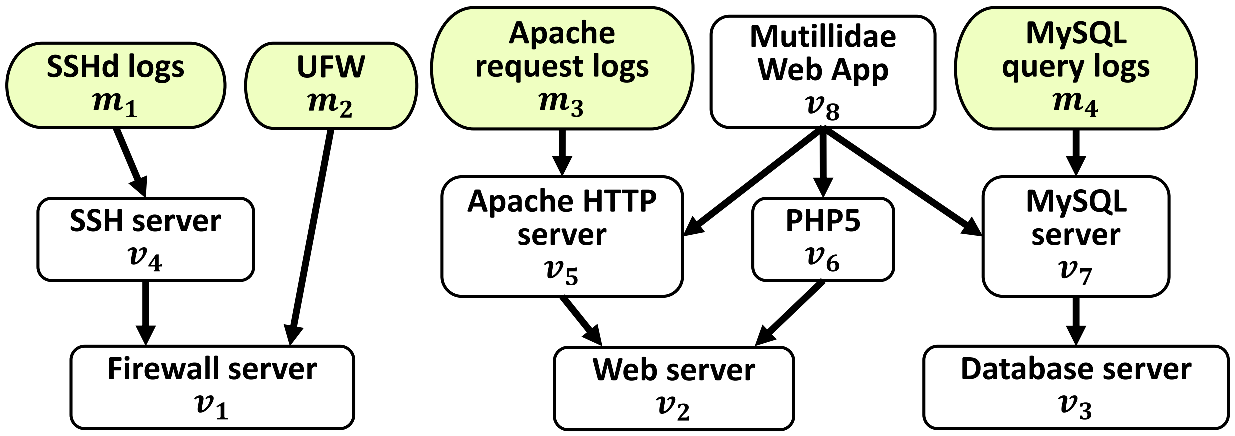

Figure 6.2: System model for the case study experiment. White shapes represent assets, green shaded shapes represent monitors, and arrows represent dependency relationships.

Corresponding to the flowchart in Figure 6.4, our implementation consists of the following parts:

1. **Raw log parsers**: The raw log parsers convert the raw logs into a machine-understandable format. We have implemented these using C++ code for the logs in our experiment. This component is used in stage 2 of the flowchart.

2. **Indicator extraction code**: This code converts the parsed raw logs into indicators that can be represented within our model. We have implemented this in C++. The indicators are generated in the format used by the system model Python code described below. This component is used in stage 3 of the flowchart.

3. The **system and data model, metrics, and optimal deployment program**: We have implemented our model, metrics, and deployment program in Python. The Python code allows a user to construct and configure the model and all intrusion detection parameters, which define the optimal deployment program for a given situation. We implement the algorithms described in Section **??** to compute the values of the metrics and solve the optimal deployment program. We use this component to build the system model in stages 4 and 5 of the flowchart and to solve for the optimal deployments.

To explain how we use monitor log data to generate indicators, we now describe our implementation of the indicator extraction code.

$$
\begin{aligned}
v_1 &= \text{Firewall server} & e_1 &= (v_8, v_5)\\
v_2 &= \text{Web server} & e_2 &= (v_8, v_6)\\
v_3 &= \text{Database server} & e_3 &= (v_8, v_7)\\
v_4 &= \text{SSH server process} & e_4 &= (v_4, v_1)\\
v_5 &= \text{Apache HTTP server process} & e_5 &= (v_5, v_2)\\
v_6 &= \text{PHP5} & e_6 &= (v_6, v_2)\\
v_7 &= \text{MySQL server} & e_7 &= (v_7, v_3)\\
v_8 &= \text{Mutillidae Web application}
\end{aligned}
$$

$$
\begin{aligned}
V_S &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}\\
E_S &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}
\end{aligned}
$$

$$
\begin{aligned}
m_1 &= \text{SSHd logs} & e_{m_1} &= (m_1, v_4)\\
m_2 &= \text{UFW} & e_{m_2} &= (m_2, v_1)\\
m_3 &= \text{Apache access logs} & e_{m_3} &= (m_3, v_5)\\
m_4 &= \text{MySQL query log} & e_{m_4} &= (m_4, v_7)
\end{aligned}
$$

$$
\begin{aligned}
M &= \{m_1, m_2, m_3, m_4\}\\
E_M &= \{e_{m_1}, e_{m_2}, e_{m_3}, e_{m_4}\}
\end{aligned}
$$

Figure 6.3: Equations describing the case study system model illustrated in Figure 6.2.

6.2.1 Indicator Extraction

As discussed earlier in this thesis, indicators are defined by logical predicates over the information generated by monitors. Specifically, in our implementation, each indicator is defined by a logical predicate, which we refer to as the *indicator logic*, over the fields provided by the monitors. A monitor m can generate an indicator if it is possible to evaluate the indicator logic over the fields generated by m. As some of the attacks within the set of events of interest for our case study rely on access thresholds within a given period of time, we use

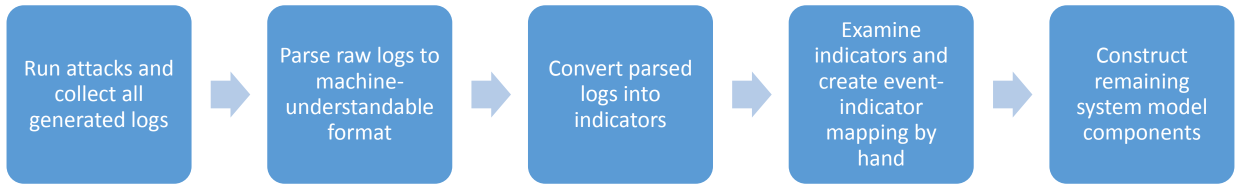
Run attacks and collect all generated logs
Parse raw logs to machine-understandable format
Convert parsed logs into indicators
Examine indicators and create event-indicator mapping by hand
Construct remaining system model components

Figure 6.4: Flowchart describing the steps we take to generate the system model for the case study example given the system model and events defined in Section 6.1.

temporal as well as logical predicates to define the indicator logic.

For this experiment, we implemented the indicator logic described below for the four monitors we used in our experiment. We observe that some indicators, such as "SSH attempt to log in as an invalid user" change in importance depending on the username that is used. For example, an attempt to log in with a misspelled version of a valid username would be relatively benign, but an attempt to log in using a username that is commonly found in brute-force dictionaries, such as "alice" or "bob", could be malicious. Therefore, we uniquely identify indicators by their type and by the set of unique identifiers associated with the indicator, which we specify with the description of the indicator logic. Indicator extraction corresponds to stage 3 in the flowchart given by Figure 6.4.

