IMPROVING RELIABILITY AND SECURITY MONITORING IN ENTERPRISE AND CLOUD SYSTEMS BY LEVERAGING INFORMATION REDUNDANCY

BY

UTTAM THAKORE

DISSERTATION

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

Urbana, Illinois

Doctoral Committee:

Professor William H. Sanders, Chair
Professor Indranil Gupta
Professor Klara Nahrstedt
Dr. Rohit Ranchal, IBM Cloud
Dr. Harigovind V. Ramasamy (non-voting), Amazon

ABSTRACT

As computing has become critical to all areas of modern life, the need to ensure the security and reliability of the underlying information technology infrastructures is greater than ever before. Large-scale enterprise and cloud systems, which form the backbone for the majority of computing activity, consist of many components and services interacting in complex and sometimes unpredictable ways. As such systems have grown in size, scale, and complexity, they have become increasingly difficult to protect against security and reliability incidents, resulting over recent years in ever more frequent service disruptions, failures, and data breaches, the financial and societal implications of which are massive. System owners have a strong desire to prevent such incidents.

Incident detection and response and compliance audit are the two primary mechanisms by which organizations enforce reliability and security policies and make their systems more resilient. Both the academic and professional communities have focused considerable attention on developing techniques to improve incident detection, incident root cause analysis, and compliance auditing, often with little consideration for the cost of the monitoring that is required to support them. Furthermore, as the scale and complexity of systems have increased, so too have the scale and complexity of their monitoring infrastructures. Monitors can fail or be compromised, and monitor data must be selectively collected to avoid exceeding storage and processing limits. Consequently, it has become increasingly important to explicitly consider the efficiency, efficacy, and resiliency of monitoring systems when one is designing large-scale enterprise and cloud systems.

In this dissertation, we address inefficiencies and inadequacies in reliability and security monitoring in enterprise and cloud systems by leveraging redundancy of information across diverse monitors. In particular, we use the redundancy of data generated by different monitors 1) to facilitate more effective and efficient use of the data in meeting reliability and security objectives, and 2) to improve the resiliency of the monitoring infrastructure itself against failures and attacks.

First, we present a framework for simplifying the complexity of data analysis for incident response in enterprise cloud systems. As a foundation for the framework, we define a general taxonomy for fields within monitor data that administrators can use to label both structured and unstructured components of data. We then present a method to automatically extract

time series features based on labels from our taxonomy, remove uninformative features, and reduce the overall number of features by clustering together related and redundant features. We apply our framework to logs and metrics collected during reliability incidents from all levels of an experimental platform-as-a-service cloud at a large computing organization, and demonstrate that our approach enables efficient coordinated analysis of both metric data and log data. Such analysis typically presents a challenge to cloud support engineers, but can identify meaningful relationships between features that can aid in incident response.

Next, we present a systematic methodology that enables system administrators to design monitoring systems that are resilient to missing data. We develop a model-based approach to quantify the resilience of a system's monitoring and incident detection infrastructure design against missing data, using which we develop a method to find monitor deployments that maximize resilience subject to monitoring cost constraints. We illustrate how our approach can be applied to production systems by using a datacenter network case study model based on monitors employed in production systems, and we evaluate its scalability by using randomly generated models of varying sizes and structures. We compare our approach to the current state of the art and demonstrate that our approach consistently finds monitor deployments that are more resilient under the same constraints.

Finally, we address the inefficiencies faced by a cloud service provider (CSP) during audit evidence collection as a result of a poor understanding of evidence requirements. We motivate our analysis by developing a taxonomic framework for understanding the causes of and potential solutions to uncertainty in audit. We present a model-driven method to learn evidence sufficiency requirements directly from historical audit records. We then apply our cost-optimal resilient monitoring approach to the evidence sufficiency model to determine an efficient evidence collection strategy for the CSP. We apply our approach to the historical audit records from an enterprise infrastructure-as-a-service cloud system at a large computing organization and demonstrate how use of our approach could have enabled more efficient evidence collection.

We believe that our work clearly demonstrates the need to critically examine the resiliency and efficiency of monitoring infrastructures in enterprise and cloud systems. This dissertation presents solutions to specific challenges faced by practitioners when monitoring their systems for reliability and security objectives, but our work addresses only part of the larger problem space of resilient monitoring system design. We hope that this dissertation paves the way for future research that focuses on the resilience of the monitoring infrastructure itself.

To my parents, sister, and late grandmother, for their limitless love and support.

ACKNOWLEDGMENTS

First, I would like to thank my adviser, Prof. William H. Sanders, for his consistent support throughout my Ph.D. Bill gave me the freedom to explore and pursue my own research interests and provided a supportive environment in which I could grow into an independent researcher. I am grateful for his guidance, encouragement, and patience over the years. I would also like to thank the other members of my doctoral committee, Prof. Klara Nahrstedt, Prof. Indranil Gupta, Dr. Rohit Ranchal, and Dr. Harigovind V. Ramasamy, for their insightful feedback and guidance on my dissertation work and for their advice as I decided on my career path.

I would especially like to thank my mentors at IBM, Dr. Harigovind V. Ramasamy and Dr. Rohit Ranchal. Much of the work in this dissertation emerged from projects I started while interning at IBM over two summers. Since I first reached out to Hari early in my Ph.D. journey, he has been one of my strongest and most enthusiastic advocates. It was through Hari's support that I first interned at IBM and established a collaboration that extended throughout my Ph.D. Both Hari and Rohit spent countless hours mentoring me and providing detailed technical feedback and hands-on guidance on my work. They also spent many a late night with me working on paper and patent submissions. I am eternally grateful for all of the opportunities that they created for me and the time they so generously gave me. I also thank my many other IBM colleagues, including Dr. Yu (Jason) Gu, Dr. Mahesh Viswanathan, and Dr. Yi-Hsiu Wei. This work would not have been possible without all of them.

I feel very fortunate to have been part of the Performability Engineering Research Group (PERFORM), where I have made many lifelong friends. Their company has enriched my time at Illinois, and their feedback and discussions have helped me become a better presenter and researcher. I thank Mohammad Noureddine for being a great officemate and friend in the last few years of my Ph.D. and for opening my eyes to the world of philosophy. I also thank Ahmed Fawaz, Carmen Cheh, Atul Bohara, and Ben Ujcich for being fantastic collaborators and wonderful friends. I thoroughly enjoyed all of the long conversations we had over the years about research, food, arts, politics, and life in general. Additionally, I thank Varun Badrinath Krishna, Michael Rausch, Brett Feddersen, Ken Keefe, Robin Berthier, Gabriel Weaver, Ron Wright, and David Huang for all manner of lively discussions, both inside and outside the office. Finally, I thank Jenny Applequist for her editorial comments on all of my

papers and this dissertation. No matter how tight the deadline, Jenny never failed to provide feedback and help me improve the quality of my writing. I am immensely grateful for her patience with me over the years.

My time in graduate school was made much more enjoyable by all of the friends I made at Illinois. In particular, I extend my warmest thanks to my friends in the DEPEND group, Arjun Athreya, Saurabh Jha, Subho Banerjee, Zachary Stephens, Krishnakant Saboo, Yogatheesan Varatharajah, Valerio Formicola, and Zak Estrada, who were always there to help me escape the office and decompress after a long day of work. Some of my fondest memories at Illinois are from the time I spent with all of them. I also thank the friends I made through ICPC, Jingbo Shang, Timothy Smith, Yong Hong, and many others; and my friends from back home, Danielle Howard and Yasmin Khan.

Finally, most of all, I would like to thank my wonderful family for being a source of emotional support, motivation, and unconditional love throughout my life. Since I was little, my father and mother both instilled in me a strong work ethic and an appreciation for the value of knowledge. They always encouraged me to reach higher and work harder. Over the course of my Ph.D., whenever I encountered obstacles or felt overwhelmed, I could always count on my father to provide words of wisdom to help me power through and my mother to help me keep perspective. My younger sister, Juhi, has always been my closest friend and my loudest cheerleader. She has always been there whenever I wanted to vent my frustrations, needed a laugh, or just generally wanted to geek out about our latest obsessions and hobbies. Conversations with her never failed to put a smile on my face. I owe all of my achievements, both past and future, to my family. Without their support and encouragement, I would never have come as far as I have.

This dissertation is based on work supported by the Army Research Office under Award No. W911NF-13-1-0086; by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084; and by the generous support of an IBM Ph.D. Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the Army Research Office or 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

As the use of computing has expanded into domains involved in all aspects of life (e.g., healthcare, finance, the military, and critical infrastructure), the need to ensure the security and reliability of the underlying information technology (IT) infrastructures is greater than ever before. Today, large-scale enterprise and cloud systems form the backbone of most organizations. Such systems consist of many different hardware and software components that are controlled and managed by multiple parties interacting in complex and sometimes unpredictable ways.

Enterprise systems can contain hundreds of employee-controlled hosts that may run arbitrary software and perform arbitrary roles in the network, requiring complex permissions and interaction with other services in the network. Enterprise systems often contain sensitive information about employees and customers that must be protected against exfiltration, and services that must be protected against loss of availability and poor performance, whether due to malicious attacks or random faults or failures.

Cloud systems are shared IT resources that provide IT functionality to customers as a service. The hosts and network hardware within a cloud system are owned and operated by the cloud service provider, and customers access the services provided by the cloud remotely over the Internet. The software, services, and hosts in cloud systems are generally less diverse than those in enterprise systems, but they still have many, complex interdependencies, and clouds often have much larger scale. Cloud services underpin an increasing percentage of commercial computing services, including those in sensitive domains, so high availability, reliability, and security of information stored within cloud systems are of the utmost importance. Furthermore, cloud service models such as hybrid and multi-cloud, which blur the line between enterprise and cloud systems, are becoming more popular as companies move away from in-house systems.

Clouds and enterprise systems are faced with many security and reliability threats that span multiple system components. On the reliability side, performance and reliability issues, resource exhaustion, and intermittent or persistent bugs in one part of the system (e.g., the network) can often impact the behavior of other parts of the system (e.g., the application software), or even cause the system to fail [1]. On the security side, sophisticated, stealthy attackers called Advanced Persistent Threats (APTs) [2] use difficult-to-detect means such as social engineering to gain a foothold into a system, after which they move laterally through

the system, take unauthorized actions, and ultimately exfiltrate sensitive information or cause service disruptions for financial or political motives. One defining characteristic of APTs is their strong desire to avoid detection.

The financial and societal implications of service disruptions, failures, and data breaches, collectively called *security and reliability incidents*, can be massive. According to a recent report by Lloyd's of London, a cyber incident that causes 3 – 6 days of service unavailability of a top-three cloud provider (i.e., Amazon EC2, Microsoft Azure, or Google Cloud) could cause industry losses of up to $15 billion [3]. As a consequence, system owners have a strong desire to prevent such incidents.

1.1 RELIABILITY AND SECURITY IN ENTERPRISE AND CLOUD SYSTEMS

In large enterprise and cloud systems, reliability and security objectives are traditionally specified at the organizational level in terms of IT security policies and service-level agreements. IT security policies establish the rules and procedures that must be imposed on all individuals who access the system in order to preserve the confidentiality and integrity of sensitive information stored within the system while maintaining availability of the services provided by the organization [4]. Security policies also often establish requirements for auditability of user actions and timely detection of violations of security policy. Service-level agreements (SLAs) are agreements that the enterprise or cloud service provider makes with its customers that specify the availability and reliability requirements for the services provided to the customer. SLAs may also be established between divisions or business units within an organization for IT services provided for internal consumption. Reliability objectives are also sometimes specified by IT disaster recovery policies, which establish strategies on how to recover from the loss of system components while maintaining organizational service level requirements. Organizational policies and SLAs are defined based on business requirements. In many domains, they may also be defined based on external regulatory or industry standards that specify minimum system requirements for operation in the given domain (e.g., HIPAA [5] for healthcare, and PCI-DSS [6] for credit card processing).

The two primary mechanisms that are used within enterprise and cloud systems to enforce organizational policies and SLAs are *incident detection and response* and *compliance audit*. Figure 1.1 illustrates the components comprising the two mechanisms. Incident detection and response constitute the in-band, operational component of system reliability and security, by which policy or SLA violations (or issues that could lead to future violations) are detected

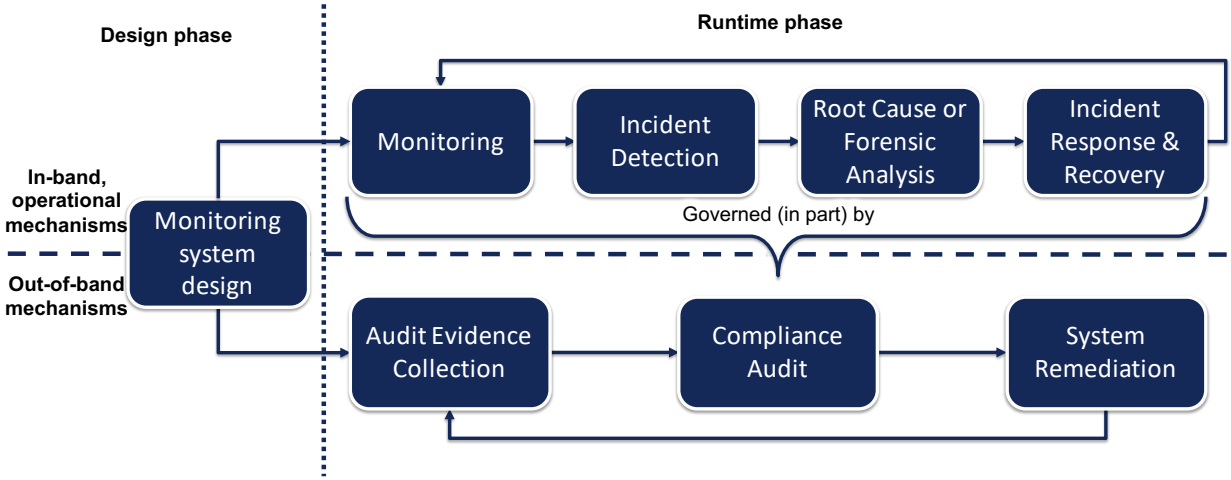


Figure 1.1: Illustration of the components of security and reliability policy enforcement.

and mitigated in a timely manner. Compliance audit constitutes the out-of-band component, by which the organization or its customers can verify that mechanisms are in place to enact the reliability and security policies, detect violations of said policies, and establish audit trails for policy violations. Both are necessary components for proper enforcement of security and reliability policies.

Incident detection, investigation, and response. The goal of incident detection is to identify the signs of attacker activity or performance and reliability issues as *early* as possible, and ideally before the attacker is able to cause a data breach or reliability issues cascade into a catastrophic failure. When problems in cloud and enterprise systems do result in a breach or failure, the circumstances leading up to the disruption must be investigated post hoc to identify the underlying root causes of the incident so that engineers can fix the associated problems. That process is called *root cause analysis* (*RCA*) or *digital forensic analysis*.

In many enterprise and cloud systems, incidents are often detected by customers and reported to the service provider, where they are handled by a dedicated team of administrators called an *incident response team* (*IRT*), *incident management team* (*IMT*), or *information security team* (*IST*). In some enterprise systems, security incident management is handled by a third-party organization called a *security operations center* (*SOC*). Administrators in IRTs, IMTs, ISTs, and SOCs use an array of alerting tools, dashboards, and log analysis tools to identify potentially relevant log events and dig into logs, often in an ad hoc manner, to trace the system's behavior backwards in time to the point at which the initial failure

or attack took place. Often, the procedure requires attempts to replicate the issue to test hypotheses and collect data that were not originally collected during the incident.

Compliance audit. The goal of compliance audit is to verify that an IT system implements controls to prevent reliability and security incidents and that it has the ability to detect, respond to, and recover from such incidents. During compliance audit, the organization that owns the system collects evidence demonstrating that controls are in place to enforce a given set of reliability and security requirements, and an auditor reviews the evidence to verify compliance. Undergoing compliance audit can uncover weaknesses in a system's enforcement of reliability and security policies that may not be discovered during investigation of and response to operational incidents, so audit serves as an crucial mechanism to achieve reliability and security objectives.

The evidence used for compliance audit takes myriad forms, ranging anywhere from organizational policy and process documents to host and network configuration files to user activity and system event logs. Due in large part to the diversity of evidence types, evidence collection generally constitutes the bulk of the time needed for compliance audit.

Organizations that offer IT services undergo compliance audit to provide assurances to their clients and customers that their systems are indeed trustworthy and reliable, so as to increase customer adoption of their services. In enterprise and cloud systems, compliance audit is often conducted for the purpose of acquiring compliance certifications for regulatory and industrial standards, such as HIPAA [5] for healthcare systems, PCI-DSS [6] for credit card handlers, and FedRAMP for U.S. federal agencies [7]. Compliance certification has large financial implications; for example, in order to compete for the recent $10 billion JEDI contract with the U.S. federal government, cloud providers were required to achieve FedRAMP High Baseline certification for their cloud services [8].

The importance of monitoring. Proper monitoring is necessary for incident detection, root cause/forensic analysis, and compliance audit, which we collectively call the reliability and security objectives of system administrators. In the case of incident detection, in order to observe the signs of reliability and security issues, it is necessary for the correct monitors to be deployed in the system. In the case of root cause and forensic analysis, sufficient monitor data must be collected during system operation to reveal after the fact what took place, but administrators must also be able to sift through the information in a timely manner to identify which data are meaningful. In the case of compliance audit, sufficient

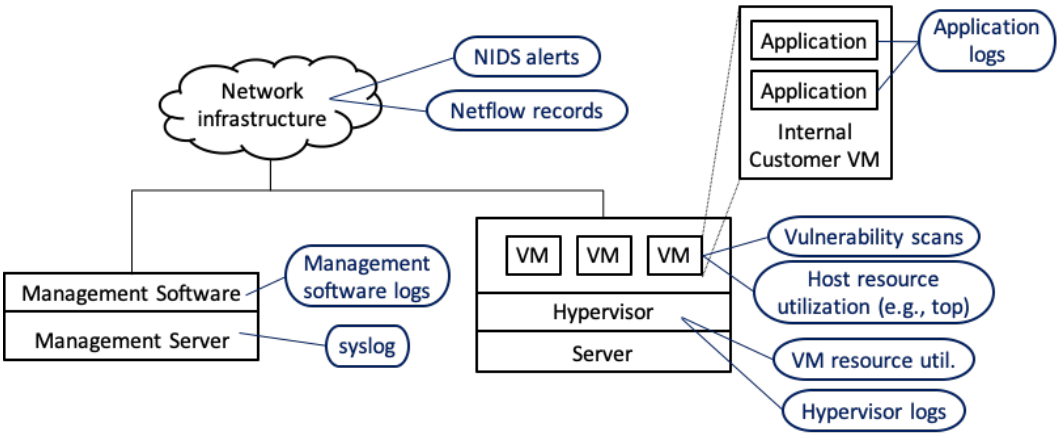


Figure 1.2: Illustration of monitor diversity in a hypothetical IaaS cloud environment. Monitors are denoted by blue ovals beside the system components on which they would run. The monitors shown constitute a fraction of the possible monitors that could be deployed.

evidence must be collected to pass the audit, but compliance experts must also be able to satisfactorily demonstrate to auditors that the evidence collected indeed satisfies the compliance requirements within the standards.

Thus, we observe that *adequate and actionable monitor data are prerequisites for ensuring the security and reliability of enterprise and cloud systems.* In this dissertation, we discuss the challenges present in reliability and security monitoring for enterprise and cloud systems and how we address them by leveraging redundancy of monitor data across heterogeneous monitors.

1.2 CHALLENGES OF ENTERPRISE AND CLOUD MONITORING

Proper monitoring for reliability and security objectives is challenging because monitoring is expensive, heterogeneous, and redundant. In enterprise and cloud systems, data for incident detection and response and audit compliance can come from an incredibly diverse set of monitors. Figure 1.2 illustrates some of the types of monitors that could be available in an infrastructure-as-a-service (IaaS) cloud system. Within IBM and the National Center for Supercomputing Applications at the University of Illinois, we have seen all of the following being used: network flow information, operating system logs, various application and event logs, application and host configuration files, periodic network scan logs, intrusion detection system (IDS) and Security Information and Event Manager (SIEM) rule sets (configurations) and alerts, e-mail communications between employees, and incident ticket records. It is not

feasible to collect or efficiently analyze all the information from all the monitors that could be deployed, nor is it necessarily known a priori which set of features will be important in detecting or responding to incidents that occur during system operation. Worse, much of the information across many monitor types is redundant, so blindly increasing monitoring may not provide additional ability to meet resilience objectives.

In production enterprise and cloud systems, we have observed that the largest factors in a system designer's decision about whether to deploy a monitor are 1) the *engineering costs* associated with installing the monitor, maintaining (i.e., updating, debugging, and refreshing) the monitor, storing the data generated by the monitor, and integrating the monitor with existing detection frameworks; and 2) the *human costs* (i.e., time and attention) associated with responding to alerts generated by the monitor and sifting through the monitor's data during incident analysis. The resource utilization costs of running the monitor also play a role, albeit a smaller one. Thus, the choice of what monitors to deploy is often influenced by the expertise of the system administrators and engineers, the existence of tools that automate the deployment and operation of the monitor, whether existing detection tools such as SIEMs and log analysis frameworks can ingest the monitor's data, and even whether a particular type of monitor is already deployed anywhere else in the system. Often, system designers default to monitoring based on vendor-specified best practices or commercial, off-the-shelf monitoring tools, and the decision of what to collect is often determined by default parameters.

Monitoring is often not treated as a first-class design requirement because it is not a functional requirement. Consequently, monitoring for internal systems (i.e., those that are not customer-facing) is often bare-bones and is re-evaluated only when IRTs are unable to respond to incidents because of insufficiency of monitoring. In systems that must be compliant with regulatory or industry standards, the bar for monitoring is often set by the evidence required for compliance audit.

Research in enterprise and cloud monitoring has focused primarily on methods for better utilization of already-collected semi-structured or unstructured log information [9] or for utilizing a larger number of already-collected data sources in domain-specific ways to accomplish specific detection goals [10, 11]. However, work on how to design the underlying monitoring infrastructure, or deal with its susceptibility to attack or failure, has been very limited.

Poorly reasoned decision-making about monitoring can lead to inefficiencies and inadequacies in attempting to meet resilience objectives. When monitoring is overprovisioned,

Challenge 1. Data are underutilized during detection and analysis because analysts cannot

identify meaningful features, combine heterogeneous information, or easily remove redundant information.

Challenge 2. Too much data are collected during monitoring because it is not clear a priori what is *required* for detection, forensics, or audit compliance, and what is *redundant*. The result is a slowing down of incident analysis and response times.

Challenge 3. If monitoring requirements are given in terms of legal documents, as in the case of regulations and compliance standards, for which satisfying human auditors is the ultimate objective, uncertainties about the requirements due to nontechnical factors result in repeated evidence collection and verification, which add unnecessary monitoring cost.

When monitoring is underprovisioned,

Challenge 4. Crucial data for incident response are not available when the system fails or an attack succeeds because sufficient monitoring had not been enabled.

Challenge 5. Overdependence on limited data and limited understanding of said overdependence breed a false sense of resilience against failures or compromise of the monitoring infrastructure itself, so early warning signs for catastrophic failures and serious attacks go undetected when monitors fail or are maliciously compromised [12, 13].

Challenge 6. Failure to collect sufficient evidence for compliance audit in a timely manner results in failure to achieve certification, which can have financial and legal consequences for a large cloud or enterprise.

1.3 DISSERTATION CONTRIBUTIONS

In this dissertation, we address the aforementioned inefficiencies and inadequacies of current cloud and enterprise monitoring approaches by leveraging the redundancy of monitor data across diverse monitors to our advantage. Our thesis is that:

The efficiency, efficacy, and resilience of reliability and security monitoring in enterprise and cloud systems can be improved by principled collection and fusion of redundant data across heterogeneous monitors.

In brief, our research contributions in this dissertation are the following:

- We develop a framework to semiautomatically process heterogeneous monitor data from multiple levels of a cloud platform into a manageable set of meaningful time series features useful in incident root cause analysis (addressing Challenges 1 and 2 above).
- We develop a model-based methodology for quantifying the resilience of a distributed system's incident and intrusion detection infrastructure against missing monitor data and suggesting changes to the monitoring that can maximize said resilience under monitoring cost constraints (addressing Challenges 2, 4, and 5).
- We develop a taxonomic framework for understanding the causes of and potential solutions to uncertainty in the enterprise and cloud system compliance audit process (addressing Challenge 3).
- We develop a technique for addressing compliance control interpretation-related uncertainties in evidence collection for compliance audit by learning a model of evidence sufficiency requirements from historical audit records, and using the model to enable cost-efficient evidence collection (addressing Challenges 2, 3, and 6).

In each of our contributions, we use the redundancy of data generated by different monitors to our advantage 1) to facilitate more effective and efficient use of the data in meeting reliability and security objectives, and 2) to improve the resiliency of the monitoring infrastructure itself against failures and attacks. Specifically, we employ the following principles about redundancy of monitor data in enterprise and cloud systems:

- Learning what data are redundant can enable more efficient monitoring and analysis by allowing system designers to avoid collecting duplicate information and by reducing the amount of data upon which analysis techniques must operate.
- Where data are redundant across *heterogeneous* monitors or monitors running on different hosts, the redundancy can provide insights about system behavior that facilitate more effective incident detection, incident analysis, and compliance audit.
- Collection of redundant monitor data, particularly from heterogeneous monitors, can enable resilience against loss of data.

To address the aforementioned monitoring challenges, we use a combination of model-based and data-driven techniques. Model-based approaches are good at representing detection

requirements and compliance requirements in unambiguous ways, and that facilitates quantitative analysis and decision-making. However, model-based approaches rely on accurate model construction, which can be prohibitively expensive and is subject to modeler assumptions, subjective interpretations, and bias. Manual modeling of log message formats, for example, is known to be challenging and error-prone in both anomaly and incident detection [14] and audit compliance [15]. In contrast, data-driven machine learning techniques are well-suited to problems involving uncertainty, noise, and scale, and can adapt to the unique characteristics of a particular system. Thus, we employ both model-based approaches and machine learning techniques in this dissertation.

1.3.1 Dissertation Organization

The rest of this dissertation can be divided by stage of system operation: Chapter 2 provides a data-driven technique that can be applied during system runtime, and Chapters 3 and 4 provide model- and data-driven techniques that can be applied during monitoring system design. The contributions in this dissertation can be summarized as follows.

Coordinated Analysis of Heterogeneous Monitor Data

In Chapter 2, we propose a framework for simplifying the complexity of data analysis for incident response in enterprise cloud systems. Our framework enables coordinated analysis of both metric (numerical) data and log (semi-structured, textual) data and exposes salient features within those data. As a foundation for the framework, we define a taxonomy for fields within monitor data based on insights gained from analyzing logs and metrics collected from all levels of an experimental platform-as-a-service (PaaS) cloud (EPC) at a large computing organization. Using the taxonomy, we lay out a method for semiautomated feature extraction and discovery across heterogeneous monitors. We then describe a method for feature clustering to promote effective analysis of the data, and to remove redundant and uninformative features. We discuss the application of our framework for effective incident response within the EPC, including root cause analysis. Clustering redundant time series features across different cloud monitors allows us to uncover relationships between features that are relevant for incident root cause analysis.

Resilience Against Missing Monitor Data

In Chapter 3, we present a model-based approach for 1) quantifying the resilience of a system's monitoring and incident detection infrastructure design against missing data, and 2) hardening system monitoring by maximizing resilience under monitoring cost constraints. We illustrate how our approach can be applied by using a datacenter network model that is based on monitors employed in production systems, and we evaluate its scalability by using randomly generated models of varying sizes and structures. We compare our approach to the current state of the art and demonstrate that our approach consistently finds more resilient monitor deployments under the same constraints. Strategically deploying redundant monitors using our approach allows system administrators to make more effective monitoring decisions and achieve resilience against the threat of missing monitor data.

Improved Efficiency of Audit Evidence Collection

In Chapter 4, we address inefficiencies faced by a cloud service provider during audit evidence collection that are caused by a poor understanding of what evidence is required to be certified as compliant. We first present a taxonomic framework for understanding the causes of and potential solutions to uncertainty in the audit process. We then describe our approach for learning a model of sufficient evidence from historical audit records and for determining more efficient evidence collection strategies for subsequent audits using the model. We apply our approach to the historical audit records from an enterprise infrastructure-as-a-service (IaaS) cloud system at a large computing organization and demonstrate how use of our approach could have enabled more efficient evidence collection. By identifying redundancy in the sets of evidence that have historically been used to demonstrate system compliance to auditors, we improve our understanding of evidence collection requirements and improve the efficiency of audit evidence collection.

Finally, in Chapter 5, we present our concluding remarks and discuss potential avenues for future work.

CHAPTER 2: ENABLING COORDINATED ANALYSIS OF HETEROGENEOUS MONITOR DATA

In large-scale enterprise platform-as-a-service (PaaS) clouds, large volumes of system data are collected for detection and investigation of undesirable incidents, such as failures, performance bottlenecks, and attacks. As cloud platforms grow in size and complexity, the diversity and quantity of monitors that can be deployed to collect data about system operation for incident analysis, as well as the number of features that can be extracted from the data, increase dramatically [14]. It is not always possible to efficiently analyze information from all monitors that can be deployed, nor is it necessarily known a priori which features will be important in detecting incidents that will occur during system operation.

Existing monitoring and analysis frameworks can automatically parse and analyze data from only a limited set of cloud monitors, and most segregate analysis of numerical and text data, limiting the types of analyses that can be done. Furthermore, existing tools provide limited guidance about which features are salient for reliability and security incident response. For a cloud provider collecting data from multiple monitors, most features are redundant or uninformative, often increasing incident response time. Downtime due to incident response can result in lost revenue and customer dissatisfaction.

Overview

In this chapter, we introduce a framework to semi-automatically process heterogeneous monitor data from multiple levels of a cloud platform into a manageable set of meaningful time series features useful in incident root cause analysis. We propose a general taxonomy for monitor data fields that administrators can use to easily label both structured and unstructured components of monitor data. We then present a method to automatically extract time series features based on labels from our taxonomy, remove uninformative features, and reduce the overall number of features by clustering together related and redundant features.

Our framework enables *coordinated analysis* of both metric (numerical) data and log (semi-structured, textual) data, which to the best of our knowledge is not supported by existing techniques or tools, and typically presents a challenge to cloud support engineers. It also (1) supports feature extraction from arbitrary cloud monitor types, (2) requires limited annotation or custom parsing by domain experts to operate on parameters within

unstructured log messages, (3) aids in the discovery of meaningful features while minimizing feature redundancy, and (4) identifies meaningful relationships between features that can aid in incident response. We demonstrate the utility of our framework on incident data collected from an experimental PaaS cloud service (EPC), which was serving as a development-test environment at a large computing organization. Our approach significantly reduces the manual effort needed of administrators to ingest and operationalize monitor data.

The rest of this chapter is organized as follows. In Section 2.1, we explain how monitoring and incident analysis are performed in practice and illustrate the challenges faced in the EPC. In Section 2.2, we describe the EPC service that motivated our work and provided the dataset we use for evaluation. Section 2.3 details our taxonomy of cloud monitor data fields and Section 2.4 describes our procedures for feature extraction and clustering. Finally, in Section 2.5, we illustrate our approach's ability to aid in feature discovery and incident analysis.

2.1 BACKGROUND AND CHALLENGES

Large-scale PaaS systems consist of many different hardware and software components interacting in complex and sometimes unpredictable ways. Performance and reliability issues, resource exhaustion, and intermittent or persistent bugs or malicious activity in one part of the system (e.g., the network) can often impact the behavior of other parts of the system (e.g., the application software), or even bring the system down.

In practice, monitor data are conventionally classified as either *metric data* or *log data* [16, 17]. At a high level, periodically sampled numerical measurements are considered to be metrics, and records of discrete events, whether structured or unstructured (i.e., textual), are considered to be logs. For example, host-level resource utilization values, such as those generated by the Linux TOP utility, are considered metrics, and logging messages generated by applications, such as those accumulated by the Linux SYSLOG utility, are considered logs.

Our work is motivated in part by the fact that existing log analysis approaches are specialized to individual log types, and cannot be easily adapted to operate on the wide range of heterogeneous data that are present in large-scale cloud systems. Some of the tools more commonly used in practice to process and analyze diverse log and metric data include Microsoft Azure Kusto [18], VMWare LogInsight [19], GrayLog [20], Prometheus [21], and Splunk [22]. Such tools are sometimes called AIOps (artificial intelligence for IT operations) tools if they use machine learning to process the data. Each of the tools can consume a

variety of logs by using community-provided input plugins. However, in most cases, the text components of log messages are not parsed by the plugins because maintainance of the complex regular expressions required for parsing is intractable, as logging statements are constantly being updated [9, 14]. That prevents utilization of parameter values present in the messages, even though those are often important, as we show later in this chapter. In addition, input plugins may be unavailable for many commercial software applications because of licensing constraints.

Further, logs and metrics are traditionally stored and analyzed separately. Within the EPC, metrics were stored in a Graphite [23] time series database and used to generate dashboards in Grafana [24] that administrators could monitor in real time, and to generate e-mail alerts when certain undesirable conditions were met. Logs were parsed and indexed using the ELK stack [25], after which some logs were used to create dashboards in Kibana [25] and some were filtered for error or failure keywords and used to generate e-mail alerts.

That dichotomy between log and metric analysis caused a number of problems. First, analysis was fragmented; since system administrators were required to maintain separate dashboards for log and metric data, it was not possible to directly compare the two, so investigation of incidents required an admin to switch back and forth between dashboards and matching alert timestamps. Similarly, since the data were stored in different databases in different formats, there was no way to query log and metric data conjunctly.

Second, log data were heavily under-utilized. In order to parse log messages, engineers needed to construct complex regular expressions. Some applications produce highly structured and detailed request and error logs, while others produce only logs with limited structured information (such as message severity level and error code) and predominantly unstructured information, which was typically examined only during manual incident analysis. That left important information unused, especially within parameter values embedded in log messages.

Importantly, the dichotomy between log and metric data was artificially imposed by the tools available to collect and analyze the data, rather than by how the data needed to be used during incident response. In many instances, administrators treated logs like "metrics." For example, the frequencies of different Apache response status codes were counted per 30-minute interval and monitored using a Kibana dashboard, as were the frequencies of the application server error log messages at `ERROR` and `FAILURE` severity levels. The admins needed a coordinated method to convert the data from different monitors in the EPC into a meaningful set of time series features that they could easily monitor during system operation

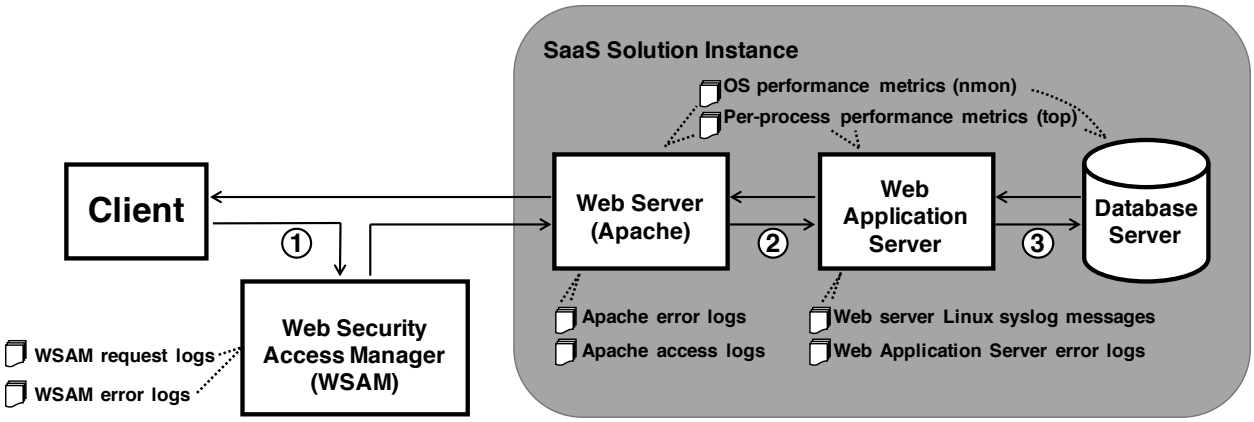


Figure 2.1: Solution architecture of each solution instance provided by the Enterprise PaaS Cloud (EPC). Monitors are indicated by file icons.

and use during incident analysis to expedite discovery of incident root-causes.

2.2 SYSTEM AND DATA DESCRIPTION

To demonstrate the application of our approach to a real system and evaluate its efficacy, we use a case study dataset containing three performance and reliability incidents that arose during end-to-end testing of the EPC. The data was collected by administrators responsible for incident response and root cause analysis.

Each solution instance on the PaaS cloud consists of three Linux VMs networked together: a Web server running an Apache HTTP server process, an application server running a Java-based Web application server (WAS) process, and a database server running a SQL server process. All client requests to the Web server are authenticated and authorized by an interceding third-party Web application Security Access Manager (WSAM). The solution architecture is shown in Figure 2.1.

During operation of the EPC, HTTP requests made by clients are first directed to the WSAM, which acts as a proxy and forwards only authenticated requests to the Web server. (Consequently, within the Web server access logs, all requests appear to originate from a single source.) When serving a given request, a plugin on the Web server determines if the response requires dynamic content. If it does, the plugin issues all necessary application requests to the WAS and waits for their response before responding back to the client. The WAS serves application requests using both local and remote application content. For the subset of application requests that require database queries, the WAS issues the queries to

Table 2.1: Monitor types available per incident in the Enterprise PaaS Cloud (EPC) dataset.

