A Quantitative Methodology for Security Monitor Deployment

Uttam Thakore, Gabriel A. Weaver, William H. Sanders
Computer Science Department, Information Trust Institute, and Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{thakore1, gweaver, whs}@illinois.edu

***Abstract*—Intrusion detection and forensic analysis techniques depend upon monitors to collect information about possible attacks. Since monitoring can be expensive, however, monitors must be selectively deployed to maximize their overall utility. This paper introduces a methodology both to evaluate monitor deployments quantitatively in terms of security goals and to deploy monitors optimally based on cost constraints. First, we define a model that describes the system assets, deployable monitors, and the relationship between generated data and intrusions. Then, we define a set of metrics that quantify the utility and richness of monitor data with respect to intrusion detection and the cost associated with deployment. Finally, we formulate a method using our model and metrics to determine the cost-optimal, maximum-utility placement of monitors. We present an enterprise Web service use case and illustrate how our metrics can be used to determine optimal monitor deployments for a set of common attacks on Web servers. Our approach is scalable, being able to compute within minutes optimal monitor deployments for systems with hundreds of monitors and attacks.**

I. Introduction

Despite advances in intrusion detection and prevention systems, attacks on networked computer systems continue to succeed. Intrusion prevention and detection are not on their own enough to secure computer systems; mechanisms for both online response and forensic analysis are required to deal adequately with attacks that are successful.

Intrusion tolerance and forensic analysis techniques depend on the information contained in logs. It is therefore essential that logs be correct and contain adequate information about events that may take place in the system [1], [2]. Log 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. Different monitors generate heterogeneous information, as a result of which the utility of the logs may depend on the algorithms that are used to analyze the data. Deploying monitors efficiently and effectively to support intrusion detection is a challenging problem.

Unfortunately, there is no principled, widely applicable methodology for evaluating the effectiveness of monitoring tools or monitor deployments. Existing approaches to coordinated security monitoring include deployment based on domain expert knowledge and partially automated tools for monitor deployment and analysis, which are often referred to as *Security Information and Event Management* (*SIEM*) systems. The *cost* of monitoring also limits one's ability to deploy monitors, as monitoring infrastructure can be expensive to purchase and maintain; monitors must be *selectively* deployed based on cost and utility tradeoffs.

To address the problem of monitor deployment, we devise a methodology that practitioners can use to determine an optimal deployment of monitors based on intrusion detection requirements and monitor cost. Our methodology consists of three parts. First, we define a *system and data model* that describes the system we aim to protect and the monitors that can be deployed in the system. Second, we define a set of *metrics* that evaluate both the utility and richness of monitor data and the cost associated with deployment. The metrics can be used to characterize the ability of a deployment of monitors to meet intrusion detection goals, and provide a baseline for quantitative evaluation of monitor deployments. Then, we use our model and metrics to formulate a nonlinear optimization problem to determine the optimal placement of monitors. The optimization problem is based on the tradeoff between intrusion detection requirements and cost constraints. We show the utility of our approach by programmatically constructing a model for a case study based on an enterprise Web service system and finding optimal monitor deployments for the system for different intrusion detection requirements. Finally, we perform a theoretical and experimental evaluation of our algorithms and show that our approach is scalable to systems with hundreds of monitors and attacks.

II. Related Work

Interest in the topic of resource-limited security monitor and intrusion detection system (IDS) placement and evaluation has increased in recent years. The research 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.

Attack graphs have been used extensively in the literature to describe attacks and their relation to monitors [3]–[7]. Many researchers have proposed methods for determining detection requirements from attack graphs. Sheyner et al. attempt to find the minimal set of attacks that must be blocked to prevent an attacker from reaching his objectives [4]. Albanese et al. merge multiple attack graphs obtained from different sources and use them to predict which attacks are forthcoming based on observed IDS alerts [7]. Some researchers have also used attack graphs to predictively determine optimal monitor deployments. For example, Almohri et al. transform attack graphs into an optimization problem, based on a probabilistic network and threat model, that can yield deployments of security resources that minimize attack success probability [8]. Schmidt et al. and Zhu et al. apply game-theoretic principles to attack graphs, modeling attacker and defender behaviors to determine where to optimally place monitors to minimize the utility of an intelligent attacker [5], [9]. All of these approaches focus on deployment of network IDSes, neglecting lower-level monitors, such as access logs, that can yield valuable information for intrusion detection and forensic analysis. Additionaly, those approaches may rely on models with a large number of parameters that must be approximated; such approximation is infeasible in real, large-scale systems.

Other researchers have proposed the use of network information flow graphs to determine where to place monitors. Talele et al. model the flow of security information among hosts and network IDSes in terms of information flow graphs and attempt to determine monitor placements that prevent information flows disallowed by security policies [3], [10]. Bloem et al. model general network traffic using information flow graphs and determine optimal network filtering policies that ensure adherence to security policies [11]. Again, these 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 to use other monitor data in detecting intrusions.

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 off-the-shelf (COTS) coordinated monitoring and analysis tools that in turn deploy monitors. Such tools offer a wide list of features, including compliance with federal regulations, consolidated management of monitoring, and automated monitor reconfiguration [12]–[15]. However, it is unclear how the tools differ in functionality or what intrusion detection guarantees the tools provide. 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.

III. System and Data Model

A. Guiding Observations and Assumptions

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

1) Monitor faultiness and compromise: We observe that monitors, once deployed, become part of a system's attack surface. That understanding motivates our research in two ways. First, we take the effect of monitor compromise into account in the development of our set of metrics, as we cannot assume that the presence of a monitor implies that the monitor will always generate indicators correctly. Second, we quantify both the ability to detect intrusions when all monitors are functioning correctly and the redundancy of monitors, as we believe that increased redundancy in data collection can increase the ability to detect intrusions when some monitors are unavailable or are providing erroneous data.