6.2.1.1 Indicator Logic for SSHd Logs

For the SSHd logs, we define the following indicators:

- High frequency of authentication attempts for a given IP address: This indicator captures an attempt by a single IP address to perform an SSH login on the firewall machine more than $threshold_{\mathrm{SSH}}$ times within a $window_{\mathrm{SSH}}$-second time interval, whether the attempt was successful or unsuccessful. For this experiment, we set $threshold_{\mathrm{SSH}}$ to 5 and $window_{\mathrm{SSH}}$ to 10 based on the speed of the tool we use to perform SSH brute force. Within the SSHd logs, a failed login attempt generates the following types of entries:

```
Jun 26 11:04:29 firewall sshd[26865]: Failed password for invalid user admin
     from 10.100.0.200 port 42656 ssh2
Jun 26 11:04:29 firewall sshd[26865]: pam\_unix(sshd:auth): authentication
    failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=10.100.0.200 user=
    root
```

  The unique identifier for this indicator is the IP address attempting the SSH logins.

- Reverse DNS lookup fails for an IP address attempting SSH login: This indicator captures an attempt by an IP address that fails reverse DNS lookup to perform an SSH login on the firewall machine. While such failures do not always have malicious causes, failure of a reverse DNS lookup could indicate that the IP address being provided is being spoofed, or that the SSH request is coming from the darknet. Within the SSHd logs, a reverse DNS lookup failure generates the following type of log entry:

```
Jun 26 11:06:04 firewall sshd[26842]: Address 213.144.202.125 maps to
    125-202-144-213.altitudetelecom.fr, but this does not map back to the
    address - POSSIBLE BREAK-IN ATTEMPT!
```

The unique identifier for this indicator is the IP address attempting the SSH login.

- SSH login attempt from a blocked IP address: This indicator simply captures an attempt to perform an SSH login on the firewall machine from an IP address that has for some reason been blacklisted. Blacklisting could be done automatically by SSHd if the process has been configured to do so, or could be done manually. Within the SSHd logs, such an attempt generates the following type of log entry:

  ```
  Jun 26 11:06:29 firewall sshd[26865]: refused connect from 10.100.0.200
      (10.100.0.200)
  ```

  The unique identifier for this indicator is the IP address attempting the SSH login.

- SSH login attempt using an invalid username: This indicator captures an attempt to perform an SSH login on the firewall machine using a username that does not exist on the machine. In some cases, such an attempt reflects misspelling of a valid username and is benign. However, in most cases, such an attempt is an attempt by an attacker to discover weak user accounts on the system that can be used to gain access to the system. SSHd generates a host of different log messages when an invalid username is used. This indicator is generated if any of the following types of entries are generated by SSHd:

  ```
  Jun 26 11:06:04 firewall sshd[26842]: Invalid user admin from 10.100.0.200
  Jun 26 11:06:04 firewall sshd[26842]: input_userauth_request: invalid user
      admin [preauth]
  Jun 26 11:06:04 firewall sshd[26842]: pam_listfile(sshd:auth): Refused user
      admin for service sshd
  Jun 26 11:06:04 firewall sshd[26842]: pam_unix(sshd:auth): check pass; user
      unknown
  Jun 26 11:06:06 firewall sshd[26842]: Failed password for invalid user admin
       from 10.100.0.200 port 42465 ssh2
  ```

  The unique identifiers for this indicator are the IP address attempting the SSH login and the invalid username used.

- High frequency of authentication failures using the sudo command: This indicator captures an unsuccessful attempt by a user to perform a command using sudo on the firewall machine more than $threshold_{\text{sudo}}$ times within a 10-second time interval. For this experiment, we set $threshold_{\text{sudo}}$ to 5.

The unique identifier for this indicator is the username performing the sudo attempts.

6.2.1.2 Indicator Logic for UFW Logs

For the UFW logs, we define the following indicators:

- UFW BLOCK alert: This indicator captures any packet that is blocked by UFW. As we have configured UFW to allow only Web and SSH traffic from the external "Internet" to the internal IP addresses, any other attempts at sending traffic will generate a UFW block attempt. We have also configured UFW to add a limit rule to port 22 (SSH) to block SSH access for an IP address temporarily if it exceeds a threshold number of new network flows to the SSH port within a certain period of time.

  As the set of rules we have created use destination ports to distinguish between types of traffic, the unique identifiers for this indicator are the source and destination IP addresses and the destination port.

- UFW AUDIT alert: This indicator captures a packet that does not exactly match a firewall rule, but is allowed through the firewall by the default policy; such an occurrence could indicate that a hole has been found in the firewall.

  As with the UFW BLOCK alert indicator, the unique identifiers for this indicator are the source and destination IP addresses and the destination port.

6.2.1.3 Indicator Logic for Apache Access Logs

For the Apache access logs, we define the following indicators:

- HTTP access to an administrative URL: This indicator captures an attempt by an IP address to access an administrative resource. While not malicious on its own, if performed by an IP address that is not permitted to access the resource, or by an unauthenticated user, it could indicate compromise of the system. For this indicator, we have constructed a list of directories and files (from the Mutillidae website) that are restricted for administrative use. We consider any access attempt (any HTTP verb) to one of the administrative URLs from an IP address outside the "intranet" to be malicious.

The unique identifiers for this indicator are the requester IP address and the URL of the resource requested. For this and all other Apache access log indicators, the URL is stripped of its query string, leaving just the path to the requested resource.

- High frequency of HTTP accesses: This indicator captures the actions of an IP address that makes a large number of HTTP requests within a $window_{\text{HTTP requests}}$-second time interval. A count of HTTP requests beyond a threshold, $threshold_{\text{HTTP requests}}$, likely indicates that a user (or machine) is attempting to perform brute-force requests on the Web server for one of multiple possible reasons (e.g., denial of service, vulnerability discovery, or discovery of unprotected administrative pages). For this experiment, we set $threshold_{\text{HTTP requests}}$ to 20 and $window_{\text{HTTP requests}}$ to 10.

  The unique identifier for this indicator is the requester IP address.

- High frequency of bad HTTP status codes: This indicator captures the actions of an IP address that makes a large number of HTTP requests (an amount that exceeds $threshold_{\text{HTTP bad status}}$) to the Web server within a 10-second time interval that return with bad status codes (bad status codes are 301, 4XX, and 5XX). The requests may be to invalid pages, may be made without valid authorization, or may result in a server error. For this experiment, we set $threshold_{\text{HTTP bad status}}$ to 5.

  The unique identifier for this indicator is the requester IP address.

- Malicious POST request: This indicator captures an attempt to perform an HTTP POST to the Web server machine. While the Apache access logs do not provide the contents of the POST, they do provide the URL to which the POST was sent. If a website's API is clearly defined, POST requests should be sent only to a specific set of URLs, so any POSTs to other URLs could indicate an attempt to upload a backdoor or exploit a vulnerability in the website. In the case of Mutillidae, almost all POSTs will be malicious, so we do not distinguish in our indicator logic between valid and invalid URLs.

  The unique identifiers for this indicator are the requester IP address and the URL to which the POST request was sent.