	Incident		
Monitor type	**1**	**2**	**3**
OS performance metrics (NMON) (all 3 VMs)	✓	✓	✓
Per-process performance metrics (TOP) (all 3 VMs)	✓	✗	✗
Linux SYSLOG (Web server VM only)	✓	✗	✗
Apache Web server access logs	✓	✓	✗
Apache Web server error logs	✓	✓	✗
Application server error logs	✗	✓	✗
Web Security Access Manager request logs	✗	✗	✓
Web Security Access Manager error logs	✗	✗	✓

the database server and waits on their response before proceeding with its application logic.

During the testing process, test clients generate a constant stream of HTTP service requests over a period of between two and seven days. By design, none of the requests are expected to receive a response from the Web server with a 500-level status code, and all servers are expected to maintain a steady state of resource utilization and request service rates. A test failure is therefore denoted by the presence of responses with 500-level status codes and/or a significant change (e.g., more than 10% difference) in the request service rate. To detect whether a test has failed, the administrators of the EPC visually examine dashboards displaying the resource utilization (CPU, memory, disk, network) on each of the three servers and the count of error log entries generated by each application, and look for noticeable trends or level shifts in resource utilization or spikes in error log messages. Incident response teams subsequently investigate the metric and log data for each incident *manually* to uncover the incident's root cause.

The case study dataset contains system and application log and metric data for three separate test failure incidents, each with a different root cause. The exact set of monitor data available differs for each incident; a summary of the relevant monitor types is given by Table 2.1. Since the VMs in the EPC use network time synchronization, the timestamps present in the data are already synchronized.

To illustrate the feature extraction challenges faced by administrators of the EPC, we provide sample log lines from the WAS error logs [26] (Listing 1) and the WSAM error logs [27] (Listing 2). The WAS error logs contain exceptions raised by the application, so they can provide insight into both application and database issues. Each of the logs contains

```
[XX/XX/XX ##:##:26:266 UTC] 0000000a com.ibm.ws.session.WASSession
   E SESN0042E: An error occurred when trying to insert a session into the
   database.
[XX/XX/XX ##:##:29:384 UTC] 0000000b com.ibm.ws.webcontainer.util.
   ApplicationErrorUtils     E SRVE0777E: Exception thrown by application class
   'org.apache.jasper.runtime.PageContextImpl.handlePageException:1234'
com.ibm.websphere.servlet.error.ServletErrorReport: ERROR: An unknown or
   unexpected error occurred when requesting the current page.
    at org.apache.jasper.runtime.PageContextImpl[...]
```

Listing 1: Sample Web application server (WAS) log lines.

```
201X-XX-XX-##:##:56.087+00:00I----- 0x38AD54BA webseald WARNING wiv ssl
   SSLConnection.cpp 860 0x0000000a
DPWIV1210W   Function call, gsk_secure_soc_init, failed error: 0000019e
   GSK_ERROR_BAD_CERTIFICATE.
201X-XX-XX-##:##:24.414+00:00I----- 0x132120DD webseald WARNING ias authsvc
   pdauthn.cpp 2233 0x0000aaaa
HPDIA0221W   Authentication for user bob@a.com failed. You have used an invalid
   user name, password or client certificate.
```

Listing 2: Sample Web Security Access Manager (WSAM) log lines.

structured and unstructured fields, where the unstructured messages can span multiple lines and are of variable length, complicating automated log analysis.

For each incident, we also have an incident report written by the incident response team that performed manual root cause discovery on the data. Specifically, the incident report includes the root cause for each incident, the initial signal that alerted the administrators to the problem, a rough description of how they traversed the logs (using auxiliary references as necessary) to uncover the root cause of the incident, the logs they identified as relevant and irrelevant to the root cause discovery process, and what actions they took to mitigate the problem.

One of the main challenges faced by the incident response teams is identifying the data source and often individual log messages that are relevant for diagnosing the root cause of a particular test failure. Because each HTTP request to the Web server can depend on a tree of subsequent WAS and database requests, an error on any one of the three servers can result in a series of error responses that propagate through all preceding servers. Thus, observing a drop in request service rates or a spike in 500-level error responses on the Web server does not necessarily indicate a problem on the Web server itself. Furthermore, since the specific tree of requests generated by a given HTTP request is not well-defined, it is not easy to

isolate the subset of error log messages that underly a particular issue without being able to separately correlate error log messages of different types and/or with different parameter values with the failing requests to identify meaningful correlations. That challenge directly motivated the design of our framework.

2.3 TAXONOMY OF CLOUD MONITOR DATA

2.3.1 Types of Data Present in Enterprise and Cloud Systems

Fundamentally, we consider there to be two types of monitor data – metrics and logs. Traditionally, most numerical features, including event-based counts and categories, are treated as metrics, whereas logs are most frequently used to refer to semi-structured or unstructured text logs, such as application logs or system logs. However, we feel that such a classification obfuscates the underlying difference between the types of data and limits clear understanding on when and how to analyze different types of data. We therefore clearly define the different types of data and how they relate to each other, and discuss the considerations that should be made when deciding how to treat monitor data.

Metric data

Metric data consists of numerical or categorical data that are periodically sampled by a process and therefore contains a value for every timestamp within a particular time interval. (We focus in this section on numerical data; see Section 2.3.2 for details about categorical data.) The data are inherently a time series, so time series properties, like stationarity, rate of change, moving averages, etc. can be studied directly. Examples of metric data include process monitor data such as that from Linux top, which provides metrics such as CPU and memory utilization, number of active processes, and per-process memory and CPU utilization.

Within metric data, there are two primary types of metrics: *cumulative* and *instantaneous*. Cumulative metrics consist of numerical values that capture the cumulative behavior of the underlying process over the sampling time interval. Examples include CPU utilization, network traffic volume, and request counts. Within cumulative metrics, there are two subtypes that depend on whether the values are normalized (averaged) with respect to the sampling interval. If normalized, they represent the *rate* or proportion of activity. If not normalized,

they represent the *count* (quantity) of activity.

Instantaneous metrics consist of numerical or categorical values that measure the current state of the underlying process or attribute, independent of the sampling time interval. Examples include current memory utilization and number of processes running.

The key distinction between cumulative and instantaneous metrics is their relationship to the sampling interval. If changing the sampling interval would have no impact on the measurement of the metric—that is, the duration of the measurement operation and the formula used to compute the metric are independent of the sampling interval—the metric is an instantaneous metric. Otherwise, it is a cumulative metric.

One "test" to distinguish between cumulative and instantaneous metrics is to examine what types of aggregation operations would maintain consistent meaning of the value of the metric. If upon downsampling, more than one value from within the downsampled time period must be used to compute a consistent new value of the metric, the metric is likely cumulative. For example, when resampling from a 1-second sampling interval to a 10-second sampling interval, for the average CPU utilization metric, the only consistent method to compute a new average CPU utilization would be to average all utilization values within the expanded time interval, so average CPU utilization is a cumulative metric. Similarly, to resample the current resident memory used metric in the same manner as above, one could simply take the single memory utilization value corresponding to the new timestamp corresponding to the metric value, thereby identifying current resident memory used as an instantaneous metric.

Log data

Log data consists of data that is generated by the occurrence of an event and are not periodically sampled. Log data can be of many different forms: they could be numerical values; categorical values; pre-coded, parameterized log messages; or freeform text, as in the case of human-written notes or reports.

Logs are often multidimensional, containing multiple feature values for each event occurrence. We delve into more detail regarding feature types in Section 2.3.2. Logs can also be *semi-structured* or *unstructured*. Semi-structured logs contain a combination of non-textual features and text, and unstructured logs contain only text.

Any form of event occurrence activity can be classified as a log, and log data need not necessarily be textual at all. The true distinction between metrics and logs is that metrics

reflect a sample of the state or utilization of some resource for which the underlying discrete events cannot be observed or for which such observation would not prove meaningful, and logs reflect the impulse events that correspond to the discrete event occurrences.

This ultimately means that the distinction between log and metric is based on judgment – in some cases, the dynamics of individual event occurrences may be of importance, so an otherwise metric may be better analyzed as log data. Log representations of data inherently contain more information, but as a result, can also be more costly to store and more difficult to analyze. Examples of features only calculable over log data are event interoccurrence times (e.g., request interarrival time) and request/response size measurements. Consequently, the conversion from log to metric is lossful, so it is not possible to reconstruct the log representation of a metric without additional information.

In practice, what is considered a metric and what is considered a log is often driven by the format of the data provided by an existing measurement tool or by the goal of the analysis. In many cases, it may actually possible to collect the data as either a log or metric by way of resampling and aggregating the underlying log data into a metric representation, so the decision on how to collect the data must be made on the basis of utility and cost.

2.3.2 Field Types within Our Taxonomy

To support coordinated analysis of heterogeneous monitor data, we define a general taxonomy of the types of fields present in the data. The taxonomy has two purposes: 1) to support labeling of all fields present in metric and log data so both may be handled uniformly, and 2) to facilitate automated extraction of time series features.

We identify the following field types across all data sources:

1. **Timestamp**: Fields representing the time at which a metric was sampled or a logged event occurred.

2. **Identity**: Fields that uniquely identify a metric measurement or log entry. For metric data, the monitor type and server name are often the only identity features required, but when multidimensional metrics are sampled conjunctly, additional identity features may be required. For example, for the Linux TOP utility, process ID must be used as an identity field to uniquely identify the per-process memory utilization metric.

3. **Metadata**: Fields with values that do not change over time for a given feature or data source. Typically, metadata describe some property of the generating process (e.g., the

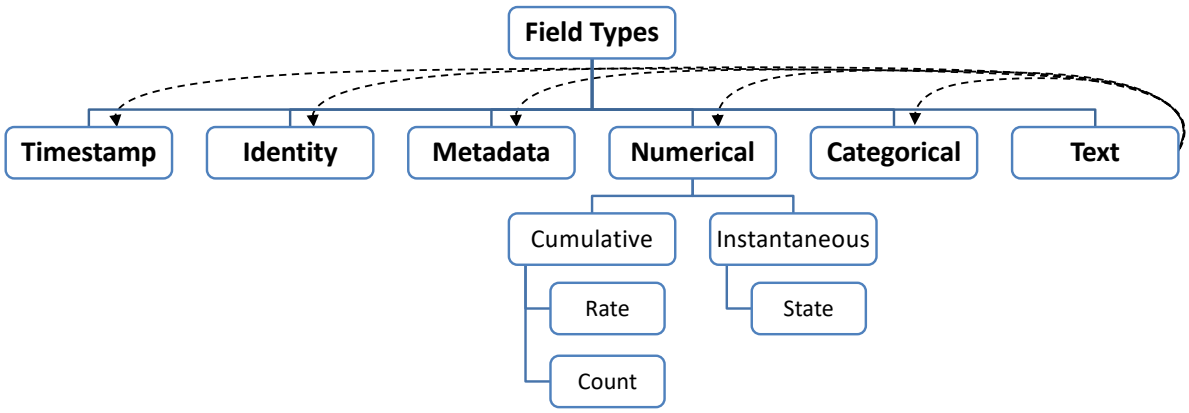


Figure 2.2: Field type taxonomy diagram. The dashed arrows emanating from text fields indicate that the message formats and parameter values within text fields are reclassified into fields of the other types within the taxonomy prior to feature extraction.

hostname for SYSLOG messages). Metadata fields can be recorded exactly once for a particular feature.

4. **Numerical**: Most fields from metric data sources are either numerical or categorical (see below). Numerical fields describe quantitative measurements. We consider ordinal fields with a discrete range to be numerical, as they can be aggregated with comparison operations.

5. **Categorical**: Fields that describe qualitative measurements with a discrete, often small range of possible values. Examples include error code and HTTP method.

6. **Text**: Fields that contain semi-structured or unstructured messages of arbitrary length or value, such as those coded into software (e.g., logging statements), generated by users (e.g., request URL strings), or generated randomly.

Numerical fields can be further classified into one of two types: *cumulative* or *instantaneous*. Cumulative fields capture the cumulative behavior of the underlying process over a given sampling time interval. Examples include CPU utilization, network traffic volume, application request counts, and kernel system call counts. Within cumulative fields, there are two subtypes that depend on whether the values are normalized (averaged) with respect to the sampling interval. If they are, they represent the *rate* (or proportion) of activity. If they are not, they represent the *count* (or quantity) of activity. Instantaneous fields measure the *state* of the underlying process at the time of sampling. Examples include current memory utilization and running process count. The full taxonomy is shown in Figure 2.2.

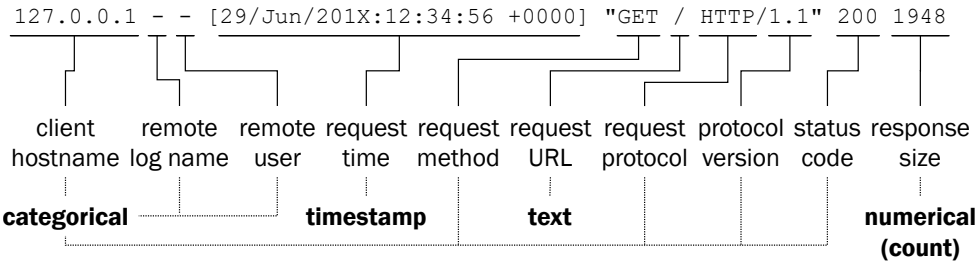


Figure 2.3: Field type labels for all fields in Apache access logs. The names of fields are given below the sample log line, and the field types are shown in bold below the field names.

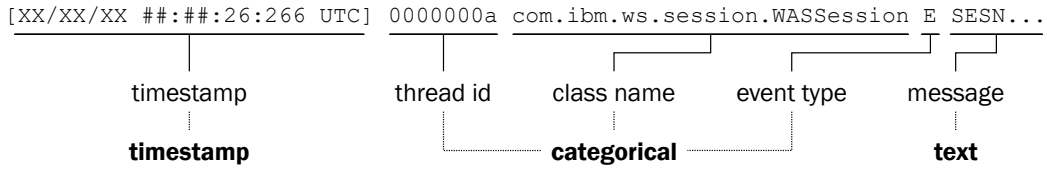


Figure 2.4: Field type labels for all fields in the Web Application Server (WAS) error logs. The names of fields are given below the sample log line, and the field types are shown in bold below the field names.

To illustrate the use of our taxonomy, consider the field labels for the Apache access logs shown in Figure 2.3. Aside from the obvious timestamp field, we would classify the response size field as a numerical feature describing a count (of bytes sent) and the request URL as a text field, as it can contain arbitrary data. We would classify all other fields as categorical. Figures 2.4 and 2.5 show the field labels for the WAS and WSAM error logs, respectively.

2.4 AUTOMATED FEATURE EXTRACTION AND REDUCTION

2.4.1 Feature Extraction

Our monitor field taxonomy enables us to automate feature extraction and discovery by defining steps to be taken when processing fields of each type.

Timestamp fields are used directly as the timestamps for extracted features. **Identity** fields are used to separate monitor data into logically distinct *streams*, each of which can be thought of as a separate monitor. **Metadata** fields must be unique per identity field, so they serve to validate the uniqueness requirement of the identity fields.

The next two field types describe observations and measurements made by the monitors, so they are used to generate the values of the extracted features. The values of **numerical** fields are used directly as the values of extracted features. **Categorical** fields, on the other

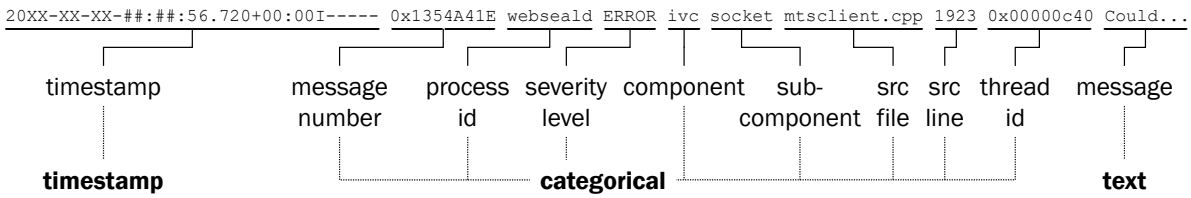


Figure 2.5: Field type labels for all fields in Web Security Access Manager (WSAM) error logs. The names of fields are given below the sample log line, and the field types are shown in bold below the field names.

hand, are similar to identity fields in that they describe classes of activity in the system. We therefore treat them much like identity fields.

Finally, **text** fields constitute the unstructured component of log messages. Text fields often contain information very pertinent to incident analysis, but extracting the information can be difficult because of format issues. For example, in the case of the WAS logs in Listing 1, admins were interested in distinguishing errors by their application exception type, but this information was buried in the text component of the logs, which made robust feature extraction difficult.

Rather than reinvent the wheel, we leverage existing techniques in unsupervised log parsing [9][28] to process text fields. These techniques identify log message formats and parameters by distinguishing between strings that are constant across all messages of a particular format (labeled as *log keys*), and those that are variable (labeled as *parameters*). We note that a recent survey [9] shows that for most log types, it is possible to achieve high parsing accuracy with limited pre-processing and tuning, which we found to be the case, as we describe below.

After the message formats and parameters have been extracted, our framework allows administrators to label relevant parameters for message formats of interest with the field types from our taxonomy. Once the parameters have been labeled, we handle them as if they were structured fields, aggregating, separating, or ignoring them accordingly. That is one respect in which our approach improves upon previous work. We treat the message formats as categorical fields.

We claim that manual labeling of relevant parameters is tractable for most cloud log sources because the number of unique message formats for a given cloud system is relatively limited and changes slowly; in our dataset, the three monitors for which we processed text fields contained 89, 32, and 79 unique message formats. Where the number of log formats is too large, we believe that labeling of a small number of initial parameters as categorical

Table 2.2: Sample regular expressions used to preprocess text fields.

Parameter type	**Regular expression used to parse parameter**
Integer	`[+-]?\d+`
Date (many formats)	`(\d{2,4}[\-/]\d{2}[\-/]\d{2,4}}`
Unix-style path	`/[\w\.:]*)+}`
Fully qualified Java class name	`([a-zA-Z_$][a-zA-Z\d_$]*\.){2,}[a-zA-Z_$][a-zA-Z\d_$]*`

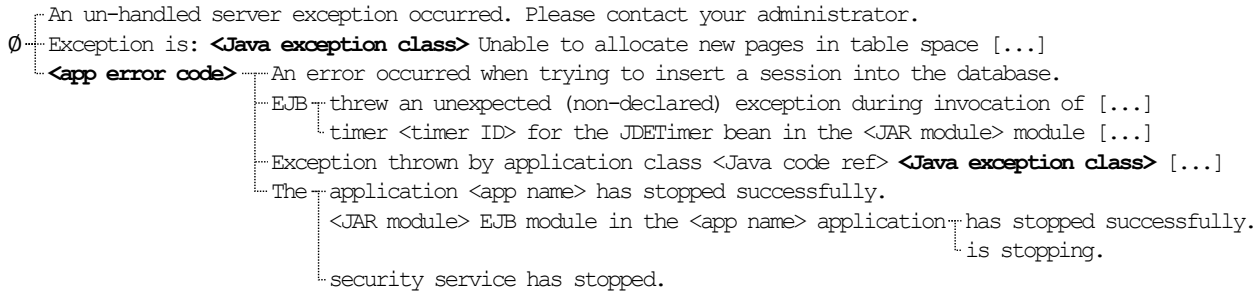


Figure 2.6: Example of the hierarchy of message formats for the unstructured text field in the WAS error logs. Parameters are denoted by parameter descriptions enclosed in angle braces.

features can still provide valuable feature separation, as early positional parameters often contain message codes that fully specify the remaining message format.

In our implementation, we used a modified version of the longest common subsequence (LCS) based parsing approach proposed in Spell [28]. Because the text fields in our log messages could take values with multiple lines (separated by newline characters) and could have a large number of tokens (e.g., exception stack traces), to improve parsing efficiency, we chose to examine the longest common prefix instead of the LCS, and bounded the number of tokens considered. We also preprocessed our logs by replacing tokens matching some common parameter types (e.g., IP addresses, standalone numbers, Java class names, dates) with placeholders, as suggested in [9]; some examples of the regular expressions we used are given in Table 2.2.

Figure 2.6 shows a sample of the hierarchy of message formats we obtained for the WAS error logs. As we were primarily interested in the `app error code` and `Java exception class` parameters, we labeled those parameter values as categorical fields within our taxonomy and ignored all others. In general, however, a user of our framework could label all parameters and rely on the feature reduction stage we describe in Section 2.4.2 to remove redundant and non-descriptive features; we discuss this further in Section 2.6.

The aforementioned approach works well in enterprise cloud scenarios for a number of reasons. First, Spell was designed to parse structured system event and application logs, which are the exact types of logs present in enterprise clouds; in a recent comparative analysis of log parsers [9], Spell had among the highest parsing accuracies on system and application logs among all log parsers evaluated. Second, Spell is a highly efficient algorithm and can run online, allowing it to handle new types of alerts as the system is running.

Procedure for automated feature extraction

Our framework performs feature extraction in two stages. First, we process data from each monitor individually per the procedure below to extract a set of time series, which we refer to as *features*. Second, to allow features from different monitors to be analyzed uniformly, we *summarize* features from all monitors by resampling them to a common sampling interval and aggregating them based on the field type.

Stage 1: Extraction. The procedure to extract features for a monitor is as follows:

1. Convert all text fields to categorical log keys and parameters as described above and drop the text fields.
2. Split the data into separate *streams*, grouping by identity field values.
   (a) Each metadata field should take the same value per stream. Verify that this is true and save the metadata field value for each stream.
   (b) Each numerical or categorical field now defines a *proto-feature* keyed on the identity field values, with *samples* for each timestamp value.
3. Identify the *feature ID field sets* for which features should be extracted. A feature ID field set is a set of categorical fields (including those derived from text fields) that, together with the identity field(s), will uniquely identify a feature. By default, a singleton feature ID field set is created for each categorical field. Domain knowledge can be used to define additional ID field sets for some monitors.
4. For each feature ID field set, split the proto-features into *features* by grouping samples that have the same combination of values for each field in the feature ID field set. Features that correspond to numerical proto-features take the value of the proto-feature; those that correspond to categorical ones take a value of 1 for each sample.

To demonstrate, we describe the feature extraction process for the Apache access logs within the EPC dataset. Recall the fields for Apache access logs from Figure 2.3.

First, we identified and removed the metadata fields. The EPC's web server did not use remote user authentication (instead relying on the WSAM for authentication) or remote logging, so values for the remote log name and remote user fields were constant. Furthermore, the web server accepted only HTTP requests, so the request protocol was always `HTTP`. Additionally we knew from domain knowledge that the request URL was not useful in diagnosing issues, so we preemptively excluded it from feature extraction.

Thus, the remaining fields that we extracted for the EPC web server Apache access logs were request time, response size, client hostname, request method, protocol version, and status code. We created proto-features for response size (the only numerical field) and request count (taking a value of 1 for all samples). The last four fields are categorical, so we created singleton feature ID field sets for each. Based on our domain knowledge, we also chose to create feature ID field sets for {status code, request method} and {client hostname, status code, request method}. For incident 1's Apache access logs, feature extraction using our framework yielded 98 distinct features.

In Table 2.3, we show the total number of features extracted for each incident in our case study dataset. Our feature extraction approach intentionally generates a large number of features, as our objective is to extract as many meaningful features from the data as possible; in Section 2.4.2, we show how we reduce the features to a manageable number.

Stage 2: Summarization. The features we obtain from feature extraction will naturally have varied timestamps, and, in the case of metric data, may also have varied sampling intervals. In order to facilitate cross-monitor feature analysis, we subsequently *summarize* the features by resampling all features. That is, we bucket data points for each feaure into contiguous windows of fixed size (the resampling interval) and aggregate their values to produce the summarized features.

In general, the resampling interval should be chosen based on domain knowledge of the dynamics of the system in question—smaller values allow for finer granularity in feature comparison, whereas larger values support meaningful analysis of behaviors that manifest over long timescales. It may also be valuable to try multiple resampling intervals for different types of analysis. Based on our knowledge of the EPC's system, in our experiments, we chose a resampling interval of 10 seconds.

The appropriate aggregation method for each feature is defined by our taxonomy: features

representing rates are *averaged*; those representing counts and categorical proto-features are *summed*; and those representing states have a representative value chosen (e.g., maximum or most recent).

2.4.2 Feature Clustering

Our feature extraction method yields what at first glance looks to be an unmanageable number of features – in our dataset, as shown in Table 2.3, we end up with an average of over 2600 features per incident. To promote effective analysis of the data, and to remove redundant and uninformative features, we propose a hierarchical feature clustering method that groups highly correlated features across all monitors. We now describe the method, and illustrate its effectiveness using the EPC dataset.

Feature clustering is useful for two reasons. First, many features are likely redundant. Within each monitor, many fields represent similar values, and in some cases, splitting on a categorical field does not yield a meaningfully different feature. Furthermore, some monitors collect redundant data (e.g., some OS-level vs. per-process performance metrics). Clustering can reduce the number of such features an admin must examine. Second, as we illustrate in Section 2.5, clustering can uncover relationships between features across different monitors that are important in incident detection and root cause analysis.

Before clustering the features, we first eliminate uninformative features and separate dense and sparse features. Some features extracted by our framework will either have no value over the entire dataset (*empty* features), or have a constant value (*constant* features). Such features exist because some combinations of feature ID field set values never appear in the data or may actually represent metadata. For example, we found that some combinations of status code and request method did not appear within the EPC's web access logs, but our framework created a feature for them. In our dataset, empty and constant features made up roughly 15% of all features.

Next, we observe that many features derived from logs are sparse (i.e. rare events)—they are zero-valued the vast majority of the time—and should be clustered separately from dense features. We define a feature $f(t)$ to be *sparse* if $\sum_{t_i \in [t,t+w)} [f(t_i) \neq 0] < R$ for some t, where R is the rare event occurrence threshold and w is the rare event window, and *dense* otherwise. R and w should be defined based on domain knowledge; in our dataset, we chose $R = 3$ and $w = 500$s to separate dense and sparse features. We show the feature clustering results for each incident separately in Table 2.3.

We now propose a hierarchical feature clustering method that groups highly correlated features at three levels: (1) within each unique feature ID field set, (2) within each monitor, and (3) across all monitors. At each level, we compute the pairwise Spearman rank correlation between all features within the level and identify maximal cliques with pairwise correlations that exceed a *feature similarity threshold*. Each clique represents a set of *redundant* features. For each clique, we choose the feature with the highest sum of pairwise intra-clique correlations as the clique's representative feature to be used in the next level of clustering. Hierarchical clustering improves scalability by drastically reducing the number of features at lower levels, where they are more likely to be redundant.

We chose to use Spearman rank correlation because unlike Pearson correlation, as used in [29], which can only identify linearly correlated features, Spearman correlation accurately groups together features that trend together non-linearly (such as request count and request size). We propose the use of a high feature similarity threshold (i.e., above 0.9) so as to limit the number of spurious groupings; based on examination of our dataset, we chose the value 0.96.

For example, for our EPC Apache access logs for incident 1, our framework first found 11 of the 98 original features to be empty, 68 to be dense, and 19 to be sparse. After clustering was performed, the original features were grouped into 15 dense clusters and 1 sparse cluster. Closer examination of the clusters revealed that all but two of them grouped together requests from hostnames that corresponded to different test roles, and the two anomalous clusters grouped various hosts with error status codes that corresponded to the failure incident.

Table 2.3 shows the results of feature clustering for the EPC data. Overall, clustering reduced the number of features by 66%.

Our approach was able to semi-automatically parse, extract features from, and correlate features across over half a dozen different monitor types of widely varying data types, processing both metric and log data jointly. The resulting feature set succinctly describes activity in the system..

2.5 APPLICATION TO INCIDENT RESPONSE

We now illustrate some of the ways our framework can improve the effectiveness of enterprise cloud administrators and IRTs in reliability and performance incident data analysis.

Table 2.3: Number of features per incident at each stage of feature reduction.

	Incident 1		**Incident 2**		**Incident 3**		**Total**	
Feature red. stage	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse
Original features	3693		2132		2231		8056	
Without empty features	3327		1495		1704		6526	
Pre-clustering	2238	1089	484	1011	426	1278	3148	3378
After clustering (1)	1302	347	357	340	306	625	1965	1965
After clustering (2)	1232	320	325	314	287	494	1844	1128
After clustering (3)	1108	273	296	287	260	475	1664	1035
Reduction %	50%	75%	39%	72%	39%	63%	47%	69%
No. of total clusters	1381		583		735		2699	
Overall reduction %	63%		73%		67%		66%	

2.5.1 Feature Discovery

Our approach enables administrators to discover meaningful features more easily across their heterogeneous monitors, reducing the time and domain expertise required in deciding what features are relevant for analysis.

As we describe in Section 2.1, prior to the development of our approach, administrators of the EPC needed to decide a priori which logs and metrics they considered useful for incident analysis, and needed to construct dashboards manually for each. Because of the engineering cost of developing tools to perform complex analyses of the log data, the administrators often used rudimentary filters, such as message severity level, token count, and keyword presence, to define metrics based on the logs. In some cases, where documentation stated that fields from the structured part of the log described the unstructured messages, they ignored the unstructured messages outright.

In our examination of the EPC dataset, we found that those simplifying assumptions overlooked many features salient to the incidents that took place.

Identifying unique feature ID field sets: The documentation for the WSAM error log states that the message number field uniquely identifies the unstructured message format. As a result, the EPC admins used the message number field to count error messages of different types. However, our analysis revealed that message number was common to multiple error message formats, and that the set of (source line, message number) was required to uniquely distinguish error types.

Feature separation by parameter value: Even within errors of the same type, meaningful signals were sometimes hidden within parameters of the unstructured message. For example, for incident 3, EPC admins found that during periods of service unavailability, there was a spike in a WSAM error message that corresponded to loss of network connection with another server, but as the IP address of the unresponsive server was contained in the unstructured message, the admins were forced to manually investigate the logs each time the error message spiked to identify the server. Our framework enabled us to automatically separate the features by server IP address; the resulting subfeature for the WAS IP address clustered with an error that corresponded to an SSL version issue, which had been identified as the root cause of the incident.

2.5.2 Feature Deduplication

Another major challenge the administrators faced was identification of meaningfully unique features within the data. While it was obvious that OS performance metrics collected from the NMON and TOP monitors provided duplicate information, it was not clear whether, for example, failed requests in the web access logs were similarly correlated with WAS errors. The admins therefore made assumptions based on past experience about which of the thousands of candidate features to monitor.

Our framework automatically groups together strongly correlated features, allowing redundant features to be monitored once. We found many instances in the EPC where the use of our framework would have prevented the creation of dashboards for multiple log message types that were ultimately redundant.

2.5.3 Root Cause Analysis

Finally, an understanding of which features are strongly correlated can help admins uncover insights about their system that can aid in root cause analysis. For example, in incident 2, the database server ran out of free disk space, causing a spike in WAS errors for application requests that required database writes. As a consequence, clients could not use the application, as the web server started responding to all such requests with a status code of 500 Server Error.

When we used our framework to examine the features clustered together with the feature for status code = 500, we found that it strongly correlated with a feature for a specific WAS error

Table 2.4: Subset of features in feature cluster containing root cause for Incident 2. Parameter values are denoted by parameter descriptions enclosed in angle braces, or by an asterisk (*) if ignored.

Monitor	**Feature ID Field Sets**	**Feature Value**
WSAM error log	short name="com.ibm.ws.ejbcontainer.osgi.internal.EJBRuntimeImpl"	COUNT
WSAM error log	short name="com.ibm.ws.webcontainer.servlet"	COUNT
WSAM error log	event type=3	COUNT
WSAM error log	message="SESN0042E: An error occurred when trying to insert a session into the database."	COUNT
WSAM error log	message="Session Object is: com.ibm.ws.session.store.db.DatabaseSession * hashCode *"	COUNT
WSAM error log	*message="Exception is: com.ibm.db2.jcc.am.SqlException Unable to allocate new pages in table space '*'. * * *"*	*COUNT*
Apache access log	*status code=500, client ip=<test host>, http method=POST*	*COUNT*
Apache access log	status code=500, client ip=<test host>, http method=POST	response size
...	...	...

type. Examination of the clusters that corresponded to the subfeatures for that error type revealed a large cluster of 12 features, one of which was a SQLException with the message "`Unable to allocate new pages in table space`", which indicated that the root cause of the incident was likely a disk space issue. Table 2.4 shows a subset of the features found in the large cluster, with the two most relevant features emphasized. While explicit monitoring of disk utilization would have identified it as a possible cause, our approach enabled us to trace the errors back to the disk space issue directly.

2.6 DISCUSSION

Generality. Our approach can generalize well to other cloud systems and log types. We have demonstrated its use for 8 considerably different log types in the EPC; practitioners interested in applying the approach can follow the procedure we have laid out to the monitors in their own system. In other work, we have successfully applied our framework to SYSLOGs, firewall logs, and security event logs.

The utility of our feature clustering approach for incident root cause analysis will likely vary depending on the type of incidents present in a system. Our approach works well for reliability and performance incidents that cause changes (both sudden and incremental) in the

steady-state value of system performance metrics or the frequency of occurrence of different types of system or application events. Many reliability and performance issues encountered in production systems, including those caused by resource exhaustion, application exceptions, and intermittent network failures, exhibit such dynamics. However, our clustering approach does not fare as well for silent failures (i.e., those that do not produce log entries) or for incidents in which the cause of the incident is very loosely correlated with service performance indicators.

Scalability. The most expensive operation in our framework is the clustering-based feature reduction, which consists of two distinct steps performed repeatedly: 1) computing pairwise cross-correlation values across all features within each feature cluster at each level of clustering, and 2) finding all maximal cliques within each cluster. The cross-correlation step scales quadratically in the number of features and log-linearly in the length (number of samples) of the data. We chose to implement the clique-finding step using the Bron-Kerbosch algorithm; though the worst-case runtime complexity of the algorithm is exponential in the number of features in a given cluster, the average case runtime complexity is highly sensitive to the sparsity of the cross-correlation graph.

We found that because there is high redundancy at levels 1 and 2 of the clustering, the number of features clustered across all monitors (level 3) is reduced to a manageable level. Our Python implementation utilizes the Pandas library [30] to compute cross-correlation and a custom implementation of the Bron-Kerbosch algorithm. It was able to process all features in our dataset (which contained data from over 2 weeks of system operation in total), in a few hours while running on a single server with an 8-core Intel Xeon CPU E5-2450 processor and 64 GB of memory. To further improve scalability, we propose adding additional levels to the clustering between levels 2 and 3 to cluster features within, for example, the same physical or virtual machine, the same network subnet, etc.

Extensions to feature extraction and clustering. We have identified many alternatives ways to extract features, and we intend to examine them in future work. They include joining of features across different monitors (e.g., on ID fields as in [31]) to further distinguish activity, and parsing of fields in alternative ways, such as by counting the number of unique values taken by categorical features or by computing interarrival times for sparse features.

We also plan to extend our feature clustering approach. Currently, we use rank-based correlation with no time lag to group features, but there are many cases in which redundant

or related features will have small time lags between them. We plan to study how we can methodically investigate lagged features to further reduce redundancy and uncover relationships relevant to root cause analysis.

2.7 RELATED WORK

We have discussed the limitations of many commercial tools for coordinated analysis of heterogeneous monitor data in Section 2.1. Here, we focus on the two most prominent and related examples: VMWare LogInsight and Microsoft Azure Kusto. Though each supports processing of a wide range of cloud data sources, none can process arbitrary, unstructured log data in an unsupervised manner or support flexible feature discovery and extraction. Our approach does both, improving visibility into a wider array of data sources.

VMWare LogInsight purports to provide functionality very similar to our approach, but our approach is meaningfully different in the following ways. First, LogInsight only supports clustering within a log type, whereas we perform cross-log type clustering. Second, LogInsight allows extraction of features on combinations of field values, but in addition to performing the same semi-automatically for a wide range of field value combinations, we *also* perform feature correlation and reduction to identify the subset that actually provide meaningful information and group together those that are effectively redundant or related. It is not clear whether it is possible to perform our analysis within LogInsight as a plugin, but even if so, our approach is not currently supported by LogInsight natively.

Microsoft Azure Kusto also provides functionality similar to some aspects of our approach, but our approach differs in the following ways. Most significantly, we provide a method to extract structured information from text fields, which Kusto does not support. Furthermore, Kusto's AutoCluster plugin groups features with common discrete values, as opposed to our use of pairwise rank correlation, which allows us to discover cross-feature correlations that Kusto cannot. Kusto also supports clustering only within a single log type, whereas we perform cross-log type clustering.

The problem of extracting meaningful signals from heterogeneous log data for failure detection has been studied extensively in high performance computing (HPC) literature. LogDiver [29], Desh [32], and LogAider [33] all extract features from HPC datasets including hardware error, reliability, workload, and scheduler logs using extensive domain knowledge of the log semantics and structure, then cross-correlate features spatially and temporally. While these works are useful, they are highly specialized for HPC problems, and do not provide a

method to *automatically* extract features from arbitrary monitors.

Among commercial tools for cloud log analysis, VMWare vRealize Log Insight [19] and Microsoft Azure Kusto [18] provide the most similar functionality to our framework, but our work is still distinct. Kusto does not support extraction of fields from unstructured log messages, and while Log Insight does, it does not allow parameters to be treated like structured fields. Furthermore, while both support clustering of features—Kusto by common discrete field values and Log Insight by message format—clustering is limited to features within individual monitors, whereas we perform cross-monitor clustering using rank correlation.