2) Monitor heterogeneity: We observe that different monitors produce heterogeneous information. 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) Intrusion detection ability: We approach the problem of monitor deployment from a *proactive* perspective. That is, we consider the problem of deploying monitors *before* the system is running in order to *support* intrusion detection.

Because our analysis is performed prior to monitor data generation, we cannot guarantee that events will always be correctly detected by an intrusion detection mechanism or quantify the probability that they will be detected. Thus, to simplify our characterization of the intrusion detection approach taken, we make the idealized assumption of *perfect forensic ability* of the detection mechanism. Under this assumption, if an attack occurs in the system, the intrusion detection mechanism, whether it be an algorithm or a forensic analyst, will be able to detect the attack if given access to a set of monitors that generates some minimal amount of information about the attack. We define the minimal amount of information needed for detection in our model.

B. Definitions

In order to quantitatively evaluate the usefulness of a set of security monitors towards intrusion detection, we first define a model on top of which the quantitative metrics can be defined.

We define the following components of the 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. Some examples of monitors include the Apache server access log, Snort, and the Linux syslog.

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 a system and that a security administrator would want to detect. Events could be explicit violations of security policy, or actions that an attacker might take while conducting an attack. Since our analysis takes place offline, we elide the notion of time in our definition of an event.

Event definitions may include source and/or target information for an intrusion, or may represent any manifestation of the intrusion, depending on the 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 log entries that are directly generated by monitors. Rather, detection of an event may require 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.

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 (OS). The monitor will generate log entries in an OS-specific format for each system call. 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 [16]–[18].

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 of logs. The approach also facilitates enumeration of indicators by security domain experts, who may not be able to specify the exact format for the logs, but understand what semantic information is provided by each log.

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. Assets may include host machines, servers, virtual machines, applications, and network hardware. Our definition of assets is the same as that introduced by Thakore et al. in [19].

C. System Model

We now describe the formalization with which we represent the set of system components. We define the base system components as follows:

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

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 as directed edges between assets. Note that these are not the same as network connections between the assets; instead, they represent a dependence between an asset v_i and another asset v_j that we can use to understand the effects of compromise. For example, if a machine running an SSH server process were compromised by an attacker, the attacker could tamper with actions taken by the SSH process and any data in the SSHd logs, making both untrustworthy. In that case, we would represent the dependence relationship as a directed edge e in E_S from SSHd to the machine.

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:

$$\begin{aligned} M &= \{m \mid m = \text{a monitor that can be deployed in the system}\} \\ E_M &= \{e_m \mid e_m = (m, v)\,,\ m \in M,\ v \in V_S, \\ &\qquad\qquad m \text{ depends on } v \text{ for correct operation}\} \end{aligned}$$

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.

E_M represents the dependence relationships between the monitors in M and the system assets in V_S. Specifically, a directed 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.

Definition 5. *We define the* system graph*, S, as a dependency graph constructed from the components in the system and their dependence relationships:*

$$S = (V, E) \tag{1}$$

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

D. Event and Indicator Model

We have defined events and indicators above. We now define the sets of each of them that we use to describe how monitor data corresponds to detection of events.

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 for which it is worthwhile to dedicate monitoring resources to detect them) 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 VIII.

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

As with Φ, we leave the enumeration of I to future work, but some possibilities include examining the monitor configuration or source code and performing data mining over vast quantities of existing logs.

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 by generating indicators, corresponding to the relationship mentioned in Section III-B.

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

$$\alpha(m) = \{\iota \in I \mid \textit{monitor } m \textit{ can generate } \iota\} \tag{2}$$

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 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.

Next, we define a representation for *the evidence required to detect an event*. This relationship stands in for the actual methods by which intrusion detection algorithms relate monitor information to attacks, corresponding to the relationship between events and indicators mentioned in Section III-B.

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. 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*

$$\beta(\phi) = \{\sigma \mid \sigma \text{ can be used to detect } \phi\} \tag{3}$$

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. Like α, β is deterministic, which is related to the assumption of perfect forensic ability.

Our model of event-to-indicator mappings supports a wide range of attack definitions. Most directly, our approach supports signature-based approaches, such as LAMBDA [16]. 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.

E. Detectability

For the purpose of representing intrusion detection goals within our model, we must define the conditions under 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 deployed in the system. That is, detectability is a function $\delta : \Phi \times \mathcal{P}(M_d) \to \{0,1\}$ such that*

$$\delta(\phi, M_d) = 1 \leftrightarrow \exists \sigma \in \beta(\phi) \wedge \sigma \subseteq \bigcup_{m:\, m \in M_d} \alpha(m) \tag{4}$$

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

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.

IV. Monitor Deployment Metrics

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.

A. Coverage

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

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

$$\mathbf{Cov}(\Phi, M_d) = \frac{|\{\phi \mid \delta(\phi, M_d) \wedge \phi \in \Phi\}|}{|\Phi|} \tag{5}$$

where M_d is the set of monitors deployed, and Φ is the set of important events to detect in the system.

Coverage provides a sense of how much of what a system administrator wants to detect can be detected using a set of monitors. For example, if a set of events must all be detected, the coverage of those events can be required to be 100%.

B. Redundancy

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

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

$$\varsigma(\phi, M_d) = \{\sigma \mid \sigma \in \beta(\phi) \wedge \delta(\sigma, M_d)\} \tag{6}$$

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,*

$$\mathbf{Red}(\phi, M_d) = \sum_{\sigma \in \varsigma(\phi, M_d)} \min_{\iota \in \sigma} |\{m \mid m \in M_d,\ \iota \in \alpha(m)\}| \tag{7}$$

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.

C. 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 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 define the truthfulness of the monitors.

Definition 15. *The* **truthfulness** *of a monitor is the extent to which 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, the 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 [20], 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.

$$\gamma_M : M \to \{n \in \mathbb{R} \mid 0 \leq n \leq 1\} \tag{8}$$

We project the truthfulness of a monitor onto 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.

$$\gamma_I(\iota, M_d) = \begin{cases} \max\limits_{m \in M_d \mid \iota \in \alpha(m)} \gamma_M(m), & \delta(\phi, M_d) \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

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 define the confidence in detecting σ as the MTL conjunction over the truthfulness values for all indicators in σ. Finally, we 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.