- URL string containing SQL code: This indicator captures an attempt to perform SQL injection by providing SQL code as a URL query string parameter. To simplify detection of SQL code, we look for substrings that we would never expect to see under normal circumstances, but would likely be present in SQL code. That technique is used in practice to detect SQL injection attempts, often using regular expressions instead of exact-match substrings. An example of a tool that uses such techniques is Apache scalp [56].

For the purposes of this experiment, we looked for the following substrings:

- `--`
- `@@version`
- `varchar`
- `char`
- `union`

The unique identifiers for this indicator are the requester IP addresses and the URL to which the request was sent.

• URL string containing JavaScript code: Similar to the SQL code URL indicator, this indicator captures an attempt to perform cross-site scripting (XSS) by injecting Javascript code as a URL query string parameter. As with the detection of SQL code, while it is possible to use tools like Apache scalp and Apache mod_security to perform regular expression analysis of the access logs to detect XSS attempts, in our implementation, we simply examined the URL query string for the following characters and substrings:

- backslash ("\")
- opening or closing angle brackets ("<", ">", or "`/>`")
- opening or closing parentheses ("`(`" or "`)`")
- semicolon ("`;`")
- "`script`"

The unique identifiers for this indicator are the requester IP address and the URL to which the request was sent.

6.2.1.4 Indicator Logic for MySQL Query Logs

The indicator logic for the indicators extracted from MySQL query logs depends heavily on the actual MySQL queries performed by the Web application, as in order to identify anomalous queries, it is necessary first to establish a baseline. Thus, we first describe the MySQL queries that Mutillidae generates upon Web requests.

Mutillidae contains an array of different pages that can be used to perform initialization of the database, authentication, update of the database, and retrieval of database information. When a user visits a page, Mutillidae creates a connection to the MySQL database; executes the queries pertaining to the page itself; logs the IP address, user agent string, referer, and time of access of the request to a hitlog table; and then closes the connection. Thus, each connection containing a query to the hitlog table can be treated as a unique page request, providing the IP address of the requester of the page. Mutillidae also stores informational log messages to the hitlog table for some user actions, including all successful and unsuccessful authentication attempts. For example, upon a hit to a page that does not require any page-specific SQL queries, the following entries are added to the MySQL query log:

```
150626 9:25:28     43 Connect    root@localhost on owasp10
       43 Query      INSERT INTO hitlog(hostname, ip, browser, referer, date)
          VALUES ('10.100.0.200', '10.100.0.200', 'Mozilla/5.0 (X11; Linux
          x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.6.0', 'User
          visited', now() )
       43 Quit
```

In addition, for pages that store information to be retrieved later, such as `capture-data.php` and `add-to-your-blog.php`, Mutillidae generates a SQL query that inserts data captured by the URL query string and fields on the page into MySQL tables corresponding to the page visited. For example, upon a visit to the `capture-data.php` page, the following entries are added to the MySQL query log:

```
150626 10:09:53    68 Connect    root@localhost on owasp10
       68 Query      SELECT * FROM accounts WHERE cid='1'
       68 Query      INSERT INTO captured_data(ip_address, hostname, port,
          user_agent_string, referrer, data, capture_date) VALUES
          ('10.100.0.200', '10.100.0.200', '53333', 'Mozilla/5.0 (X11; Linux
          x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.6.0', '', '
          page = capture-data.php
    showhints = 2
    username = admin
    uid = 1
    my_wikiUserID = 2
    my_wikiUserName = Test
    my_wiki_session = 6f9d602e9031eb75474500799fe76bed
    PHPSESSID = d633mpp1img52iie4bishda4q4
    showhints = 2
username = admin
uid = 1
my_wikiUserID = 2
my_wikiUserName = Test
my_wiki_session = 6f9d602e9031eb75474500799fe76bed
PHPSESSID = d633mpp1img52iie4bishda4q4
', now())
    68 Query      INSERT INTO hitlog(hostname, ip, browser, referer, date)
        VALUES ('10.100.0.200', '10.100.0.200', 'Mozilla/5.0 (X11; Linux
        x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.6.0', 'User
        visited', now() )
    68 Quit
```

Since the queries used for all Mutillidae pages are static, we can create regular expressions that attempt to find anomalous queries or those with unexpected clauses or characters. Thus, for the MySQL query logs, we define the following indicators:

- Unexpected SQL query: This indicator captures an attempt to perform a form of SQL injection in which additional clauses are appended to a SQL query by way of a user-provided string field that is not properly escaped by the PHP webpage. For the pages that we visited for this experiment, we collected all of the SQL queries performed by visits to those pages, excepting all hitlog table queries. We constructed a regular expression that matches all such valid queries, where user-supplied string parameters have been replaced with a regular expression matching any possible properly-escaped MySQL string. Any query that does not match the regular expression is potentially malicious or evidence of a SQL injection attempt.

  The unique identifiers for this indicator are the requester IP address and the table on which the query is being performed.

- SQL query containing HTML: This indicator captures an attempt to perform XSS by inserting HTML code into the database to be retrieved later and displayed by the Web application. To identify HTML code, the indicator looks for matches of the regular expression string "`<[^>]*?>`", which identifies XML tags within the query.

  The unique identifiers for this indicator are the requester IP address and the table on which the query is being performed.

- SQL query containing injection keywords: This indicator captures an attempt to perform SQL injection using common SQL injection keywords. For the purposes of this experiment, we looked for the following substrings, which we know would not be present in a valid query:

– `--`
– `@@version`
– `varchar`
– `schema`
– `union`

The unique identifiers for this indicator are the requester IP address and the table on which the query is being performed.

By running the code implementing the indicator logic described above, we were able to extract the observable indicators shown in Figure 6.5 from the logs in our system.

The monitor-indicator generation relationship for the indicators listed in Figure 6.5 is given by the following set of equations:

$$\begin{aligned} \alpha(m_1) &= \{\iota_1, \iota_2, \iota_3, \iota_4, \iota_5, \iota_6\} & \alpha(m_3) &= \{\iota_9, \iota_{10}, \iota_{11}, \iota_{12}, \iota_{13}, \iota_{14}, \iota_{15}, \iota_{16}, \iota_{17}, \iota_{18}, \iota_{19}, \iota_{20}\} \\ \alpha(m_2) &= \{\iota_7, \iota_8\} & \alpha(m_4) &= \{\iota_{21}, \iota_{22}, \iota_{23}, \iota_{24}, \iota_{25}\} \end{aligned}$$

6.2.2 Creation of the Event-Indicator Mapping

Next, given the indicators extracted from the logs, we generated the mappings for the evidence required to detect events by hand, which corresponds to stage 4 in the flowchart given by Figure 6.4. We did so using our understanding of how the responses by the system to attack actions map to the indicators generated when we ran the attacks.