Our work relies on unsupervised approaches for identifying message formats and parameters within unstructured log data (also called log clustering or automated log parsing). Zhu et al. [9] examine 13 mainstream approaches that utilize various techniques (data mining, clustering, heuristics, etc.), and they evaluate the parsing accuracy of the techniques over a wide array of diverse log types. They find that many cloud log types can be accurately parsed with existing approaches. We extend the utility of those techniques by enabling the extraction of meaningful time series features from parsed logs in conjunction with metric data.

There is also literature in the space of anomaly detection on heterogeneous log data. Yen et al. [10] extract various manually-specified features from host- and network-level monitors in an enterprise system logs collected from web proxies, DHCP servers, VPN servers, domain controllers, and host antivirus scanners and combine them to proactively detect suspicious activity, but their approach requires significant domain knowledge. Du et al. [34] use Spell to parse various application and system logs and train a recurrent neural network for each log type to identify anomalous workflows. While their approach is semi-automated, it does not support cross-monitor data analysis as ours does.

2.8 CONCLUSIONS

In this chapter, we presented a monitor data analysis framework for enterprise clouds that allows administrators to semi-automatically process heterogeneous data from multiple levels of a cloud platform into a manageable set of meaningful time series features that are useful in incident analysis. In contrast to existing tools and techniques for incident root cause analysis, our framework enables the *coordinated* analysis of diverse data types while requiring limited manual administrator effort. We illustrated the use of our framework on data from an experimental PaaS cloud and demonstrated its utility in reducing the complexity of monitor

data analysis for incident response.

Our results show that principled use of redundancy through coordinated analysis of log and metric data, as our framework enables, facilitates more rapid root cause analysis in two ways. First, by enabling the clustering and removal of redundant features across *different* monitors, as well as redundant features within a single monitor, our framework dramatically reduces the number of features that analysts must sift through during analysis. Second, examining clusters of statistically redundant features originating from heterogeneous monitors can allow analysts to directly connect an incident's key performance indicators to the logs necessary to diagnose the incident's root cause.

CHAPTER 3: RESILIENCE AGAINST MISSING DATA THROUGH REDUNDANT MONITOR DEPLOYMENT

As networked computer systems have grown in size and complexity, so too has the need to adequately monitor them for failures and attacks. Insufficient monitoring can cause system administrators to miss critical security and reliability events, which hinders system recovery and can result in devastating financial and societal consequences [3]. Such challenges come at a time when more infrastructure providers (e.g., energy systems, industrial systems, healthcare providers, and municipal governments) are connecting their internal networks to the Internet in spite of sophisticated and stealthy attackers.

In modern, large-scale enterprise and cloud systems, system administrators often use centralized intrusion detection systems, security information and event management (SIEM) tools, and network and server monitoring tools to detect security and reliability incidents. Although such tools are capable of ingesting and processing data from an ever-increasing array of diverse network and server log sources, practical limitations, such as monitoring costs and resource utilization overhead, often limit the data that are actually collected for incident detection. As a result, in order for most incident detection tools to function correctly, they implicitly require that the data provided by monitors be trustworthy and complete. System administrators rarely consider the resilience of the monitoring infrastructure itself when designing monitoring systems.

We posit that *placing trust in the availability of monitor data is ill-founded* and that *monitoring infrastructures should be designed to be resilient to missing information.* In large-scale distributed systems, monitor data may be temporarily unavailable for myriad reasons. First, in an adversarial context, attackers compromise monitors. Stealthy attackers often disable monitoring or hide their tracks to hinder the ability of security analysts to observe their malicious activity. Recent high-profile attacks [35, 36] have demonstrated circumvention of monitors, and hiding techniques are well-documented [37, 38]. Second, in a non-adversarial context, monitors may fail due to software or hardware faults or configuration issues [39], and monitor data can be lost or indeterminately delayed in transit to a centralized incident detection tool due to packet losses, network misconfigurations, or failures of network devices [40]. Absence of critical monitor data can prevent detection of important events or hinder post-hoc incident investigation, with huge financial implications [12, 13].

We find shortcomings in existing approaches that might be used to mitigate the impact of missing monitor data. Although many recent approaches to enterprise and cloud security

and reliability monitoring [10, 11, 31, 34, 41, 42, 43] combine data from multiple, potentially redundant data sources, such approaches do not consider the effects of monitor failures or compromises on detection. State-of-the-art optimal monitor placement techniques [44, 45] do not directly examine the effect of missing monitor information on a system's incident detection ability, which can cause them to overlook overreliance on a limited set of monitor data for the majority of incident detection.

Overview

We believe that monitoring infrastructures should be explicitly protected against missing monitor data by *preemptively enabling data collection from redundant monitors*, an approach we call *resilient monitoring*. Such an approach is practically applicable to real systems, as it does not require changing the incident detection mechanisms already in place or developing specialized detection algorithms, as is the case with related defenses proposed in adversarial machine learning literature [46].

To this end, we present a systematic methodology that enables system administrators to design resilient monitoring systems. The central components of our approach are 1) a general model for system monitoring and incident detection, and 2) metrics that quantify the effort required to subvert incident detection, the damage potential of an attacker with a particular level of capability, and the resilience of a monitoring system against missing monitor data. We show how our model can be used to determine which monitors should be deployed to maximize resilience against missing monitor data, subject to the operational reality of cost constraints in monitoring.

We demonstrate the practicality of our approach by applying it to a realistic datacenter network case study model based on monitors that are employed in production systems, and we study its scalability and performance against the current state of the art [45]. We find that our approach can help system administrators make more effective monitoring decisions in line with their incident detection objectives and priorities. To the best of our knowledge, our work is the first to methodically examine the problem of missing monitor data and propose mitigation techniques that are complementary to existing anomaly and intrusion detection approaches.

Contributions

In summary, our main contributions are:

1. A general **model** for system monitoring and incident detection.
2. A set of **metrics** that quantify the effect of missing monitor data on incident detection ability and the resilience of a monitoring system.
3. A **cost-constrained methodology** for **maximizing the resilience of monitor deployments**.

3.1 BACKGROUND AND MOTIVATION

In large enterprise and cloud systems, security and reliability incident detection is often performed centrally in SIEMs and infrastructure monitoring tools. Such tools (e.g., Splunk [22]), collect logs and metric data from distributed sources in the system (i.e., *monitors*) to detect anomalous or malicious activity. There have also been recent trends in literature in enterprise security monitoring [10, 11, 31, 47], cloud failure monitoring [43, 48, 49], and datacenter network failure monitoring [40, 50, 51, 52] towards techniques that combine data from multiple, often heterogeneous sources to uncover novel signals for incident detection.

Ideally, a system administrator would collect and analyze all monitor data available to them to maximize their chances of detecting incidents. However, practical limitations often guide monitoring decisions. First, monitors impose a performance overhead to the systems and network, and data collection, storage, and analysis budgets are often limited. For example, most hosted SIEM products—those that provide the SIEM functionality as a third-party service—and security operations centers (SOCs) charge by the events per second (EPS) or gigabyte of data processed. Second, increasing the complexity of the monitoring infrastructure can introduce considerable engineering costs to deploy and maintain monitors. Without clear reason to build in redundancy, most practitioners opt for simpler monitoring infrastructures.

In practice, administrators often prioritize data collection and analysis based on perceived utility for incident detection and cost [53]. As a result, their incident detection systems may come to rely heavily on data from a limited number of monitors. Consequently, detection systems may fail to detect incidents if even a select few monitors fail to generate data.

The problem of missing data

It is well understood that monitor data in large-scale systems is prone to temporary unavailability. Monitors may crash due to software or hardware faults [39]. Monitoring code may contain bugs, so monitors may fail to generate logs when otherwise anticipated [51]. Even when monitors function correctly, packet losses, misconfigurations, and failures of network devices may cause monitor data to be lost or indeterminately delayed in transit to a centralized incident detection tool [40, 44]. In many of the aforementioned scenarios, the data loss may be silent and otherwise undetectable.

Monitor data may also be intentionally hidden by malicious actors. Stealthy attackers, for example, wish to remain undetected while moving laterally through a system. To do so, they may disable monitors. Such techniques have been demonstrated by rootkits within malware such as Nemesis [35] and Stuxnet [36], which explicitly hide files and processes they create from operating system-level monitors. Alternatively, attackers aiming to cause denial of service may instead hide data necessary to preemptively detect a catastrophic system failure. There are many possible mechanisms by which an attacker could attempt to compromise a given monitor in a distributed system. Some examples include:

- The attacker could target the infrastructure layer on which the monitor runs (e.g., use a bootkit or rootkit to hide activity from operating system logs, as in VM introspection subversion)
- The attacker could corrupt or kill the monitoring software (e.g., replace the monitoring software binary with a malicious version, change IDS rule or configuration files, or kill a monitoring binary)
- The attacker could corrupt the monitor data files prior to analysis (e.g., tamper with user command history or log directory files before they are remotely synchronized)
- The attacker could target the monitor data collection process (e.g., dropping network packets or killing remote log synchronization processes to temporarily disable monitoring)

We refer to the problem of missing monitor data, whether due to failure or attack, as that of *monitor compromise*.

3.2 RELATED WORK

The problem of designing cost-efficient probing systems has been well-studied in tomography literature. However, none of the prior approaches take into account compromise of the monitors themselves. For example, NetBouncer [40] and deTector [51] attempt to maintain localization ability under network component failure, but assume that monitors are always available. NetNORAD [50] simply deploys a fixed number of monitors per rack and does not examine their placement strategy any further. Ma et al. explore the placement of the monitors themselves [54], but only take into account the failure of non-monitor nodes. [55] and [56] minimize monitor count under minimum link coverage constraints, but they assume that monitors are fault-tolerant.

There are few approaches that explicitly aim to incorporate redundancy into their monitor placement strategies. Holloway [44] considers redundant monitor deployment in the context of cloud availability monitoring, using a probabilistic model for the failure of the links and hosts that monitor data must traverse while being centralized. While the approach addresses many of our objectives, it cannot generalize beyond the cloud availability monitoring architecture the authors define.

Importantly, none of the aforementioned prior work consider malicious monitor compromise. A stealthy, dedicated attacker may identify and drop just a *single* alert that is needed to detect their attack. Such targeted behavior cannot be captured using probabilistic models of monitor failure. In previous work [45] (DSN16), we proposed a state-of-the-art general methodology for monitor deployment that takes into account monitoring cost, monitor utility towards incident detection, and monitor susceptibility to compromise by a malicious attacker. However, as DSN16 does not allow an administrator to study the effect of the compromise of individual alerts, the optimal monitor deployments it finds can still be highly susceptible to monitor compromise, as we demonstrate in Section 3.9.3.

3.3 GOALS AND OVERVIEW

Based on the aforementioned limitations of existing monitor placement strategies to provide resilience against monitor compromise, we formulate two design goals that a system designer wants to consider when they design their monitoring system:

DG1: Resilience. Is my current monitoring system design sufficiently resilient against monitor compromise?

DG2: Cost-efficiency. Given my monitoring system design, how should I efficiently deploy monitors to maximize the resilience against monitor compromise, subject to monitoring cost constraints?

We find these goals to be reflective of operational realities that system designers must face. Monitoring is typically a non-functional requirement of most systems, which means that monitoring infrastructures collect and analyze just enough data to meet detection objectives. Thus, system designers should be able to take into account the resilience of the monitoring against failure or compromise (goal **DG1**). Monitoring systems can be expensive in terms of resource utilization, management, data storage, and analysis [45, 53]. Thus, system designers should be able to efficiently use cost-constrained limited resources (goal **DG2**).

To achieve these goals, we propose a model-based framework. Our framework provides a system designer with the capabilities to model the monitoring system (Section 3.5.1), the incident detection system (Section 3.5.2), the attacker's capabilities (Section 3.6.1), and the cost of compromise to the attacker (Section 3.6.2), and to quantify the resilience of the monitoring infrastructure against monitor compromise (Section 3.7). Based on this modeling, system designers can determine if their system design is resilient (goal **DG1**) and cost-efficient (goal **DG2**) (Section 3.8).

3.4 CASE STUDY: DATACENTER NETWORK FAILURE LOCALIZATION

To illustrate our modeling approach, throughout this chapter, we consider the design of the monitoring infrastructure for the hypothetical simple datacenter network with a topology as shown in Figure 3.1. The datacenter provides a distributed object storage service (similar to Lustre [57]) to internal client hosts (Cl_{1-4}). The storage service consists of multiple data servers (DS_1 and DS_2) and a metadata server (MD). To request an object, a client first requests the location of the object from the metadata server, then makes a subsequent request to the appropriate data server to obtain the file. The network has a two-tier fat tree topology with two spine switches (Sw_1 and Sw_2), and the clients and servers are distributed across four racks, each with its own top-of-rack switch (ToR_{1-4}).

The datacenter's administrator wants to detect and localize failures of all switches in the network, individual links between the ToR and spine switches, and the storage service servers. In large-scale datacenter networks, such failures occur frequently as a result of bugs, misconfigurations, and resource contention [40]. Recent research [1, 58] has revealed that

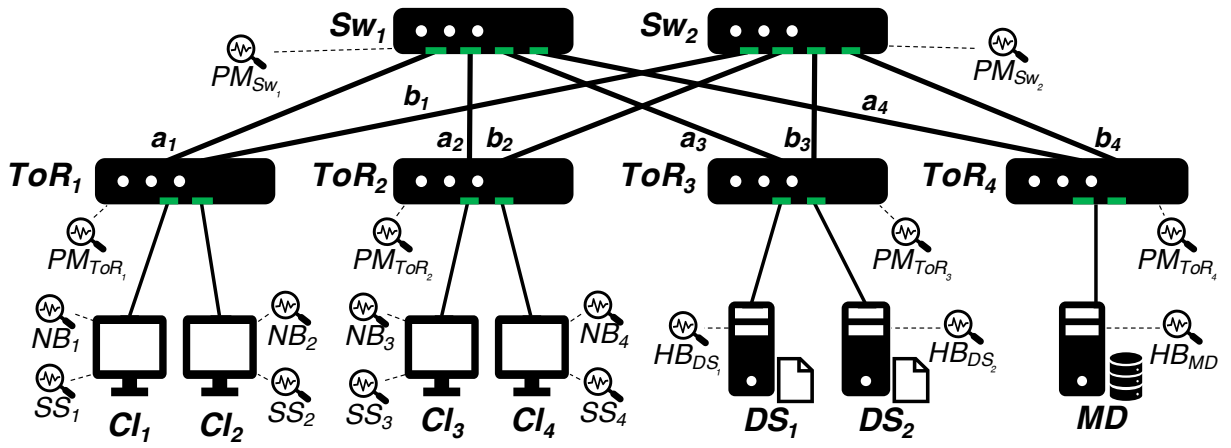


Figure 3.1: Case study example datacenter network topology. Sw_{1-2} are core switches, ToR_{1-4} are top-of-rack switches. Cl_{1-4} are client hosts, DS_{1-2} are data servers, and MD is the metadata server. Each magnifying glass represents an instance of a monitor that can be deployed in the system. Network links are shown as solid lines between components, and monitor deployment relationships are shown as dotted lines.

"gray failures", which are partial failures that cause temporary or intermittent unavailability, are behind most major availability and performance issues in large-scale datacenters. As such, the administrator wants the detection system to be able to diagnose both crash-stop and partial failures that manifest as loss of availability for all components and links in the system. For the sake of simplicity, we exclude gray failures that manifest as increases in end-to-end request latency but not outright unavailability.

While designing a monitoring infrastructure for failure detection may seem like a simple and straightforward problem, in practice, the types of monitors that can be deployed are diverse, and each comes with advantages and drawbacks. Traditional passive monitoring methods, such as heartbeat monitoring for servers and network performance and port health monitors for switches (e.g., over SNMP), have been shown to be insufficient to reliably identify gray failures because such failures may be perceived differently by end hosts and network components. For example, a switch may falsely report that a link is up because of software bugs, but end hosts will be unable to send packets over the link [51, 59]. Other reasons why passive monitoring approaches can be insufficient include the following:

- The heartbeat process for a given component or application is often separate from the service the component provides, so heartbeats may be generated even if the service is unavailable.

- If a component (or link) is failing intermittently (e.g., link flapping), if the central observer never probes the component during the period of failure, it would never believe the component to be failing, even as end-to-end requests experience the failure.

Inspired by recent literature in datacenter failure monitoring [40, 50, 51, 52], the administrator has decided to employ active monitoring techniques that use end-to-end probes to inspect the network. Such techniques are inspired by prior work in network tomography (e.g., [60]) and use lightweight, client-side monitoring modules to either generate end-to-end, ping-like probes of network or service availability or collect already present service requests as end-to-end probes. During periodic probing intervals, active monitoring techniques aggregate the success or failure status of all probes and feed them into a topology-aware latent factor or probabilistic model to compute the probability that a given component has failed. To limit the probing overhead on the system, active monitoring techniques choose to deploy monitoring modules be deployed to only a subset of hosts and/or choose to exercise only a subset of probe paths such that there is sufficient probe coverage of the datacenter network to be able to localize failures for each of the components.

The administrator considers the use of the following types of monitors:

- Basic, accrual-based heartbeat monitors [61] (denoted as HB in Figure 3.1) running on each storage service server (DS_{1-2} and MD) that centrally record a single percentage of expected heartbeats received during a probing interval.

- Port health monitors (denoted as PM) running on each of the switches (ToR_{1-4} and Sw_{1-2}) that periodically poll the status of each link connected to the switch (`UP` or `DOWN`) and centrally record the percentage of polls for which the link was seen as `UP` during a probing interval.

- Storage service monitors (denoted as SS) running on each client (Cl_{1-4}) that observe the status of each storage service request made by the client to each server and centrally record the percentage of requests per server that received a response during a probing interval.

- NetBouncer [40] agents (denoted as NB) running on each client that use IP-in-IP packet bouncing to probe the paths to each spine switch (Sw_{1-2}) and centrally record the probe success probability during a probing interval.

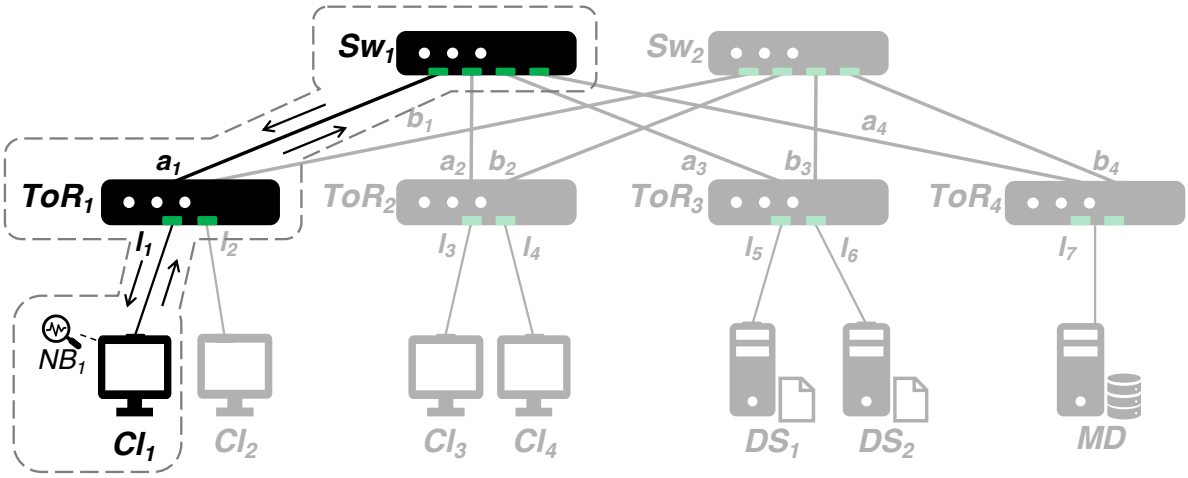


Figure 3.2: Illustration of the "probe path" for the NetBouncer probe from NB_1 to Sw_1. Because there is only one path (indicated by the arrows) between Cl_1 and Sw_1, the probe *must* traverse that path. The full probe path is therefore $l_1 \rightarrow ToR_1 \rightarrow a_1 \rightarrow Sw_1$.

All probe and heartbeat data is transmitted over an out-of-band management network (not shown), failures of which are assumed to be uncorrelated to those of the components being monitored. For simplicity, we assume that the management network is reliable.

Conceptually, each of the probe types defined above can be interpreted as testing the availability of a "probe path" consisting of the subset of components and links "traversed" by the probe message. For example, Figure 3.2 illustrates the probe path for the NetBouncer probe from monitor NB_1 to switch Sw_1. We determine probe paths for each of the monitor types in the datacenter based on the following rules:

- Each heartbeat probe "traverses" only the component its monitor is deployed on.
- Each port health probe "traverses" both the link it observes and the switch its monitor is deployed on.
- Each storage service probe traverses all components and links (inclusive) between the client it is deployed on and the server it contacts. However, since we cannot know which spine switch a request has traversed, we must treat the probe paths specially, as we describe below.
- Each NetBouncer probe traverses all components and links (inclusive) between the client it is deployed on and the spine switch it bounces off of.

The administrator has chosen initially to avoid ping-based active monitoring systems (e.g., NetNORAD [50] and deTector [51]) because of the CPU and network bandwidth overhead

they impose and because they require that a pinger *and* responder module be installed on all servers, which adds to system complexity and management cost. In contrast, heartbeat monitors and switch port health monitors can be enabled without installation of additional software, storage service requests are already being performed by clients, and NetBouncer IP-in-IP pings can be performed with client-side pingers only and have low overhead. However, as we demonstrate in Section 3.9, our methodology enables the administrator to evaluate the impact of adding ping-based monitors to the overall resilience of the monitoring infrastructure and to determine whether they are worth the cost.

3.4.1 Failure Localization Approach

Rather than use the data from each type of monitor independently, the administrator plans to combine all of the information available from the monitors that are deployed using a variant of the Tomo failure localization algorithm [60] to detect which components or links are failed. The Tomo algorithm treats the failure localization problem as one of Boolean network tomography. Each component or link in the system can either be "available" or "failed", and for an individual probe to be "successful", all components and links on its probe path must be available. The failure localization problem is that of finding the smallest set of failed components and links, F, such that the probe paths of all unsuccessful probes intersect with F, and the probe paths of all successful probes do not[1].

In the datacenter network, probes are aggregated over a probing interval, so each probe type provides a *probabilistic* measure of the availability of its probe path rather than a Boolean one. As is done by the authors of deTector [51], the administrator considers a probe type to be "unsuccessful" (and therefore indicative of a possible failure along its probe path) if the probe's success probability is below a threshold to be defined based on the datacenter's operating characteristics.

Since the storage service is agnostic of the underlying network topology and cannot pin a request to a particular path through the spine switches, we cannot know the exact path taken by each storage service request. Furthermore, within a given probing interval, different requests may traverse different paths. Thus, we assign different probe paths to successful and unsuccessful storage service probes. Unsuccessful probes indicate that a component or link along either of the two paths between the client and server has failed, so the probe path

[1]As stated, the failure localization problem is mathematically equivalent to the *Minimum Hitting Set* problem [60].

includes both spine switches and their relevant links. Successful probes indicate that at least one path is traversable (but not necessarily both), so the probe path includes all components and links at the ToR level and below, but not the spine switches or links.

Putting everything together, the failure localization algorithm (i.e., the *detector*) works as follows. At each probing interval, the detector records the success status of all probe types it receives. If probe of a particular type is not received, the detector assumes that the probe type was not enabled and ignores it. If none of the probe types are unsuccessful, the detector assumes that no component is in a failed state. Otherwise, the detector assumes that at least one component must have failed, and reports the set of possibly failed components as the intersection of the probe paths of all unsuccessful probe types minus the union of the probe paths of all successful probe types.

Based on prior observations, the administrator has found that using heartbeat messages alone to detect failures generates an untenable number of false positives, so she adds the following rule to the detector: "If the detector observes an unsuccessful heartbeat probe type in a probing interval, it should report a possible failure only if it observes at least one other unsuccessful probe type in the same probing interval."

Detection Goals

The datacenter's administrator aims to achieve 1-identifiability [51] for the failure of all components and links in the system, excluding client hosts. That is, the administrator wants to be able to detect when exactly one component or link fails, but does not explicitly consider simultaneous failures of more than one component or link[2]. Consequently, a failure is considered to have been accurately localized by the detector iff the set of failed components for a given probing interval is of size 1 and contains only the single component that has actually failed.

Deploying a particular monitor allows the datacenter administrator to observe whether the probe types provided by that monitor are successful or unsuccessful, and thereby use that information in localizing failures in the datacenter. However, the administrator's monitoring budget prevents her from deploying all monitors at her disposal; she must determine which monitors to deploy to ensure that she can localize failures for all components and links, even if some of the monitors are compromised.

[2]The authors of deTector [51] find that a probing plan that achieves 1-identifiability can localize failures in large datacenter networks (containing tens of spine switches) with >90% accuracy, even when there are

Thus, the administrator's monitoring system design goals are as follows:

1. To understand how many probe types can be made unavailable by monitor compromise across different numbers of monitors and hosts while *maintaining* 1-identifiablity, assuming all monitors are deployed (**DG1**).

2. To understand how many (and which) monitors must be deployed to guarantee 1-identifiability, and to understand how increasing deployment affects resilience against missing information (**DG2**).

3.5 MODELING MONITORING AND INCIDENT DETECTION

We now define our modeling formalism for the monitoring system and incident detection. Our model is inspired by our previous work [45], but includes important distinctions for monitor compromise analysis.

3.5.1 Monitoring System

In enterprise and cloud systems, diverse monitors collect system information. Common monitors include system logs, application request logs, operating system performance monitors (such as Linux's TOP utility), and firewalls. We collectively refer to the set of all monitors that are or can be deployed within a system, and the supporting infrastructure to collect, store, and analyze monitor data, as the *monitoring system*.

Definition 3.1. A **monitor** is a sensor within a system that generates information about the state of the system that can be used for incident detection. We denote the set of all monitors that can be deployed in a system as M.

Administrators decide which monitors to deploy based on how they intend to analyze data generated by the monitors to detect incidents. We define the concept of *alert types* to represent the types of information that monitors can generate.

Definition 3.2. An **alert type** is a semantic label for a subset of monitor data that uniquely identifies how it is used by incident detection approaches. We denote the set of all alert types within a system as A.

simultaneous failures of multiple links, so we consider 1-identifiability a reasonable failure localization goal. We also note that given the size of our example datacenter, simultaneous failure of more than one component or link would represent failure of over 9% of all components and links, which occurs very infrequently in real systems.

We represent which alert types can be generated by which monitors as an undirected, bipartite graph $G = (M, A, E_G)$, where $E_G \subseteq M \times A$ and $(m, a) \in E_G$ if and only if monitor m can generate alert type a. We require that $\forall a \in A,\ \exists m \in M \mid (m, a) \in E_G$, but it is possible for more than one monitor to generate the same alert type. For simplicity of notation, we represent the set of alert types that can be generated by a monitor by the function $g : M \to 2^A$, where 2^A denotes the power set of A and $g(m) = \{a \mid (m, a) \in E_G\}$.

We use 'alert type' to describe any general transformation on raw monitor data that distinguishes different views of system activity or discrete types of actions performed by hosts or system components. Our definition of alert type is not strictly the same as those used by intrusion detection systems (IDSes), such as Snort [62] or Zeek [63]. Alert types may include, but are not limited to, log message formats (also referred to as log event types [64], log event templates [9], or log keys [34]), time series features [43, 65], conditional and/or temporal logic-based alerts [21], and other features that may be extracted from monitor data by incident detection approaches.

Ultimately, alert types for a particular system should be driven by the types of monitor data available and the detection methods employed, as we illustrate for the datacenter network example in Section 3.5.3.

For an attacker to compromise a monitor, the system component on which the monitor runs must usually be compromised as well. We refer to such components as *hosts*.

Definition 3.3. A **host** is a system component that may run a monitor. We denote the set of hosts within a system as H.

Hosts may include, but are not limited to, physical servers, workstations, virtual machines, and network hardware.

We represent the dependence between monitors and hosts by the undirected, bipartite graph $R = (M, H, E_R)$, where $E_R \subseteq M \times H$ and $(m, h) \in E_R$ iff manipulation of the data generated by monitor m requires access to or control of host h. For simplicity of notation, we use the function $r : M \to H$ to represent the host on which each monitor depends, where $r(m) = h \mid (m, h) \in E_R$. H consists of only those hosts on which at least one monitor depends, and each monitor depends on exactly one host. Some monitors, such as firewalls and middleboxes, are distinct devices that are like hosts. To account for this distinction, we represent such monitors using a host and a monitor running on that host.

At this point, we have completely characterized a monitoring system's design, denoted as $S = (M, H, A, E_G, E_R)$. Given a fixed design, we assume that system designers wish

to determine which subset of monitors to deploy to meet resilience objectives. We do not study the effects of monitor configuration changes or alert type extraction that would require modification of any element of S. Therefore, S should represent a universe of all possible monitors and alert types under consideration. A system designer could compare different configurations for the same set of monitors (e.g., different Apache web server monitoring modules) by rerunning the same analysis over different system designs S'.

Definition 3.4. A **monitor deployment** is a subset of monitors $M_d \subseteq M$ that a system designer may choose to enable in a system. Each deployment, M_d, induces on S the subgraph $D = (M_d, H_d, A_d, E_{G[M_d]}, E_{R[M_d]})$, where $H_d = \bigcup_{m \in M_d} r(m)$, $A_d = \bigcup_{m \in M_d} g(m)$, and $E_{G[M_d]}$ and $E_{R[M_d]}$ are the edge sets of the subgraphs of G and R induced by $M_d \cup A_d$ and $M_d \cup H_d$, respectively.

For a deployment M_d, we denote the induced alert type generation graph as $G[M_d] = (M_d, A_d, E_{G[M_d]})$, and the induced monitor dependence graph as $R[M_d] = (M_d, A_d, E_{R[M_d]})$. For clarity, since all elements of D are completely characterized by M_d, we hereafter use M_d as shorthand for D.

3.5.2 Incident Detection

Incidents cause disruption or damage to an organization's service (e.g., loss of sensitive information). As early detection of incidents is strongly desirable, incident management and response teams invest monitoring and detection capabilities in the detection of early stages of incidents. As such, we define *events* within our model to encompass all such actions.

Definition 3.5. An **event** is a security or reliability incident that an organization wishes to detect, or a precursor action to an incident that might aid in early detection of the incident. We denote the set of all events being considered as E.

In order to detect events, incident management and response teams may employ various different analysis methods over subsets of data collected from the monitors within their system. Detection methods include, but are not limited to, clustering [31], decision trees [66], outlier detection techniques [67], signature-based and rule-based reasoning [62, 63, 68], and neural networks [34, 69]. Ultimately, we observe that for any detection approaches employed, it is possible to characterize the evidence required to detect each event in terms of subsets of data available from the monitors within the system. Thus, a system designer's goal is to

ensure that *sufficient* evidence is available to detect each event of importance, even under compromise of some tolerable degree of the monitoring system.

Formally, we characterize the incident detection infrastructure in terms of *minimally-sufficient sets of evidence* as follows.

Definition 3.6. A **minimally-sufficient set of evidence** (or sufficient evidence set) is a set of alert types that can collectively be used during system operation to detect an event, but no strict subset of which can be used to detect the event. We denote the set of minimally-sufficient sets of evidence for the events in a model as $\Sigma \subseteq 2^A \setminus \emptyset$.

For brevity, we hereafter refer to minimally-sufficient sets of evidence as sufficient evidence sets.

We specify *event detection requirements* in terms of the undirected, bipartite graph $\Delta = (E, \Sigma, E_\Delta)$, where $E_\Delta \subseteq E \times \Sigma$. We require that $\forall \sigma \in \Sigma, \exists e \in E \mid (e, \sigma) \in E_\Delta$, but as it may be possible to detect multiple events with the same sets of information, it is possible for more than one event to map to a sufficient evidence set. Furthermore, minimality of the sufficient evidence sets requires that, $\forall \sigma_i, \sigma_j \in \Sigma$,

$$\sigma_i \subset \sigma_j \rightarrow \nexists e \in E \mid [(e, \sigma_i) \in E_\Delta \wedge (e, \sigma_j) \in E_\Delta] \tag{3.1}$$

In our formalism, the logic underpinning the incident detection mechanisms within a system is completely characterized by the tuple (E, A, Σ, E_Δ), such that any changes to detection rules, detection techniques, or the underlying monitoring system would require changes to the model of the system. We can then define *detectability* of an event as follows.

Definition 3.7. An event is **detectable** if all of the alert types in at least one of the sufficient evidence sets for the event can be generated by the monitors deployed in the system.

Formally, we denote detectability by the function $\delta : E \times 2^{M \times A} \rightarrow \{0, 1\}$, defined as:

$$\delta\left(e, E_{G[M_d]}\right) \leftrightarrow \exists \sigma \in \Sigma \mid [(e, \sigma) \in E_\Delta \wedge \sigma \subseteq A_d] \tag{3.2}$$

where $E_{G[M_d]}$ and A_d are defined as in Definition 3.4.

Note that the domain of the second argument to δ is $2^{M \times A}$ (an edge set from monitors to alert types) rather than 2^M (a subset of monitors). In cases without monitor compromise, by Definition 3.4, the alert type generation graph is completely characterized by M_d, so for notational simplicity, we hereafter use $\delta(e, M_d)$ as shorthand for $\delta\left(e, E_{G[M_d]}\right)$. However, as we

describe in Section 3.6, monitor compromise affects $E_{G[M_d]}$ directly, so we define detectability in terms of the edge set of the alert type generation graph to support later definition of detectability under compromise.

Finally, system designers and administrators are interested in the detectability of multiple events. We define the metric of *coverage* (similarly to Thakore et al. [45]) as follows.

Definition 3.8. The **coverage** of a set of events quantifies the fraction of the events that are detectable given a particular monitor deployment. Formally,

$$\text{Coverage}\left(E^*, E_{G[M_d]}\right) = \frac{\left|\left\{e \in E^* \mid \delta\left(e, E_{G[M_d]}\right)\right\}\right|}{|E^*|} \tag{3.3}$$

where $E^* \subseteq E$.

Generality of our modeling formalism. In production cloud and enterprise systems, incident management and response teams (IMTs and IRTs) often use SIEM tools, log analysis tools, and dashboards to define alerting rules for events considered to be important. The alerting rules defined within said tools can provide the basis for enumerating the events in a given system and the evidence required for their detection.

Any signature-based or rule-based detection system should be amenable to modeling within our formalism, as each signature or rule would constitute a sufficient evidence set and the log messages or metrics that trigger the signature or rule would constitute the alert types. We believe that a system designer could model the important events and evidence required for their detection through consultation with incident management or response teams (IMTs or IRTs) and/or examination of the configuration of the incident detection tools employed.

However, our formalism would be relatively ill-suited to detection systems for which it is not easy to isolate the features or log entries responsible for triggering a detection, as is the case with some techniques that employ deep learning or probabilistic graphical models. Quantifying the value of redundant data for such detection systems would require the use of entirely different modeling techniques, so we leave support of such detectors to future work.

3.5.3 Application to Case Study

To demonstrate how a system designer could construct our model in practice, we describe the process by which we constructed the model for the datacenter network case study we introduced in Section 3.4.

Figure 3.1 depicts all of the hosts and monitors in the system, and shows all monitor-host dependence relationships (E_R) using dotted lines. To complete the characterization of the monitoring system's design, we need to enumerate the alert types generated by each of the monitors. Since the datacenter's administrator uses the Tomo algorithm for detection, which localizes failures by intersecting probe paths, the alert types are given by the unique probe paths in the system. We denote the alert types for each monitor with the lowercase for the monitor generating the alert type and either the source and destination (if applicable) for the probe path, or the generating component and port identifier for port health monitor alert types. The full specification of S for the datacenter system is given by Figure 3.3.

As described in Section 3.4.1, the datacenter's administrator aims to achieve 1-identifiability [51] for the failure of all components and links in the system. Thus, localizing (i.e., detecting) the failure of each individual component or link in the system (excluding client hosts) constitutes a unique event, and sufficient evidence sets for each event are any set of alert types, σ, that can localize the failure of the component or link and for which no strict subset $\sigma\prime \subset \sigma$ can also localize the failure of the same component.

To generate the sufficient evidence sets for the datacenter network, we use the following procedure.

1. For each component or link[3] in the system (excluding clients), we assume that the component has failed, and create an event corresponding to the detection of the *failed component*. Let all alert types that include the failed component in their probe path be the *affected alert types*, and those that do not include it be the *unaffected alert types*. We assume that all affected alert types will be "unsuccessful" (as previously defined in Section 3.4.1), and all unaffected alert types will be "successful".

2. To detect the component's failure, the detector must be able to observe at least one unsuccessful alert type. Thus, we initialize a list of *candidate evidence sets* with a singleton set for each of the affected alert types.

3. For each candidate evidence set, we apply the detector's localization approach—namely, taking the intersection of the probe paths of all unsuccessful alert types in the set and subtracting the union of the probe paths of all successful alert types in the set—and examine the resulting *path intersection set*.

[3]For simplicity, we hereafter refer to both components and links as components.