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

$$\mathbf{Conf}(\phi, M_d) = \max_{\sigma \in \beta(\phi)} \min_{\iota \in \sigma} \gamma_I(\iota, M_d) \tag{10}$$

D. 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.

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 17. *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 $ResourceCost : V_S \to \mathbb{R}^4$.*

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 18. *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* $\mathbf{ResourceUtil} : M \to \mathbb{R}^4$.

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.

We also define the purchase and management costs, which 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.

Definition 19. *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.*

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,*

$$\begin{aligned} \mathbf{Cost}(M_d) = \sum_{m \in M_d} (&\mathbf{CPUCost}(v)\,\mathbf{CPUUtil}(m) \\ &+ \mathbf{MemoryCost}(v)\,\mathbf{MemoryUtil}(m) \\ &+ \mathbf{NetworkCost}(v)\,\mathbf{NetworkUtil}(m) \\ &+ \mathbf{DiskCost}(\mathbf{LogsTo}(m))\,\mathbf{DiskUtil} + P(m)) \end{aligned} \tag{11}$$

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.

V. 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 to meet the intrusion detection requirements.

A. 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 Section IV. To support that, we allow a practitioner to specify *weights and constraints* on the values of the monitoring utility metrics for the optimal deployment algorithm. That is, we define the following constants:

$\mathbf{w}_{\text{Cov}} \in [0, 1]$: The weight for the coverage metric for the set of events Φ (12)

$\mathbf{w}_{\text{Red}_\phi}, \mathbf{w}_{\text{Conf}_\phi} \in [0, 1] \quad \forall \phi \in \Phi$: The weights for the redundancy and confidence metrics for each of the events in Φ (13)

$\mathbf{min}_{\text{Cov}} \in [0, 1]$: The minimum value for the coverage metric for the set of events Φ (14)

$\mathbf{min}_{\text{Red}_\phi} \in \mathbb{Z} \quad \forall \phi \in \Phi$: The minimum values for the redundancy metric for each of the events in Φ (15)

$\mathbf{min}_{\text{Conf}_\phi} \in [0, 1] \quad \forall \phi \in \Phi$: The minimum values for the confidence metric for each of the events in Φ (16)

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 Φ 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 $\mathbf{w}_{\text{Redundancy}_{\phi_2}}$ 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.

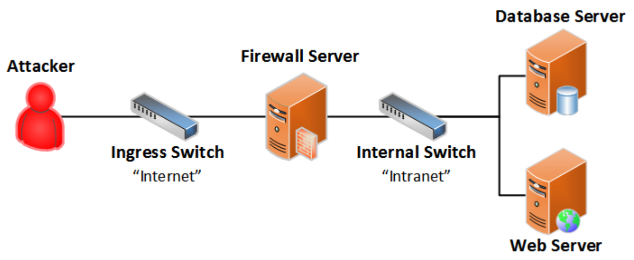

Fig. 1. Architectural diagram for the case study experiment. The attacker can access the Web server through the ingress switch and firewall, and can use SSH to log in to the firewall.

Additionally, practitioners are often bound by business constraints on the resources available for monitoring. We represent such constraints using the constant $\mathbf{maxCost}$, an upper limit on the value of the cost metric.

B. Cost-Constrained 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.

We represent our optimal deployment problem as a 0-1 integer program with inequality constraints. The program is given by Equation 17. Unless otherwise specified, by default, the $\mathbf{min}$ and $\mathbf{w}$ constants are set to 0, and $\mathbf{maxCost}$ is set to ∞.

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

$$\begin{aligned} \max_{M_d} \ & \mathbf{w}_{\text{Cov}} \mathbf{Cov}(\varPhi, M_d) + \sum_{\phi \in \varPhi} \Big(\mathbf{w}_{\text{Red}_\phi} \mathbf{Red}(\phi, M_d) + \\ & \qquad \mathbf{w}_{\text{Conf}_\phi} \mathbf{Conf}(\phi, M_d) \Big) \\ \text{s.t.} \ & \mathbf{Cost}(M_d) \leq \mathbf{maxCost} \\ & \mathbf{Cov}(\varPhi, M_d) \geq \mathbf{min}_{\text{Cov}} \\ & \mathbf{Red}(\phi, M_d) \geq \mathbf{min}_{\text{Red}_\phi}, \ \forall \phi \in \varPhi \\ & \mathbf{Conf}(\phi, M_d) \geq \mathbf{min}_{\text{Conf}_\phi}, \ \forall \phi \in \varPhi \\ & M_d \in \{0,1\}^{|M|} \end{aligned} \tag{17}$$

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.

C. Unconstrained-Cost Optimal Deployment

The maximum utility program given by Equation 17 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 17 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. The program is given by Equation 18.

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

$$\begin{aligned} \min_{M_d} \ & \mathbf{Cost}(M_d) \\ \text{s.t.} \ & \mathbf{Cov}(\varPhi, M_d) \geq \mathbf{min}_{\text{Cov}} \\ & \mathbf{Red}(\phi, M_d) \geq \mathbf{min}_{\text{Red}_\phi}, \ \forall \phi \in \varPhi \\ & \mathbf{Conf}(\phi, M_d) \geq \mathbf{min}_{\text{Conf}_\phi}, \ \forall \phi \in \varPhi \\ & M_d \in \{0,1\}^{|M|} \end{aligned} \tag{18}$$

VI. Case Study: Enterprise Web Service

A. Experimental Setup

To evaluate our approach to modeling and evaluating the efficacy of monitor deployment, we present a use case that models an enterprise system wherein the enterprise provides a Web service to customers. The use case system contains common software platforms that are 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 system to one each of the aforementioned components and restrict the event space to a few attacks for each of the components.