First, for each event, we grouped the indicators that were generated by each monitor based on the type of evidence they provided to support detection of the event. For example, for ϕ_5, all of the SSH invalid user indicators (where each indicator describes an attempt under a different username) for the same IP address would provide equivalent evidence to support detection of an SSH brute-force attack by that IP address. We could treat all such indicators as equivalent. Thus, we grouped all of these indicators together when examining the indicators generated by SSHd logs for ϕ_5.

Then, we used our judgment, backed by existing IDS signatures and the literature on intrusion detection techniques, to determine which groups of indicators would need to be detected in order to detect an attack. For each group, we considered the generation of some

$\iota_1 =$ SSH auth attempts $> \mathit{threshold}_{\text{SSH}}$ for 10.100.0.200
$\iota_2 =$ SSH login attempt from blocked IP 10.100.0.200
$\iota_3 =$ SSH login attempt with bad username `test`@10.100.0.200
$\iota_4 =$ SSH login attempt with bad username `admin`@10.100.0.200
$\iota_5 =$ SSH login attempt with bad username `alice`@10.100.0.200
$\iota_6 =$ sudo auth failures $> \mathit{threshold}_{\text{sudo}}$ for user `user`
$\iota_7 =$ UFW BLOCK for source 10.100.0.200 to 10.100.0.100:22
$\iota_8 =$ UFW BLOCK for source 10.100.0.200 to 172.16.3.70:80
$\iota_9 =$ HTTP access to admin URL `/mutillidae/index.php?page=phpinfo.php` by 10.100.0.200
$\iota_{10} =$ HTTP access to admin URL `/mutillidae/phpinfo.php` by 10.100.0.200
$\iota_{11} =$ HTTP access to admin URL `/mutillidae/index.php?page=/etc/passwd` by 10.100.0.200
$\iota_{12} =$ HTTP access to admin URL `/phpMyAdmin` by 10.100.0.200
$\iota_{13} =$ HTTP brute force by 10.100.0.200
$\iota_{14} =$ HTTP bad requests $> \mathit{threshold}_{\text{HTTP bad status}}$ for 10.100.0.200
$\iota_{15} =$ malicious `POST` to `/mutillidae/index.php?page=login.php` by 10.100.0.200
$\iota_{16} =$ malicious `POST` to `/mutillidae/index.php?page=text-file-viewer.php` by 10.100.0.200
$\iota_{17} =$ SQL injection attempt through URL string on resource `/mutillidae/index.php?page=capture-data.php` by 10.100.0.200
$\iota_{18} =$ SQL injection attempt through URL string on resource `/mutillidae/index.php?page=login.php` by 10.100.0.200
$\iota_{19} =$ XSS attempt through URL string on resource `/mutillidae/index.php?page=capture-data.php` by 10.100.0.200
$\iota_{20} =$ XSS attempt through URL string on resource `/mutillidae/index.php` by 10.100.0.200
$\iota_{21} =$ unexpected SQL query on table `accounts` by 10.100.0.200
$\iota_{22} =$ unexpected SQL query on table `captured_data` by 10.100.0.200
$\iota_{23} =$ XSS attempt via injection on table `captured_data` by 10.100.0.200
$\iota_{24} =$ SQL injection attempt on table `accounts` by 10.100.0.200
$\iota_{25} =$ SQL injection attempt on table `captured_data` by 10.100.0.200

Figure 6.5: Equations listing the indicators extracted from logs for the case study experiment.

minimum threshold number of indicators from the group to be sufficient for detection of the group. In most cases, the threshold number was one. For example, for ϕ_5, we considered brute force to be defined as one of 1) a threshold number of failed SSH login attempts from the same IP address within a given time period, 2) a threshold number of failed sudo attempts for the same user within a given time period, 3) a threshold number of SSH login attempts using invalid usernames from the same IP address, 4) a UFW BLOCK alert for port 22 on the firewall server (since the UFW SSH limit rule causes UFW to generate BLOCK alerts for SSH connection attempts exceeding a threshold number), or 5) an SSH login attempt that appears to be a break-in attempt coupled with a login attempt from the same IP address using an invalid username. We constructed the minimal indicator sets and evidence mapping for ϕ_5 using the conditions described above and the actual indicators that were generated.

The equations provided below describe the evidence required to detect the events for our case study.

$$\begin{aligned}
\beta(\phi_1) &= \{\{\iota_{17}\}, \{\iota_{22}\}, \{\iota_{25}\}\} \\
\beta(\phi_2) &= \{\{\iota_{15}, \iota_{21}\}, \{\iota_{15}, \iota_{24}\}, \{\iota_{18}, \iota_{21}\}, \{\iota_{18}, \iota_{24}\}, \{\iota_{21}, \iota_{24}\}\} \\
\beta(\phi_3) &= \{\{\iota_{19}\}, \{\iota_{20}, \iota_{22}\}, \{\iota_{20}, \iota_{25}\}, \{\iota_{23}\}\} \\
\beta(\phi_4) &= \{\{\iota_{9}\}, \{\iota_{10}\}, \{\iota_{12}\}\} \\
\beta(\phi_5) &= \{\{\iota_{1}\}, \{\iota_{3}, \iota_{4}\}, \{\iota_{3}, \iota_{5}\}, \{\iota_{4}, \iota_{5}\}, \{\iota_{6}\}, \{\iota_{7}\}\}
\end{aligned}$$

6.2.3 Remaining System Model Parameters

After constructing the system model graph, S; the list of events, Φ; the list of indicators, I; the mapping between monitors and events, α; and the mapping between events and indicators, β, we still needed to define values for a small number of system model parameters. They are the trustworthiness values for the monitors, γ_M, and the parameters and mappings for the calculation of monitor cost, which are the monitor resource utilization, monitor purchase and management cost, asset resource cost, and monitor **LogsTo** relationship. Assigning these values corresponds to stage 5 in the flowchart given by Figure 6.4.

We assigned truthfulness values to the monitors based on the levels of indirection between the monitors and the attacker VM and the vulnerability of the assets on which they depend, as provided by Mutillidae's website.

We assigned resource utilizations to each of the monitors based on a resource usage profile of its execution during the tests and based on the size of the logs generated. We used a scaled version of Amazon's EC2 pricing model to determine resource costs for each of the

Table 6.1: System model values for case study monitors.

Monitor	$\gamma_M(m)$	Resource Utilization			
		CPU	Memory	Disk	Network
m_1	0.3	0.02	0.008	0.1	0
m_2	0.3	0.05	0.004	0.01	0
m_3	0.5	0.1	0.1	0.4	0
m_4	0.9	0.2	100	1	0

Table 6.2: System model values for case study assets.