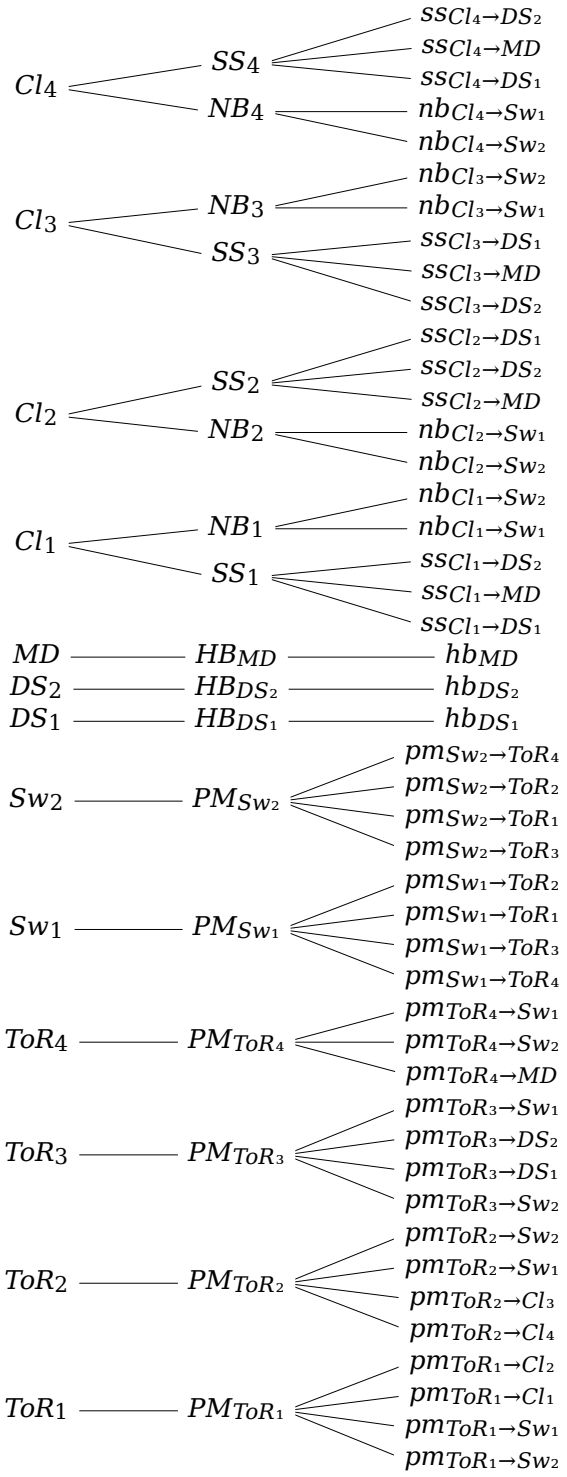


Figure 3.3: Components of the monitoring system model, $S = (H, M, A, E_R, E_G)$, for the datacenter network case study. From left to right, the graph shows the elements of H, the edges in E_R, the elements of M, the edges in E_G, and the elements of A.

(a) If the path intersection set is empty or does not contain the failed component, we skip the candidate evidence set in question, as neither the candidate evidence set nor its supersets can localize the failure of the failed component.

(b) If the path intersection set contains only the failed component[4], we mark the candidate evidence set in question as a sufficient evidence set for the event.

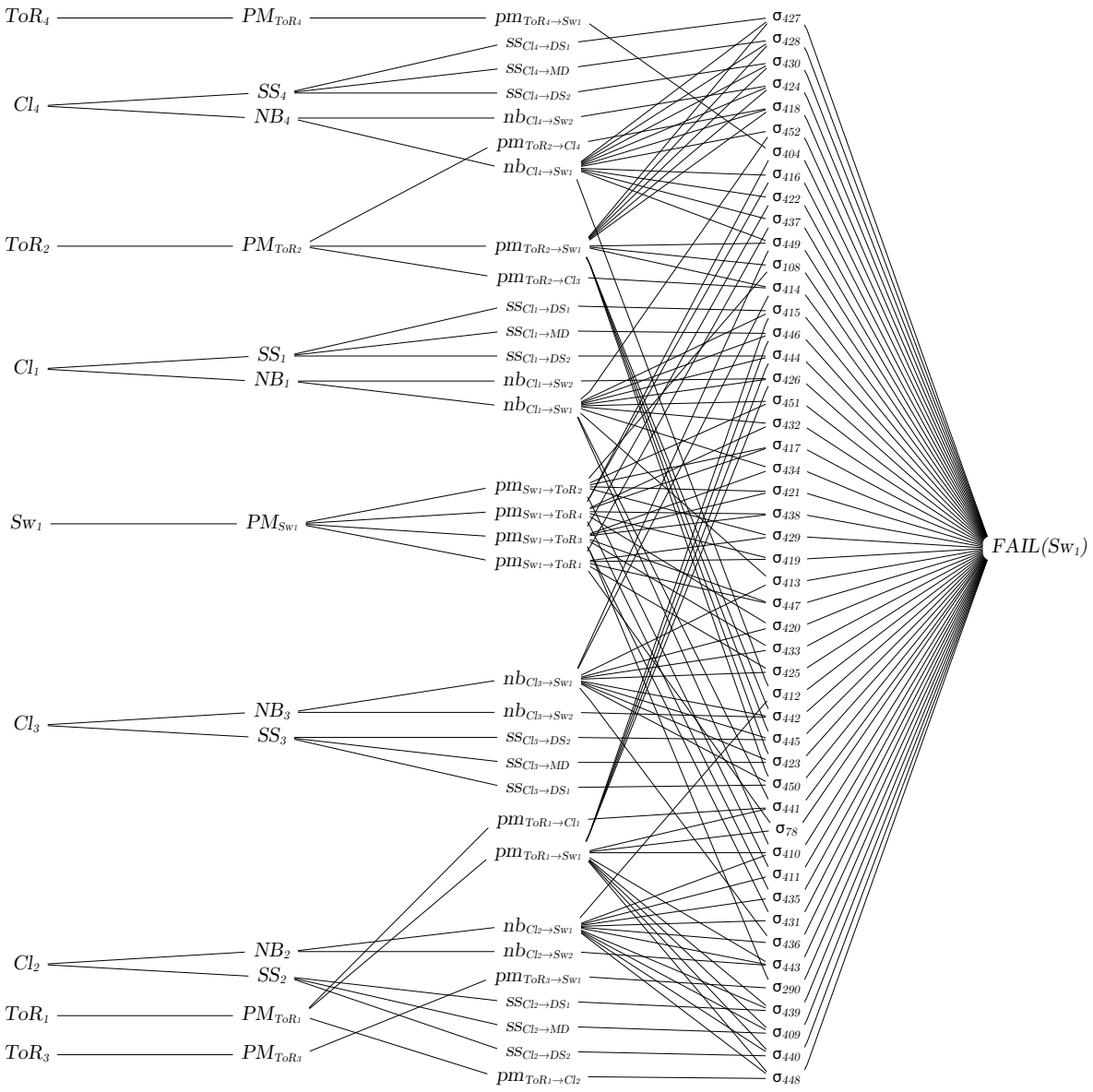
(c) Otherwise, the path intersection set contains more than one component, including the failed component. While the candidate evidence set in question may not be sufficient to localize the failure, one of its supersets might. Thus, for each alert type not already in the candidate evidence set, we create a new candidate evidence set by adding the alert type to the candidate evidence set. If the new candidate set is not a superset of any existing sufficient evidence sets for the event, we add it to the list of candidate evidence sets.

4. We repeat Step 3 until there are no further candidate evidence sets to examine.

Our procedure effectively performs a guided complete search of the space of all possible subsets of alert types. In network tomography literature, our procedure for constructing sufficient evidence sets is akin to algorithms for probe plan construction (e.g., deTector's PMC algorithm [51]) and suffers from the same scalability challenges. Tomo [60], deTector, and NetBouncer [40] all use heuristics and structural properties of the network (e.g., decomposability, path uniqueness) to improve scalability. We consider addressing scalability challenges in model construction to be out of scope for our work, but we note that in our implementation of the case study, through caching and opportunistic pruning of the search tree, we are able to generate the sufficient evidence sets for the case study model within a reasonable amount of time. We also find that bounding the size of the sufficient evidence sets at a sufficienctly large number (e.g., 9 alert types) and implementing the procedure as a breadth-first search can improve scalability without affecting the results of the monitor compromise resistance analysis we perform in Section 3.9.

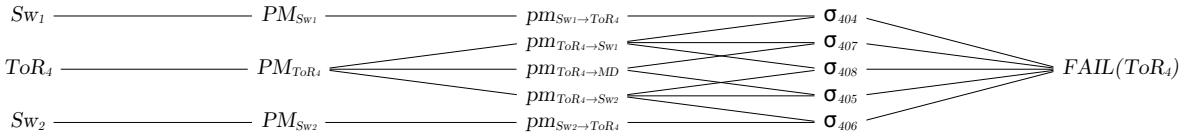
The resulting model contains 5115 unique sufficient evidence sets, with the number of sufficient evidence sets per event ranging from 5 to 1450. As the full model is too large to display, we show subgraphs for a subset of events in Figure 3.4. Of the 5115 unique sufficient evidence sets, 46 can be used to detect two different events, 24 can be used to detect three different events, and all others can only be used to detect a single event. For example,

[4]Because of the special rule for detection using heartbeat messages defined in Section 3.4.1, if the candidate evidence set in question contains only a heartbeat alert type, it is handled using Step 3c instead.

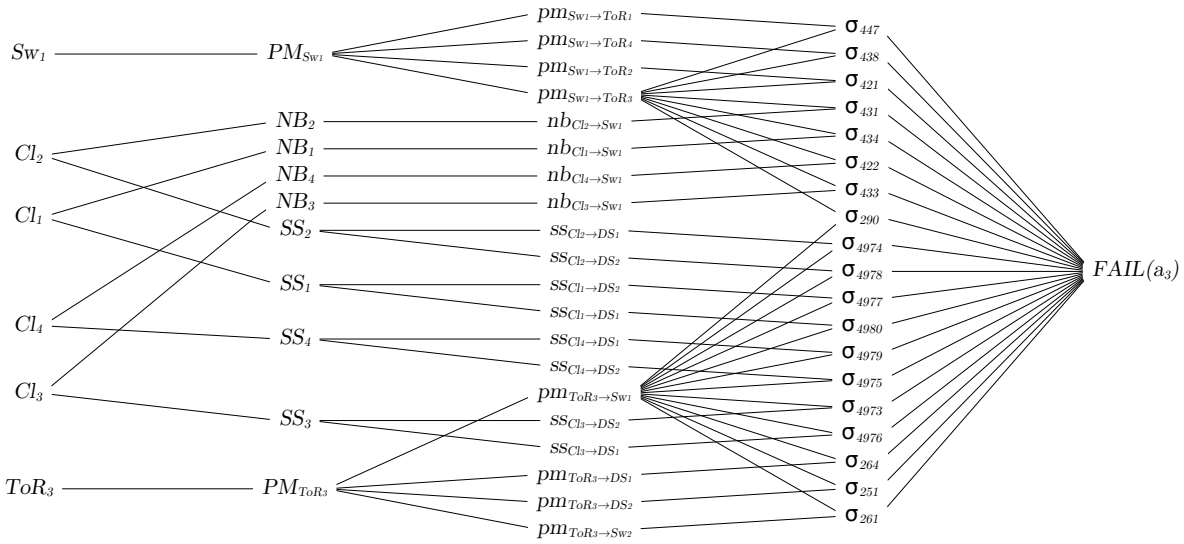


(a) Subgraph of monitoring and incident detection system design graph for $FAIL(Sw_1)$.

Figure 3.4: Subgraphs of the datacenter network case study monitoring and incident detection system design graph for various events. The layers of nodes represent (from left to right) the hosts (H), monitors (M), alert types (A), sufficient evidence sets (Σ), and events (E). Lines between layers in the graph represent (from left to right) edges in E_R, edges in E_G, inclusion in Σ, and edges in E_Δ.



(b) Subgraph of monitoring and incident detection system design graph for $FAIL(ToR_4)$.



(c) Subgraph of monitoring and incident detection system design graph for $FAIL(a_3)$.

Figure 3.4: (cont.) Subgraphs of the datacenter network case study monitoring and incident detection system design graph for various events.

the sufficient evidence set $\{pm_{ToR_2 \to a_2}, pm_{Sw_1 \to a_2}\}$ can be used to detect $FAIL(ToR_2)$, $FAIL(Sw_4)$, and $FAIL(a_2)$.

3.6 ATTACKER AND THREAT MODEL

We now model our attacker and the threats that we consider for monitoring and incident detection. Our model allows system designers to quantify an attacker's costs.

3.6.1 Monitor Compromise Model

We assume that the attacker's ultimate goal is to hamper an organization's ability to detect events within a set of events of interest, E^*, by targeting the system's monitoring system. The loss of these events' detectability could disrupt the organization's operations or could have monetary or security-related consequences.

We assume that the attacker knows our models of the monitoring system and detection mechanisms, but not any priorities or weights placed on the importance of different events. We assume that the mechanism the attacker uses to perform compromise is to *prevent* the generation of a particular alert type by a particular monitor so as to reduce the total set of alert types generated by the monitoring system and cause an event to go undetected.

Formally, we define monitor compromise in terms of the *set* of manipulated monitor and alert type pairs:

Definition 3.9. A specific monitor compromise scenario is characterized by a **compromise set** $C \subseteq E_G$, where the inclusion of $(m, a) \in C$ implies that the attacker can prevent monitor m from generating alerts of type a.

A particular compromise set is only applicable to a monitor deployment M_d if $C \subseteq E_{G[M_d]}$.

Our definition of compromise covers any run-time scenario where the attacker chooses to completely or selectively drop alert types. Note that we intentionally do not consider detection mechanism efficacy metrics (e.g., precision/recall, FPR/TPR) within our approach, as they are essentially unknowable at system design time, even with detailed knowledge of the detection mechanisms. As a result, we do not consider it meaningful in this work to consider attacker injection of alerts of existing alert types, since those would only serve to change detector efficacy rates. Any attempt to drop alerts of existing types is, however, covered by our compromise model. We do not explicitly represent attacker attempts to inject new alert

types, but we argue that most detectors will either ignore such messages or will trigger false positive alerts to alert defenders of the attackers' attempts at compromise.

Monitor compromise affects detectability and coverage of events by modifying the alert type generation graph G. Let $E_G^C = E_G - C$ for a particular compromise set C. Then, the set of alert types that can be generated under compromise, $g^C(m)$ is given by:

$$g^C(m) = \left\{a \mid (m,a) \in E_G^C\right\} \tag{3.4}$$

and the compromised alert type generation graph is given by $G^C = (M, A^C, E_G^C)$, where $A^C = \bigcup_{m \in M} g^C(m)$. A compromised deployment is given by $D^C = \left(H_d, M_d, A_d^C, E_{G[M_d]}^C, E_{R[M_d]}\right)$, which we also denote as M_d^C. Detectability and coverage under compromise can then be computed by simply substituting $E_{G[M_d]}^C$ and A_d^C in place of $E_{G[M_d]}$ and A_d in Definitions 3.7 and 3.8, respectively.

3.6.2 Attacker Cost Model

We assume that an attacker desires to expend minimum effort in the process of compromising the monitoring system. This is because any attempts to compromise the monitoring system generate activity that could be used to detect the attacker's presence, so a stealthy attacker would want to minimize such activity. We thus quantify attacker effort (i.e., cost) as follows.

Definition 3.10. The **compromise cost** is the cost for an attacker to perform the compromise defined by a compromise set C, and is given by the number of hosts and monitors that must be manipulated to prevent the generation of the alert types in C, and by the size of C itself. Formally,

$$\text{CompromiseCost}\ (C) = (\mathbf{h}, \mathbf{m}, \mathbf{a}) \tag{3.5}$$

where $\mathbf{h} = \left|\bigcup_{(m,a)\in C} r(m)\right|$, $\mathbf{m} = \left|\bigcup_{(m,a)\in C} m\right|$, and $\mathbf{a} = |C|$.

The image of the CompromiseCost function is restricted by the design of the monitoring system, S, to the lattice values $(h, m, a) \in \mathbb{N}_{[0,|H|]} \times \mathbb{N}_{[0,|M|]} \times \mathbb{N}_{[0,|E_G|]}$ subject to the following

constraints:

$$h \leq m \leq \max_{H' \subseteq H \mid |H'|=h} \left| \bigcup_{h_j \in H'} \{m_i \mid (m_i, h_j) \in E_R\} \right| \tag{3.6}$$

$$m \leq a \leq \max_{M' \subseteq M \mid |M'|=m \wedge \left|\bigcup_{m_i \in M'} r(m_i)\right|=h} \left| \bigcup_{m_i \in M'} g(m_i) \right| \tag{3.7}$$

We denote the image of CompromiseCost as $\mathbb{N}_S^3$.

We implicitly assume that each host, monitor, and alert type is equally expensive or difficult for the attacker to compromise, and that each host and monitor is reachable by the attacker. Given that the susceptibility of hosts to compromise is variable and dynamic, there is no definitive way to accurately estimate individual probabilities a priori. Furthermore, we note that many enterprise networks, and indeed some cloud networks, are not properly segmented and thus have a relatively flat architecture. Thus, the reachability assumption is an acceptable approximation, and we leave the formulation of a more sophisticated compromise cost to future work.

We believe that given the assumptions we have previously made, our metric for compromise cost is reasonable. First, as is true in real systems, the cost of a particular compromise set is independent of which monitors are deployed, and instead depends solely on the system design and the compromise set itself. The compromise cost metric does not require any prior presumption of relative importance between hosts, monitors, and alert types, so the effect of different weight assignments on the relative ranking of deployments can be studied at analysis time (as we illustrate in Section 3.9). As a consequence, compromise costs are partially ordered—that is, for any two costs, $\mathbf{c}_1 = (\mathbf{h}_1, \mathbf{m}_1, \mathbf{a}_1)$ and $\mathbf{c}_2 = (\mathbf{h}_2, \mathbf{m}_2, \mathbf{a}_2)$, $\mathbf{c}_1 \leq \mathbf{c}_2$ iff $\mathbf{h}_1 \leq \mathbf{h}_2$, $\mathbf{m}_1 \leq \mathbf{m}_2$, and $\mathbf{a}_1 \leq \mathbf{a}_2$; otherwise, $\mathbf{c}_1$ and $\mathbf{c}_2$ are not directly comparable without additional information.

We use our compromise cost metric to represent upper bounds on the capability or willingness of attackers to perform compromises associated with different compromise sets. Recall that we assume that an attacker will choose to limit the compromise cost of their actions based on their concern about being detected. The compromise cost can also serve to describe limits on an attacker's capability to control sufficient numbers of hosts and monitors, or the difficulty of performing such a compromise.

Ultimately, regardless of a system designer's choice of interpretation of compromise cost, in our approach, we assume that an attacker with the ability or desire to perform monitor

compromise of a particular cost $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ will choose the compromise set C that causes the greatest reduction of coverage for which CompromiseCost $(C) \leq (\mathbf{h}, \mathbf{m}, \mathbf{a})$. We thus define the attacker's goal as maximizing their *damage potential* for a particular compromise cost.

Definition 3.11. An attacker's **damage potential** for a given compromise cost $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ is the amount by which the attacker can reduce the defender's coverage of a set of events using any compromise set with cost no greater than $(\mathbf{h}, \mathbf{m}, \mathbf{a})$. Formally,

$$\text{Damage } (E^*, M_d, \mathbf{h}, \mathbf{m}, \mathbf{a}) = \text{Coverage } (E^*, M_d) - \min_{C \mid \text{ CompromiseCost } (C) \leq (\mathbf{h}, \mathbf{m}, \mathbf{a})} \text{Coverage } \left(E^*, M_d^C\right) \tag{3.8}$$

Note about non-malicious compromise. We describe our threat model above in terms of worst-case damage by a malicious attacker intentionally sabotaging monitor data. However, even in the case of non-malicious unavailability of monitor data, it is still generally true that the number of hosts that will fail will be much smaller than the number of monitors, and smaller still than the number of alert types, and loss of a particular alert type at a point in time (a failure) requires that its generating monitor and/or dependent host experience some type of fault. The cost of a compromise can be seen as a lower bound on the number of hosts, monitors, and alert types that must fail in conjunction for the consequences of said compromise (loss of detectability) to manifest. We believe that as a result, our compromise model is equally well-suited to examination of a monitoring system's susceptibility to non-malicious failures.

3.7 QUANTIFYING MONITORING RESILIENCE

We now consider the resilience of a monitor deployment in terms of the system designer's (i.e., the defender's) ability to detect events, even under monitor compromise. To quantify the defender's objective, we define a deployment's *compromise tolerance* as follows.

Definition 3.12. The **tolerance** of a monitor deployment against compromise by an attacker with a given capability is the maximum coverage that can be achieved assuming the attacker

chooses the compromise set that maximizes their damage potential. Formally,

$$\begin{aligned}\text{Tolerance}\ (E^*, M_d, \mathbf{h}, \mathbf{m}, \mathbf{a}) &= \text{Coverage}\ (E^*, M_d) - \text{Damage}\ (E^*, M_d, \mathbf{h}, \mathbf{m}, \mathbf{a}) \\ &= \min_{C \mid \text{CompromiseCost}(C) \leq (\mathbf{h},\mathbf{m},\mathbf{a})} \text{Coverage}\ \left(E^*, M_d^C\right)\end{aligned} \tag{3.9}$$

If we fix the set of events of interest, E^*, the tolerance of a deployment is a scalar field that assigns a value in $\mathbb{R}_{[0,1]}$ to each possible value of compromise cost. Let us denote said function as $\text{Tolerance}_{E^*,M_d} : \mathbb{N}_S^3 \to \mathbb{R}_{[0,1]}$. We define a deployment M_d as being $(\mathbf{h}, \mathbf{m}, \mathbf{a})$-*compromise tolerant* for a set of events E^* if $\text{Tolerance}_{E^*,M_d}\ (\mathbf{h}, \mathbf{m}, \mathbf{a}) > 0$.[5]

$(\mathbf{h}, \mathbf{m}, \mathbf{a})$-compromise tolerance provides a lower bound on the minimum compromise cost required for an attacker to make *all* events in E^* undetectable. However, for a set of events of high importance, such as those that might serve as early indicators of a catastrophic failure or highly damaging attack, a defender would likely instead desire to know what level of compromise the monitoring system could tolerate without loss of detectability of *any* of the events. We thus consider a deployment M_d to be $(\mathbf{h}, \mathbf{m}, \mathbf{a})$-*compromise resistant* for a set of events E^* if $\text{Tolerance}_{E^*,M_d}\ (\mathbf{h}, \mathbf{m}, \mathbf{a}) = 1$.

An attacker's capability is not known in advance, so the defender wishes to find a monitor deployment that maximizes the values of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for which the monitoring system is resistant to monitor compromise, subject to cost constraints on monitoring. As compromise costs are partially ordered, finding the maximal value of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for a deployment's compromise resistance can be formulated as a multi-objective optimization problem with each of $\mathbf{h}$, $\mathbf{m}$, and $\mathbf{a}$ serving as objective functions. However, multi-objective optimization may not always produce a single, globally optimal solution that simultaneously optimizes each objective function. Thus, we look to find the set of *non-dominated* or *Pareto optimal* solutions, defined as follows.

Definition 3.13. For a multi-objective minimization problem consisting of n objective functions $\{f_1, ..., f_n\}$ evaluated over a set X of feasible decisions, a feasible solution $x_1 \in X$ is said to *dominate* another feasible solution $x_2 \in X$ if:

$$\begin{aligned}&\forall i \in \{1, ..., n\}, f_i(x_1) \leq f_i(x_2), \text{ and} \\ &\exists j \in \{1, ..., n\}, f_i(x_1) < f_i(x_2)\end{aligned} \tag{3.10}$$

[5] We observe that $\text{Tolerance}_{E^*,M_d}$ is monotonically nonincreasing in each dimension of CompromiseCost and is therefore an order-preserving function, but isosurfaces of $\text{Tolerance}_{E^*,M_d}$ are non-convex.

A solution x is considered **non-dominated** (or **Pareto optimal**) if no other solution $x' \in X$ dominates x. The set of all such solutions is called the **Pareto boundary** or **Pareto surface**.

There may thus be multiple non-dominated values of $(\mathbf{h}, \mathbf{m}, \mathbf{a}) \in \mathbb{N}_S^3$ for which a deployment is compromise resistant. The Pareto boundary for compromise resistance completely characterizes the region specified by $[\text{Tolerance}_{E^*,M_d} = 1]$ since, by definition, all values of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for which M_d is resistant must be dominated by at least one of the Pareto optimal values. We refer to the set of such $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ as the *resistance surface* of the deployment. The resistance surface of a deployment is the lower bound on the effort required for an attacker to cause *any* reduction to the coverage of E^*.

To find a single, maximally compromise-resistant monitor deployment for a given cost constraint, it is necessary to define a function that imposes a strict ordering over different resistance surfaces. The function should weight the relative importance of resistance against different levels of compromise cost.

As we noted in Section 3.6.2, we believe that the system designer should be able to provide such a function based on their own requirements and domain expertise and be able to compare the effects of using different weights on the optimal monitor deployment chosen. Thus, we allow the system designer to provide an *importance score*, $\text{Importance} : \mathbb{N}_S^3 \to \mathbb{R}$, that weights the importance of resilience against compromise at each level of compromise cost. The importance score can be interpreted as an inverse of the difficulty for an attacker to perform monitor compromise of a particular compromise cost. We can then collapse the resistance surface for each deployment to a single number, which we call the *resistance score*.

Definition 3.14. The **resistance score**, denoted as RS, is the sum of the importance score for each compromise cost against which the deployment is resistant:

$$\begin{aligned} \text{RS}\ (E^*, M_d) &= \sum_{(\mathbf{h},\mathbf{m},\mathbf{a}) \in \mathbb{N}_S^3} \left([\text{Tolerance}_{E^*,M_d}\ (\mathbf{h}, \mathbf{m}, \mathbf{a}) = 1] \times \text{Importance}\ (\mathbf{h}, \mathbf{m}, \mathbf{a}) \right) \\ &= \sum_{\substack{(\mathbf{h},\mathbf{m},\mathbf{a}) \in \mathbb{N}_S^3 \\ \text{Tolerance}_{E^*,M_d}\ (\mathbf{h},\mathbf{m},\mathbf{a})=1}} \text{Importance}\ (\mathbf{h}, \mathbf{m}, \mathbf{a}) \end{aligned} \tag{3.11}$$

It is possible to similarly define a tolerance score as the weighted sum of all tolerance values and importance scores, but unlike the resistance score, which can be computed by finding the Pareto boundary of the region defined by $[\text{Tolerance}_{E^*,M_d} = 1]$, computing such a tolerance

score would require computing $\text{Tolerance}_{E^*,M_d}$ for all values of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ within the region defined by $[\text{Tolerance}_{E^*,M_d} > 0]$, which is significantly more computationally expensive and can quickly become intractable even for small models. Furthermore, such a score does not have as well-defined a meaning, as it includes compromise costs for which the coverage of the events in E^* is small. We also note that for singleton sets of events (i.e., $E^* = \{e_i\}$ for some i), the tolerance score is the same as the resistance score.

While we allow the system designer to provide any arbitrary importance score function, we propose the following baseline function that places larger importance on attacks that are easier/cheaper to perform, and weights hosts, monitors, and alert types in proportion to the sizes of the relations between them in the system design:

$$\text{Importance } (\mathbf{h}, \mathbf{m}, \mathbf{a}) = \left(1 + \ln\left(1 + \mathbf{h}\right) \ln\left(1 + \mathbf{m}\frac{|H|}{|E_R|}\right) \ln\left(1 + \mathbf{a}\frac{|M|}{|E_G|}\right)\right)^{-1} \tag{3.12}$$

Note that we intentionally do not use a priori methods such as scalarization [70] to convert the multi-objective problem of finding the maximal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$-resistance to a single-objective problem. That is because we are interested in quantifying the resistance of a deployment against *all* possible levels of monitor compromise, and not simply the highest level of resistance achievable. The problem with applying scalarization is that a poor choice of function used to weight the compromise costs could cause undesired or unexpected effects on the deployment considered optimal. For example, an importance score that is a linear function of the number of alert types required for a compromise would consider a deployment that is resistant to compromises with a low number of hosts and arbitrary number of alert types but susceptible to any compromise with even a few hosts to be more desirable than one that is resistant to compromises with a high number of hosts but that requires slightly fewer alert types to compromise, even if ultimately the marginal difference in difficulty between attacking all alert types generated by a monitor and attacking all but a few is nominal.

By using the a posteriori method of finding the Pareto boundary first, then applying a weighting function to the boundary, we allow the system designer to discover the effects of a given choice of importance function by comparing it against others and evaluating whether the assumptions behind the function are reasonable. For some choices of importance function, it may be possible to convert the function into a scalarization function on the values of the objective function for which an a priori method could find the optimal deployment. We do not explore this any further in this paper, but consider it to be a possible avenue for future work.

3.8 RESILIENT MONITOR DEPLOYMENT

We revisit the earlier design goals from Section 3.3 to analyze how they influence our problem formulation for monitor deployment. We also describe how we efficiently compute resistance scores and maximally-resistant deployments.

3.8.1 Goals and problem formulation

Design Goal DG1 (Resilience)

If a full deployment of monitors (i.e., $M_d = M$) is highly susceptible to monitor compromise, then the system designer will need to consider adding new monitors, new detection mechanisms, or both, to the monitoring system design before considering which subset of monitors to deploy.

If a system is found to be insufficiently resilient, or "underprovisioned", the designer can compare different system designs by comparing the resistance surfaces/scores and/or tolerance values of full deployments for different system designs until one with sufficient capability to tolerate compromise is identified. We show how this could be done for our case study datacenter network in Section 3.9.

One analysis that would be of particular importance for reliability engineering of datacenter and cloud environments is the boundary of the resistance surface and tolerance plots taken at the maximum value for **a** for each value of **h** and **m**. Those boundaries show the effect on resistance and tolerance, respectively, of different numbers of monitors being taken completely offline (i.e., have all alerts compromised) across different numbers of hosts.

Design Goal DG2 (Cost-efficiency)

In the case where the monitoring system design is "overprovisioned," the system designer will then need to identify a subset of monitors to deploy to maximize resilience against monitor compromise, subject to some cost constraint on the monitoring system.

We thus formulate the problem faced by a system designer of finding a resilient monitor deployment as follows:

Problem 3.1. For a monitoring system $S = (M, H, A, E_G, E_R)$, the *maximally compromise-resistant monitor deployment* for a monitoring cost constraint MC_{max} can be found by solving

the following optimization equation:

$$\begin{aligned} \underset{M_d}{\text{argmax}} \quad & \text{RS}\ (E^*, M_d) \\ \text{s.t.} \quad & \text{MonitoringCost}\ (M_d) \leq \text{MC}_{max} \\ & M_d \in \{0,1\}^{|M|} \end{aligned} \tag{3.13}$$

where MonitoringCost : $2^M \rightarrow \mathbb{R}$ defines a cost function for a deployment of monitors.

In production enterprise and cloud systems, we have observed that the largest factors in a system designer's assessment of the cost of deploying a monitor are 1) the *engineering costs* associated with installing the monitor, maintaining (i.e., updating, debugging, refreshing) the monitor, storing the data generated by the monitor, and integrating the monitor with existing detection frameworks, and 2) the *human costs* (i.e., time and attention) associated with responding to alerts generated by the monitor and sifting through the monitor's data during incident analysis. Resource utilization costs, while not inconsequential, often play much less of a role in monitoring system design. As such, much of the true cost imposed by adding a new monitor is influenced by the expertise of the system administrators and engineers, the existence of tools that automate the deployment and operation of the monitor, whether existing detection tools can ingest the monitor's data, and even whether a particular type of monitor is already deployed anywhere else in the system. It is fundamentally difficult to devise a single cost model that accurately represents all of the various costs that a system designer would need to consider during monitoring system design.

While it is possible to consider sophisticated formulations for monitoring cost, such as the one proposed in [45], we use a simple cost model in which the cost of a deployment M_d is given by the number of monitors, $|M_d|$. We note, however, that the algorithm we describe below to find the optimal deployment is capable of supporting arbitrary monitor cost assignments.

3.8.2 Solution algorithms

Achieving goal **DG1** requires us to efficiently compute the Pareto boundary for the resistance surface for the full deployment of monitors for a system design, which (in theory) requires a search over the entire space of $2^{|E_G|}$ compromise sets. Achieving goal **DG2** further requires us to search over all possible $2^{|M|}$ deployments for a given system design,

which requires the computations of the resistance score and the resistance surface for each deployment.

For our approach to be practically applicable, we devise methods to efficiently solve both of the optimization problems. To compute the resistance surface and score, we leverage properties of the tolerance metric to come up with an algorithm to walk the resistance surface, and convert the multi-objective optimization problem of finding points on the surface into a constraint programming problem that can be quickly solved by a satisfiability solver. To find the maximally compromise-resistant monitor deployment, we use a simple genetic algorithm [71].

Computing the resistance score

Recall that the resistance surface is defined as the set of values of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for which $\text{Tolerance}_{E^*, M_d}(\mathbf{h}, \mathbf{m}, \mathbf{a}) = 1$ that are not dominated by any other value in the set. To compute the resistance surface, it is necessary to search over the space of compromise sets and check for each compromise cost $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ whether there are any C for which $\text{CompromiseCost}(C) \leq (\mathbf{h}, \mathbf{m}, \mathbf{a})$ and $\text{Coverage}\left(E^*, M_d^C\right) < 1$.

Because all three of $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ are defined as set cardinalities in Definition 3.10,

$C \subset C' \rightarrow \text{CompromiseCost}(C) < \text{CompromiseCost}(C')$. Thus, if we can find some C for which $\text{Coverage}\left(E^*, M_d^C\right) < 1$, then we can prune from the search space all C' for which $\text{CompromiseCost}(C') \geq \text{CompromiseCost}(C)$, and similarly prune all $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ that dominate $\text{CompromiseCost}(C)$ from our search for the resistance surface.

Furthermore, observe that if $\text{Tolerance}_{E^*, M_d}(\mathbf{h}, \mathbf{m}, \mathbf{a}) < 1$, it must be the case that for some $e \in E^*$, $\text{Tolerance}(\{e\}, M_d, \mathbf{h}, \mathbf{m}, \mathbf{a}) = 0$, so we can further simplify the search by decomposing it into separate searches for the resistance surface for each singleton event in E^*. Since the set of alert types relevant for each event is usually much smaller than the union across all events, decomposing the problem in this way should improve efficiency.

Therefore, rather than search for the set of maximal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for which $\text{Tolerance}_{E^*, M_d}(\mathbf{h}, \mathbf{m}, \mathbf{a}) = 1$, it is more efficient to search for the set of minimal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for which $\text{Tolerance}_{E^*, M_d}(\mathbf{h}, \mathbf{m}, \mathbf{a}) < 1$. To further simplify the search, we observe that Definition 3.13 implies the following:

Theorem 3.1. If, for some monitor deployment M_d and set of events E^*, the point $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ is on the Pareto boundary of the region for which $\text{Tolerance}_{E^*, M_d} < 1$, then for all other

points $(\mathbf{h}', \mathbf{m}', \mathbf{a}')$ on the Pareto boundary, (a) at least one of $\mathbf{h}' > \mathbf{h}$, $\mathbf{m}' > \mathbf{m}$, or $\mathbf{a}' > \mathbf{a}$, and (b) at least one of $\mathbf{h}' < \mathbf{h}$, $\mathbf{m}' < \mathbf{m}$, or $\mathbf{a}' < \mathbf{a}$.

Proof. Assume by way of contradiction that there exists a Pareto optimal point $(\mathbf{h}', \mathbf{m}', \mathbf{a}')$ violating (a). Then, $(\mathbf{h}', \mathbf{m}', \mathbf{a}') \leq (\mathbf{h}, \mathbf{m}, \mathbf{a})$, so by definition, $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ is not Pareto optimal, so it could not have been on the Pareto boundary. Similarly, if there exists an $(\mathbf{h}', \mathbf{m}', \mathbf{a}')$ violating (b), then $(\mathbf{h}, \mathbf{m}, \mathbf{a}) \leq (\mathbf{h}', \mathbf{m}', \mathbf{a}')$, so $(\mathbf{h}', \mathbf{m}', \mathbf{a}')$ is not Pareto optimal and is therefore not on the Pareto boundary. QED.

If we can efficiently find a single non-dominated, minimal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ on the Pareto boundary, there are 12 explicit possible categories for all other points on the boundary: 6 categories where one of $\mathbf{h}$, $\mathbf{m}$, or $\mathbf{a}$ is the same, and 6 where all three values are different. We can further reduce the number of categories to 6 by requiring instead that $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ be lexicographically minimal. Without loss of generality, if $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ is minimal for the lexicographical ordering given by minimizing $\mathbf{h}$, then $\mathbf{m}$ then $\mathbf{a}$, all other points $(\mathbf{h}', \mathbf{m}', \mathbf{a}')$ on the Pareto boundary must fall into one of the following categories: $(\mathbf{h}-, \mathbf{m}\uparrow, \mathbf{a}\downarrow)$, $(\mathbf{h}\uparrow, \mathbf{m}-, \mathbf{a}\downarrow)$, $(\mathbf{h}\uparrow, \mathbf{m}\uparrow, \mathbf{a}\downarrow)$, $(\mathbf{h}\uparrow, \mathbf{m}\downarrow, \mathbf{a}\downarrow)$, $(\mathbf{h}\uparrow, \mathbf{m}\downarrow, \mathbf{a}-)$, or $(\mathbf{h}\uparrow, \mathbf{m}\downarrow, \mathbf{a}\uparrow)$, where, e.g., $\mathbf{h}\uparrow$ denotes $\mathbf{h}' > \mathbf{h}$.

We use the properties above to "walk" the entire Pareto boundary of the region for which $\text{Tolerance}_{E^*, M_d} < 1$. The resistance surface is then given by the maximal points in $\mathbb{N}_S^3$ that are not dominated by any points on the boundary.