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 show that the use of our metrics can provide unexpected optimal placements, which lends credence to our approach of quantifying the contribution of monitors towards intrusion detection goals.

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." The system architecture for the case study is illustrated in Figure 1. 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 [21] 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) 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.

We ran the Mutillidae Web application [22] for our case study. Mutillidae is a deliberately vulnerable Web application running on a LAMP stack (Linux, Apache, MySQL, and PHP) that implements vulnerabilities that correspond directly to the Open Web Application Security Project (OWASP) Top 10 list of Web application security weaknesses from 2013 [23].

1) 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

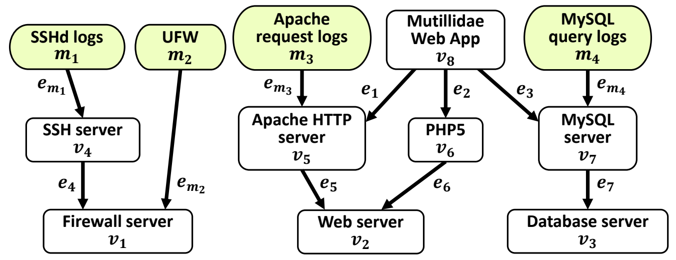

Fig. 2. System model graph for the case study experiment. White rounded rectangles represent assets; green shaded ovals represent monitors; and arrows represent dependence relationships. The component labels correspond to those used in Section VI.

server, and SSH server processes. The system model for the case study and the associated component labels are illustrated in Figure 2. For the case study, the system model is given by:

$$\begin{aligned} V_S &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\} \\ E_S &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \\ M &= \{m_1, m_2, m_3, m_4\} \\ E_M &= \{e_{m_1}, e_{m_2}, e_{m_3}, e_{m_4}\} \end{aligned}$$

2) Security Events: We now describe the events of interest for our case study. We used the OWASP Top 10 list [23] as the source for the events. The OWASP Top 10 provides a set of the most common and 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 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

B. Implementation

The process by which we constructed our model of the system from the raw logs is described next. 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 those indicators, we constructed by hand the minimal indicator sets and the event-indicator mappings for each event. Finally, we used the experiment topology and the Mutillidae and OWASP resources to define the remaining components of our model.

1) Indicator Extraction: As discussed earlier in this paper, indicators are defined by logical predicates over the information generated by monitors. 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

ι_1 = SSH auth attempts $> \mathit{threshold}_{\text{SSH}}$
ι_2 = SSH login attempt from blocked IP 10.100.0.200
ι_3 = SSH login attempt with bad username `test`@10.100.0.200
ι_4 = SSH login attempt with bad username `admin`@10.100.0.200
ι_5 = SSH login attempt with bad username `alice`@10.100.0.200
ι_6 = sudo auth failures $> \mathit{threshold}_{\text{sudo}}$ for user `user`
ι_7 = UFW BLOCK for source 10.100.0.200 to 10.100.0.100:22
ι_8 = UFW BLOCK for source 10.100.0.200 to 172.16.3.70:80
ι_9 = HTTP access to admin URL /mutillidae/index.php?page=phpinfo.php
ι_{10} = HTTP access to admin URL /mutillidae/phpinfo.php
ι_{11} = HTTP access to admin URL /mutillidae/index.php?page=/etc/passwd
ι_{12} = HTTP access to admin URL /phpMyAdmin
ι_{13} = HTTP brute force
ι_{14} = HTTP bad requests $> \mathit{threshold}_{\text{HTTP bad status}}$
ι_{15} = malicious `POST` to /mutillidae/index.php?page=login.php
ι_{16} = malicious `POST` to /mutillidae/index.php?page=text-file-viewer.php
ι_{17} = SQL injection attempt through URL string on resource /mutillidae/index.php?page=capture-data.php
ι_{18} = SQL injection attempt through URL string on resource /mutillidae/index.php?page=login.php
ι_{19} = XSS attempt through URL string on resource /mutillidae/index.php?page=capture-data.php
ι_{20} = XSS attempt through URL string on resource /mutillidae/index.php
ι_{21} = unexpected SQL query on table `accounts`
ι_{22} = unexpected SQL query on table `captured_data`
ι_{23} = XSS attempt via injection on table `captured_data`
ι_{24} = SQL injection attempt on table `accounts`
ι_{25} = SQL injection attempt on table `captured_data`

Fig. 3. Definitions of the indicators extracted for the case study experiment. All indicators assume the host at 10.100.0.200 to be the source of the activity.

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 our case study's set of events of interest rely on access thresholds within a given period of time, we use temporal as well as logical predicates to define the indicator logic.

For this experiment, we implemented the indicator logic described in [24] for the four monitors we used in our experiment. By running the code implementing the indicator logic, we were able to extract the observable indicators shown in Figure 3 from the logs in our system.

The monitor-indicator generation relationship for the indicators listed in Figure 3 is given by the following equations:

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

2) Creation of the Event-Indicator Mapping: Next, given the indicators extracted from the logs, we generated by hand the mappings for the evidence required to detect events. 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

TABLE I. Monitor truthfulness and resource utilization values for the case study experiment.