Assets	Resource Cost			
	CPU	Memory	Disk	Network
v_1, v_4	0.5	0.002334	0.0347	0.1
v_2, v_3, v_5, v_6, v_7, v_8	1.0	0.005668	0.0694	0.1

resources on the machines. Furthermore, as the firewall server is not crucial to the operation of the Web server, its resource costs were reduced in comparison to those of the Web server and database server. As the monitors used in our case study are open-source and do not require extensive configuration, we set all purchase and management costs to zero. In our case study, all monitors log locally. All of the values we used in the case study for the system model parameters are provided in Tables 6.1 and 6.2.

6.3 Results

We use the following four sets of intrusion detection requirements as example cases to compare the deployment obtained through our methodology with one that might be chosen by a system administrator. For each example case, we also show the complete set of possible deployments and the values of the relevant metrics for each deployment, to prove that our equations indeed yield the optimal result. For all cases, the deployment is subject to a cost constraint of maxCost = \$1.00, which we chose to be insufficient to deploy all monitors, but sufficient to deploy at least two.

Monitors deployed				Cost	**Redundancy**(ϕ_1)	**Coverage**(Φ)
m_1	m_2	m_3	m_4			
No	No	No	No	\$0.000	0	0.0
Yes	No	No	No	\$0.013	0	0.2
No	Yes	No	No	\$0.025	0	0.2
Yes	Yes	No	No	\$0.039	0	0.2
No	No	Yes	No	\$0.128	1	0.6
Yes	No	Yes	No	\$0.142	1	0.8
No	Yes	Yes	No	\$0.154	1	0.8
Yes	Yes	Yes	No	\$0.167	1	0.8
No	No	No	Yes	\$0.836	2	0.6
Yes	No	No	Yes	\$0.850	2	0.8
No	Yes	No	Yes	\$0.862	2	0.8
Yes	Yes	No	Yes	\$0.875	2	0.8
No	No	Yes	Yes	\$0.965	3	0.8
Yes	No	Yes	Yes	\$0.978	3	1.0
No	Yes	Yes	Yes	\$0.990	3	1.0
Yes	Yes	Yes	Yes	\$1.003	3	1.0

Table 6.3: Relevant metrics' values for all possible monitor deployments for Case 1.

6.3.1 Case 1

A practitioner wishes to maximize the number of events that can be detected, but must ensure detectability of ϕ_1 (SQL injection through `capture-data.php`). Using his or her intuition, the practitioner would deploy all of the monitors, but would not know if this meets the cost constraints.

Using our approach, we can describe the set of events of interest as $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5\}$. The intrusion detection requirements can be captured by setting $\mathbf{w}_{\text{Coverage}} = 1$ to maximize the number of events that can be detected and by setting $\mathbf{min}_{\text{Redundancy}_{\boldsymbol{\phi_1}}} = 1$ to ensure detectability of ϕ_1. All other parameters are set to 0.

Solving the optimal deployment equation generated by the requirements provided above and the model constructed for this experiment yields the deployment $M_d = \{m_1, m_3, m_4\}$. Table 6.3 shows the values of **Redundancy**(ϕ_1), **Coverage**(Φ), and monitor cost for all possible monitor deployments. Because of cost constraints, it is not possible to deploy all four monitors. However, m_1 and m_2 both provide coverage for ϕ_5, so only one of them needs to be deployed to maximize coverage. From examination of the table, it is evident that the deployment chosen by our algorithm is indeed cost-optimal.

6.3.2 Case 2

A practitioner wishes to ensure that all unauthorized access events (ϕ_2, ϕ_4, ϕ_5) can be detected. The other events are not important. Based on intuition, the practitioner would deploy monitors m_1, m_3, and m_4, since the unauthorized access events deal with SSH accesses, Web page accesses, and logins using the database.

Using our approach, since we do not have any requirements regarding ϕ_1 or ϕ_3, we can restrict the set of events of interest to $\Phi = \{\phi_2, \phi_4, \phi_5\}$. Doing so increases the efficiency of the solver, as it does not need to keep track of as many intermediate variables. By setting $\mathbf{min}_{\text{Coverage}} = 1$, we can ensure that the monitor deployment will completely cover all events in Φ. All other parameters are set to 0.

Solving the optimal deployment equation generated by the requirements provided above and the model constructed for this experiment yields the deployment $M_d = \{m_1, m_3, m_4\}$. Table 6.4 shows the values of $\mathbf{Coverage}(\Phi)$ and monitor cost for all possible monitor deployments. From examination of the table, it is evident that the monitor deployment chosen by our algorithm is indeed optimal. In this case, the deployment does not differ from the deployment chosen via intuition. However, from the fact that the optimal deployment required both m_3 and m_4, we can be sure that m_3 and m_4 cannot independently provide enough information to detect both ϕ_2 and ϕ_4. That inference is corroborated by Table 6.4.

6.3.3 Case 3

A practitioner believes that monitors may become unavailable and wishes to maximize the number of ways in which events ϕ_1 and ϕ_4 can be detected. All other events are of lesser importance. Using intuition, the practitioner would deploy m_3 and m_4, as both monitors should provide information about both events.

As with Case 2, using our approach, we can set $\Phi = \{\phi_1, \phi_4\}$. To maximize the number of ways to detect the two events, we can set $\mathbf{w}_{\text{Redundancy}_{\boldsymbol{\phi_1}}} = \mathbf{w}_{\text{Redundancy}_{\boldsymbol{\phi_4}}} = 1$. All other parameters are set to 0.

Solving the optimal deployment equation generated by the requirements provided above and the model constructed for this experiment yields the deployment $M_d = \{m_3, m_4\}$. Table 6.5 shows the values of $\text{Redundancy}_{\phi_1}$ and $\text{Redundancy}_{\phi_4}$; the monitor cost for all possible monitor deployments; and the value of the utility function (the function to maximize) for the optimal deployment equation. Here, again, the deployment is the same as the intuitive deployment, and examination of the table verifies that the deployment chosen by our algorithm is optimal. However, our approach provides the additional utility of quantifying the

Monitors deployed				**Cost**	**Coverage**(Φ)
m_1	m_2	m_3	m_4		
No	No	No	No	\$0.000	0.00
Yes	No	No	No	\$0.013	0.33
No	Yes	No	No	\$0.025	0.33
Yes	Yes	No	No	\$0.039	0.33
No	No	Yes	No	\$0.128	0.33
Yes	No	Yes	No	\$0.142	0.67
No	Yes	Yes	No	\$0.154	0.67
Yes	Yes	Yes	No	\$0.167	0.67
No	No	No	Yes	\$0.836	0.33
Yes	No	No	Yes	\$0.850	0.67
No	Yes	No	Yes	\$0.862	0.67
Yes	Yes	No	Yes	\$0.875	0.67
No	No	Yes	Yes	\$0.965	0.67
Yes	No	Yes	Yes	\$0.978	1.00
No	Yes	Yes	Yes	\$0.990	1.00
Yes	Yes	Yes	Yes	\$1.003	1.00

Table 6.4: Relevant metrics' values for all possible monitor deployments for Case 2.

redundancy values for the two events. For the case study model constructed as described above, Redundancy$(\phi_1, M_d) = 3$ and Redundancy$(\phi_4, M_d) = 3$.