To efficiently find the lexicographically minimal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$ for a given deployment, we transform the problem of finding the minimal-cost compromise set for which $\text{Tolerance}_{E^*, M_d} < 1$ into a constraint programming problem by observing that our definition of detectability can be represented as a Boolean satisfiability problem over the edges in E_G. We have implemented the constraint programming problem using the Google OR-Tools library [72] in Python, and use the CP-SAT solver to find each minimal $(\mathbf{h}, \mathbf{m}, \mathbf{a})$.

The last step required to compute the resistance score is to take the sum of importance scores for all points in $\mathbb{N}_S^3$ dominated by the resistance surface. The worst case time complexity for this operation is $O\left(|H_d|\,|M_d|\,\left|E_{G[M_d]}\right|\right)$, but for a given choice of importance function, it can be reduced to linear in the size of the resistance surface by precomputing the cumulative sum of importance scores dominated by each value of $(\mathbf{h}, \mathbf{m}, \mathbf{a}) \in \mathbb{N}_S^3$, then walking the Pareto front given by the resistance surface (in any order) and computing the resistance score using the principle of inclusion-exclusion. We have implemented all steps of the solution approach we describe above in Python. We discuss its performance in Section 3.9.

Computing maximally-resistant deployments

Because Problem 3.1 is a 0-1 nonlinear optimization problem, an enumerated search would not be efficient to find the optimal deployment for even moderately-sized models. We find that the problem is amenable, however, to a simple genetic algorithm (SGA) [71] search. We implemented the SGA search using the pygmo library [73] in Python. To determine the parameters for the genetic algorithm, we performed a grid search, varying population size in the range $[|M|, 5|M|]$, crossover probability in the range $[0.7, 0.9]$, and mutation probability in the range $[0.01, 0.6]$. For each set of parameter values, we ran ten trials for various randomly-generated models of varying sizes and observed the number of generations required for convergence and the optimal resistance score obtained for each. We used the tournament selection method and binomial crossover strategy in our experiments.

We observed that the genetic algorithm was not particularly sensitive to the crossover probability, and that increasing the population size corresponded with faster convergence, but otherwise did not affect the outcome of the search. However, the approach was highly sensitive to the mutation probability—at values less than 0.05, the SGA was very slow to converge, and at values much larger than 0.25, it often yielded suboptimal results for models with a larger number of monitors. Thus, we used population size $= 5|M|$, crossover probability $= 0.9$, mutation probability $= 0.15$, and number of generations $= 75$.

While the SGA search is agnostic of the definitions of the monitoring cost and importance score, it tends to converge more quickly for definitions that are convex, such as those we provide in Section 3.7. We leave detailed examination of convergence characteristics of the SGA search to future work, but note that in our experiments, the approach converges to near the globally optimal deployment relatively quickly for moderate-sized models.

3.9 EXPERIMENTAL EVALUATION

To illustrate how our approach achieves the two goals we lay out in Section 3.3, which require examining properties of a particular model in detail, we generated a smaller random model, which we hereafter refer to as Model X, whose graph structure mimics that of the case study system but can be shown in its entirety. Model X contains 4 hosts, 10 monitors, 35 alert types, and 10 events. Figure 3.5 shows the complete system design for Model X.

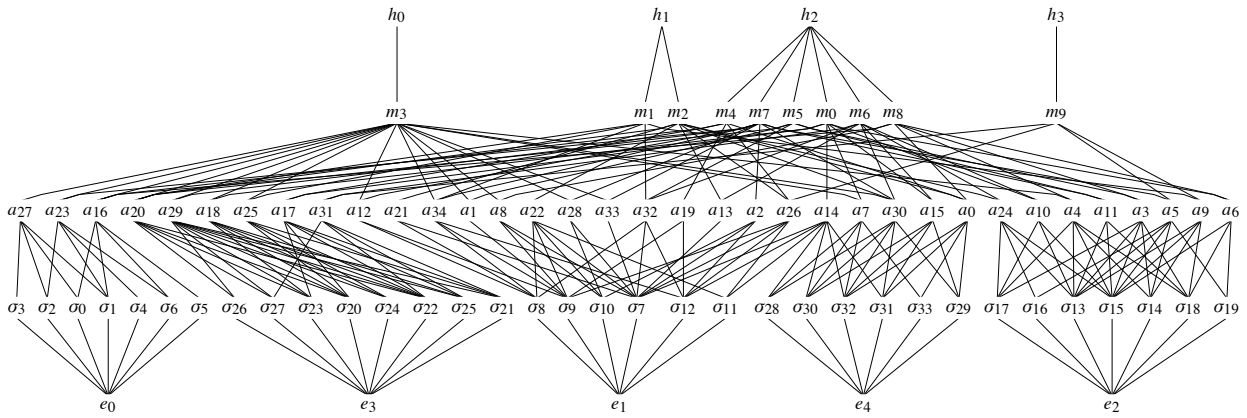


Figure 3.5: Monitoring and incident detection system design graph for Model X. Hosts are denoted as h_i, monitors as m_i, alert types as a_i, sufficient evidence sets as σ_i, and events as e_i. Lines between layers in the graph represent (from top to bottom) edges in E_R, edges in E_G, inclusion in Σ, and edges in E_Δ.

3.9.1 Examining resilience of system design

To address **DG1**, a system designer can look at the resistance surface for their system design. Figure 3.6 shows the the resistance surface for Model X. From examining the surface, it is apparent that Model X is somewhat tolerant to monitor compromise, as it can tolerate compromises that affects any number of alert types across any number of monitors on at most one host. However, to reduce the defender's coverage, an attacker would need only be able to compromise one monitor each on two different hosts.

A system designer determining how to protect the system described by Model X may consider setting up additional monitors on hosts other than h_2, as most of the monitors in the model depend on that host according to Figure 3.5. Indeed, as we show in Section 3.9.2, the system designer would need to deploy most of the current monitors to ensure any amount of tolerance against compromise.

3.9.2 Finding cost-optimal monitor deployments

To answer **DG2**, we run the SGA search described in Section 3.8.2 for Model X. Table 3.1 shows the best resistance scores obtained by each of 25 independent runs of the SGA for the model for important values of MC_{max}.

The resistance score of the maximally compromise-resistant monitor deployment is 0 for $\mathrm{MC}_{max} \leq 2$, meaning that no deployment of 2 or fewer monitors can detect any of the events. For $3 \leq \mathrm{MC}_{max} \leq 5$, the $\mathrm{RS} = 1$, meaning that while deployments exist that enable

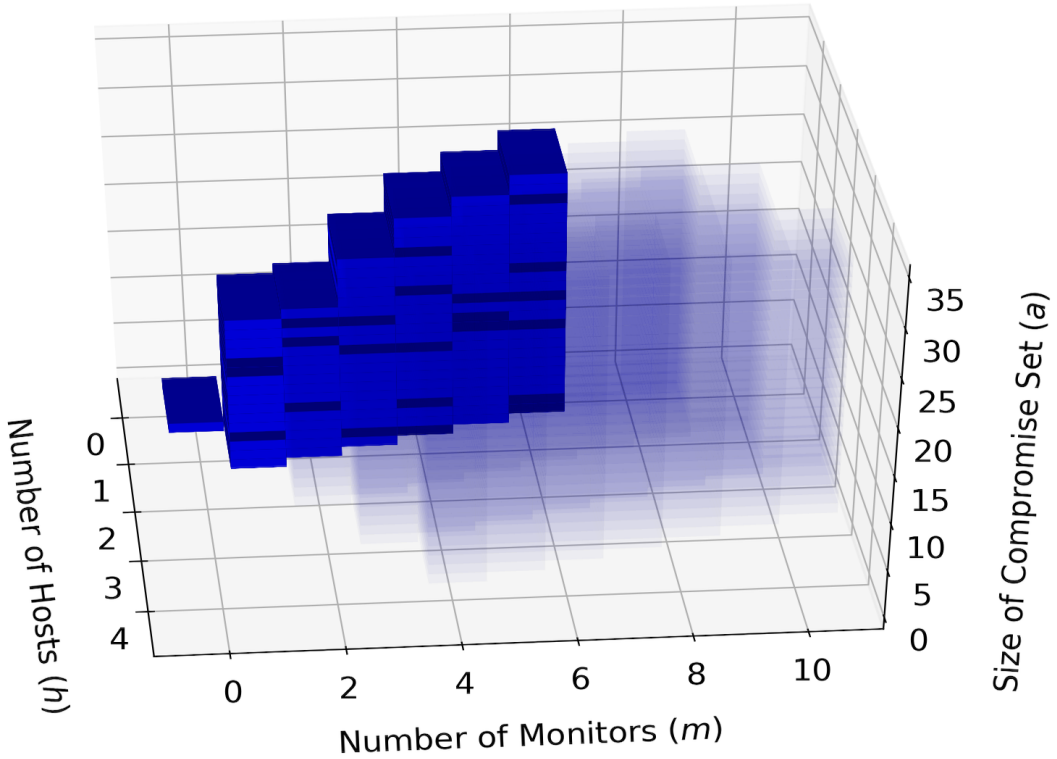


Figure 3.6: Resistance surface for full deployment of monitors for Model X. The opaque regions describe the set of compromise costs against which the models are resistant to compromise, and the shaded regions delineate $\mathbb{N}_S^3$.

detection of all of the events, compromise of even one alert type would make some event undetectable. Only when at least 6 monitors are deployed can the monitoring system tolerate some compromise, but a system designer may decide to deploy more monitors to improve the system's resistance to compromise.

Examining the specific sets of monitors within the maximally compromise-resistant monitor deployments for each value of MC_{max} reveals that the optimal deployments do not always include the same monitors. For example, notice in Table 3.1 that though m_6 is optimal for $\mathrm{MC}_{max} = 5$, it is not part of subsequent optimal deployments for $6 \leq \mathrm{MC}_{max} \leq 7$.

3.9.3 Comparison with DSN16

As we previously described, one of the key distinguishing factors between our approach and DSN16, is that our definition of tolerance accurately reflects the minimum effort required to compromise a monitoring system. In DSN16, resilience of a deployment against compromise is quantified through the **Redundancy** and **Compromise** metrics, both of which treat

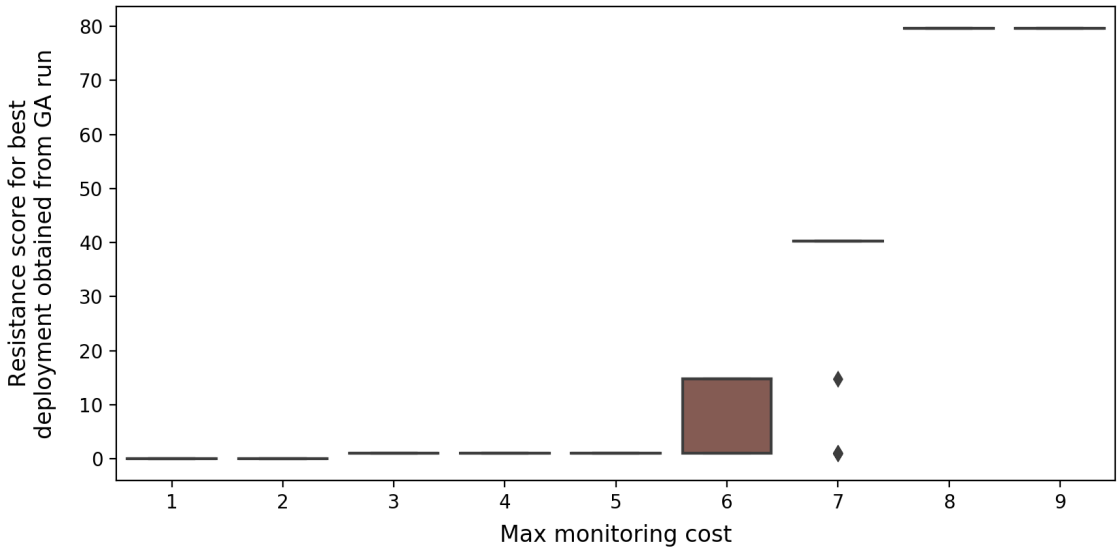


Figure 3.7: Resistance scores for optimal deployments of monitors for different values of MC_{max} for Model X.

MC_{max}	RS	Monitors in the optimal deployment
3..5	1.0	$\{m_2, m_6, m_7\}$
6	14.738	$\{m_0, m_2, m_3, m_4, m_5, m_7\}$
7	40.265	$\{m_0, m_2, m_3, m_4, m_5, m_7, m_9\}$
8	79.664	$\{m_0, m_1, m_2, m_3, m_4, m_6, m_7, m_9\}$

Table 3.1: Monitors in the maximally compromise-resistant monitor deployment for different values of MC_{max} and their corresponding resistance scores.

different sufficient evidence sets as *independent* mechanisms to detect an event. Therefore, if a model contained sufficient evidence sets $\sigma_1 \subset \sigma_2$ such that $\exists e \in E, (e, \sigma_1) \in E_\Delta \wedge (e, \sigma_2) \in E_\Delta$, then given an M_d for which $\delta(e, M_d) = 1$, $\mathbf{Redundancy}(e, M_d) = 2$, despite the fact that detectability of both σ_1 and σ_2 require generation of the same alert types (called 'indicators' in DSN16) and could be compromised with the same compromise set. In contrast, our definitions of resistance and tolerance explicitly capture the amount of effort required by an attacker to compromise detectability of events, thus allowing the resiliency of a monitoring system to be studied properly.

As a result, the optimal deployment obtained by attempting to maximize **Redundancy** while requiring $\min_{\mathbf{Coverage}} = 1$ using the approach presented in DSN16, as suggested in that paper, does not necessarily yield the most resilient monitor deployment. To compare our approach against that of DSN16, we implemented the cost-constrained optimal deployment problem described in the paper with the weights and constraints described above, and solve

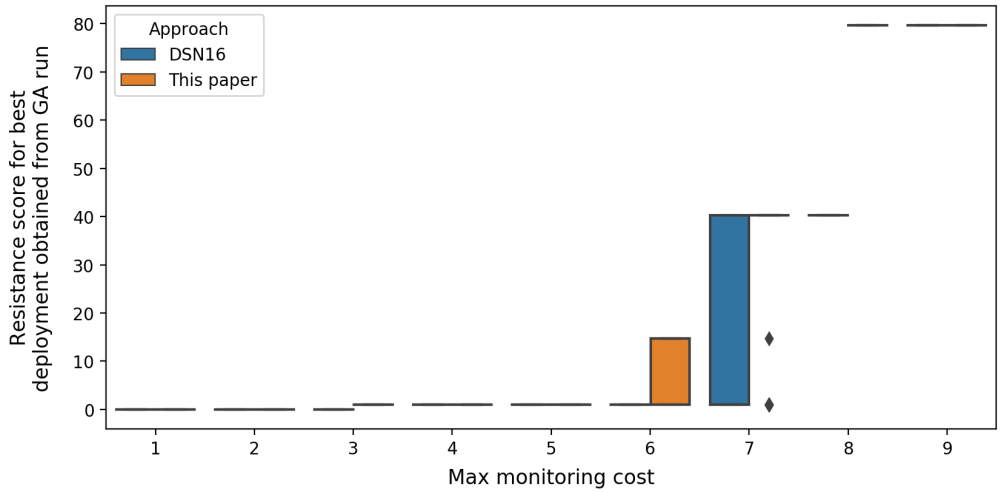


Figure 3.8: Boxplot of resistance scores of optimal deployments obtained using DSN16 approach and our approach for Model X. Notice that for $\text{MC}_{max} = 3$, 6, and 8, the DSN16 approach is unable to find the most resilient deployment obtained by our approach.

the optimization problem using the same SGA as for our approach. Figure 3.8 compares the resistance scores for the optimal deployments obtained using the DSN16 approach against the ones obtained using our approach for Model X. For $\text{MC}_{max} = 3$, 6, and 8, the DSN16 approach is unable to find the most resilient deployment—for $\text{MC}_{max} = 3$, it is unable to even find a deployment that is able to detect all events, despite one existing. We find the same to be true for other models, for the reason described above.

3.9.4 Scalability

Finally, we provide some results that demonstrate the scalability of our approach. For this evaluation, we randomly generated 50 models with varying numbers of monitors, hosts, alert types, and events and varying graph structures. Figure 3.9 shows the distribution of parameters for the models. We ran our experiments on a server with an 8-core Intel Xeon CPU E5-2450 processor and 64GB of memory. Figure 3.10 shows the boxplot of the time to compute the resistance score (in seconds) against the value of MC_{max}. As can be seen from the plot, the time to compute the resistance score does not vary significantly based on the number of monitors to deploy. However, as shown by Figure 3.11, the time does vary linearly with the number of points on the Pareto boundary of the region given by $\text{Tolerance}_{E^*,M_d} < 1$, as described in Section 3.8.2. The average time to compute a resistance score across all models was under 0.2 seconds.

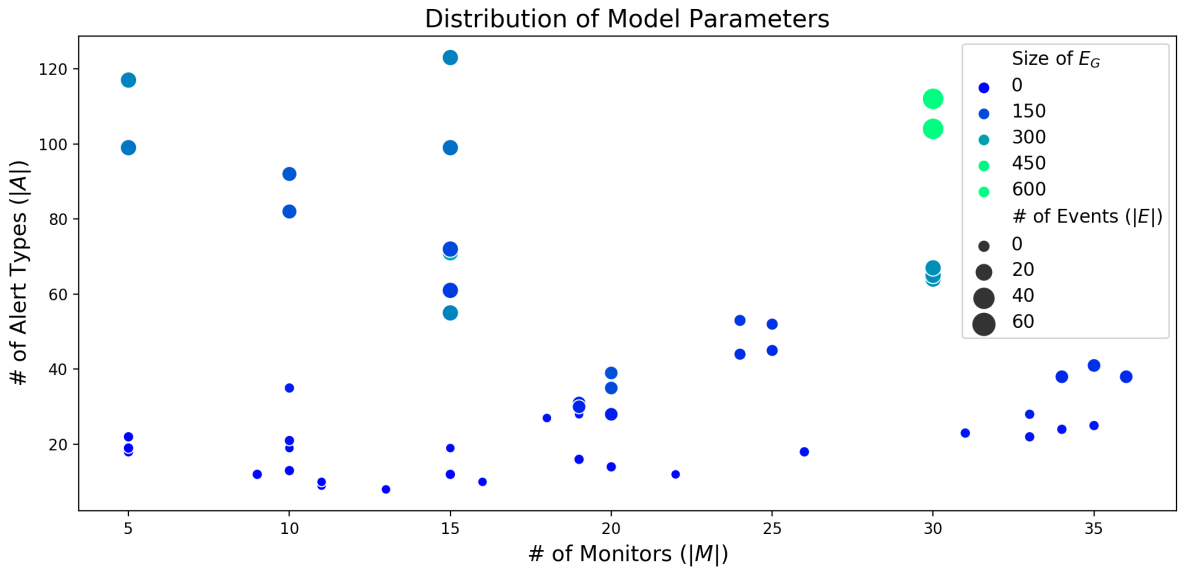


Figure 3.9: Distribution of model parameters for scalability experiment models. Each circle corresponds to a single, randomly generated model.

For a given model, the time to run the SGA search is linearly proportional to the number of function evaluations performed by the algorithm, which varies linearly with the number of monitors in the model, $|M|$. The time to run the search did not seem to follow any discernable pattern with respect to any of the other model parameters. For Model X (10 monitors), the average single-threaded time to perform 50 generations of the genetic algorithm was 1090 seconds. Note that we did not attempt to optimize the implementation of our approach; we expect such efforts could improve the runtime by an order of magnitude.

3.10 DISCUSSION

Here, we discuss the reasoning behind various assumptions and limitations of our modeling approach, and suggest avenues for future work to address those limitations.

Assumption about detection ability. In this work, we consider the system at design time, at which point the actual dynamics of monitor data generation are not necessarily known. Therefore, we assume an idealized model of detection ability, in which the incident or intrusion detection mechanisms in place will always be able to detect a given reliability incident or intrusion (that is, with no false negatives) provided that the monitors available in the system can generate some minimally-sufficient set of evidence about the event, which

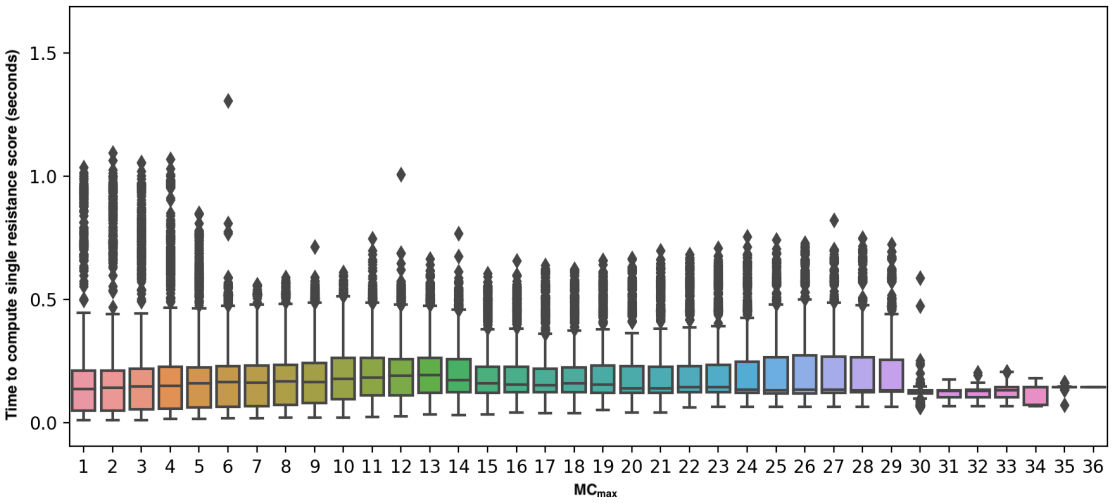


Figure 3.10: Boxplot of time to compute resistance scores (in seconds) vs. MC_{max}.

we define in Section 3.5, and that the information is not missing due to monitor failure or compromise.

Even given access to sufficient evidence, intrusion and incident detection mechanisms can fail to detect an event that is taking place (that is, produce false negatives) for many reasons, including difficulties in distinguishing between normal and anomalous or malicious traffic, misconfiguration of monitors or detection mechanisms, and poorly-chosen or intentionally subverted detection thresholds. We consider modeling inadequacies in detection approaches out of scope for this work. Instead, we argue that our assumption of idealized detection ability given sufficient evidence can be seen as a necessary condition for the detectability of events, since if an event is not detectable under our assumption, it would not be detectable under more realistic assumptions, which would be more stringent. Thus, our approach provides *lower bounds* on attacker damage potential and monitoring required for resilience to compromise, which is itself useful, as many current monitoring decisions are made without awareness of or ability to understand these.

Our proactive perspective does not preclude use of our monitors to evaluate the effectiveness of monitor placement after the system has started running, however. In fact, we intend 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.

Alternate compromise cost definitions. To facilitate scalable analysis, we consider attacker cost to compromise hosts, monitors, and alert types uniform and static, and consider

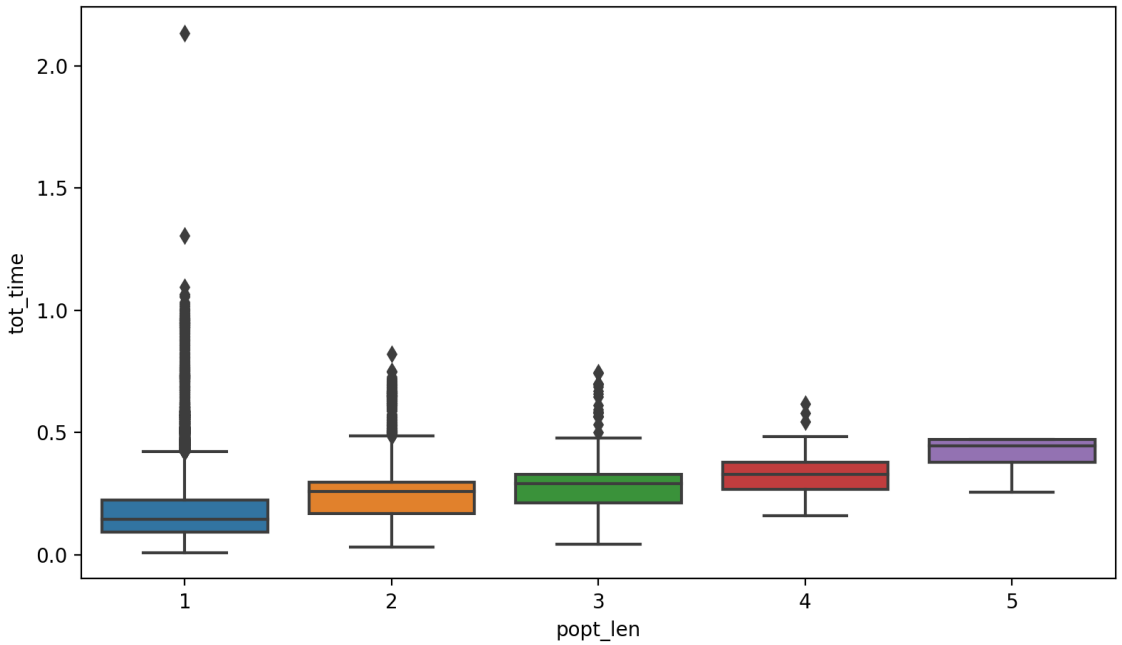


Figure 3.11: Time to compute resistance scores (in seconds) vs. the length of the Pareto boundary.

compromise of hosts to be independent. Such assumptions may be considered valid for random failures, but less so for attacks. While for this work, we have considered the incorporation of more sophisticated measures for compromise cost to be out of scope, we have considered extensions to the approach that support more sophisticated costs.

For example, we have considered the use of a dynamic reachability or vulnerability graph model, such as TVA [74] or ADVISE [75], for attacker ability to compromise monitors given a current set of capabilities. Given such a model, one could define attacker costs as a composite of the incremental effort required to compromise additional hosts, monitors, or alert types, and the increased potential of being detected upon doing so.

Significance of minimality of sufficient evidence sets. A model violating the minimality constraint on E_Δ will not ultimately affect the detectability or coverage of an event, nor will it cause inflation of the resistance of a deployment to compromise as we define in Section 3.6. However, any non-minimal edge in E_Δ is superfluous, as detectability of its sufficient evidence set necessarily implies detectability of the corresponding minimal sufficient evidence set, so it can be removed from the model as it would otherwise only decrease the efficiency of solution techniques. Indeed, the presence of such edges in a model reflect poor modeler system understanding.

3.11 CONCLUSIONS

In this chapter, we presented a methodology to examine a system and network monitoring infrastructure in the context of incident and intrusion detection to quantify susceptibility to monitor compromise and suggest changes to the monitoring that can improve the resiliency of the monitoring infrastructure. Our approach can be used by a system designer to identify monitors and evidence without which particular incidents would be undetectable, and to defensively monitor systems by determining which monitors should be deployed to maximize resilience against attackers of arbitrary capabilities subject to monitoring cost constraints. Analysis of monitor compromise using our approach, as we have demonstrated using a case study of failure localization in a datacenter network, can help system administrators make more effective monitoring decisions in line with their incident and intrusion detection objectives and priorities.

CHAPTER 4: RESILIENCE AGAINST UNCERTAINTIES IN AUDIT EVIDENCE COLLECTION

As the use of computing has expanded into sensitive domains (e.g., healthcare, finance, military, and critical infrastructure), the need for security assurance of the underlying information technology (IT) infrastructures has increased. One way enterprises and IT service providers provide such assurances is through certifications of compliance with mandatory regional or sector-specific regulations and optional global industrial and commercial standards. However, with the emergence of numerous, unique IT security standards to satisfy slightly different requirements in different domains, (e.g., HIPAA [5] for healthcare, PCI-DSS [6] for credit card handlers, and FedRAMP for U.S. Federal agencies [7]) compliance auditing has become increasingly expensive, time-consuming, and error-prone for service providers that operate in multiple domains, such as cloud service providers (CSPs). More generally, audit serves as the out-of-band mechanism to enforce reliability and security policies and SLAs in cloud systems.

From a service provider's perspective, compliance audit consists of three main phases: (i) pre-audit evidence collection, (ii) audit interview, and (iii) remediation in response to any findings of noncompliance in the second phase. In the first phase, the service provider collects evidence from its systems based on internal understanding, i.e., the understanding of its own compliance subject matter experts (SMEs) of what constitutes acceptable evidence of compliance to controls in a given regulation. Once the evidence has been collected, the service provider then submits the evidence to the auditor for review. During the audit interview phase, the auditor evaluates the evidence against the standard in question, and may request clarifications or additional evidence from the service provider. At the end of this phase, the auditors summarize their findings into an audit report, which they submit to the service provider. During the remediation phase, the service provider is given a fixed amount of time to rectify the issues laid out in the audit report and demonstrate compliance by providing additional evidence to the auditors. Finally, in the case of regulatory audits, the auditors make their recommendations to a regulator, which makes final decisions on certification.

In this chapter, we apply our monitoring lens to inefficiencies in compliance audit for infrastructure-as-a-service (IaaS) clouds. Our observation is that evidence collection for compliance audit is yet another monitoring problem with uncertainties, and if we can unambiguously model evidence collection requirements, our resilient monitoring approach from Chapter 3 can be used to determine what evidence to collect.

How is evidence collection a monitoring problem?

In order to receive audit certification for a particular standard or regulation, the cloud service provider needs to collect evidence demonstrating system compliance with each compliance control within the standard or regulation. Evidence takes the form of evidence artifacts collected from various data sources, which are akin to (in fact, inclusive of) logs and metrics, and demonstrating compliance with a compliance control is akin to detecting an incident. Cloud service providers want to collect sufficient evidence, but not spend more resources than are necessary in doing so.

In Chapter 3, we were trying to be resilient against *missing information*, but in the compliance audit context, we want to be resilient against *uncertainty in evidence requirements*. That is, whereas, for resilient monitoring, we were concerned about particular alert types not being available during online incident detection, in compliance audit, we are concerned about particular information within an evidence artifact, which is akin to an alert type, being considered uninterpretable or invalid by a subsequent auditor, and as a consequence, a particular set of evidence artifacts being deemed insufficient to prove compliance with a particular control, which is akin to data being insufficient to detecting a particular event.

The consequence of insufficiency is that it prolongs the audit process, making compliance more expensive for the cloud service provider. When an auditor deems evidence insufficient, they will generate an entry in the audit report that requires organizational review and additional evidence collection. Especially if the system is already compliant, proactive collection of additional, redundant evidence during the initial evidence collection stage can save countless hours during the audit interview and remediation stages of audit.

Determining how to prove audit compliance from logs has been tackled using model-based approaches in some limited contexts, which we describe later in this chapter. However, the requirement for what is "sufficient" evidence is poorly defined because of various uncertainties in audit process. The consequence is that in order to enable reasoning about sufficient evidence collection, we first must have an *unambiguous* model of evidence sufficiency.

We propose the following approach to learn such a model of evidence sufficiency from historical audit records. For a particular operational IaaS cloud system that is relatively unchanged across multiple audits, we model the evidence that has been collected for historical audits, and *learn evidence sufficiency for each control* (based on a standardized set of controls) as sets of evidence that have previously satisfied auditors. Historical audits provide a record of what evidence was collected and what subsets of that evidence satisfied compliance

requirements for a given auditor. We use those as "labels" for sufficiency to infer what would be sufficient in the future for the same system. Subsequent audits either verify or disprove the assumed sufficiency of a given set of evidence, triggering updates to the model. Given a sufficiently large sample size of audits for which evidence was collected for the same control, we can suggest what to collect to achieve sufficiency for the control based on a desired tolerance of uncertainty. We can then apply our approach from Chapter 3 to determine what evidence to collect to meet sufficiency requirements across *multiple controls* while minimizing evidence collection efforts.

Contributions

In summary, our main contributions in this chapter are:

1. An in-depth discussion of the challenges present in evidence collection for compliance auditing, especially in large-scale IT systems (such as public clouds). We specifically focus on the uncertainty and unpredictability introduced by humans.

2. A **taxonomic framework** for understanding the causes of and potential solutions to **uncertainty in the audit process**. We classify uncertainties by the stages of audit in which they arise, party or parties involved, target, and impact of ambiguities in language. We also explain the underlying causes of uncertainties and describe why existing model-driven approaches for audit evidence collection are ill-suited to addressing them.

3. A **model-driven method** to **learn evidence sufficiency requirements** from historical audit records. We describe how we model compliance requirements and audit evidence, and how we infer sufficiency of evidence to demonstrate compliance based on audit reports from prior audits. Our method learns an unambiguous representation of the evidence required to demonstrate compliance for a particular auditor.

4. Using the evidence sufficiency requirements learned from prior audits, we apply the cost-optimal resilient monitoring approach we presented in Chapter 3 to determine a **cost-optimal evidence collection strategy** for a cloud service provider to use during the evidence collection phase of compliance audit.

The rest of the chapter is organized as follows. First, in Section 4.1, we provide background on the role compliance audit plays in the security and reliability of enterprise cloud systems

and identifies various challenges cloud service providers face when attempting to attain compliance certification with multiple regulatory and industrial standards. In Section 4.2, we provide a primer on the compliance audit process from the perspective of a cloud service provider, detailing the stages of audit, parties involved, and the central role evidence collection plays throughout the process. Next, in Section 4.3, we present our taxonomic framework for classifying uncertainties in the audit process and explains various causes of uncertainties in compliance audit. Then, in Section 4.4, we contextualize the problem of evidence collection for compliance audit within our uncertainty framework and lay out our approach to 1) learn a model of evidence sufficiency requirements from historical audit records and 2) use the model to determine a cost-optimal evidence collection strategy. We provide the details of our experimental evaluation of our approach on compliance audit records for an IaaS cloud in Section 4.5. Finally, we discuss some limitations and possible extensions to the work in Section 4.6 and conclude in Section 4.7.

4.1 BACKGROUND

Popular CSPs such as Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure, are currently certified with tens of certifications [76, 77, 78, 79]. These certifications require periodic renewals, so cloud providers need to go through several audits every year. Failure to achieve certification for a standard may have legal and financial consequences for a CSP and its customers.

Certain regulations have a large number of controls that must be met. For example, to satisfy the requirements of the Federal Risk and Authorization Management Program (FedRAMP) [7], CSPs must implement several hundred security controls, enhancements, parameters, and requirements within the cloud computing environment as detailed in the NIST SP 800-53 Rev. 4 catalog of controls [80]. Some of these controls have strict requirements and tight timelines, e.g., the ability to provide continuous monitoring, quick application of systems updates, and generation of monthly compliance reports.

The compliance audit process remains expensive for service providers that support clients in multiple industry verticals because it involves many technical and nontechnical challenges. On the technical side, the challenges include:

- The heterogeneity of data demanded by auditors as acceptable evidence of compliance even within a given regulation, not to mention, across diverse standards,

- The scale of evidence collection and validation,
- Limited time availability for evidence collection, and
- Varying levels of automated evidence collection supported by systems from different vendors, which run the gamut from manual screenshots of command outputs to full API support for automated queries.

On the nontechnical side, the challenges arise from human factors. Uncertainties and unpredictability in the audit process stem from the high level of subjectivity involved in interpreting the requirements for a given compliance control. That may lead to an incomplete or incorrect understanding of the evidence needed and the type of validation that would be acceptable. Those factors are compounded by the reality that auditors have varying levels of skill and experience even within the same auditing organization. The sheer number of controls (even in a single standard like FedRAMP [7]) and the amount of evidence to sift through make the audit process prone to human error both on the provider side and the auditor side. Implementation and verification of controls that relate to people or processes is inherently challenging.

The compliance burden on CSPs is increasing because of the release of new regulatory standards and because of changes in the requirements of existing standards. Failure to collect sufficient evidence may result in failure to achieve certification for a standard, and that may have legal and financial consequences. Manual evidence collection and remediation are costly and error-prone. It is too expensive to collect (or even identify) all potentially useful evidence, so evidence collection must be done with sufficient care and planning to pass audit without unnecessary collection or wasted effort. Hence, automated evidence collection and validation for compliance auditing is considered a holy grail for CSPs.

Despite the challenges and inadequacies, compliance auditing and certifications serve an essential purpose: demonstrating that service providers or enterprises meet a common minimum set of security requirements as laid out in specific standards. Thus, industry and academia alike have focused on addressing challenges in compliance auditing. Modern security event and information management systems (SIEMs) include continuous scanning tools and audit support, and help in automation of compliance processes. There are active efforts to standardize controls and evidence collection requirements across different standards (e.g., the Cloud Security Alliance's Cloud Control Matrix (CCM) [81]). Those works have contributed to reduction in the cost of some pre-audit evidence collection and validation.

Existing solutions for automated audit evidence collection and validation, which are predominantly model-based, by and large cannot solve the nontechnical challenges of compliance auditing, as they are dependent on the assumptions made when constructing the models, and those assumptions may be inaccurate due to subjectivity in interpretation of compliance standards and regulations and disagreement between different parties involved in the audit process.

4.2 COMPLIANCE AUDIT PRIMER

Compliance auditing as defined by the International Organization for Standardization (ISO), is a systematic, independent, and documented process for obtaining objective evidence and evaluating the evidence objectively to determine the extent to which audit criteria are fulfilled [82]. Enterprises go through various types of auditing, such as financial, tax, environmental, and fraud investigations.

Industrialization has led to increased oversight of corporate practices and business operations, giving rise to regulatory compliance and auditing. The need for global standards to establish requirements and assess any corporation's ability to meet these requirements resulted in the formation of organizations like the ISO. In the field of information technology, the ISO 27000 series standards were developed to provide best practice recommendations on information security and evaluation of security controls.