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

provided to support detection of the event. For example, for ϕ_5, all of the SSH invalid user indicators for unique usernames 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 minimum threshold number of indicators from the group to be sufficient for detection of the group. 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. A more detailed description of how we obtained the mappings is given in [24].

$$\beta(\phi_1) = \{\{\iota_{17}\}, \{\iota_{22}\}, \{\iota_{25}\}\}$$
$$\beta(\phi_2) = \{\{\iota_{15}, \iota_{21}\}, \{\iota_{15}, \iota_{24}\}, \{\iota_{18}, \iota_{21}\}, \{\iota_{18}, \iota_{24}\}, \{\iota_{21}, \iota_{24}\}\}$$
$$\beta(\phi_3) = \{\{\iota_{19}\}, \{\iota_{20}, \iota_{22}\}, \{\iota_{20}, \iota_{25}\}, \{\iota_{23}\}\}$$
$$\beta(\phi_4) = \{\{\iota_9\}, \{\iota_{10}\}, \{\iota_{12}\}\}$$
$$\beta(\phi_5) = \{\{\iota_1\}, \{\iota_3, \iota_4\}, \{\iota_3, \iota_5\}, \{\iota_4, \iota_5\}, \{\iota_6\}, \{\iota_7\}\}$$

C. Remaining System Model Parameters

We assigned truthfulness values, γ_M, to the monitors based on the 1) levels of indirection between the monitors and the attacker VM and 2) the vulnerability of the assets on which they depend, as provided by Mutillidae's website. For example, since Mutillidae exposes many PHP vulnerabilities that can cause compromise of the Web server, but no such vulnerabilities for the database server (beyond SQL injection), we consider m_3 to be considerably less truthful than m_4.

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 resources on the machines. As the monitors used in our case study are open-source, we set all purchase and management costs to zero. All monitors log locally. All of the values we used in the case study for the system model parameters are provided in Tables I and II.

D. 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 is intuitive and might be chosen by a system administrator. For all cases, the deployment is subject to a cost constraint of maxCost = \$1.00, which is insufficient to deploy all monitors, but sufficient to deploy at least two. All parameters not specified are set to 0.

TABLE II. Asset resource costs for the case study experiment. Costs are given in dollars per unit of utilization per unit time.

Assets	Resource Cost			
	CPU	Memory	Disk	Network
v_1, v_4	\$0.50	\$0.002334	\$0.0347	\$0.10
v_2, v_3, v_5, v_6, v_7, v_8	\$1.00	\$0.005668	\$0.0694	\$0.10

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 might 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 $\mathbf{w}_{\text{Cov}} = 1$ to maximize the number of events that can be detected and by setting $\mathbf{min}_{\text{Red}_{\phi_1}} = 1$ to ensure detectability of ϕ_1.

Solving the optimal deployment equation yields the deployment $M_d = \{m_1, m_3, m_4\}$. 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.

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 might 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 $\Phi = \{\phi_2, \phi_4, \phi_5\}$. By setting $\mathbf{min}_{\text{Coverage}} = 1$, we can ensure that the monitor deployment will cover all events in Φ.

Solving the optimal deployment equation yields the deployment $M_d = \{m_1, m_3, m_4\}$. In this case, the deployment does not differ from the intuitive deployment. 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.

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 might 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 $\mathbf{w}_{\text{Red}_{\phi_1}} = \mathbf{w}_{\text{Red}_{\phi_4}} = 1$.

Solving the optimal deployment equation yields the deployment $M_d = \{m_3, m_4\}$. Here, again, the deployment is the same as the intuitive deployment, but our methodology also quantifies the redundancy values for the two events. For the case study model constructed as described above, $\text{Red}(\phi_1, M_d) = 3$ and $\text{Red}(\phi_4, M_d) = 3$.

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 might 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 his or her 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, setting $\mathbf{w}_{\text{Red}_{\phi_1}} = \mathbf{w}_{\text{Red}_{\phi_2}} = \mathbf{w}_{\text{Red}_{\phi_5}} = \mathbf{w}_{\text{Conf}_{\phi_1}} = \mathbf{w}_{\text{Conf}_{\phi_2}} = \mathbf{w}_{\text{Conf}_{\phi_5}} = 1$.

Solving the optimal deployment equation yields the deployment $M_d = \{m_1, m_2, m_4\}$. Further examination shows that compared to deploying only m_1 and m_4, deploying m_3 increases the redundancy of ϕ_1 by 1 and deploying m_2 increases the redundancy of ϕ_5 by 1. Since m_2 has a lower cost than m_3 and the added utility from each is identical, deploying m_2 is more cost-effective, even if this result may not seem obvious.

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 a study of monitor deployment with different sets of intrusion detection requirements or different parameter values for the same system.

VII. Scalability Analysis

We now evaluate the scalability of our approach to models of realistic systems, which can contain hundreds of monitors, indicators, and events.

A. Complexity Analysis

To solve the optimal deployment programs, we implement a version of the branch-and-bound algorithm that performs a search over the hypercube representing the state space of all possible monitor deployments and prunes the set of possible deployments when the minimum metric value constraints are not met. The algorithm, given by Algorithm 1, starts at the deployment representing all monitors and traverses the edges of the hypercube using a breadth-first search, keeping track of the best deployment it has seen up to that point and terminating branches of the search when it encounters a deployment that does not meet the minimum metric value requirements or does not improve utility beyond the current best deployment. The utility function used is given by the objective function of Equation 17.

Since our optimal deployment programs use the metrics' values as objective and constraint functions, the programs have nonlinear, non-convex objective and constraint functions. Therefore, it is not possible to use convex optimization techniques (such as interior point methods) or linear 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 not useful, since the metric functions are discontinuous over the search space.

The algorithms used to compute the metric values are direct implementations of the metric definitions, so we elide them in this paper. However, we provide their worst-case computational complexities here for analysis of the algorithms used to solve the optimization problems. The complexity of the algorithm to compute coverage is $O(|M||I| + B|I|)$, where $B = \sum_{\phi\in\Phi}|\beta(\phi)|$. The complexity of the algorithms to compute redundancy and confidence for an event ϕ is $O(|M||I| + |\beta(\phi)||I|)$. Finally, the complexity of the algorithm to compute cost is $O(|M|)$. The full algorithms and derivations of these complexities can be found in [24].