6.3.4 Case 4

A practitioner believes the system may be compromised, and wishes to maximize the ability to detect SQL injection attacks (ϕ_1 and ϕ_2) and ϕ_5. Based on intuition, the practitioner would deploy m_4 to detect the two SQL injection attacks and m_1 to detect ϕ_5. However, it is unclear which of the other two monitors would best meet the practitioner's requirements.

Using our approach, we first set $\Phi = \{\phi_1, \phi_2, \phi_5\}$. We can maximize the ability to detect the events under compromise by maximizing redundancy and confidence for each. That is, we set $\mathbf{w}_{\text{Redundancy}_{\phi_1}} = \mathbf{w}_{\text{Redundancy}_{\phi_2}} = \mathbf{w}_{\text{Redundancy}_{\phi_5}} = \mathbf{w}_{\text{Confidence}_{\phi_1}} = \mathbf{w}_{\text{Confidence}_{\phi_2}} = \mathbf{w}_{\text{Confidence}_{\phi_5}} = 1$. All other parameters are set to 0.

Solving the optimal deployment equation generated by the requirements provided above and the model constructed for this experiment yields the deployment $M_d = \{m_1, m_2, m_4\}$. Table 6.6 shows the values of the redundancy and confidence metrics for ϕ_1, ϕ_2, and ϕ_5; the monitor cost for all possible monitor deployments; and the value of the utility function (the

Monitors deployed				**Cost**	**Redundancy**(ϕ_1)	**Redundancy**(ϕ_4)	Total utility
m_1	m_2	m_3	m_4				
No	No	No	No	\$0.000	0	0	0
Yes	No	No	No	\$0.013	0	0	0
No	Yes	No	No	\$0.025	0	0	0
Yes	Yes	No	No	\$0.039	0	0	0
No	No	Yes	No	\$0.128	1	3	4
Yes	No	Yes	No	\$0.142	1	3	4
No	Yes	Yes	No	\$0.154	1	3	4
Yes	Yes	Yes	No	\$0.167	1	3	4
No	No	No	Yes	\$0.836	2	0	2
Yes	No	No	Yes	\$0.850	2	0	2
No	Yes	No	Yes	\$0.862	2	0	2
Yes	Yes	No	Yes	\$0.875	2	0	2
No	No	Yes	Yes	\$0.965	3	3	6
Yes	No	Yes	Yes	\$0.978	3	3	6
No	Yes	Yes	Yes	\$0.990	3	3	6
Yes	Yes	Yes	Yes	\$1.003	3	3	6

Table 6.5: Relevant metrics' values for all possible monitor deployments for Case 3. Total utility for this case is the sum of **Redundancy**(ϕ_1) and **Redundancy**(ϕ_4).

function to maximize) for the optimal deployment equation. Examination of the table shows that compared to deploying only m_1 and m_4, deploying m_2 increases the redundancy of ϕ_1 by 1 and deploying m_3 increases the redundancy of ϕ_5 by 4. Since the added utility from deploying m_3 is greater than that of deploying m_2, deploying m_3 is more beneficial. While a practitioner could use intuition or domain knowledge to try to arrive at the same conclusion, our deployment explicitly quantifies the benefit, eliminating guesswork and simplifying the analysis for the practitioner. The deployment chosen by our algorithm can be seen to be optimal for this set of requirements.

6.3.5 Expressiveness of Approach

The evaluation we perform here illustrates not only the feasibility and ease-of-use of our approach in determining cost-optimal monitor placements, but also its expressiveness. Each of the four scenarios represents a realistic intrusion detection goal a practitioner may have for a system. As shown above, the scenarios can be directly transformed into intrusion detection requirements within our methodology, and the resulting optimization equation can be solved to obtain an optimal deployment. Using our methodology, a practitioner could perform

Monitors deployed				**Cost**	**Redundancy**(ϕ_1)	**Confidence**(ϕ_1)	**Redundancy**(ϕ_2)	**Confidence**(ϕ_2)	**Redundancy**(ϕ_5)	**Confidence**(ϕ_5)	Total utility
m_1	m_2	m_3	m_4								
No	No	No	No	\$0.000	0	0.0	0	0.0	0	0.0	0.0
Yes	No	No	No	\$0.013	0	0.0	0	0.0	5	0.3	5.3
No	Yes	No	No	\$0.025	0	0.0	0	0.0	1	0.3	1.3
Yes	Yes	No	No	\$0.039	0	0.0	0	0.0	6	0.3	6.3
No	No	Yes	No	\$0.128	1	0.8	0	0.0	0	0.0	1.8
Yes	No	Yes	No	\$0.142	1	0.8	0	0.0	5	0.3	7.0
No	Yes	Yes	No	\$0.154	1	0.8	0	0.0	1	0.3	3.0
Yes	Yes	Yes	No	\$0.167	1	0.8	0	0.0	6	0.3	8.1
No	No	No	Yes	\$0.836	2	0.9	1	0.9	0	0.0	4.8
Yes	No	No	Yes	\$0.850	2	0.9	1	0.9	5	0.3	10.1
No	Yes	No	Yes	\$0.862	2	0.9	1	0.9	1	0.3	6.1
Yes	Yes	No	Yes	\$0.875	2	0.9	1	0.9	6	0.3	11.1
No	No	Yes	Yes	\$0.965	3	0.9	5	0.9	0	0.0	9.8
Yes	No	Yes	Yes	\$0.978	3	0.9	5	0.9	5	0.3	15.1
No	Yes	Yes	Yes	\$0.990	3	0.9	5	0.9	1	0.3	11.1
Yes	Yes	Yes	Yes	\$1.003	3	0.9	5	0.9	6	0.3	16.1

Table 6.6: Relevant metrics values for all possible monitor deployments for Case 4. Total utility for this case is the sum of **Redundancy**(ϕ) and **Confidence**(ϕ) for all ϕ in $\Phi = \{\phi_1, \phi_2, \phi_5\}$.

a study of monitor deployment with different sets of intrusion detection requirements or different parameter values for the same system. For a small system like the one used in our experiments, it would be possible to perform the analysis manually, but as the size of the system and the numbers of monitors and events grow, our approach significantly simplifies such an analysis.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this thesis, we presented a quantitative methodology to determine maximum-utility, cost-optimal deployments of security monitors in networked distributed systems in terms of intrusion detection goals. We showed the need for our methodology by observing that current approaches to monitor deployment do not consider the cost of monitoring, deal only with the deployment or configuration of network IDS monitors, or do not consider the effect of compromise on the ability of monitors to support intrusion detection.