Auditing can be conducted by an internal party (first-party audits) or an external party (second- or third-party audits) [83]. An internal corporate audit of business units within a corporation may be done to ensure that the business adheres to corporate policies, regulatory procedures, and client requirements. For instance, companies listed on the New York Stock Exchange (NYSE) [84] are required to "maintain an internal audit function to provide management and the audit committee with ongoing assessments of the company's risk management processes and system of internal controls." An external second-party audit is typically conducted by entities, such as customers, with an interest in the corporation. An external third-party audit involves examination of an organization by an independent entity to produce an unbiased report in accordance with a reporting framework, such as FedRAMP for CSPs offering services to the US government.

Our focus in this paper is on information system auditing that involves assessment of security- and reliability-related controls relevant to the IT systems of an organization. Furthermore, we consider only external third-party auditing because it is the predominant

mechanism for certification with regulatory and industrial standards, and is therefore of major importance to operators and providers of large IT infrastructures like CSPs. An external third-party audit involves examination of an organization by an independent entity to produce an unbiased report in accordance with a reporting framework, such as FedRAMP for CSPs offering services to the US government.

Different regulatory and industrial standards have been developed to address providers in different domains (e.g., HIPAA for healthcare, Sarbanes-Oxley for finance, PCI-DSS for credit card handlers, and FedRAMP for U.S. Federal agencies). CSPs that operate in multiple domains must be certified on multiple standards, and that is time-consuming and expensive. Further, there is a significant overlap among controls across different standards [81]. CSPs strongly desire to reduce the complexity and cost incurred in achieve certification with multiple standards.

Compliance enforcement mechanisms can be classified in terms of the time at which security or reliability compliance violations are caught and protective measures are taken. In their survey of the current state-of-the-art in compliance verification techniques found in literature, Majumdar et al. [85] define three classes of approaches to achieving compliance: retroactive, intercept-and-check, and proactive. Third-party audit is a form of *retroactive compliance*.

In general, CSPs and regulators alike desire to move to continuous auditing and proactive compliance mechanisms in order to reduce the time during which the system is noncompliant or vulnerable and to reduce inefficiencies in protecting the system by preventing the system from becoming noncompliant in the first place. Despite advancements in research towards that end [86, 87, 88], however, the primary mechanism for compliance in clouds is through auditing, the majority of which is performed as third-party audit. Hence, in this work, we focus on challenges that arise during the third-party compliance audit process.

4.2.1 Description of the compliance audit process

Here, we provide a primer on the compliance audit process that frames the rest of our analysis. We identify the different parties involved in audit and describe the stages of the audit process in more detail. We also illustrate some of the complexities of evidence collection and validation.

Parties involved

Audit involves the following *parties*:

- **Auditor**: The organization or individuals, internal or external, who perform the audit.
- **Compliance SME**: The team internal to the service provider (the auditee) that consists of subject matter experts (SMEs) on compliance audit and liaises with both the auditor and system owners.
- **System owner**: The team(s) internal to the service provider that operate and manage the system and have the technical expertise to collect evidence and change system configurations to remediate noncompliance.
- **Tool vendors**: Third-party vendors of tools that are designed to help system owners automate audit evidence collection and validation, such as Cloud Custodian [89] and Tenable Nessus [90].

Stages of audit

We separate the compliance audit process into three main stages: **initial evidence collection**, **audit interview**, and **reporting and remediation** of noncompliance. Figure 4.1 shows a high-level view of the audit process and the parties involved in each stage.

Initial evidence collection. In the first stage, the service provider collects evidence from its system (during system operation) based on its own compliance SMEs' understanding of what is sufficient to prove compliance. For a first-time audit, evidence must be collected for all controls. However, for subsequent recertifications, evidence to show continued compliance may be sufficient. Often, compliance SMEs will use an audit checklist provided by the auditor as a guide for what needs to be collected. The evidence is collected using a combination of manual collection by the system owners, third-party automated audit evidence collection tools, and software developed in-house to collect evidence automatically. During the evidence collection process, the compliance SMEs evaluate the evidence based on their prior experience with audits and their knowledge of the standards and the service provider's systems. Evidence collection is an iterative process. If the service provider's compliance SMEs believe the evidence to be insufficient, they request additional evidence from the system owners.

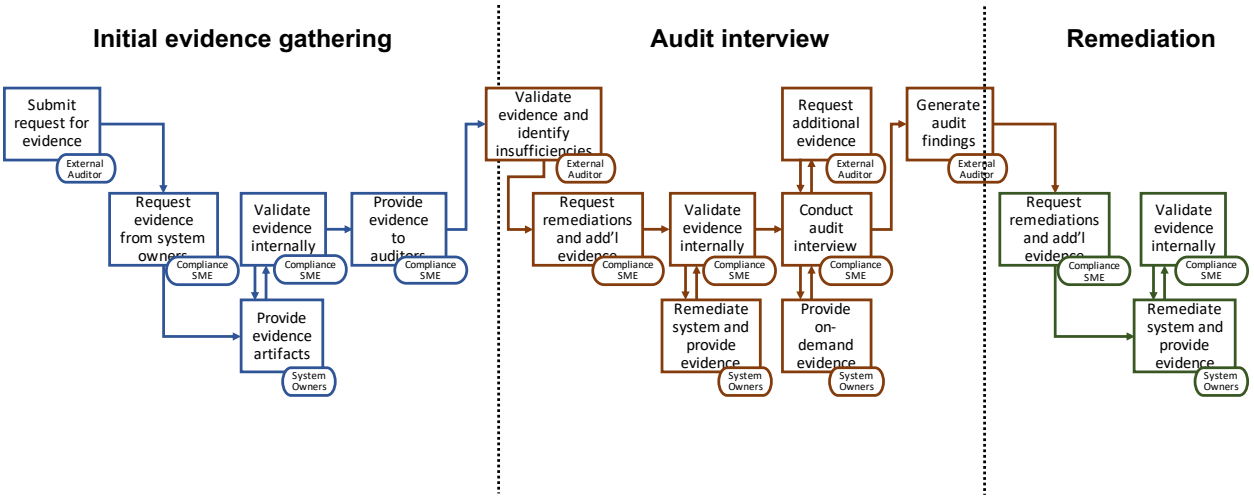


Figure 4.1: High-level view of the compliance audit process. Each box denotes an audit-related task that is completed by the party shown at the bottom right of the task. Tasks are temporally ordered from left to right, such that all tasks to the left of a given task are completed before the given task begins, and tasks that are vertically aligned occur in parallel. Tasks are further grouped into three stages of audit, separated in the figure by dotted lines. From top to bottom, tasks that appear at the same horizontal level are performed by the same party.

Audit interview. Once the evidence has been collected, the service provider, usually represented by its compliance SMEs, then submits the evidence to the auditor for review. During the audit interview phase, the auditor evaluates the evidence against the standard in question, and may request clarifications or additional evidence from the service provider. The audit interview process takes place within an allotted duration, so it is important for both the auditor and service provider that evidence is collected and presented in a timely manner. The audit interview may include an "on-site" component, in which the auditors directly interact with employees and system owners to verify process-related controls. The audit interview itself can be an iterative process in which the auditor and compliance SMEs may go back and forth until the auditor is satisfied that the service provider has shown sufficient evidence or the audit deadline has passed.

Reporting and remediation. Once the allotted audit duration has passed, the auditors compile their findings into an audit report, which they give to the service provider. The report contains a list of inadequacies and instances of noncompliance that should be remediated. During the remediation phase, the service provider is given a fixed amount of time to rectify the issues laid out in the audit report and demonstrate compliance by providing additional

evidence to the auditors. If additional evidence must be collected, it is collected in the same iterative manner (between the compliance SMEs and system owners) as in the previous phases. Finally, in the case of external audits, the auditors make their recommendations to a regulator, which makes final decisions on certification.

4.3 CLASSIFICATION OF UNCERTAINTIES IN COMPLIANCE AUDIT

In order to structure our discussion of the uncertainties that arise in compliance auditing, we answer the following questions:

- What do we mean by "uncertainty"?
- How can we classify uncertainty?
- What are the causes of uncertainty?

4.3.1 Definition of "uncertainty" in compliance audit

We define *uncertainties* in the audit process as any respects in which a party involved in the audit has an insufficient interpretation or understanding of the system, evidence, or other parties in the audit process that can lead to inefficiencies or errors.

For instance, recall that during initial evidence collection, compliance SMEs submit requests for evidence to system owners. Depending on the compliance SMEs' level of understanding of system components, for some controls, they may ask for specific pieces of information, while for others, their requests may be more open-ended, leaving the system owners to decide what relevant information to provide. If the system owners' understanding is different from that of the compliance SMEs, they may provide information that the compliance SMEs deem incorrect, which increases the overall evidence collection time and cost.

4.3.2 Taxonomy of uncertainties in compliance audit

Based on our observations of the audit process within a large cloud provider and our discussions with various security and compliance professionals, we have developed a taxonomy for examining and classifying uncertainties in the audit process. We summarize our taxonomy in Figure 4.2, and we provide descriptions of each of the components of the taxonomy below.

Parties involved. We identified the four parties involved in audit in Section 4.2.1, namely, the auditor, the compliance SME, the system owner, and the tool vendor. In our taxonomy, we use those parties to identify the source(s) of a given type of uncertainty and the stakeholder(s) who would desire to reduce it.

Interpersonal interaction. We further classify uncertainties in two ways related to human involvement, which is the source of much of the uncertainty in auditing. First, we distinguish between two kinds of *cross-party involvement*: whether the uncertainty involves interactions between different parties (i.e., **multi-party**) or is internal to one party (i.e., **single-party**). Second, we look at the role that *human interaction* plays; that is, whether the uncertainty arises because of **interpersonal** interactions (whether single-party or multi-party) or if it is due to some **inherent** ambiguity, vagueness, or challenge that is irrespective of the individuals involved.

Stage of audit. We also attribute uncertainties to the *stage* in the compliance audit process in which they arise. Recall from Section 4.2.1 that the three stages are initial evidence collection, audit interview, and reporting and remediation.

Target of uncertainty. In addition, we identify five *targets* for uncertainty that describe what, within a cloud system, an individual can be uncertain about:

1. **System compliance**: Parties may be uncertain about whether a particular control has been satisfied by the current *state* of the system, which refers to policies and processes in place, configuration of system components, data being collected, and the monitoring setup of the system, among other elements.

2. **Evidence necessity**: Parties may be uncertain about what evidence they are required to collect for compliance with a particular control.

3. **Evidence availability**: Parties may be uncertain about what evidence is available in the system that could potentially be used to prove compliance with a control.

4. **Evidence utility**: Parties may be uncertain about how to use the evidence available in the system to prove compliance with a particular control.

5. **Control interpretation**: Parties may be uncertain about whether the evidence collected will be considered sufficient by another party involved in the audit.

Each of the aforementioned targets of uncertainty affects each of the parties involved in an audit in different ways because of the parties' different responsibilities. The first, second, and third targets of uncertainty are predominantly felt by the parties internal to the service provider (namely, the compliance SMEs and system owners) because they control the operation of the system and would be directly affected if the system were to be found noncompliant or compromised and because they are solely responsible for evidence collection, whereas the rest of the above targets of uncertainty affect all parties.

The fourth target of uncertainty affects all parties in an audit fairly equally. Auditors must be able to evaluate diverse types of evidence, so they must be able to understand the meaning of the evidence provided and ask for the appropriate clarifying evidence. Compliance SMEs must be able to act as internal auditors to determine, prior to the audit interview phase, whether they need to collect more evidence. System owners must be able to understand the intent of a request from compliance SMEs in order to respond with the appropriate evidence. Third-party tool vendors must target the correct types of information to be able to convince auditors and service providers to use their tools in the audit process.

Last, the fifth target of uncertainty is again felt by all parties in an audit, but particularly by the auditors and compliance SMEs. While system owners and tool vendors must be able to understand how auditors and compliance SMEs will use the evidence they collect to prove compliance, the auditors and compliance SMEs must be directly engaged during all phases of the audit in a dialogue in which they must agree on the interpretations of the standards.

Language-derived uncertainties. Finally, for aspects of audit involving natural language, such as regulatory controls, policy documents, and communications between parties, we identify *language-derived* classes of uncertainty. Unfortunately, regulations are often intentionally written with ambiguous or vague language to allow for broad legal interpretation and prevent dependence on specific technologies or practices [91, 92]. While others have developed more granular classes for language-related uncertainties in policies and regulations [92], we boil them down to two basic classes: **ambiguity** and **vagueness**.

Ambiguity arises when a statement or concept has multiple but finitely many interpretations. More generally, ambiguity can be caused if words have multiple meanings or statements have multiple grammatically correct but semantically distinct readings [93]. For example, §170.302(f) of the HITECH Act [94] requires that electronic health record systems "*plot and display growth charts [...] for patients 2–20 years old.*" The regulation does not specify the inclusiveness of the range given, so individuals could interpret it in four distinct ways (i.e.,

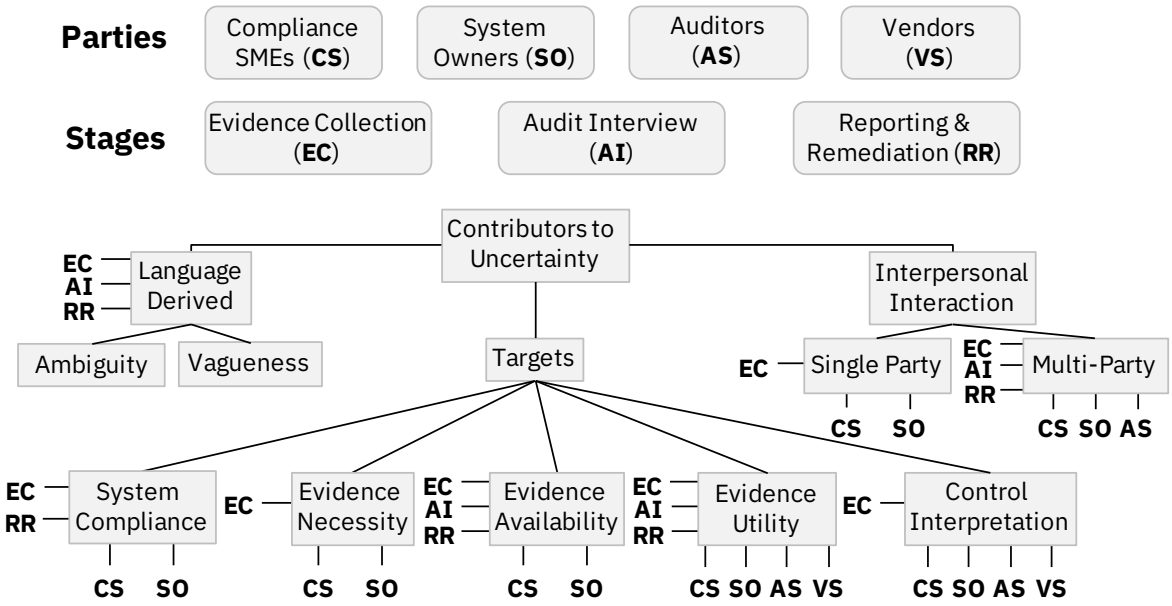


Figure 4.2: Our taxonomy of types of uncertainties that arise during compliance audit. Each subtree describes a classification of uncertainty, and leaf nodes denote classes. Parties affected are annotated underneath each class, and stages of audit affected are annotated to the left.

depending on whether the range includes 2 and/or 20). Ambiguity can be mitigated in part with additional context.

Vagueness arises when a statement provides insufficient detail to convey its meaning or can have borderline cases. For example, consider control AU-6(a) within the NIST SP 800-53 standard [80], which requires that: *"The organization: (a) Reviews and analyzes information system audit records [Assignment: organization-defined frequency] for indications of [Assignment: organization-defined inappropriate or unusual activity]."* Every individual within an organization may have his/her own definition of "inappropriate or unusual activity," with some cases being dependent on context and/or an arbitrarily-specified threshold (e.g., too many failed authentication attempts). Hence, the control's definition is vague. The majority of language-derived uncertainties in regulations are due to vagueness [92]. Vagueness can be mitigated in part by using formal specifications.

Classifying uncertainties in the way we have done within our taxonomy is important because it enables practitioners to *localize* uncertainties experienced in practice to a particular set of parties or individuals and stage of audit, and thereby guide mitigation approaches. It also enables practitioners to understand the inherent assumptions and limitations of a given mitigation approach, as we demonstrate in the next section.

4.3.3 Underlying causes of uncertainty in compliance audit

Here, we list some of the underlying causes of uncertainty in the audit process. While we do not claim that this list is complete, we believe that it covers the vast majority of causes. For each cause, we describe (with examples) the classes of uncertainty it introduces at different stages of audit based on the classification we have defined above. We then discuss solutions that are present in the literature and commercial tools, most of which are model-driven, and point out their inadequacies. Our goal here is to motivate the need for future work in this area, including our own approach to improve the efficiency of audit evidence collection that we describe throughout the rest of this chapter.

Auditor domain perspective

Third-party auditors in different domains often have different perspectives that guide their interpretations of the standards and hence the evidence they deem sufficient/insufficient. For service providers that operate across multiple layers, such as IaaS and PaaS CSPs, and in multiple domains, such as healthcare and finance, variations in auditors' interpretations of controls due to domain perspective makes it difficult to translate knowledge about evidence collection across standards, causing increased inefficiency in evidence collection. Attempts to standardize controls across standards, like the CCM [81], do not sufficiently account for auditor domain perspective.

The uncertainties caused by auditor domain perspective are multi-party, as they cause differences in control interpretation between auditors and compliance SMEs, and they arise from inherent domain-specific assumptions held by auditors about how controls should be enforced. Tool vendors are also affected, as they must account for differences in control intent if they intend to support standards in different domains.

Despite our own observations that such uncertainties caused by auditor domain perspective are a problem for cloud providers, there is limited literature specifically addressing that cause. Automated scanning and compliance testing tools like Chef InSpec [95], which specify relationships between tests and the controls they validate by using handcrafted mappings written in a domain-specific language, reduce interpretation-related uncertainties because the mappings they specify can easily be vetted by auditors and are therefore more likely to be accepted by auditors in different domains. However, there is still a need for more work in this area.

Variability of technical knowledge

Of course, individuals involved in compliance audit have varying levels of technical knowledge. That influences their abilities to identify, for example, what evidence is necessary to demonstrate compliance and whether controls can be circumvented in ways that call for additional evidence. While an individual's lack of technical knowledge can cause some single-party and inherent uncertainties, they can be mitigated through increased training or the watchful eye of more knowledgeable individuals in the same party. The more difficult-to-mitigate uncertainties caused by variability of technical knowledge are multi-party and interpersonal.

Depending on which parties are involved and whose knowledge is lacking, different issues may arise during the audit process.

A lack of knowledge on the part of the auditor may cause a system to pass audit despite noncompliance or lack of sufficient evidence. As a result, past certification of compliance may not always indicate that the evidence collection is sufficient for future recertification even for the same control, thus introducing uncertainties for the compliance SMEs in system compliance, evidence utility, and control interpretation.

A lack of knowledge on the part of the compliance SMEs could introduce uncertainties about all targets of uncertainty and may cause them to be unaware about what evidence to provide to auditors or to request from their system owners. That would increase the number of times auditors request additional evidence during the audit interview and the number of noncompliance results reported by auditors during reporting and remediation. For example, a lack of knowledge on the part of the system owners may cause them to have inherent uncertainties about evidence necessity, availability, and utility. Those uncertainties may result in increased time to collect evidence and in system owners providing insufficient or incorrect evidence to compliance SMEs, therefore causing internal validation of evidence during evidence collection to take more rounds of interaction.

Automated scanning and compliance testing tools can reduce the amount of uncertainty introduced by variability of knowledge between parties by reducing the amount of discretion involved in collecting or interpreting evidence, but their scope is limited to a subset of controls. Uncertainties caused by lack of auditor knowledge can be partially mitigated if regulators provide guidance for auditors, such as FedRAMP's Security Assessment Framework (SAF) [96]. However, variability of technical knowledge is still a problem for controls that require more interpretation, such as process-related controls, and those for which evidence from multiple sources in a system must be combined to demonstrate compliance, such as for

disaster recovery capabilities.

Diversity and scale of evidence collection

A large CSP may have thousands of system components that may be managed internally by separate teams (e.g., network, compute, and storage). As a result, during compliance audit, a CSP's compliance SMEs may need to coordinate the efforts of dozens of individuals (system owners, auditors, and other compliance SMEs) to collect evidence in a timely manner and verify that the evidence collected is correct and sufficient. That introduces interpersonal, multi-party uncertainties, as there is increased room for duplicated effort, miscommunication, and oversight that can lead to noncompliance findings.

A number of inherent uncertainties may also arise because of the diversity and scale of evidence collection for CSPs. Evidence for some controls, particularly process- and design-related controls, is not easy to identify, collect, or evaluate automatically, as it may be contained in unstructured, natural-language documents such as design and process documents, e-mails, presentation slide decks, etc. That may cause uncertainties about evidence availability and utility for compliance SMEs and system owners. Evidence for which collection *can* be automated can take many forms (e.g., logs, configuration files, and periodic scans), so compliance SMEs and system owners may face further uncertainties about evidence necessity. The same uncertainties also affect auditors and tool vendors, as auditors must know how to interpret the evidence provided to them, and tool vendors must make determinations about what evidence sources they should target and must ensure that their interpretations of evidence utility align with those of auditors.

Furthermore, monitors available for different applications that serve the same function (e.g., Postgres vs. MySQL vs. MongoDB), or for different versions of the same application, may provide semantically different information, so collecting the same evidence from different systems may require vastly different procedures.

A number of commercial tools aim to reduce the challenges posed by diversity and scale of evidence collection in audit. The automated scanning tools we have discussed reduce some of the burden in evidence discovery, collection, and mapping to controls by curating lists of known evidence sources and scans pertaining to controls. However, as their models must be actively maintained and specified by hand, their coverage of controls is limited. RSA Archer [97] addresses some of the multi-party, interpersonal uncertainties in coordinating evidence collection and identifying potential oversights by assisting with audit management.

In addition, many approaches have been proposed in the research literature for automating evidence collection and validation. Amazon has developed a formal model-checking approach called Zelkova [87] to automatically verify access control policies in many parts of AWS's infrastructure. Ullah et al. [98] claim to have built an automated tool to collect and validate evidence against standards by combining API scanning, vulnerability scanning, log analysis, and manual data entry, but they are sparse on details about how they validate the evidence. Majumdar et al. [15, 99] translate controls into a formal language representation that they map to evidence available from OpenStack; they use the mapping to automatically collect evidence and verify compliance for identity and access management controls, but their approach requires one to do manual analysis of log data in order to perform their modeling. Stephanow et al. [100] developed a formal framework to quantitatively evaluate the performance of continuous compliance testing tools, and they evaluated it for resource availability controls.

All of the abovementioned formal modeling-based approaches make strong assumptions that the formal models of controls and evidence that they manually construct are authoritative. They are therefore still susceptible to control interpretation issues. Furthermore, the test cases and case studies used by each approach are from control families that are relatively easy to model formally, so it is likely that the approaches would not generalize to the larger array of controls present in standards.

Language of standards and regulations

The problem of vagueness and ambiguity in the wording of regulatory standards has been well-studied, as we illustrated in Section 4.3.2. Most directly, vague or ambiguous wording in standards can cause inherent uncertainty in how controls should be interpreted and what mechanisms should be present to ensure system compliance, which affects all parties in an audit. In addition, vagueness in standards can cause the same auditor to identify different gaps in evidence sufficiency upon a repeat audit of the same system. In some cases, requirements may even be inconsistent across controls within a single standard, introducing further ambiguity to control interpretation. For example, some controls require a process document to be under change control, with approvals and proper distribution, while others do not.

Differences in control interpretation resulting from poor wording of standards can also cause disagreements (i.e., multi-party, interpersonal uncertainties) between auditors, compliance SMEs, and system owners during all stages of audit. Furthermore, differences among

individuals in the same role can cause single-party uncertainties that delay evidence collection and validation.

There is a body of work that attempts to explicitly identify and reduce language-related uncertainties in standards.

Massey et al. explicitly taxonomize all instances of ambiguity and vagueness in HIPAA and HITECH [101] and survey parties from different domains about what types of ambiguities they perceive in the standards' texts [92, 102]. They develop metrics to quantify ambiguities in the text of regulations, but they primarily focus on how to guide regulators to make the documents less unclear [93].

Breaux et al. attempt to remove uncertainties by formally classifying the constraints and rules present in compliance standards [91] and by manually extracting formal semantic specifications of those rules for a subset of controls within HIPAA and HITECH [103]. They then propose a method to construct finite state machines from the semantic specifications that they claim can be used as compliance monitors [104].

While Breaux et al. address language-related uncertainties more directly than other researchers have done, like the other formal language-based approaches we describe above [15, 99, 100], their approach makes strong assumptions about how controls should be interpreted. While such interpretations may be obvious or well-accepted for some families of controls (e.g., access control and availability), it is unclear how general or useful such interpretations would be for other families (e.g., configuration management and risk assessment).

Poor communication or miscommunication

Poor communication and miscommunication can affect all parties and all stages of audit with both single-party and multi-party interpersonal uncertainties. For example, poor communication or miscommunication between system owners can cause duplication of evidence collection. This is an example of single-party uncertainties caused by interpersonal interactions. Communication-related uncertainties during evidence collection can be mitigated by the use of well-defined processes for evidence collection and the use of formal specifications for evidence requirements (as opposed to natural language).

Communication-related uncertainties during evidence collection can be mitigated by the use of well-defined processes and better tools for evidence collection. The use of formal specifications for evidence requirements (as opposed to natural language) can also reduce lack of clarity during all stages of audit. Many of the approaches we describe above that are

effective against uncertainties caused by auditor domain perspective, variability of technical knowledge, and vague/ambiguous wording of standards are useful here as well.

Dependencies between cloud service levels

When audit is done for a CSP at a particular level, the CSP is responsible for evidence collection for its own components. However, a CSP may also itself utilize cloud services from providers at lower levels and depend on evidence collected by the lower-level provider for the CSP's own compliance auditing. When the dependencies span organizational boundaries (e.g, a SaaS provider using a public IaaS infrastructure), uncertainties may be introduced in terms of evidence utility towards audit and evidence availability, as the dependent provider may not have visibility into the underlying provider's evidence collection procedures and may be limited by its monitoring capabilities [105]. Even when the lower-level provider is within the same organization, however, the cloud services may be audited by different third-party auditors, so the uncertainties related to auditor control interpretation that we describe above may still apply.

The adoption of the multi-cloud paradigm introduces another avenue for uncertainty, as the different kinds of evidence provided by underlying providers must also be coordinated/consolidated during evidence collection. This is a parallel problem to that of different software versions within an infrastructure providing different types of information (discussed above), and introduces inherent uncertainties for auditors and compliance SMEs about evidence availability and utility.

Uncertainties caused by dependencies between cloud service levels can be partially mitigated by clear delineation of threats and compliance responsibilities between the different CSPs. For example, Thakore et al. [105, 106] develop a service level-aware cloud threat model and discuss techniques to monitor assets managed by lower-level CSPs. In some cases, regulatory standards themselves separate compliance responsibilities by service provider level – §3.1.3.2 of the FedRAMP Security Assessment Framework [96], for example, allows service providers to specify controls that are handled by a lower-level service provider as inherited controls; the service provider is then not required to prove compliance of the underlying infrastructure provided that the lower-level provider it itself certified to be compliant, which may still cause uncertainties for the dependent service provider.

Furthermore, some CSPs provide monitoring tools (e.g., Amazon CloudWatch [107]) that provide high visibility into the underlying infrastructure and integrate well with SIEMs, thus

reducing the effort involved in coordinating evidence. Amazon also allows cloud consumers to perform model-based queries on the access control configurations of their cloud assets [87]; such well-specified evidence can reduce uncertainties about evidence utility and promote adoption of multi-clouds.

4.4 LEARNING EVIDENCE COLLECTION REQUIREMENTS FROM HISTORICAL AUDITS

4.4.1 Problem description

Recall from the introduction of this chapter that we are interested in improving the efficiency of evidence collection for compliance audit by first learning an unambiguous model of what audit evidence is sufficient to demonstrate compliance based on historical audit records, then choosing a cost-optimal sufficient subset of evidence to collect based on the model.

Using the terminology of our taxonomy, we can contextualize where this problem falls among the larger set of uncertainties that arise in the compliance audit process. In our work, we take the perspective of the compliance SMEs (**CS**) during the initial evidence collection (**EC**) and audit interview (**AI**) stages of audit. We aim to reduce inter-party uncertainties between the compliance SMEs and auditors (**AS**) regarding evidence necessity, evidence utility, and control interpretation. Of the underlying causes of uncertainty that we enumerate in Section 4.3.3, we specifically aim to address issues caused by auditor domain perspectives, the language of standards and regulations, and poor communication or miscommunication.

To simplify the problem, in this work, we do not take into account the system owners, nor any uncertainties introduced by interactions between compliance SMEs and system owners, when determining the evidence that must be collected. Instead, we focus on the boundary between the service provider (inclusive of both the compliance SME and system owners) and the auditors. Since we aim through our model to make explicit the actual pieces of evidence that must be collected, we expect that compliance SMEs could present the model to system owners as the explicit specification of requirements for evidence collection and thereby reduce intra-service provider uncertainties as well.

4.4.2 Evidence collection terminology

Here, we define the terms we use throughout the rest of this chapter to refer to specific elements of the compliance audit evidence collection process.

Evidence collection requirements are driven by compliance controls. A *compliance control* specifies policies or guidelines that a system being audited must implement in order to be considered compliant with the containing standard or regulation. For example, the control AC-7 within the NIST SP 800-53 standard [80], which regulates unsuccessful logon attempts, requires that an organization "*a. Enforces a limit of [Assignment: organization-defined number] consecutive invalid logon attempts by a user during a [Assignment: organization-defined time period]; and b. Automatically [Selection: locks the account/node for an [Assignment: organization-defined time period]; locks the account/node until released by an administrator; delays next logon prompt according to [Assignment: organization-defined delay algorithm]] when the maximum number of unsuccessful attempts is exceeded.*" We have provided other examples of compliance controls earlier in this chapter in Section 4.3. Compliance controls may be grouped together within standards and regulations into families of similar controls. For example, the NIST SP 800-53 control AC-7 is contained within the Access Control control family.

As shown in the example above, individual compliance controls may specify multiple distinct policies. Thus, in our analysis, we use the term *compliance requirement* to refer to an individual policy or guideline specified by a compliance control. The NIST SP 800-53 control AC-7, for example, specifies two compliance requirements.

In theory, a CSP's system should be *compliant* with a particular regulation or standard if the CSP correctly implements measures to satisfy all compliance requirements contained within the regulation or standard. However, as we explained in Sections 4.2 and 4.3, for myriad reasons, "being compliant" is not an objective property of a system — two individuals may come to different conclusions about what a compliance requirement means or whether a system satisfies a given compliance requirement.

In the context of compliance audit, what matters to a CSP is whether the *auditor* deems the system to be compliant. To demonstrate compliance to an auditor, once the CSP has implemented mechanisms to enforce compliance requirements within their system, they must collect *audit evidence*. Section 3.13 of the ISO 9000 [82] defines audit evidence as "records, statements of fact, or other information, which are relevant to the audit criteria and verifiable."

Audit evidence takes many forms, depending on the compliance requirement and system characteristics. For example, for compliance requirements specifying policies around communication between employees, audit evidence may include records of communication, including e-mails, memoranda, and presentations given. For requirements specifying policies around system monitoring, evidence may include records of system health or vulnerability scans conducted, system configuration files for log centralization or intrusion detection tools, or query records demonstrating that a particular type of analysis was conducted in the past.

More specifically, audit evidence can be separated into discrete *evidence artifacts*, which we define as individual files, documents, screenshots, or other pieces of information that are provided to an auditor for evaluation. Evidence artifacts are themselves pulled from *data sources*, which we define as a service, interface, resource, or repository that is already present in a system and from which a system owner may retrieve an evidence artifact using a set of queries or other subselection methods, such as taking a screenshot. For example, for NIST SP 800-53 control AC-7, we have seen each of the following be provided as an evidence artifact: the "Account Lockout Policy" section of the Windows Active Directory configuration for the domain controller for the system (for which the data source would be the entire Active Directory Default Domain Policy); the "aaa" section of the RADIUS network access server configuration file (for which the data source would be the entire RADIUS configuration file); and the list of timestamped SIEM tool log entries from the previous year for which the log message contained "invalid logon attempt" (for which the data source would be the complete list of SIEM tool log entries).

The CSP collects each evidence artifact for the purpose of demonstrating compliance with a particular compliance requirement, and the artifacts are generally labeled as such when provided to the auditor. In some cases, the same artifact can be used to demonstrate compliance with multiple requirements. Thus, we assume that for each historical audit, the audit records contain a set of labels for each evidence artifact corresponding to the compliance requirement(s) for which the artifact was collected.

Once provided the evidence artifacts, the auditor assesses the artifacts for each compliance requirement to determine if the system is indeed in compliance, based on the auditor's understanding of the compliance requirements. At the end of their assessment, the auditor may make one of four determinations for each compliance requirement. Either 1) the system is deemed compliant; 2) the system is deemed non-compliant with the requirement, in which case the CSP must modify the system to make it compliant; 3) the auditor is unable to make a

determination because of issues with the evidence provided, in which case additional evidence must be provided to demonstrate compliance, or 4) the auditor makes a partial determination about the compliance of the system, but still requires additional evidence to make a complete determination (i.e., a mix of the first 3 outcomes). If the latter two outcomes occur after the last stage of evidence collection and/or remediation, it is functionally equivalent to the system being deemed non-compliant. The results of the auditor's assessment, which include the auditor's recommendations to remediate the system or requests for clarifying evidence, are provided in the *audit report*.

As we described in Section 4.2.1, a single audit consists of multiple rounds of evidence collection and auditor assessment, at the end of each of which an audit report is generated. Collectively, the evidence artifacts collected for each round, their compliance requirement labels, and the auditor's reports from each round consistute the audit records for a single audit.

We define the *evidence requirements* for a given compliance requirement as the set of evidence artifact types that must be collected for an auditor to deem the system compliant with the compliance requirement within their audit report. In our work, we focus on learning and modeling solely the evidence requirements, rather than the compliance requirements. This is because the evidence requirements encapsulate the auditor's understanding of evidence necessity, evidence utility, and control interpretation, thus circumventing the inter-party uncertainties between the compliance SMEs and auditors that we aim to address. Furthermore, the evidence requirements provide an unambiguous indication of the artifacts that must be collected in order to pass audit.

4.4.3 Evidence sufficiency vs. validity

It is important to note that what we care to understand is what artifact types are *sufficient* to demonstrate compliance, rather than whether they are *valid*. Here, we must call out the distinction between the two. A party in audit would consider an evidence artifact to be *valid* if it demonstrates compliance with a compliance requirement for a particular audit. Validity applies to a specific artifact collected for an individual audit, and requires that the party believe that the system is compliant, which as we explained earlier can be subjective. In contrast, an evidence artifact or set of artifacts is *sufficient* to demonstrate compliance if it can be used, without any additional evidence, to make a determination about whether the system is compliant. Sufficiency is a property of sets of artifact types, and for the

same auditor and same system, can span multiple audits. Using the terminology from our taxonomy of uncertainties, sufficiency encompasses evidence necessity, evidence utility, and control interpretation, whereas validity encompasses system compliance, evidence utility, and control interpretation.

The CSP strives to collect both valid and sufficient evidence. A valid evidence artifact may, on its own, be insufficient, but a set of sufficient evidence artifacts will always be valid if the system is compliant with the standard and may be invalid if it is not. There may be instances where evidence artifacts that are sufficient but assumed by the CSP to be invalid pass audit because of one of the many uncertainties described in Section 4.3.2 (e.g., auditor oversight), but as the number of historical audits increases, either 1) the same invalid evidence will pass repeatedly, signaling that there is likely a difference in control interpretation between compliance SME and auditor, or 2) the evidence will be found to be insufficient in a later audit.

We assume in our analysis that the CSP's system is, to the best knowledge of the compliance SMEs, compliant, except as explicitly called out by auditors in the audit reports. We do not aim to learn how a non-compliant system can skate by an audit undetected, though that can and likely does happen in practice for the many reasons described in Section 4.3.2.

4.4.4 How evidence sufficiency manifests in audit reports

Recall that our objective is to develop a method to reliably learn an unambiguous model for evidence sufficiency requirements from historical audit records. Given the audit records, in order to accomplish that objective, we must be able to identify when prior models of evidence requirements and compliance requirements are incorrect and require updating. We now describe the logic by which we interpret the findings in audit reports to infer evidence sufficiency and validity.

To summarize from earlier, for each compliance requirement in the audit report, the following properties may either be true or false about the system, the CSP's compliance SMEs, and the auditor:

- **Whether the system is compliant with the auditor's interpretations of the compliance requirements.** This is unknowable by the CSP's compliance SMEs and must be inferred from the audit report.
- **Whether the compliance SMEs believe the evidence artifacts are valid.** This

property depends on the compliance SMEs' interpretations of the compliance requirements. As it would not be rational for compliance SMEs to knowingly provide invalid evidence to auditors when valid evidence exists, we assume that if the SMEs believe the system to be compliant, they will always believe the artifacts they provide to be valid.

- **Whether the compliance SMEs believe the evidence artifacts are sufficient.** While it would be counterproductive for the compliance SMEs to knowingly provide insufficient evidence to auditors, there are legitimate reasons why this occurs in practice (e.g., the compliance SMEs run out of time to collect evidence).

- **Whether the auditor believes the evidence is sufficient.** As described above, if this property holds true, the auditor will be able to make a determination about compliance given the evidence provided, and no additional evidence will be requested in the audit report to evaluate compliance.