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

Algorithm 1 Branch-and-bound exact solution algorithm to compute the optimal monitor deployment

```
function FindOptimalDeployment($\Phi$, $M$)
    $queue \leftarrow \varnothing$
    add $M$ to $queue$
    $best \leftarrow \varnothing$
    $visited \leftarrow \varnothing$
    while $queue$ is not empty do
        $M_d \leftarrow$ POLL($queue$)
        if $M_d$ in $visited$ then
            continue to next deployment in $queue$
        end if
        $visited \leftarrow visited \cup M_d$
        for all $\phi$ in $\Phi$ do
            $redundancy[\phi] \leftarrow$ Redundancy($\phi$, $M_d$)
            if $redundancy[\phi] < \mathbf{min}_{\text{Redundancy}_\phi}$ then
                $visited \leftarrow visited \cup \binom{M_d}{|M_d|-1}$
                continue to next deployment in $queue$
            end if
            $confidence[\phi] \leftarrow$ Confidence($\phi$, $M_d$)
            if $confidence[\phi] < \mathbf{min}_{\text{Confidence}_\phi}$ then
                $visited \leftarrow visited \cup \binom{M_d}{|M_d|-1}$
                continue to next deployment in $queue$
            end if
        end for
        $coverage \leftarrow$ Coverage($\Phi$, $M_d$)
        if $coverage < \mathbf{min}_{\text{Coverage}}$ then
            $visited \leftarrow visited \cup \binom{M_d}{|M_d|-1}$
            continue to next deployment in $queue$
        end if
        $cost \leftarrow$ Cost($M_d$)
        if maxCost exists then
            if $cost <$ maxCost and Utility $(M_d) >$ Utility $(best)$ then
                $best \leftarrow M_d$
            end if
        else
            if $cost <$ Cost($best$) then
                $best \leftarrow M_d$
            end if
        end if
        if $best = M_d$ or $\nexists$maxCost or $cost >$ maxCost then
            $visited \leftarrow visited \cup \binom{M_d}{|M_d|-1}$
        end if
    end while
    return best
end function
```

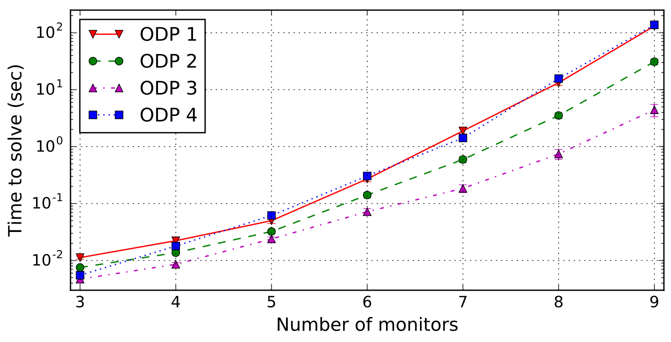

Fig. 4. Average time taken by Algorithm 1 to solve each of the optimal deployment equations given in Section VII-B with 10 events for $N = 100$ simulations. Error bars indicate the 95% confidence interval. Time to solve increases superexponentially with the number of monitors.

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 $O\left(2^{|M|}\,|I|\,(|M|+|\Phi|)\right)$.

B. Experimental Evaluation

We evaluated the scalability of our exact solution algorithm by running the algorithm on randomly generated models with differing numbers of events and monitors.

When generating test models for the scalability analysis, we observed that in our case study and in enterprise networks, many monitors produce redundant or complementary information for the detection of individual events, so it is possible to choose to deploy a small number of the monitors to ensure detection of each individual event. Using this observation, we generated models in which each event could be detected using combinations of indicators generated by a constant number of monitors, where the set of monitors is different for each event.

For all test models, all monitors have either a low, medium, or high resource utilization, with equal probability for each, and are deployed to an asset with low, medium, or high resource costs, with equal probability for each. Each monitor also has a uniformly random truthfulness, and the number of indicators generated by each monitor is randomly generated from a truncated normal distribution. The number of indicators in the model follows the truncated normal distribution with mean proportional to the number of monitors.

The indicators required to detect each event were determined as follows. First, a constant number of monitors were chosen uniformly at random from all monitors. A uniformly random number of indicators were selected from each of the monitors, and a constant number of minimal indicator sets were chosen uniformly from the power set of the selected indicators. In this way, each event can be detected using a combination of a small number of monitors.

The following intrusion detection requirements were used to construct the optimal deployment programs (ODPs) used in the scalability experiments. These requirements were chosen to evaluate the performance of the solution algorithm on a range of different types of detection goals that would be seen in enterprise systems.

ODP 1) Cost-constrained maximum coverage: $\mathbf{w}_{\text{Cov}} = 1$, $\mathbf{maxCost} = \$100$

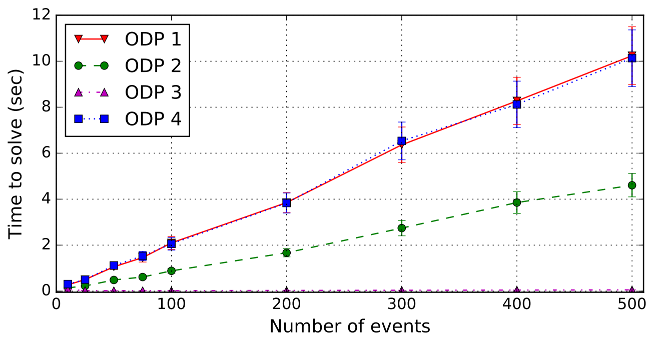

Fig. 5. Average time taken by Algorithm 1 to solve each of the optimal deployment equations given in Section VII-B with 6 monitors for $N = 100$ simulations. Error bars indicate the 95% confidence interval. Time to solve increases linearly with the number of events.

ODP 2) Cost-constrained maximum coverage, where two random events *must* be detected: $\mathbf{w}_{\text{Cov}} = 1$, $\mathbf{min}_{\text{Red}_i} = \mathbf{min}_{\text{Red}_j} = 1$ for random i and j, $\mathbf{maxCost} = \$100$

ODP 3) Unconstrained-cost detection of $\frac{1}{3}$ of the events with a constant minimum redundancy: $\mathbf{min}_{\text{Red}_i} = 2$ for i in a random selection of $\frac{1}{3}$ of the events

ODP 4) Cost-constrained maximum overall detection ability: $\mathbf{w}_{\text{Red}_\phi} = \mathbf{w}_{\text{Conf}_\phi} = 1$ for all events ϕ, $\mathbf{maxCost} = \$100$

The value for $\mathbf{maxCost}$ was chosen such that the expected number of monitors that can be deployed is 4.