Our methodology consists of three components. First, we defined a system and data model that represents 1) system components, monitors, and their dependence relationships; 2) events in the system that a security administrator would want to detect; and 3) how data generated by monitors relates to the detectability of an event. Second, we defined a set of monitor utility metrics that quantify the ability of a set of monitors to support intrusion detection and a monitor cost metric that quantifies monitor purchase and utilization costs over the lifetime of the monitor. Third, we defined an optimization problem that combines our model and metrics with intrusion detection requirements and can be solved to yield an optimal monitor deployment.

We illustrated the use of our approach through a working example based on an enterprise Web service scenario. We demonstrated the efficacy and practicality of our approach through a case study, also based on an enterprise Web service scenario, in which we programmatically generated the components of our model from logs captured from a running Web application and used a nonlinear program solver to find optimal monitor deployments for a set of four different example scenarios. Through our examples, we demonstrated that our methodology gives practitioners a way to express intrusion detection requirements and easily determine satisfactory optimal monitor deployments, which may not be intuitive. Our approach offers a novel, quantitative technique for determining where to deploy security monitors, and fills a gap in the existing range of tools and literature.

7.2 Future Work

In future work, we plan to investigate utility metrics other than the three we introduce here. Some possible metrics include those related to increasing the granularity of monitor information for forensic analysis, graph-theoretic metrics such as connectivity and robustness of the assets in the graph, and metrics related to the necessity of certain types of information for monitor data fusion and intrusion detection algorithms. We are also considering performing sensitivity analysis on assets to determine the absolute effect of unavailability of an asset on the utility metrics.

Another avenue for future work would be to address some of the assumptions we make in our approach. In our current model, monitors independently provide information about events by generating indicators. In some cases, however, absence of an indicator in the presence of other indicators may itself provide evidence supporting detection of an event. We aim to represent such a relationship between monitors and events within our model. In our current model, we also assume independence between monitors, and model dependence between assets as a dependency graph. We plan to investigate dependence relationships among assets and monitors in more depth, and accurately represent the relationships within our model.

We also aim to investigate methodical ways to set model parameters and enumerate the components of our system and data model. In this thesis, we propose a mechanism for generating indicators directly from log data, but other possible approaches include using monitor configurations and monitor source code analysis to determine the set of all possible indicators that can be generated by a monitor. We currently leave specification of events of interest and selection of target metric values to the security administrator, but it might be possible to automatically infer these parameters directly from the security policy specifications, service level agreements, and intrusion detection system rules.

Finally, we plan to adapt our approach to be used for *online* monitor deployment. As the system changes and alerts arrive from monitors, the optimal deployment of monitors will also change. We plan to investigate how fusion and intrusion detection algorithms would drive our model, metric definitions, and weights and constraints for the optimal deployment equations. We also plan to investigate the applicability of machine learning approaches to online monitor deployment. An online version of our approach could be the basis for reinforcement learning responses or a defensive monitoring system.

REFERENCES

[1] Aorato Labs, "The Untold Story of the Target Attack Step by Step," Aorato Labs, Tech. Rep., Aug. 2014. [Online]. Available: https://aroundcyber.files.wordpress.com/2014/09/aorato-target-report.pdf

[2] K. Zetter, "Sony Got Hacked Hard: What We Know and Dont Know So Far," *Wired*, Dec. 2014. [Online]. Available: http://www.wired.com/2014/12/sony-hack-what-we-know/

[3] S. Banjo, "Home Depot Hackers Exposed 53 Million Email Addresses," *Wall Street Journal*, Nov. 2014. [Online]. Available: http://www.wsj.com/articles/home-depot-hackers-used-password-stolen-from-vendor-1415309282

[4] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM Conference on Computer and Communications Security.* New York, NY, USA: ACM, 2007, pp. 552–561.

[5] Y. Deswarte, L. Blain, and J.-C. Fabre, "Intrusion tolerance in distributed computing systems," in *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991, pp. 110–121.

[6] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining audit data to build intrusion detection models," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD '98*, 1998, pp. 66–72.

[7] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transactions on Information System Security*, vol. 2, no. 2, pp. 159–176, May 1999.

[8] J. Sundqvist, "80 Linux Monitoring Tools for SysAdmins," Server Density, 2015. [Online]. Available: https://blog.serverdensity.com/80-linux-monitoring-tools-know/

[9] NetIQ, "Security management," 2015. [Online]. Available: https://www.netiq.com/solutions/security-management/

[10] IBM Corp., "Tivoli Netcool/OMNIbus," 2015. [Online]. Available: http://www-03.ibm.com/software/products/en/ibmtivolinetcoolomnibus

[11] Symantec Corporation, "Products & solutions," 2015. [Online]. Available: http://www.symantec.com/products-solutions/

[12] SilverSky, "Obtaining Fortune 500 Security without Busting Your Budget," 2015. [Online]. Available: http://www.staging.silversky.com/sites/default/files/Fortune_500_Security_ROI_White_Paper_0.pdf

[13] W. Jansen, "Directions in security metrics research," National Institute of Standards and Technology, Tech. Rep. NISTIR 7564, 2009. [Online]. Available: http://csrc.nist.gov/publications/nistir/ir7564/nistir-7564_metrics-research.pdf

[14] D. Rathbun, "Gathering security metrics and reaping the rewards," SANS Institute, Oct. 2009. [Online]. Available: http://www.sans.org/reading-room/whitepapers/leadership/gathering-security-metrics-reaping-rewards-33234

[15] N. Pappas, "Network IDS & IPS deployment strategies," SANS Institute, Tech. Rep., 2008. [Online]. Available: http://www.homeworkmarket.com/sites/default/files/q2/23/04/network-ids-ips-deployment-strategies-2143.pdf

[16] C. Calabrese, "SANS: Intrusion detection FAQ: How to place IDS sensor in redundant networks?" [Online]. Available: https://www.sans.org/security-resources/idfaq/ids_redun.php#Single_IDS_probe_placed_outside_area_of

[17] E. Carter, "Cisco IDS sensor deployment considerations," Feb. 2002. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=25327&seqNum=4

[18] IBM Corp., "IBM Security QRadar SIEM," July 2015. [Online]. Available: http://www-03.ibm.com/software/products/en/qradar-siem

[19] Tripwire, "Tripwire Enterprise." [Online]. Available: http://www.tripwire.com/it-security-software/scm/tripwire-enterprise/

[20] Splunk, "Using Splunk software as a SIEM." [Online]. Available: https://www.splunk.com/web_assets/pdfs/secure/Splunk_as_a_SIEM_Tech_Brief.pdf

[21] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 273–284.

[22] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Proceedings. of the 19th Annual Computer Security Applications Conference, 2003*, Dec. 2003, pp. 86–95.

[23] E. LeMay, M. Ford, K. Keefe, W. Sanders, and C. Muehrcke, "Model-based security metrics using ADversary VIew Security Evaluation (ADVISE)," in *Proceedings of the 2011 Eighth International Conference on Quantitative Evaluation of Systems (QEST)*, Sep. 2011, pp. 191–200.