- **Whether the auditor believes the evidence demonstrates compliance.** If this property holds true, the auditor will deem the system to be compliant in the audit report.

For the auditor to believe that a set of evidence demonstrates compliance, they must also consider it sufficient. So, we infer that if, according to the audit report, evidence demonstrated compliance, it was also sufficient. If the auditor believes that the evidence does not demonstrate compliance, it can be because the artifacts were collectively insufficient, because all or a subset of artifacts were invalid, or both. The specific details are provided in the auditor's assessment of outstanding risks in the audit report, which must be interpreted by the compliance SMEs.

The first three properties are within the control of the CSP, whereas the last two are not and must be learned from audit reports. The CSP's compliance SMEs will have expectations for how the audit report will look (i.e., the values of the last two properties) based on their understanding of the first three properties; where the audit reports deviate from expectation, we can refine our understanding of the auditor's interpretation of the compliance requirements and evidence sufficiency.

Each form of deviation has a different meaning. Table 4.1 lists the possible scenarios for how an audit may play out for a particular compliance requirement. Each row in the table corresponds to a different set of possible compliance SME expectations and audit outcomes observed in the audit report. Though there will be a distinct set of expectations and

outcomes for each compliance requirement, the explicit evidence artifacts used to demonstrate compliance may be shared across some requirements.

Inferring evidence sufficiency from audit reports

Compliance SMEs can use Table 4.1 to reason about the results of an individual stage of audit as follows, performing the steps below for each compliance requirement. First, the compliance SMEs can identify which of the four types of audit outcomes is present in the audit report (i.e., the last column in Table 4.1). The compliance SMEs can then locate the row in the table that corresponds to both the observed audit outcomes and their prior beliefs about the evidence sufficiency and evidence validity. The actions the compliance SMEs should take will depend on the explanation provided in the second-to-last column of that row:

- If **A**, then according to the auditor's interpretations of the compliance requirements, the system is non-compliant. As the compliance SMEs had believed the system to be compliant, they should update their understanding of the compliance requirements, remediate the system to make it compliant, and re-collect the evidence that had previously been collected.

- If **B**, the compliance SMEs should update their understanding of the evidence sufficiency requirements to reflect those of the auditor. If the auditor deemed the evidence sufficient, the compliance SMEs can infer that the set of evidence artifacts they had cumulatively provided through the current stage of audit are sufficient. If not, then the system may actually be compliant, but the auditor did not receive enough evidence to make a determination, so the compliance SMEs can add the evidence artifacts that were additionally requested by the auditor to their model of required evidence, but refinements to the compliance requirements should only be made after the evidence is re-evaluated by the auditor in subsequent rounds of audit.

- If **C**, then the compliance SMEs should take the same actions as for both **A** and **B**.

- If **D**, then the compliance SMEs should take the same actions as for **A**, but they should be aware that if the current audit outcome was due to auditor oversight, in future audits (or even future rounds of the same audit), the audit outcomes may not align with expectations.

Table 4.1: Interpretation of evidence sufficiency for each possible audit outcome and set of audit properties.

System compliant	**Compliance SME**		**Auditor**		**Compliance SME expectation for audit report**	**Possible reasons for audit properties**	**How audit properties manifest in the audit report**
	Evidence sufficient	**Evidence valid**	**Evidence sufficient**	**Evidence shows compliance**			
✶	✶	✶	✗	✓	Not possible: by definition, insufficient evidence cannot demonstrate compliance	Not possible	Not possible
✓	✶	✗	✶	✶	Shouldn't happen: we assume that compliance SMEs will not provide evidence they do not believe to be valid if valid evidence theoretically exists, as doing so would just result in wasted time.	Shouldn't happen	Shouldn't happen
✓	✓	✓	✓	✓	Auditors will find the evidence sufficient and the system compliant.	Audit report looks as expected.	①: The audit report shows that the system was deemed compliant. There is no remediation request recorded at the start of the subsequent stage of audit, if applicable.
✓	✓	✓	✓	✗	"	Ⓐ : Compliance SMEs' and auditor's interpretations of the compliance requirements may differ.	②: The audit report shows that the request for evidence was satisfied, but the system was deemed noncompliant. There may be a subsequent remediation request for the same compliance requirement within the same stage of audit, and/or a remediation request recorded at the start of the subsequent stage of audit, if applicable, stating that the system was deemed noncompliant, *but not that more evidence is required to demonstrate compliance.*
✓	✓	✓	✗	✗	"	Ⓑ : Compliance SMEs' and auditor's interpretations of evidence sufficiency requirements may differ.	③: The audit report shows that the request for evidence was not satisfied and that the system was (consequently) deemed noncompliant. There may be a subsequent request for additional evidence for the same control within the same stage of audit. There may also be a remediation request recorded at the start of the subsequent stage of audit, if applicable, stating that more evidence is required to demonstrate compliance, *but not that the system was deemed noncompliant.*

Table 4.1: (cont.)

System compliant	Compliance SME		Auditor		Compliance SME expectation for audit report	Possible reasons for audit properties	How audit properties manifest in the audit report
	Evidence sufficient	Evidence valid	Evidence sufficient	Evidence shows compliance			
✓	✗	✓	✓	✓	Auditors will find the evidence insufficient to demonstrate compliance.	See B.	See ①.
✓	✗	✓	✓	✗	”	C : Compliance SMEs' and auditor's interpretations of both evidence sufficiency requirements and compliance requirements may differ.	See ②.
✓	✗	✓	✗	✗	”	See B	See ③.
✗	✓	✓	✓	✓	Compliance SMEs mistakenly expect that auditors will find the evidence sufficient and the system compliant, though in reality, the system is noncompliant.	D: The auditor's interpretation of the compliance requirements may be different than that of the compliance SMEs, but the auditor may also have made a mistake or had an oversight.	See ①.
✗	✓	✓	✓	✗	”	See A.	See ②.
✗	✓	✓	✗	✗	”	See C.	④: The audit report shows that the request for evidence was not satisfied and that the system was deemed noncompliant. There may be a subsequent request for additional evidence and/or remediation request for the same control within the same stage of audit. There may also be a remediation request recorded at the start of the subsequent stage of audit, if applicable, stating that more evidence is required to demonstrate compliance and possibly also that the system was deemed noncompliant.
✗	✓	✗	✓	✓	Auditors will find the evidence sufficient, but the system noncompliant.	See D.	See ①.
✗	✓	✗	✓	✗	”	Audit report looks as expected.	See ②.
✗	✓	✗	✗	✗	”	See B.	See ④.

Table 4.1: (cont.)

System compliant	Compliance SME		Auditor		Compliance SME expectation for audit report	Possible reasons for audit properties	How audit properties manifest in the audit report
	Evidence sufficient	Evidence valid	Evidence sufficient	Evidence shows compliance			
✗	✗	✓	✓	✓	Compliance SMEs mistakenly expect that auditors will find the system compliant but may request additional evidence to do so, though in reality, the system is noncompliant.	See B.	See ①.
✗	✗	✓	✓	✗	”	See C.	See ②.
✗	✗	✓	✗	✗	”	See A.	See ④.
✗	✗	✗	✓	✓	Auditors will find the evidence insufficient and the system noncompliant.	See D.	See ①.
✗	✗	✗	✓	✗	”	See B.	See ②.
✗	✗	✗	✗	✗	”	Audit report looks as expected.	See ④.

- Otherwise, the auditor's determinations are in line with those of the compliance SMEs.

For the purposes of learning about evidence sufficiency, we are most interested in the scenarios in which the evidence sufficiency requirements must be updated — specifically, those with explanations **B** and **C**.

4.4.5 Constructing the evidence sufficiency model from historical audit records

The above describes the process for an ongoing audit, when compliance SMEs have knowledge of their assumptions pre- auditor assessment. In practice, most of those assumptions are not well-documented, so for historical audits, we can only assume that the evidence artifacts and audit outcomes for each stage are available.

Thus, we must make the following simplifying assumptions about past evidence. First, we assume that all requests for evidence artifacts by the auditors (during all stages of the audit process) and all evidence artifacts that were provided to and evaluated by the auditors to come to their final decision about evidence sufficiency and compliance *are present in the audit records*. Second, we assume that there are no errors in the labels within the audit records of whether the evidence provided for a request for artifacts was considered sufficient by the auditors to evaluate compliance for the associated control.

That is, we interpret all audit reports as rows in the table corresponding to the compliance SMEs believing that evidence is both sufficient and valid. That simplifies the analysis of audit outcomes using Table 4.1. If the audit outcomes correspond to ①, then we must assume the system is compliant. If they correspond to ②, then evidence artifacts were considered sufficient but invalid. The exact same artifact types should be provided again in the subsequent round of audit. If they correspond to ③, then evidence artifacts were considered insufficient. In the subsequent round of audit, the auditor will examine the union of the evidence artifacts from this round and the next round. If they correspond to ④, then evidence artifacts were considered insufficient but partially invalid. Some of the same artifact types should be provided again in the subsequent round of audit, and the auditor will evaluate the most recent of those plus all other artifacts provided this round and the next round.

We use only the empirical labels from the audit reports as ground truth regarding whether a set of artifact types is sufficient. Until the auditor has deemed the system compliant, we do not consider any subset of evidence collected prior to be sufficient.

Model components. We now describe how we represent the compliance audit evidence collection concepts we have just defined using the components of the resilient monitoring methodology modeling formalism we defined in Chapter 3. We denote evidence collection terms using sans serif, and model components from Chapter 3's formalism using *italics*.

In our evidence sufficiency model, we represent each compliance requirement as an *event*. We represent evidence artifact types as *alert types* and the data sources from which artifact types are pulled as *monitors*. Each unique set of evidence artifact types that is found to be sufficient to demonstrate compliance is represented as a *sufficient evidence set*. Thus, satisfying evidence sufficiency requirements is equivalent to collecting enough artifacts to make the compliance requirement *detectable*. For the purposes of modeling evidence collection, we elide the concept of *hosts* by assigning each *monitor* to its own *host*.

Algorithm. There are two aspects to constructing the evidence sufficiency model from historical audit records: constructing a model from a single audit, and refining the model by merging models from multiple audits. From the records for a single audit, for each compliance requirement, it is possible to identify at most a single set of sufficient evidence artifact types. Across multiple audits, it is possible to combine the single audit models to identify multiple, potentially overlapping sets of sufficient evidence artifact types for each compliance requirement.

First, we establish the baseline procedure for determining what evidence artifact types were considered sufficient within a single audit, but across multiple rounds of evidence collection conducted during successive stages of the audit. The procedure is given by Algorithm 4.1. During model construction, in addition to the sufficient evidence set, denoted as σ, we also keep track of the largest set of evidence artifact types that was explicitly deemed to be *insufficient*, which we denote as $\bar{\sigma}$. Figure 4.3 illustrates the outcome of running the procedure for a small example.

The preconditions to running Algorithm 4.1 are as follows. First, we expect that all stages of audit have completed and that the final compliance/noncompliance results have been provided for each compliance requirement by the auditor. Second, we expect that the compliance SMEs have interpreted all of the audit report outcomes into one of the four categories we define above – namely, COMPLIANT, INVALID $\wedge$ SUFFICIENT, INSUFFICIENT, or INVALID $\wedge$ INSUFFICIENT.

Next, we describe the procedure by which we merge and refine the evidence sufficiency model across multiple, distinct audits. As a precondition, we assume that for each audit,

Algorithm 4.1 Procedure to learn an evidence sufficiency model across multiple stages of a single audit

Input: all sets of evidence artifacts $\{a_0 \dots a_N\}$ collected and their associated audit outcome labels $\{l_0 \dots l_N\}$ for compliance requirement e for audit stages $0 \dots N$
Output: $(\sigma, \bar{\sigma})$ for e

```
1: function ConstructAuditModel({a_0 ... a_N}, {l_0 ... l_N})
2:     σ ← ∅, σ̄ ← ∅
3:     for i ← 0 to N do
4:         if COMPLIANT in l_i then
5:             σ ← σ ∪ a_i
6:             return (σ, σ̄)
7:         else if INVALID in l_i then
8:             if SUFFICIENT in l_i then
9:                 σ ← σ ∪ a_i
10:            end if
11:        else                                   ▷ (i.e., INSUFFICIENT in l_i)
12:            σ̄ ← σ ∪ a_i
13:            σ ← a_i
14:        end if
15:    end for
16:    return (∅, σ̄)
17: end function
```

the compliance SMEs have already obtained independent evidence sufficiency models per compliance requirement per audit using the procedure given by Algorithm 4.1. The procedure to merge evidence sufficiency models across audits is given by Algorithm 4.2. Figure 4.4 illustrates the outcome of running the procedure for a small example. Specifically, the algorithm implements the following rules:

- If a set of evidence artifacts is sufficient for a compliance requirement in one audit but explicitly labeled as insufficient in another, it is removed from the combined model.
- If the sufficient evidence set for a compliance requirement obtained from one audit is a superset of the set obtained for the same compliance requirement from another audit, the superset is added to the combined model.
- If sufficient evidence sets obtained from any two audits are not strict subsets of each other, then both are added to the combined model.

Our approach makes explicit when evidence is deemed insufficient and what evidence was ultimately collected across all stages of audit to demonstrate compliance. As a result,

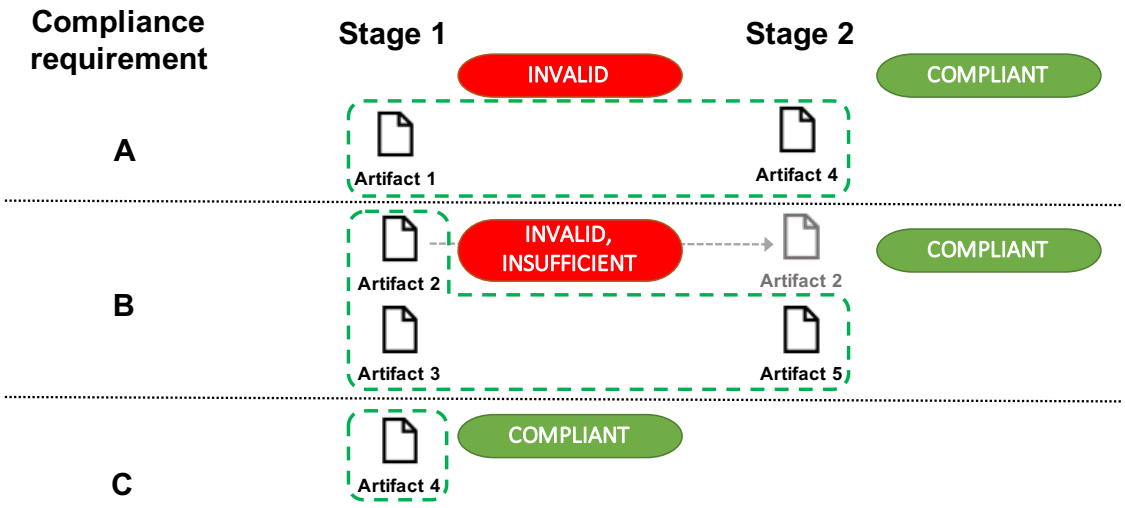


Figure 4.3: Diagram illustrating the procedure by which we determine what evidence is considered sufficient across multiple rounds of evidence collection for a single audit. The colored ovals denote auditor labels for the evidence provided at each stage of audit, and the dashed green lines outline the sets of evidence artifacts our algorithm infers to be sufficient for each compliance requirement.

it prevents compliance SMEs from making an error we commonly observed in practice, in which compliance SMEs tended to look to prior audit records to guide evidence collection for subsequent audits, but would replicate the evidence collection from the initial stages of audit without realizing when that evidence was insufficient. The other benefit of our approach is that by centralizing the model of what evidence was considered sufficient for a given system, it enables compliance SMEs to transfer that knowledge across audits, auditors, and standards, and to update it as the system or auditors change. If the model is deemed incorrect upon subsequent audits, use of our approach automatically refines it.

Cost-optimal audit evidence collection. To determine what evidence to collect, we apply the resilient monitoring approach we define in Chapter 3 to find the most redundant evidence collection strategy for a given budget for evidence collection costs. By examining at the different results at different levels of monitoring cost, compliance SMEs can see which evidence artifacts would successively provide the most marginal utility for a particular set of compliance requirements, and use the resulting sequence of artifacts to drive their evidence collection strategy. The same analysis can be performed on a requirement-by-requirement basis to determine what to collect, but since evidence tends to be shared across multiple compliance requirements, it is more efficient to obtain optimal evidence collection strategies

Algorithm 4.2 Procedure to merge evidence sufficiency models across multiple audits

Input: sets of sufficient evidence artifacts $\{\sigma_0 \dots \sigma_N\}$ and explicitly insufficient sets $\{\bar{\sigma_0} \dots \bar{\sigma_N}\}$ for compliance requirement e for audits $0 \dots N$
Output: combined Σ and $\bar{\Sigma}$ for e

function MERGEAUDITMODELS($\{\sigma_0 \dots \sigma_N\}$, $\{\bar{\sigma_0} \dots \bar{\sigma_N}\}$)
  $\Sigma \leftarrow \emptyset$, $\bar{\Sigma} \leftarrow \emptyset$
  for $j \leftarrow 0$ to N **do**
    if $\sigma_j \neq \emptyset$ **then**
      $\Sigma \leftarrow \Sigma \cup \{\sigma_j\}$
      $\bar{\Sigma} \leftarrow \bar{\Sigma} \cup \{\bar{\sigma_j}\}$
    end if
    for σ_k in Σ **do**
      if $\exists \bar{\sigma} \in \bar{\Sigma} \mid \sigma_k \subseteq \bar{\sigma}$ **then**
        $\Sigma \leftarrow \Sigma - \{\sigma_k\}$
      end if
    end for
  end for
  return Σ, $\bar{\Sigma}$
end function

for entire families of related requirements or entire audits at once.

Evidence collection costs. When a particular data source is available and its data is collected, we assume that all artifacts that can be pulled from it are available for use during compliance audit. Thus, evidence collection costs are specific to data sources rather than individual evidence artifact types. We allow the compliance SMEs to assign arbitrary evidence collection costs to each data source prior to determining the cost-optimal evidence collection strategy. The compliance SMEs and/or system owners can use their discretion and understanding about difficulty of collecting evidence from each source to assign said costs, but we do not provide any guidance on how to do that; we consider it to be out of scope for this work.

There are many reasons why evidence collection costs can vary across different data sources. First, collecting evidence of different types can require different amounts of human effort. For example, in some enterprise and cloud systems, logs are automatically centralized within an SIEM or log ingestion tool (e.g., Splunk, GrayLog). In systems with log centralization, evidence artifacts that consist of subsets of log data would be relatively easy to collect, whereas in systems without it, log-related evidence collection may require an extensive manual process. There are also many automated tools (e.g., Nessus Tenable, Ansible) that

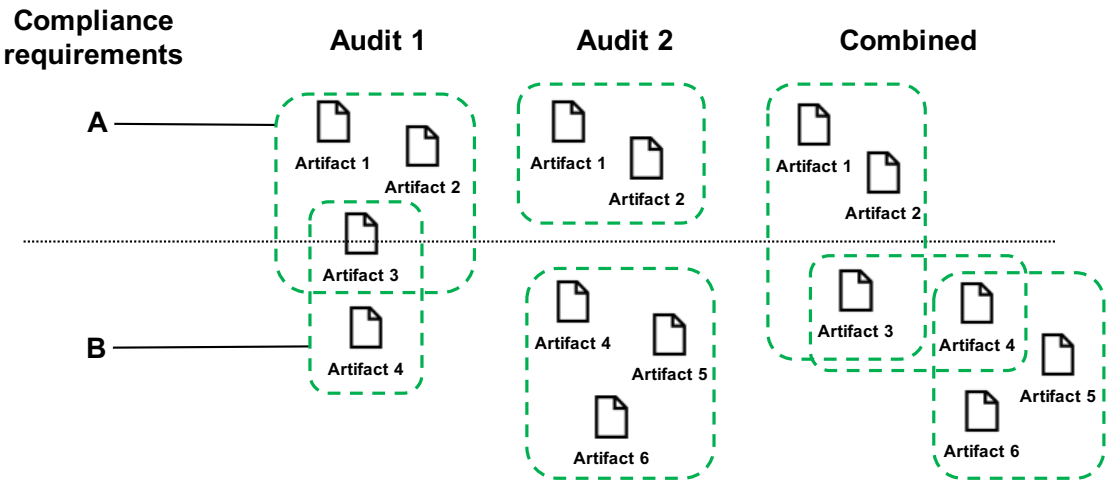


Figure 4.4: Diagram illustrating the procedure by which we merge evidence sufficiency models across multiple audits for the same system and same set of compliance requirements. The dashed green lines outline the sets of evidence artifacts included in the evidence sufficiency model for each individual audit and those that end up in the merged evidence sufficiency model.

enable collection of some types of evidence artifact using scripts. Evidence artifacts that can be collected through such tools would be cheaper to collect than those that must be collected manually. For manual evidence collection processes, the cost of collection can depend on various factors, such as the ease of use of the interface to the data source.

Furthermore, costs may change from audit to audit. The engineering effort or effort to establish a manual collection process may be expensive to perform the first time, but once it is established, collecting the same evidence for subsequent audits would be much cheaper. Thus, as an organization continues to build out its evidence collection infrastructure, cost-optimal evidence may change for the same system from audit to audit.

4.5 EVALUATION: IAAS CLOUD SECURITY AUDIT

4.5.1 Case Study Dataset

To demonstrate the efficacy of our approach, we apply it to a case study dataset consisting of audit records from recent security audits of an enterprise infrastructure-as-a-service (IaaS) cloud (EIC) service offered by a large computing organization, which we hereafter refer to as the EIC provider.

The EIC provider offers its customers the ability to requisition compute and storage capacity within the EIC provider's datacenters with the guarantee of physical isolation of compute hardware and virtual isolation of network resources from other customers (i.e., a private cloud service model[1]). Customers are given the ability to provision bare metal machines using a platform powered by the Xen hypervisor [108] or virtual machines (VMs) and containers using the RedHat OpenShift platform [109]. They also have access to a managed object storage service. All customer access to their private cloud resources takes place through a dedicated secured virtual network assigned to the customer. The EIC provider manages the datacenters, compute hardware, network infrastructure, object storage service, and the virtualization platforms, and is responsible for network monitoring, network vulnerability scanning, bare machine/VM/container health monitoring, virtualization service monitoring, object storage service monitoring, and access control to the EIC system. The EIC provider does *not* have access to the customer machines, VMs, or containers, or any of the workloads that the customers run on top of the EIC service. Therefore, in the context of compliance audit of the EIC system, the EIC provider's access control, monitoring, and identity management responsibilities extend only to the activities of their own employees who manage the EIC service.

The EIC provider restricts access to the EIC system to a limited set of privileged accounts, which are managed using a combination of a small number of different identity and access management protocols, including Windows Active Directory. All privileged account access is secured using multi-factor authentication, and all remote access is additionally secured using VPN. The EIC provider uses automated provisioning, configuration management, and application deployment tools to manage its infrastructure services. All management operations take place over a dedicated management virtual network. The EIC provider centrally logs all network connections to, from, and within the EIC system on all physical and virtual networks; privileged account access to and system logs from the hosts running the management tools, virtualization platforms, and object storage service; and health metrics for the services it manages. It also maintains a central repository of configuration files used to provision the EIC system. At the time of our analysis, the EIC system consisted of many hundreds of physical servers.

As is common for many CSPs, compliance SMEs for the EIC system undergo compliance audits for numerous cloud and domain-specific regulations and standards. In our case study

[1]Customers can also choose to use the EIC service as a hybrid cloud.

evaluation, we limit our focus on a pair of successive audits that were performed on the system in 2018 and 2019 to achieve FedRAMP authorization[2] [7].

FedRAMP is a program within the U.S. federal government that provides a standard for security assessment that agencies throughout the federal government can use to set minimum requirements for governmental use of cloud services. Compliance requirements for FedRAMP are pulled from the controls defined in the NIST SP 800-53 Rev. 4 standard [80]. FedRAMP authorization is required for a cloud service provider's service offering to be used to store any U.S. federal government data. Cloud service providers can choose to undergo assessment for one of two types of FedRAMP authorization (called an Authority to Operate, or ATO): 1) an agency-specific partnership for a particular use of the cloud service [110], such as the Joint Enterprise Defense Infrastructure (JEDI) contract put forth by the U.S. Department of Defense in 2018 [8], or 2) general-purpose, cross-agency authorization by a Joint Authorization Board (JAB) that allows the cloud service provider to advertise their service offering in the FedRAMP Marketplace [111] and can expedite assessment for future, agency-specific authorizations [112]. For both types of authorization, the actual security assessment (i.e., audit) of the system is performed by a third-party auditing organization. The security assessment proceeds as we previously described in Section 4.2.1.

In our evaluation, we examine the audit records for the security assessment undergone by the EIC provider for JAB authorization of the EIC system. In 2018, the system was audited for and achieved the FedRAMP Moderate authorization, then was subsequently audited for and achieved FedRAMP High authorization. We refer to the FedRAMP Moderate authorization as Audit 1, and the FedRAMP High authorization as Audit 2. To facilitate ease of explanation, we limit our evaluation to the controls and control enhancements within the Audit and Accountability (AU) control family.

The exact differences between the requirements for each level of FedRAMP certification are given in the catalog of FedRAMP baseline security controls [113], but we briefly summarize them here. First, the set of controls with which compliance must be demonstrated for the FedRAMP High baseline is a superset of those for the FedRAMP Moderate baseline. Second, the minimum parameter values (e.g., number of allowed concurrent sessions for privileged users, frequency of review of user accounts for inactivity) for some of the overlapping controls are stronger or more restrictive in the FedRAMP High baseline as compared to the FedRAMP Moderate baseline. Table 4.2 shows the number of controls present within each of

[2]For the FedRAMP program, certification is referred to as authorization.

Table 4.2: Number of controls per control family within each FedRAMP baseline level.

	FedRAMP High	**FedRAMP Moderate**	
Control Family	All controls	All controls	Controls w/diff. param. vals than High baseline
Access Control (AC)	54	43	5
Audit & Accountability (AU)	31	19	4
Awareness & Training (AT)	7	5	2
Configuration Management (CM)	36	26	3
Contingency Planning (CP)	35	24	3
Identification & Authentication (IA)	31	27	3
Incident Response (IR)	26	18	3
Maintenance (MA)	14	11	1
Media Protection (MP)	12	10	4
Personnel Security (PS)	10	9	7
Physical & Environmental Protection (PE)	27	20	2
Planning (PL)	6	6	2
Risk Assessment (RA)	12	10	2
Security Assessment & Authorization (CA)	16	15	1
System & Communications Protection (SC)	39	32	3
System & Information Integrity (SI)	39	28	3
System & Services Acquisition (SA)	26	22	3
Total	**421**	**325**	**51**

the FedRAMP High and Moderate baselines, as well as the number of controls within the FedRAMP Moderate baseline for which the parameter value differs from the High baseline.

Consequently, to ensure that the audit evidence that was deemed sufficient is comparable across the two audits, in our evaluation, we examine only those controls and control enhancements that appear in both FedRAMP Moderate and FedRAMP High baselines and for which the evidence artifacts required to demonstrate compliance would be identical for both. The latter is true for most shared controls, even if they have different minimum parameter values, since in most cases, the parameter value directly maps to a configuration setting or property of a log file, and the same configuration file or log file constitutes evidence for both Moderate and High baselines.

4.5.2 Analysis Results

In all, we consider evidence artifacts for the 20 individual controls and control enhancements within the AU family that meet the abovementioned criteria. Because the audit evidence

Table 4.3: Evidence artifact types collected for each AU family compliance requirement under consideration for Audit 1.

	Audit Stage 1		Audit Stage 2	
Compliance Requirement	Artifact types collected	Outcome	Artifact types collected	Outcome
AU-1	1	COMPLIANT	–	–
AU-2	2, 3	INVALID, SUFFICIENT	2	COMPLIANT
AU-2 (3)	2, 4	INSUFFICIENT	28	COMPLIANT
AU-3	3, 5	COMPLIANT	–	–
AU-3 (1)	3, 5, 6	COMPLIANT	–	–
AU-4	7	INVALID, SUFFICIENT	7	COMPLIANT
AU-5	8, 9	INSUFFICIENT	29, 30	COMPLIANT
AU-6	8, 9	COMPLIANT	–	–
AU-6 (1)	8, 9, 10	COMPLIANT	–	–
AU-6 (3)	11	INSUFFICIENT	31	COMPLIANT
AU-7	3, 12, 13	INSUFFICIENT	13, 32	COMPLIANT
AU-7 (1)	3, 14, 15	INVALID, INSUFFICIENT	14, 33	COMPLIANT
AU-8	16, 17	COMPLIANT	–	–
AU-8 (1)	18	COMPLIANT	–	–
AU-9	19, 20	COMPLIANT	–	–
AU-9 (4)	19, 21, 22	INSUFFICIENT	34	COMPLIANT
AU-11	7, 23	COMPLIANT	–	–
AU-12	2, 24, 25	COMPLIANT	–	–
AU-12 (1)	2, 25, 26	INVALID, INSUFFICIENT	26, 35, 36	COMPLIANT
AU-12 (3)	2, 24, 25, 27	COMPLIANT	–	–

artifacts collected by the EIC provider's compliance SMEs were labeled by the control identifier (e.g., "AU-7 (1)"), we use control identifiers as compliance requirements within our model. For the 20 compliance requirements we consider, in Audit 1, the compliance SMEs collected 36 unique artifact types from 25 data sources across 2 rounds of evidence collection. In Audit 2, they collected 29 unique artifact types from 22 data sources across 2 rounds of evidence collection.

For each audit, we manually examined each artifact type and data source and assigned it a unique identifier, then examined the audit reports using the method we described in Section 4.4.4 to identify the audit report outcomes for each compliance requirement. Table 4.3 lists all artifact types and audit outcome labels for each control identifier and audit stage for Audit 1, and Table 4.4 does the same for Audit 2. If the auditor found that the system was compliant with a given control at the end of an stage 1 of an audit, no additional evidence was collected for the control, so the cells for stage 2 for those controls are empty.

Of the 41 distinct artifact types, the following 16 were the sole artifact type pulled from their corresponding data source: 1, 2, 3, 4, 11, 14, 15, 16, 17, 18, 23, 28, 31, 34, 37, 40. The

Table 4.4: Evidence artifact types collected for each AU family compliance requirement under consideration for Audit 2.

	Audit Stage 1		Audit Stage 2	
Compliance Requirement	Artifact types collected	Outcome	Artifact types collected	Outcome
AU-1	1	COMPLIANT	–	–
AU-2	2, 3	COMPLIANT	–	–
AU-2 (3)	2, 4	INVALID, INSUFFICIENT	4, 28	COMPLIANT
AU-3	3, 5	COMPLIANT	–	–
AU-3 (1)	3, 5	COMPLIANT	–	–
AU-4	23, 37	INVALID, SUFFICIENT	37	COMPLIANT
AU-5	8, 29, 38	INVALID, SUFFICIENT	29	COMPLIANT
AU-6	8, 29	COMPLIANT	–	–
AU-6 (1)	8, 29, 39	COMPLIANT	–	–
AU-6 (3)	11	INSUFFICIENT	31	COMPLIANT
AU-7	3, 12, 13	COMPLIANT	–	–
AU-7 (1)	33, 40, 41	INVALID, SUFFICIENT	33, 41	COMPLIANT
AU-8	16	COMPLIANT	–	–
AU-8 (1)	18	COMPLIANT	–	–
AU-9	19, 20	COMPLIANT	–	–
AU-9 (4)	19, 21, 22	INSUFFICIENT	34	COMPLIANT
AU-11	7, 29	COMPLIANT	–	–
AU-12	2, 24, 25	COMPLIANT	–	–
AU-12 (1)	2, 25, 41	INVALID, SUFFICIENT	41	COMPLIANT
AU-12 (3)	2, 24, 25	COMPLIANT	–	–

remaining 25 artifacts can be grouped together by data source as follows: (5, 6), (7, 8), (9, 10), (12, 13), (19, 20), (21, 22), (24, 25, 41), (26, 27), (29, 30), (32, 33), (35, 36), (38, 39).

We observe that there is considerable overlap between the artifacts used to demonstrate compliance with different compliance requirements. The average number of different compliance requirements for which each artifact type was used in the case study example exceeds 1.5. Furthermore, we observe that the set of artifact types used for Audit 1 and Audit 2 are not identical. For example, the artifact types used for controls AU-4, AU-6 (1), AU-7 (1), and AU-12 (1) were considerably or entirely different between the two audits. Nevertheless, in both audits, the auditor ultimately deemed the evidence sufficient to demonstrate compliance.

Table 4.5 shows the evidence sufficiency models we obtained by applying Algorithm 4.1 to the evidence from each audit individually, as well as the combined evidence sufficiency model obtained by subsequently applying Algorithm 4.2.

One interesting property of the merged model is that it demonstrates a case of likely auditor oversight. For control AU-7, the set of evidence artifacts that passed Audit 2 was explicitly deemed insufficient in Audit 1. Rather than keep both sufficient evidence sets in

Table 4.5: Sets of sufficient evidence artifact types inferred from Audit 1, Audit 2, and their combination.

Compliance Requirement	**Audit 1**	**Audit 2**	**Merged**
AU-1	$\{1\}$	$\{1\}$	$\{1\}$
AU-2	$\{2, 3\}$	$\{2, 3\}$	$\{2, 3\}$
AU-2 (3)	$\{2, 4, 28\}$	$\{2, 4, 28\}$	$\{2, 4, 28\}$
AU-3	$\{3, 5\}$	$\{3, 5\}$	$\{3, 5\}$
AU-3 (1)	$\{3, 5\}$	$\{3, 5\}$	$\{3, 5\}$
AU-4	$\{7\}$	$\{23, 37\}$	$\{7\}, \{23, 37\}$
AU-5	$\{8, 9, 29, 30\}$	$\{8, 29, 38\}$	$\{8, 9, 29, 30\}, \{8, 29, 38\}$
AU-6	$\{8, 9\}$	$\{8, 29\}$	$\{8, 9\}, \{8, 29\}$
AU-6 (1)	$\{8, 9, 10\}$	$\{8, 29, 39\}$	$\{8, 9, 10\}, \{8, 29, 39\}$
AU-6 (3)	$\{11, 31\}$	$\{11, 31\}$	$\{11, 31\}$
AU-7	$\{3, 12, 13, 32\}$	$\{3, 12, 13, 32\}$	$\{3, 12, 13, 32\}$
AU-7 (1)	$\{3, 14, 15, 33\}$	$\{33, 40, 41\}$	$\{3, 14, 15, 33\}, \{33, 40, 41\}$
AU-8	$\{16\}$	$\{16, 17\}$	$\{16\}, \{16, 17\}$
AU-8 (1)	$\{18\}$	$\{18\}$	$\{18\}$
AU-9	$\{19, 20\}$	$\{19, 20\}$	$\{19, 20\}$
AU-9 (4)	$\{19, 21, 22, 34\}$	$\{19, 21, 22, 34\}$	$\{19, 21, 22, 34\}$
AU-11	$\{7, 23\}$	$\{7, 29\}$	$\{7, 23\}, \{7, 29\}$
AU-12	$\{2, 24, 25\}$	$\{2, 24, 25\}$	$\{2, 24, 25\}$
AU-12 (1)	$\{2, 25, 26, 35, 36\}$	$\{225, 41\}$	$\{2, 25, 26, 35, 36\}, \{225, 41\}$
AU-12 (3)	$\{2, 24, 25\}$	$\{2, 24, 25, 27\}$	$\{2, 24, 25\}, \{2, 24, 25, 27\}$

the model, our approach excludes the smaller set because we cannot safely assume that it will be sufficient in future audits (as it was already insufficient in a previous audit).

Finally, we examine the optimal evidence collection strategy we obtain by applying our approach. For the sake of simplicity, we assigned a constant evidence cost to all data sources, and looked only at the minimum evidence collection cost required to achieve full coverage of all compliance requirements, which was the intention of the EIC provider's compliance SMEs during both Audit 1 and Audit 2. The total cost of the evidence that was actually collected for Audit 1 was 25 data sources, and the total cost for Audit 2 was 22 data sources. Using the evidence sufficiency model inferred from the records of both audits, we were able to find a more cost-efficient strategy for evidence collection that requires collecting 29 artifacts from only *20 data sources*. Therefore, in a future audit, if the EIC provider were to use our approach to determine which evidence artifacts to collect rather than collect the same artifacts collected for either Audit 1 or Audit 2, they would reduce their overall evidence collection costs, thereby confirming the utility of our approach for the case study system.

4.6 DISCUSSION

Applicability across compliance standards. In our evaluation, we consider the application of our approach to only a single compliance regulation (namely, FedRAMP). In practice, a CSP may want to use evidence sufficiency models learned for a given standard or regulation to determine evidence collection strategies for audits of the same system but for other standards or regulations. Our approach is entirely capable of supporting such an application, provided that the CSP uses a common model for compliance requirements when labeling sufficiency of evidence across different audits. There have been some efforts to define such common models of compliance requirements across multiple standards. For example, the CSA Cloud Controls Matrix [81] defines a standard set of compliance requirements and maps them to the equivalent controls within multiple cloud-relevant standards and regulations (e.g., SOC, FedRAMP, PCI DSS).

Incorporating a notion of confidence in sufficiency labels. Compliance SMEs may desire to assign confidence values to the sufficient evidence sets learned by our model from historical audits in order to inject out-of-band information into evidence sufficiency model. For example, if the SMEs were to believe that the auditor had made an oversight in a previous audit based on their own internal analysis of the evidence, they may wish to decrease confidence in the sufficiency labels learned from that audit, rather than exclude the records from that audit entirely.