We ran the exact solution algorithm for each of the four optimal deployment equations on 100 randomly generated models, noting the time taken to find the solution or decide that no valid solution existed. The code to generate the models and solve the optimal deployment equations was written in Python, and the experiments were run on a single thread of a Windows 7 machine with an Intel Core i7 950 3.20 GHz quad-core processor and 9 GB of RAM. The results of the experiment are shown in Figures 4 and 5.

The time taken to solve the optimal deployment equations increased to the order of hours per test model for models containing 10 monitors, so the simulations were run only for models with up to 9 monitors. From the graph, it is evident that Algorithm 1 scales superexponentially in the number of monitors, and linearly in the number of events, as expected.

C. Heuristic Algorithm

The exact solution algorithm given above quickly becomes intractable for models with a large number of monitors, limiting its usability in real systems. In order to improve the scalability of our approach, we use a greedy algorithm that attempts to maximize the utility gained by adding a monitor at each step of its operation. The greedy algorithm utility is given by Equation 19. Adding the constraint values to the utility allows the algorithm to consider progress towards meeting the minimum metric value requirements in addition to increasing the overall utility when choosing which monitors to deploy.

Algorithm 2 starts with an empty monitor deployment, and continues choosing the monitor with the highest value in each iteration until either the cost of adding another monitor exceeds the maximum cost for cost-constrained deployment, or the minimum metric value constraints are met for unconstrained-cost deployment. The value of a monitor is given by the ratio

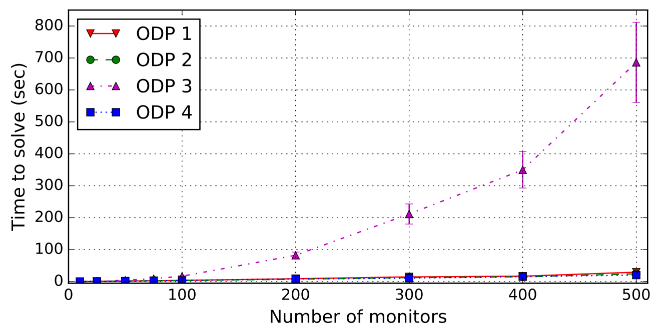

Fig. 6. Average time taken by Algorithm 2 to solve each of the optimal deployment equations given in Section VII-B with 50 events for $N = 100$ simulations. Error bars indicate the 95% confidence interval. Time to solve increases polynomially with the number of monitors.

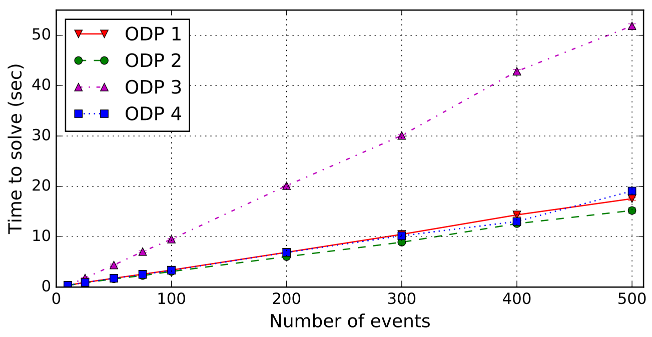

Fig. 7. Average time taken by Algorithm 2 to solve each of the optimal deployment equations given in Section VII-B with 50 monitors for $N = 100$ simulations. Error bars indicate the 95% confidence interval. Time to solve increases linearly with the number of events.

of the increase in utility obtained by deploying the monitor to the cost of the monitor.

$$\begin{aligned}\text{GreedyUtil} = {} & \mathbf{w}_{\text{Cov}}\mathbf{Cov}\left(\Phi, M_d\right) + \\ & \min\left(\mathbf{min}_{\text{Cov}}, \mathbf{Cov}\left(\Phi, M_d\right)\right)\mathbf{Cov}\left(\Phi, M_d\right) + \\ & \sum_{\phi\in\Phi}\Big(\mathbf{w}_{\text{Red}_\phi}\mathbf{Red}\left(\phi, M_d\right) + \\ & \quad \min\left(\mathbf{min}_{\text{Red}_\phi}, \mathbf{Red}\left(\phi, M_d\right)\right)\mathbf{Red}\left(\phi, M_d\right) + \\ & \quad \mathbf{w}_{\text{Conf}_\phi}\mathbf{Conf}\left(\phi, M_d\right) + \\ & \quad \min\left(\mathbf{min}_{\text{Conf}_\phi}, \mathbf{Conf}\left(\phi, M_d\right)\right)\mathbf{Conf}\left(\phi, M_d\right)\Big)\end{aligned} \tag{19}$$

1) Comparison of error with exact solution: To show that the greedy algorithm performs well when compared to the exact solution algorithm, we compare the objective function values obtained by the two algorithms when run on the same model with the same constraints. We generated 100 models with 7 monitors and 100 events, and ran both Algorithms 1 and 2 on the models. Our experimental setup was the same as for the scalability experiments for the exact solution algorithm.

The results of the experiment are given in Table III. The two error metrics of interest are the relative error in the objective function value for the solution given by Algorithm 2 and the percentage of models for which the greedy algorithm could not find a solution when one existed. Overall, for the experiments performed, the average error in the objective function was less than 2.5%. While a few models were misclassified by the heuristic algorithm, in practice, slight relaxation of the cost constraint would likely yield more accurate results.