[24] S. Cheung, U. Lindqvist, and M. Fong, "Modeling multistep cyber attacks for scenario recognition," in *Proceedings of the DARPA Information Survivability Conference and Exposition, 2003*, vol. 1, Apr. 2003, pp. 284–292.

[25] Q. Zhu and T. Başar, "Indices of Power in Optimal IDS Default Configuration: Theory and Examples," *arXiv:1110.1862 [cs]*, Oct. 2011, arXiv: 1110.1862.

[26] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian, "Scalable detection of cyber attacks," in *Computer Information Systems Analysis and Technologies*, ser. *Communications in Computer and Information Science*, N. Chaki and A. Cortesi, Eds. Springer Berlin Heidelberg, 2011, no. 245, pp. 9–18.

[27] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats*, ser. *Massive Computing*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. Springer US, Jan. 2005, no. 5, pp. 247–266.

[28] S. Jajodia and S. Noel, "Topological vulnerability analysis," in *Cyber Situational Awareness*, ser. *Advances in Information Security*, S. Jajodia, P. Liu, V. Swarup, and C. Wang, Eds. Springer US, Jan. 2010, no. 46, pp. 139–154.

[29] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *Proceedings of the DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01*, vol. 2, 2001, pp. 307–321.

[30] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02.* New York, NY, USA: ACM, 2002, pp. 245–254.

[31] S. Noel and S. Jajodia, "Optimal IDS sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 259–275, Sep. 2008.

[32] H. Almohri, D. Yao, L. Watson, and X. Ou, "Security optimization of dynamic networks with probabilistic graph modeling and linear programming," *IEEE Transactions on Dependable and Secure Computing*, to appear. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2015.2411264

[33] S. Noel, S. Jajodia, L. Wang, and A. Singhal, "Measuring security risk of networks using attack graphs," *International Journal of Next-Generation Computing*, vol. 1, no. 1, pp. 135–147, 2010.

[34] S. Schmidt, T. Alpcan, A. Albayrak, T. Başar, and A. Mueller, "A malware detector placement game for intrusion detection," in *Critical Information Infrastructures Security*, ser. *Lecture Notes in Computer Science*, J. Lopez and B. M. Hämmerli, Eds. Springer Berlin Heidelberg, 2008, no. 5141, pp. 311–326.

[35] N. Talele, J. Teutsch, R. Erbacher, and T. Jaeger, "Monitor placement for large-scale systems," in *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies.* ACM Press, 2014, pp. 29–40.

[36] N. Talele, J. Teutsch, T. Jaeger, and R. F. Erbacher, "Using security policies to automate placement of network intrusion prevention," in *Proceedings of the 5th International Conference on Engineering Secure Software and Systems, ESSoS'13*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 17–32.

[37] M. Bloem, T. Alpcan, S. Schmidt, and T. Başar, "Malware filtering for network security using weighted optimality Measures," in *Proceedings of the 2007 IEEE International Conference on Control Applications*, Oct. 2007, pp. 295–300.

[38] C. Pham, Z. Estrada, P. Cao, Z. Kalbarczyk, and R. Iyer, "Reliability and security monitoring of virtual machines using hardware architectural invariants," in *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2014, pp. 13–24.

[39] "What is Snort?," 2015. [Online]. Available: https://www.snort.org/faq/what-is-snort

[40] "Tcpdump & Libpcap," 2015. [Online]. Available: http://www.tcpdump.org/

[41] C.-Y. Ho, Y.-C. Lai, I.-W. Chen, F.-Y. Wang, and W.-H. Tai, "Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems," *IEEE Communications Magazine*, vol. 50, no. 3, pp. 146–154, Mar. 2012.

[42] D. Owen, "SANS: What is a false positive and why are false positives a problem?" [Online]. Available: http://www.sans.org/security-resources/idfaq/false_positive.php

[43] M. Bishop, "A model of security monitoring," in *Proceedings of the Fifth Annual Computer Security Applications Conference*, Dec. 1989, pp. 46–52.

[44] F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks," in *Proceedings of the 3rd International Symposium on Recent Advances in Intrusion Detection*, ser. *Lecture Notes in Computer Science*, H. Debar, L. Mé, and S. F. Wu, Eds. Springer Berlin Heidelberg, Jan. 2000, no. 1907, pp. 197–216.

[45] B. Morin, L. Mé, H. Debar, and M. Ducassé, "M2D2: A formal data model for IDS alert correlation," in *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, 2002, pp. 115–137.

[46] E. Totel, B. Vivinis, and L. Mé, "A language driven intrusion detection system for event and alert correlation," in *IFIP Security and Protection in Information Processing Systems*, Y. Deswarte, F. Cuppens, S. Jajodia, and L. Wang, Eds. Springer US, Jan. 2004, no. 147, pp. 209–224.

[47] U. Thakore, G. A. Weaver, and W. H. Sanders, "An Actor-Centric, Asset-Based Monitor Deployment Model for Cloud Computing," in *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing*. Dresden, Germany: IEEE Computer Society, 2013, pp. 311–12.

[48] "PHPIDS." [Online]. Available: https://github.com/PHPIDS/PHPIDS

[49] U. Thakore, G. A. Weaver, and W. H. Sanders, "An actor-centric, asset-based monitor deployment model for cloud computing," University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, Technical Report UILU-ENG-14-2202, July 2014.

[50] G. A. Weaver, C. Cheh, E. J. Rogers, W. H. Sanders, and D. Gammel, "Toward a cyber-physical topology language: Applications to NERC CIP audit," in *Proceedings of the First ACM Workshop on Smart Energy Grid Security, SEGS '13*. New York, NY, USA: ACM, 2013, pp. 93–104.

[51] M. Almgren, U. Lindqvist, and E. Jonsson, "A multi-sensor model to improve automated attack detection," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, ser. *Lecture Notes in Computer Science*, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds. Springer Berlin Heidelberg, Jan. 2008, no. 5230, pp. 291–310.

[52] H. H. Hosmer, "Security is fuzzy!: Applying the fuzzy logic paradigm to the multipolicy paradigm," in *Proceedings of the 1992-1993 Workshop on New Security Paradigms*. ACM, 1993, pp. 175–184.

[53] "Metasploitable 2 Exploitability Guide," Rapid7, 2013. [Online]. Available: https://community.rapid7.com/docs/DOC-1875

[54] "NOWASP (Mutillidae)," SourceForge, 2015. [Online]. Available: http://sourceforge.net/projects/mutillidae/

[55] "Top 10 2013," Open Web Application Security Project (OWASP), 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013

[56] "Apache-scalp: Apache log analyzer for security," Google Project Hosting. [Online]. Available: https://code.google.com/p/apache-scalp/