Algorithm 2 Greedy algorithm to compute the optimal monitor deployment

```
1:  function FINDOPTIMALDEPLOYMENT($\Phi$, $M$)
2:      add all monitors in $M$ to set $S$
3:      $M_d \leftarrow \varnothing$
4:      while $S \neq \varnothing$ do
5:          for all $m$ in $S$ do
6:              $m_{new} \leftarrow M_d \cup m$
7:              if $\nexists$maxCost or COST($m_{new}$) < maxCost then
8:                  $value_m \leftarrow \frac{\text{GreedyUtil}(m_{new}) - \text{GreedyUtil}(M_d)}{\text{COST}(m)}$
9:              end if
10:         end for
11:         if $\exists m_{new}$ s.t. COST($m_{new}$) < maxCost then
12:             $M_d \leftarrow M_d \cup$ monitor with highest $value$
13:             if metric value constraints are met by $M_d$ and $\nexists$maxCost then
14:                 return $M_d$
15:             end if
16:         end if
17:     end while
18:     if metric value constraints are met by $M_d$ then
19:         return $M_d$
20:     end if
21: end function
```

2) Scalability analysis: Finally, we examine the scalability of the heuristic algorithm with respect to model size. In each iteration of the outer loop of the heuristic algorithm, in the worst case, each of the metric values is computed for new monitor deployments for each of the monitors in S. Since S starts with all deployable monitors in M and decreases by one at each iteration, the two outer loops of the algorithm have a computational complexity of $O\left(|M|^2\right)$. From the complexity equations given in Section VII-A, the overall worst-case computational complexity of Algorithm 2 is therefore $O\left(\left(|M|^3 + B|M|^2\right)|I|\right)$.

Figures 6 and 7 show the results of running the experiment described in Section VII-B using Algorithm 2 instead of Algorithm 1. Since the complexity of the greedy algorithm is lower, we were able to run the experiment for a larger number of monitors and events. The results show that the algorithm scales polynomially with respect to the number of monitors and events, matching the theoretical analysis. Even for models with 200 monitors and 50 events, the heuristic algorithm takes on average less than a minute to run.

To test scalability in an extreme case, we also ran the greedy algorithm on models with 500 monitors and 250 events. For the cost-constrained programs (ODPs 1, 2, and 4), on average, for 100 runs, the heuristic algorithm took 68.06, 80.55, and 82.10 seconds, respectively. For ODP 3, on average, the algorithm took 1000.37 seconds. The difference in running time for ODP 3 stems from the fact that the number of monitors to deploy is not limited to 4 by the value of **maxCost**, as it is for ODPs 1, 2, and 4. From those results, we argue that for reasonably large systems with hundreds of monitors, our algorithm could be run in an online fashion as the system model changes to determine where best to place monitors.

We leave further investigation of other search algorithms and heuristic approaches to future work.

VIII. Conclusions and Future Work

This paper presents a quantitative methodology to determine maximum-utility, cost-optimal deployments of security

TABLE III. SOLUTION ERROR FOR ALGORITHM 2

	ODP 1	ODP 2	ODP 3	ODP 4
% difference in objective function value	1.800%	2.647%	0.362%	3.652%
% of models misclassified	0%	3%	0%	0%

monitors in networked distributed systems in terms of intrusion detection goals. We show the need for our approach by observing that current approaches to monitor deployment either do not consider the cost of monitoring or deal only with network IDS monitors, neglecting the utility of other types of monitors.

Our methodology consists of a system and data model, a set of monitor utility and cost metrics, and two optimization problems that combine the model and metrics with intrusion detection requirements and can be solved to find optimal monitor deployments. We illustrate the efficacy and practicality of our approach through a case study in which we programmatically generated the components of our model from logs captured from a running Web application. We show the scalability of our approach by providing an algorithm that can solve the optimization problems in minutes for large systems with small error. Through our examples, we demonstrate that our methodology gives practitioners a way to easily express intrusion detection requirements and determine 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.

In future work, we plan to investigate metrics other than the three we introduce here, including those related to increasing the granularity of information for forensic analysis. We are also investigating techniques to easily generate models for large systems given system and monitor configuration information to reduce the burden on system administrators.

ACKNOWLEDGMENTS

We would like to thank Brett Feddersen for his help implementing the experiments for this research. We would also like to thank the other members of the PERFORM group for their insightful comments that helped improve the work, and Jenny Applequist for her editorial revisions to the paper.

This material is based in part on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining audit data to build intrusion detection models." in *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1998, pp. 66–72.

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

[3] 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*, 2014, pp. 29–40.

[4] 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.

[5] Q. Zhu and T. Basar, "Indices of power in optimal IDS default configuration: Theory and examples," *arXiv:1110.1862 [cs]*, Oct. 2011, arXiv: 1110.1862.

[6] 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.

[7] 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.

[8] 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.

[9] S. Schmidt, T. Alpcan, A. Albayrak, T. Basar, 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. Hammerli, Eds. Springer Berlin Heidelberg, 2008, no. 5141, pp. 311–326.

[10] 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.

[11] M. Bloem, T. Alpcan, S. Schmidt, and T. Basar, "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.

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

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

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

[15] SilverSky, "Obtaining Fortune 500 security without busting your budget," SilverSky, Whitepaper, 2015. [Online]. Available: http://www.staging.silversky.com/sites/default/files/Fortune_500_Security_ROI_White_Paper_0.pdf

[16] 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. Me, and S. F. Wu, Eds. Springer Berlin Heidelberg, Jan. 2000, no. 1907, pp. 197–216.

[17] B. Morin, L. M, H. Debar, and M. Ducass, "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. [Online]. Available: http://www.springerlink.com/index/cwp428tlhf35rwba.pdf

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

[19] 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, 2013, pp. 311–12.

[20] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=283845

[21] "Metasploitable 2 Exploitability Guide | Rapid7 Community," 2015. [Online]. Available: https://community.rapid7.com/docs/DOC-1875

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

[23] "Top 10 2013: OWASP," 2015. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013

[24] U. Thakore, "A quantitative methodology for evaluating and deploying security monitors," Master's Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Jul. 2015. [Online]. Available: http://hdl.handle.net/2142/88103