In our framework, we do not provide a mechanism for compliance SMEs to express such a confidence metric. However, we have discussed in Section 3.10 how we might extend that approach to handle probabilistic interpretations of sufficiency of evidence sets towards detection of an event. In the context of compliance audit, the same techniques should support such a metric as well.

4.7 CONCLUSIONS

We presented a taxonomic framework for understanding uncertainties in the audit process created by human involvement. Using that foundation, we identified the causes of and potential solutions to those uncertainties, and we explained why existing approaches for audit evidence collection that are based on formal logic and domain-specific languages are not sufficient to address challenges created by those uncertainties. We then proposed a method

to learn an unambiguous model of evidence sufficiency requirements from historical audit records to reduce inefficiencies in audit evidence collection caused by differing interpretations of compliance controls. We described how we model compliance requirements and audit evidence, and how we infer sufficiency of evidence in demonstrating compliance from audit reports collected during prior audits. We demonstrated the efficacy of our approach on the compliance audit records from an enterprise IaaS cloud system. Our approach reduces evidence collection costs and errors and manual effort required to identify evidence needed for compliance certification.

CHAPTER 5: CONCLUSIONS

In this dissertation, we set out to demonstrate that the efficiency, efficacy, and resilience of reliability and security monitoring in enterprise and cloud systems can be improved by principled collection and fusion of redundant data across heterogeneous monitors. We identified and were guided by the following principles about redundancy of monitor data in enterprise and cloud systems. First, by understanding which data are functionally or statistically redundant, system designers can improve the efficiency of monitoring by avoiding collection of duplicate information and by reducing the amount of data that must be evaluated during incident detection and investigation. Second, identification of redundancy across heterogeneous or distributed monitors can provide insights about system behavior that facilitate more effective measures to meet reliability and security objectives. Third, the resilience of the monitoring system against missing data can be achieved through strategic collection of redundant monitor data.

To validate our thesis, we applied the aforementioned principles to address various challenges that arise during system runtime and system design time. At runtime, we focused on facilitating more efficient and effective use of heterogeneous data in enterprise cloud systems during incident root cause analysis (Chapter 2). At design time, we focused on improving the resiliency and efficiency of monitoring for incident detection (Chapter 3) and audit evidence collection (Chapter 4).

First, in Chapter 2, we presented a taxonomic framework that enables more efficient and effective coordinated analysis of heterogeneous monitor data in enterprise cloud systems for incident root cause analysis and response. The framework consists of (1) a general taxonomy of the field types present in metric and log data, (2) a method to semi-automatically extract time series features from the data based on the taxonomy labels, and (3) a method to hierarchically cluster the features across monitors and hosts based on pairwise feature cross-correlation. Our framework drastically reduced the complexity of incident response in enterprise PaaS environments by simplifying the feature discovery process and reducing the number of unique features that an administrator must wade through during incident root cause analysis. Our results from applying the framework on an experimental PaaS cloud case study dataset showed that it can uncover features that are meaningful for root cause analysis that may not be obvious to administrators from application documentation or prior experience. The results also demonstrated that our hierarchical feature-clustering technique reliably removes

duplicate and uninformative features and can facilitate discovery of relationships between features across different monitors that are highly pertinent for root cause analyis, thus improving both the efficacy and efficiency of the use of the data in root cause analysis.

Then, we developed techniques to improve monitoring system design for two different applications of monitor data for enterprise and cloud reliability and security: incident detection (Chapter 3) and compliance audit (Chapter 4). In Chapter 3, we focused on the susceptibility of incident detection techniques to missing monitor data. Leveraging the insight we developed in Chapter 2 that monitor data are highly redundant and can often be used interchangeably to detect incidents, we proposed a methodology to preemptively enable data collection from redundant monitors to improve the resilience of an incident detection system against missing data. We presented a general model for system monitoring and incident detection, and we defined metrics that quantify (1) the amount of data that must be missing to subvert incident detection, (2) the damage potential of an attacker with a particular level of ability to hide monitor data, and (3) the resilience of a monitoring system against missing monitor data. We then developed an optimization approach that can be used to determine which monitors should be deployed in a system to maximize resilience against missing monitor data, subject to the operational monitoring cost constraints. Using a datacenter network case study model based on monitors employed in production systems, we demonstrated how our approach could be used to identify the most resilient monitor deployments at different cost levels for a given system design, and how it could be used to compare the cost-effectiveness and resilience of different monitoring system designs. We also used randomly-generated models of varying sizes and structures to demonstrate the scalability of our approach. Our approach enables system administrators to make more effective monitoring decisions in light of the reality of missing monitor data and to improve their resilience against missing data.

In Chapter 4, we focused on the inefficiencies that arise during evidence collection for security and compliance audit. Even when supported by existing tools and state-of-the-art techniques, security and compliance auditing is expensive, time-consuming, and error-prone for cloud service providers operating in multiple domains. To clearly identify the underlying causes of the inefficiencies, we first developed a taxonomic framework of uncertainties in the audit process. Through examination using our framework, we found that existing techniques and tools for evidence collection for compliance audit do not sufficiently account for the challenges caused by human involvement, which are a major contributor to inefficiencies and mistakes in the audit process.

To reduce collection of duplicate or superfluous evidence specifically caused by a difference in interpretation of compliance controls between auditors and the cloud service provider, we developed a technique that a cloud service provider can use to learn what evidence the auditors consider to be sufficient to demonstrate compliance. Our approach constructs an unambiguous model of sufficient evidence for each compliance requirement based on historical audit records. We then applied the resilient monitoring methodology we developed in Chapter 3 to determine the cost-optimal sufficient subset of evidence to collect based on the model. We demonstrated how our approach can improve the efficiency of audit evidence collection by applying it to the historical audit records from an enterprise IaaS cloud service.

5.1 TRANSITIONING TO PRACTICE

Though we demonstrated the application of each of our approaches to real systems and datasets, effective application of our techniques to enterprise and cloud systems at scale presents a number of challenges that we did not explicitly address in the preceding chapters. Here, we present some of those challenges and our proposed solutions.

5.1.1 Coordinated Analysis of Heterogeneous Monitor Data

Labeling structured field types. Our approach to semi-automated feature extraction requires that a domain expert accurately specify the field type labels for structured fields for all monitor types from which they wish to analyze data. In practice, the practitioner must also be able to write a transformer that can convert all values taken by each field to a structured data representation format (e.g., an integer value, a structured timestamp, etc.). While not necessarily difficult, both of these tasks can require some trial and error. It can be challenging to infer the correct format for all theoretical values of a field from a limited sample of data, so as violations to the expected format arise, the practitioner may need to revise their field labels or transformers. However, once tuned in, the labels and transformers can be reused ad infinitum for all data generated by the same monitor type.

For common log and metric types, however, practitioners can benefit from the prior labeling done by users of other well-established log processing tools, such as Prometheus [21] and Logstash [25]. The user communities for those tools have developed plugins for a variety of common log formats that may be used either directly or with little modification to convert log messages into a format actionable by our framework. In general, similar to those tools,

we envision that a public, community repository of field labels and transformers could be developed for most common monitor types, and a shared repository could be created within an organization for proprietary or domain-specific monitor types.

In addition to inferring structured field type labels through data analysis, it may also be possible to directly derive said labels and transformers from configuration files. For many common applications and systems, output fields and formats are specified by logging configuration parameters. When the configuration information is available, a practitioner could manually examine the logging configuration for each monitor type to determine the field type labels and appropriate transformers. Deriving the labels and transformers from configuration rather than from data would would likely be less error-prone since it would involve less reverse engineering or human interpretation. We also propose that practitioners could develop a service that consumes the configuration file and programmatically generates the field type annotations and transformer code. Doing so would require an understanding of all possible values that the configuration file could take, but would facilitate rapid adaptation if logging configuration parameters were changed.

Labeling parameter field types within unstructured logs. Since unstructured message formats for commonly-used monitor types are often entirely determined by source code and stable for a given application version, we envision that practitioners could develop a community repository of the field types and transformers for parameters within unstructured log messages, similar to the one we described for field types for structured fields. Unlike structured fields, however, organizations would likely need to perform additional, organization-specific labeling of parameters based on their specific workload characteristics and domain knowledge. We propose that practitioners could use community labels as a baseline for their own systems, then Individual teams could extend, modify, or curtail the parameter field type annotations based on their particular analysis needs or their computational budget for analysis.

Log message instability. As shown by Zhang et al. [114], in production systems, log messages can be unstable. Over the lifetime of the system, the log message formats may change due to source code updates, and errors in log centralization and pre-processing may introduce noise into the central store of log data. Our approach would recognize such changes as the introduction of new log message formats, which may split features and reduce the efficacy of the feature clustering.

To account for log instability, for versioned applications and services, practitioners may decide to keep versioned field type annotations (for both structured fields and parameters) that parallel service versions. However, in more complex cases where log messages are more dynamic (e.g., in systems under rapid development), service versions are not readily available, or unstructured log messages depend heavily on the other services present in the system (e.g., services that surface nested exception traces), practitioners may need to establish a process to periodically re-label parameter field types (e.g., once per day or once per week), depending on how frequently log messages change. We also propose that using semantic labels for unstructured log formats, such as those derived using natural language processing techniques [114, 115, 116], in place of the structural message formats extracted by Spell that we currently use, could prevent feature splitting and improve our approach's robustness to log message instability.

Scalability. At each level of clustering, the cross-correlation analysis we perform is quadratic in the number of features within the cluster and log-linear in the total number of samples. Clustering the features, which requires finding cliques in feature cross-correlation values, is exponential in the number of features. While our centralized implementation was tractable for our datasets, in massive scale cloud and enterprise systems with data sizes and feature counts orders of magnitude larger than ours, that may not be the case. As we mentioned in Section 2.6, one extension that might improve scalability would be to introduce additional levels to the clustering to take advantage of topological or physical redundancy at the host and networking layers.

In general, however, practitioners will likely need to paralellize and distribute both the cross-correlation analysis and feature clustering. Reimplementating our approach using more modern big data tools that support scalable time series analysis, such as TimescaleDB [117] or Spark [118], may partially address the problem, as those tools internally distribute workloads. However, to achieve data processing time constraints for massive scale systems, it may also be necessary to employ heuristic algorithms [119].

5.1.2 Resilience Against Missing Monitor Data

Model construction. The primary issue practitioners would face in transitioning our approach to practice is accurate and efficient model construction. Constructing a model for an arbitrary incident detection system, as we did for our case study system, requires a

domain expert with deep knowledge. One possible solution, which we mention at the end of Section 3.5.2, would be to develop a method to take in IDS configurations (e.g., rule sets) and construct models programmatically. Such an approach is feasible for rule-based and signature-based IDSes, as we have demonstrated in previous work [45]. More work would be required to generalize such an approach to a wider array of IDSes and data sources than the ones we previously examined.

In the case where multiple detection systems are used in conjunction, a practitioner would further need to standardize the inputs and outputs of each detection system to a common representation when constructing our model. We have not studied the problem extensively, but one approach that may aid in standardization is to identify a limited set of nearly identical detection rules that analyze the same or similar inputs, have similar conditional structures or signatures, and produce semantically similar outputs. For example, most IDSes contain rule sets that perform checks for brute force authentication attempts and scanning activity. Those "standard candle" rules could then be used to translate between IDS configurations. To identify equivalent events, it may be possible to borrow from the semantic labeling literature for log message formats [114, 115, 116] to identify semantically similar IDS alert messages.

Lastly, one particularly challenging aspect of model construction for large-scale systems is the identification of redundant data. Often, rule-based incident detection systems do not readily define what data are interchangeable inputs to a particular rule, so a practitioner would need to use their own domain knowledge to manually label redundant alert types. To address the challenge algorithmically, we propose that a practitioner could use data mining techniques to identify alert types that are strongly temporally correlated or co-occurrent in distributed workflows and therefore likely redundant. Such data mining techniques have already been applied to log messages in various contexts for workflow construction and anomaly detection [120, 121]; additional work would be needed to ensure that mined relationships represent true data redundancy and not spurious correlations. For alert types that behave like time series, a practitioner may alternatively be able to apply our feature clustering approach from Chapter 2. With any data-driven approach, there is an inherent risk of false positives and false negatives that could introduce errors into the model, but we leave further examination of modeling errors to future work.

Model skew. As detection rules are updated, new data sources are added, or system architecture changes, our model would need to be updated as well. For a large-scale system, in which such updates are more frequent, keeping our model up-to-date could be incredibly

time consuming.

The automated model construction techniques we mention above would help decrease the burden of updating the model, as a practitioner could simply rerun them when the system changes. The practitioner would need to examine how frequently such system changes occur in practice to determine how frequently it would make sense to update the model. In practice, it may be necessary to limit model updates to a fixed period (e.g., once per day) rather than perform them on-demand.

Scalability. As the amount of possible redundancy of alert types increases, the size of Σ grows exponentially. Models with large Σ are more expensive for the CP solver to solve, so they take longer to compute a single resistance score and consequently to find optimal deployments.

However, in practice, there is often considerable modularity within the alert type-event graph, so to improve scalability, it might be possible to subdivide the model into disjoint subgraphs and solve them separately, similar to the approach used by Peng et al. in deTector [51]. Alternatively, one could employ a heuristic solution for computing resistance scores in which the model is divided into densely connected subgraphs with sparse interconnectivity with respect to alert types, Tolerance values are are computed separately for each subgraph, and the resulting values are combined together to compute the Tolerance value for the entire model. More work would need to be done to ensure that the heuristic approach does not violate Theorem 3.1 or to devise an algorithm to compute the resistance score that can is accurate in the presence of violations.

5.1.3 Resilience Against Uncertainties in Audit Evidence Collection

Dynamicity of system architectures. In production environments, systems can change between distinct audits or between rounds of an individual audit in such a way that the validity of a set of collected evidence artifacts may change. By focusing on the sufficiency of the evidence, irrespective of its validity, our approach for determining evidence collection requirements is robust to such changes.

However, changes to a system's *architecture* may change the evidence necessary to demonstrate compliance. In our work, we assume that the system's architecture is static across audits to be able to infer that differences in evidence sufficiency across audit stages and across distinct audits are due to differences in the sets of evidence or differences in auditor

control interpretations. To account for the effect of system changes, practitioners applying our approach in practice would need to maintain a record of all system changes. Upon each model update inferred from audit records, the practitioners would need to compare the model update against the record of system changes; for those updates that may have been caused by the system change, the practitioners would need to discard all existing sufficient evidence sets corresponding to older versions of the system, as they may not be applicable to the updated system. In general, more work would be needed to incorporate system architecture into our model for evidence sufficiency.

Handling incomplete audit records. Another important challenge is that of ensuring that the audit evidence collection history is complete and correct, in line with the assumptions we make in Section 4.4. While evidence records from the initial evidence collection and remediation stages of audit are generally well-documented because they are compiled and shared with the auditors, the same is not always the case for evidence provided during the audit interview stage. For example, if evidence is provided on-demand during an on-site audit interview, unless proper records were kept regarding the difference between the evidence that was presented to auditors *prior* to the audit interview and the evidence that was shown *during* the interview itself, historical audit records may well be incomplete and/or incorrect. A practitioner attempting to apply our approach would need to perform due diligence to verify the correctness of historical audit records and properly document such discrepancies for future audits.

5.2 FUTURE DIRECTIONS

We believe that our work clearly demonstrates the need to critically examine the resiliency and efficiency of monitoring infrastructures in enterprise and cloud systems. This dissertation presents solutions to some of the challenges faced by practitioners when monitoring their systems for reliability and security objectives. However, we believe that there are other, related problems in the larger problem space of resilient monitoring system design that warrant further exploration. Here, we propose some extensions to our work that hold the potential to address those problems. We hope that this dissertation will pave the way for future research that focuses on the resilience of the monitoring infrastructure itself.

Through our work in Chapter 3, we have demonstrated the susceptibility of incident detection systems to missing monitor data. While we focused in that chapter on improving

the resiliency of the incident detection system through monitoring system design, there is also a need to be able to *detect* missing monitor data at system runtime to enable rapid incident response. Missing monitor data may in some cases be a rare early indicator of a stealthy attack or catastrophic failure, so its timely detection may be essential to preventing damage to the system. We believe that redundancy of information can be used to detect compromise of individual monitors. In most enterprise and cloud systems, there is extensive communication between services running on different hosts and between user-controlled devices, such as workstations, and hosts running services. The resulting patterns of log messages can often be separated into distinct, repeated workflows of activity, as shown by Deeplog [34], CloudSeer [48], and Lou et al. [120]. While Deeplog and CloudSeer mine workflows from individual monitors, we propose that it would be possible to link together multiple logs across a system by identifying the messages that are generated simultaneously for individual workflow actions (e.g., those representing different ends of a connection, or host vs. network view of a request). Using those mined relationships, we propose that it would be possible to mine workflows across monitors distributed throughout the system to establish a baseline for expected log message generation, then detect missing information as violations of that baseline. The feasibility of distributed log mining has been demonstrated by existing work; HERCULE [47], OmegaLog [122], and Fu et al. [123] all stitch together heterogeneous logs from different processes running on the same host for the purpose of detecting anomalies or performing digital forensics, and lprof [49] mines workflows from distributed logs for the highly structured case of cloud services. We believe that by focusing on redundant information across logs and on discrepancies from expected log generation behavior, it is possible to detect missing data in a robust manner as well. Our preliminary work in this direction is presented in [124].

As the fields of machine learning and artificial intelligence have advanced and become more accessible, many intrusion detection systems have emerged that employ complex detection algorithms that use deep learning and probabilistic graphical models. As we mentioned in Section 3.5.2, the modeling formalism we develop for resilient monitor deployment does not lend itself well to such detection models because their detection functions can exhibit high connectivity between alert types (algorithm input features) and events, albeit with low weight assigned to many input features. Our model's representation of such algorithms would include all alert types that contribute nonzero weight to the detection of an event in the event's sufficient evidence sets, but since our model treats all elements of a sufficient evidence

set as equally necessary for detection, it would likely improperly value the importance of different alert types. It would likely be necessary to explore fundamentally different techniques than the ones we have used to facilitate analysis of the effect of redundant monitoring on the resilience of detection systems that use deep learning or probabilistic graphical models. Given the growing adoption of complex detection systems in production enterprise and cloud environments, such an extension of our work would facilitate resilient monitoring system design for a much wider set of systems.

We believe that our work could also be applied to domains beyond enterprise and cloud systems, such as cyber-physical and Internet of things (IoT) systems. In this dissertation, we have demonstrated the need for a more holistic analysis of the resilience and efficiency of monitoring infrastructures in various types of systems. While we explicitly addressed various monitoring challenges faced by administrators in enterprise and cloud systems, similar challenges exist in other types of systems. More work may be required to tailor our approaches to the unique architectures and usage patterns of such systems. Indeed, the relatively smaller scale and complexity of many such systems may make them particularly amenable to the approaches we have developed.

5.3 TOWARDS A UNIFIED MONITORING FRAMEWORK

Finally, we briefly discuss how the different components of the work in this dissertation could be combined together into a unified framework for cloud and enterprise monitoring.

Figure 5.1 illustrates a theoretical general framework for monitoring of an enterprise or cloud system. The figure shows the role that the work in each chapter of this dissertation would play within each stage of the framework and how the stages would connect together to support end-to-end monitoring decision making. To form a unified framework, we envision that our feature extraction and reduction approach from Chapter 2 could be used to convert monitor data to time series features; incident detection and audit evidence validation would be performed on the data using organization-specific detection systems and audit requirements; and that decisions on what to monitor would be driven by the approaches we present in Chapters 3 and 4.

In practice, organizations already perform most or all of the tasks shown in Figure 5.1, but they may use less effective or efficient methods than we propose in this dissertation, so we envision that our work would supplant or augment techniques already in use. In addition, as we have detailed above, the approaches we present in this dissertation do not extend

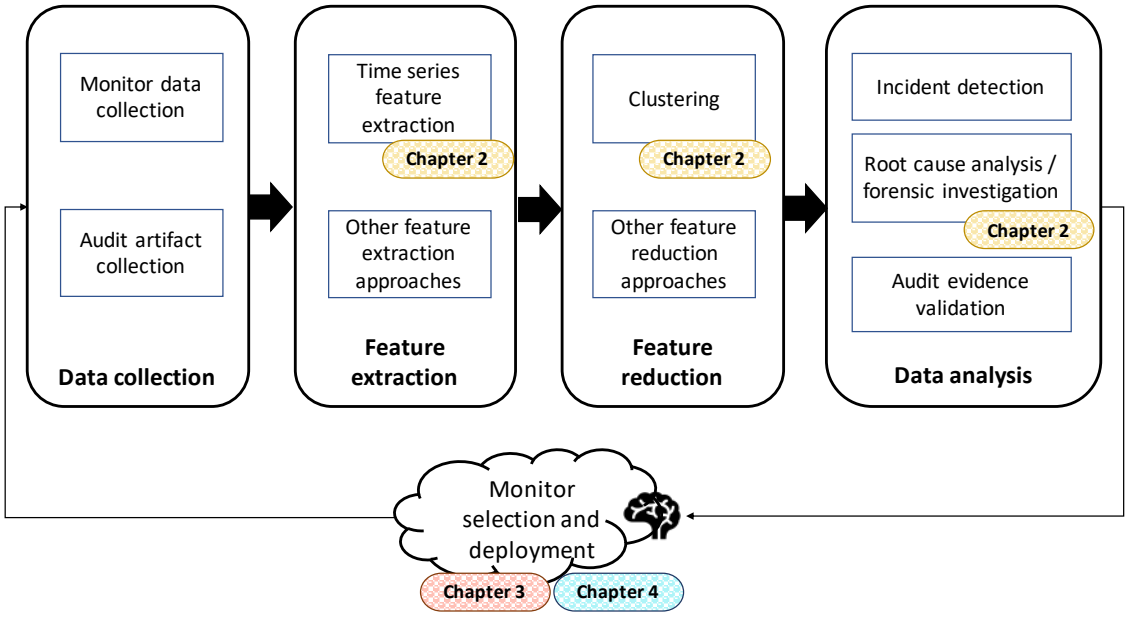


Figure 5.1: A theoretical general unified monitoring framework for enterprise and cloud systems. The roles that the work from each chapter of this dissertation would play in the larger framework are marked with colored ovals.

to all incident detection, root cause analysis, or audit evidence validation techniques an organization may employ, so, as shown in Figure 5.1, the components of our dissertation work would need to be augmented by other tools and techniques to form a complete monitoring framework.

REFERENCES

[1] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang, "Capturing and Enhancing in Situ System Observability for Failure Detection," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. Berkeley, CA, USA: USENIX Association, 2018, event-place: Carlsbad, CA, USA. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291168.3291170 pp. 1–16.

[2] "What Is an Advanced Persistent Threat (APT)? | Kaspersky." [Online]. Available: https://www.kaspersky.com/resource-center/definitions/advanced-persistent-threats

[3] Lloyd's of London and AIR Worldwide, "Cloud Down: Impacts on the US economy," Emerging Risk Report, 2018. [Online]. Available: https://www.lloyds.com/news-and-risk-insight/risk-reports/library/technology/cloud-down

[4] "What is an IT Security Policy?" library Catalog: www.paloaltonetworks.com. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-an-it-security-policy

[5] U.S. House, 104th Congress, "*H.R. 3103, Health Insurance Portability and Accountability Act of 1996*," Aug. 1996.

[6] *Requirements and Security Assessment Procedures*, PCI DSS ver. 3.2.1, May 2018.

[7] "Fedramp.gov." [Online]. Available: http://www.fedramp.gov/

[8] H. M. Peters, "The Department of Defense's JEDI Cloud Program," Congressional Research Service, Tech. Rep. R45847, Aug. 2019. [Online]. Available: https://fas.org/sgp/crs/natsec/R45847.pdf

[9] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and Benchmarks for Automated Log Parsing," in *Proc. 2019 41st ACM/IEEE Int. Conf. Softw. Eng.*

[10] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks," in *Proc. 2013 29th ACM Annu. Comput. Secur. Appl. Conf.*, pp. 199–208.

[11] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, "Detecting Credential Spearphishing in Enterprise Settings," in *USENIX Security '17*, 2017.

[12] "Azure status history | Microsoft Azure," Sep. 2019. [Online]. Available: https://status.azure.com/en-us/status/history/

[13] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why Does the Cloud Stop Computing?: Lessons from Hundreds of Service Outages," in *ACM SoCC '16*, 2016.

[14] W. Xu, "System Problem Detection by Mining Console Logs," Ph.D. Dissertation, University of California at Berkeley, Aug. 2010.

[15] S. Majumdar, Y. Jarraya, T. Madi, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "Proactive Verification of Security Compliance for Clouds Through Pre-computation: Application to OpenStack," in *Computer Security – ESORICS 2016*, vol. 9878. Springer, 2016, pp. 47–66.

[16] C. Sridharan, "Logs and Metrics," Medium, Apr. 2017. [Online]. Available: https://medium.com/@copyconstruct/logs-and-metrics-6d34d3026e38

[17] D. Reichert, "Logs and Metrics: What are they, and how do they help me?" Sumo Logic, Jan. 2018. [Online]. Available: https://www.sumologic.com/blog/logs-metrics-overview/

[18] "Reference - Azure Data Explorer," Microsoft, Accessed: 2019-08-31. [Online]. Available: https://docs.microsoft.com/en-us/azure/kusto/

[19] "Log Management Tool And Analytics — vRealize Log Insight," VMWare, Accessed: 2019-08-31. [Online]. Available: https://www.vmware.com/products/vrealize-log-insight.html

[20] "Enterprise Log Management for Any Scale," GrayLog, Accessed: 2019-08-31. [Online]. Available: https://www.graylog.org/products/enterprise

[21] "Prometheus - Monitoring system and time series database," Prometheus, Accessed: 2019-08-31. [Online]. Available: https://prometheus.io/

[22] "Machine Data Management & Analytics," Splunk, Accessed: 2019-08-31. [Online]. Available: https://www.splunk.com/en_us/software/splunk-enterprise.html

[23] Graphite, Accessed: 2019-08-31. [Online]. Available: http://graphiteapp.org

[24] "Grafana Features," Grafana Labs, Accessed: 2019-08-31. [Online]. Available: https://grafana.com/grafana

[25] "ELK Stack: Elasticsearch, Logstash, Kibana," Elastic, Accessed: 2019-08-31. [Online]. Available: https://www.elastic.co/elk-stack

[26] IBM, "JVM log interpretation," IBM Knowledge Center, Oct. 2014. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssaw57_9.0.5/com.ibm.websphere.nd.multiplatform.doc/ae/rtrb_readmsglogs.html

[27] IBM, "Message logging," IBM Tivoli Software Problem Determination Guide, Accessed: 2019-08-31. [Online]. Available: https://publib.boulder.ibm.com/tividd/td/ITAMOS/GC32-1106-00/en_US/HTML/am41_pdg08.htm#Header_78

[28] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in *Proc. 2016 IEEE 16th Int. Conf. Data Mining*, pp. 859–864.

[29] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "LogDiver: A Tool for Measuring Resilience of Extreme-Scale Systems and Applications," in *Proc. 2015 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, pp. 11–18.

[30] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. 9th Python in Sci. Conf.*, 2010, pp. 51–56.

[31] A. Bohara, U. Thakore, and W. H. Sanders, "Intrusion Detection in Enterprise Systems by Combining and Clustering Diverse Monitor Data," in *Proc. 2016 ACM Symp. and Bootcamp Sci. of Secur.*, pp. 7–16.

[32] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC," in *Proc. 2018 27th ACM Int. Symp. High-Perform. Parallel and Distrib. Comput.*, pp. 40–51.

[33] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "LOGAIDER: A Tool for Mining Potential Correlations of HPC Log Events," in *Proc. 2017 17th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Comput.*, pp. 442–451.

[34] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning," in *Proc. 2017 ACM SIGSAC Conf. Comput. and Commun. Secur.*, New York, NY, USA, pp. 1285–1298.

[35] D. Goodin, ""Nemesis" malware hijacks PC's boot process to gain stealth, persistence," Dec. 2015. [Online]. Available: https://arstechnica.com/information-technology/2015/12/nemesis-malware-hijacks-pcs-boot-process-to-gain-stealth-persistence/

[36] T. M. USA, "STUXNET Malware Targets SCADA Systems - Threat Encyclopedia - Trend Micro USA." [Online]. Available: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/54/stuxnet-malware-targets-scada-systems

[37] N. Kreculj, "How Hackers Hide Their Tracks: Part 1," June 2012. [Online]. Available: https://www.countertack.com/blog/bid/166056/How-Hackers-Hide-Their-Tracks-Part-1

[38] "Hack Like a Pro: How to Cover Your Tracks & Leave No Trace Behind on the Target System," Aug. 2013. [Online]. Available: https://null-byte.wonderhowto.com/how-to/hack-like-pro-cover-your-tracks-leave-no-trace-behind-target-system-0148123/

[39] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish, "Detecting failures in distributed systems with the Falcon spy network," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. Cascais, Portugal: Association for Computing Machinery, Oct. 2011. [Online]. Available: https://doi.org/10.1145/2043556.2043583 pp. 279–294.

[40] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang, "NetBouncer: Active Device and Link Failure Localization in Data Center Networks," 2019. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/tan pp. 599–614.

[41] H. Siadati and N. Memon, "Detecting Structurally Anomalous Logins Within Enterprise Networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134003 pp. 1273–1284.

[42] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of Early-Stage Enterprise Infection by Mining Large-Scale Log Data," in *Proc. 2015 45th Annu. IEEE/IFIP Int. Conf. Dependable Systems and Networks*, pp. 45–56.

[43] U. Thakore, H. V. Ramasamy, and W. H. Sanders, "Coordinated Analysis of Heterogeneous Monitor Data in Enterprise Clouds for Incident Response," in *IEEE ISSRE '19*, 2019.

[44] M. Holloway, *Service Level Management in Cloud Computing: Pareto-Efficient Negotiations, Reliable Monitoring, and Robust Monitor Placement*. Springer, June 2017.

[45] U. Thakore, G. A. Weaver, and W. H. Sanders, "A Quantitative Methodology for Security Monitor Deployment," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Toulouse, France: IEEE Computer Society, June 2016, pp. 1–12.

[46] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "SoK: Security and Privacy in Machine Learning," in *IEEE EuroS&P '18*, 2018.

[47] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "HERCULE: attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. Los Angeles, California, USA: Association for Computing Machinery, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2991079.2991122 pp. 583–595.

[48] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2872362.2872407 pp. 489–502.

[49] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm, "lprof: A Non-intrusive Request Flow Profiler for Distributed Systems," 2014. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/zhao pp. 629–644.

[50] P. Lapukhov, "NetNORAD: Troubleshooting networks via end-to-end probing," Feb. 2016. [Online]. Available: https://engineering.fb.com/core-data/netnorad-troubleshooting-networks-via-end-to-end-probing/

[51] Y. Peng, J. Yang, C. Wu, C. Guo, C. Hu, and Z. Li, "deTector: a Topology-aware Monitoring System for Data Center Networks," 2017. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/peng pp. 55–68.

[52] Q. Zhang, G. Yu, C. Guo, Y. Dang, N. Swanson, X. Yang, R. Yao, M. Chintalapati, A. Krishnamurthy, and T. Anderson, "Deepview: Virtual Disk Failure Diagnosis and Pattern Detection for Azure," 2018. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/zhang-qiao pp. 519–532.

[53] A. Belak, A. Barros, and A. Chuvakin, "How to Architect and Deploy a SIEM Solution," Gartner, Whitepaper G00366351, Oct. 2018. [Online]. Available: https://www.gartner.com/en/documents/3891675/how-to-architect-and-deploy-a-siem-solution

[54] L. Ma, T. He, A. Swami, D. Towsley, K. K. Leung, and J. Lowe, "Node Failure Localization via Network Tomography," in *Proceedings of the 2014 Conference on Internet Measurement Conference - IMC '14*. Vancouver, BC, Canada: ACM Press, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2663716.2663723 pp. 195–208.

[55] M. Natu and A. S. Sethi, "Probe Station Placement for Robust Monitoring of Networks," *Journal of Network and Systems Management*, vol. 16, no. 4, pp. 351–374, Dec. 2008. [Online]. Available: https://doi.org/10.1007/s10922-008-9114-0

[56] B. Patil, S. Kinger, and V. K. Pathak, "Probe station placement algorithm for probe set reduction in network fault detection and localization," in *2013 19th IEEE International Conference on Networks (ICON)*, Dec. 2013, pp. 1–6.

[57] "Lustre." [Online]. Available: http://lustre.org/

[58] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray Failure: The Achilles' Heel of Cloud-Scale Systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems - HotOS '17*. Whistler, BC, Canada: ACM Press, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3102980.3103005 pp. 150–155.

[59] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. Florianopolis, Brazil: Association for Computing Machinery, Aug. 2016. [Online]. Available: https://doi.org/10.1145/2934872.2934891 pp. 58–72.

[60] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "NetDiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data," in *Proceedings of the 2007 ACM CoNEXT conference*, ser. CoNEXT '07. New York, New York: Association for Computing Machinery, Dec. 2007. [Online]. Available: https://doi.org/10.1145/1364654.1364677 pp. 1–12.

[61] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The ϕ accrual failure detector," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, Oct. 2004, pp. 66–78.

[62] "Snort - Network Intrusion Detection & Prevention System." [Online]. Available: https://www.snort.org/

[63] "The Zeek Network Security Monitor." [Online]. Available: https://www.zeek.org/index.html

[64] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards Automated Log Parsing for Large-Scale Log Data Analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2018.

[65] J. Xue, R. Birke, L. Y. Chen, and E. Smirni, "Spatial–Temporal Prediction Models for Active Ticket Managing in Data Centers," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 39–52, 2018.

[66] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting Large-scale System Problems by Mining Console Logs," in *ACM SOSP '09*, 2009.

[67] Y. Park, I. M. Molloy, S. N. Chari, Z. Xu, C. Gates, and N. Li, "Learning from Others: User Anomaly Detection Using Anomalous Samples from Other Users," in *ESORICS '15*, 2015.

[68] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian, "Scalable detection of cyber attacks," in *Computer Information Systems: Analysis and Technologies*. Springer, 2011, pp. 9–18.

[69] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," in *IJCAI '19*, 2019.

[70] K. Miettinen, *Nonlinear Multiobjective Optimization*. Springer Science & Business Media, Dec. 2012, google-Books-ID: bnzjBwAAQBAJ.

[71] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[72] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[73] "Pagmo & Pygmo — pagmo 2.12.0 documentation." [Online]. Available: https://esa.github.io/pagmo2/

[74] 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.

[75] E. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke, "Model-based Security Metrics Using ADversary VIew Security Evaluation (ADVISE)," in *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, Sep. 2011, iSSN: null. pp. 191–200.

[76] "Compliance Programs – Amazon Web Services (AWS)." [Online]. Available: https://aws.amazon.com/compliance/programs/

[77] "Cloud Compliance – Regulations & Certifications." [Online]. Available: https://cloud.google.com/security/compliance/

[78] "Compliance on the IBM Cloud." [Online]. Available: https://www.ibm.com/cloud/compliance

[79] "Azure Compliance." [Online]. Available: https://azure.microsoft.com/en-us/overview/trusted-cloud/compliance/

[80] "Security and Privacy Controls for Federal Information Systems and Organizations," NIST, SP 800-53, Apr. 2013, doi: 10.6028/NIST.SP.800-53r4.

[81] "Cloud Controls Matrix v3.0.1," Nov. 2018. [Online]. Available: https://cloudsecurityalliance.org/artifacts/csa-ccm-v-3-0-1-11-12-2018-FINAL/

[82] *Quality management systems – Fundamentals and vocabulary*, ISO Std. 9000:2015(en), Sep. 2015.

[83] *Guidelines for auditing management systems*, ISO Std. 19 011:2018(en), July 2018.

[84] "New York Stock Exchange Listed Company Manual," Aug. 2013. [Online]. Available: http://wallstreet.cch.com/LCM/

[85] S. Majumdar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Cloud Security Auditing: Major Approaches and Existing Challenges," in *Foundations and Practice of Security*, vol. 11358. Springer, Nov. 2018, pp. 61–77.

[86] S. Majumdar, A. Tabiban, Y. Jarraya, M. Oqaily, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "Learning probabilistic dependencies among events for proactive security auditing in clouds," *Journal of Computer Security*, vol. 27, no. 2, pp. 165–202, Mar. 2019.

[87] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based Automated Reasoning for AWS Access Policies using SMT," in *Formal Methods in Computer Aided Design (FMCAD)*, Austin, TX, Oct. 2018, pp. 1–9.

[88] S. Lins, P. Grochol, S. Schneider, and A. Sunyaev, "Dynamic Certification of Cloud Services: Trust, but Verify!" *IEEE Security and Privacy*, vol. 14, no. 2, pp. 66–71, Mar. 2016.

[89] "Cloud Custodian." [Online]. Available: https://cloudcustodian.io/

[90] "Nessus Product Family." [Online]. Available: https://www.tenable.com/products/nessus

[91] T. Breaux and A. Antón, "Analyzing Regulatory Rules for Privacy and Security Requirements," *IEEE Trans. on Software Engineering*, vol. 34, no. 1, pp. 5–20, Jan. 2008.

[92] A. K. Massey, R. L. Rutledge, A. I. Anton, and P. P. Swire, "Identifying and classifying ambiguity for regulatory requirements," in *IEEE 22nd International Requirements Engineering Conference*, Aug. 2014, pp. 83–92.

[93] A. K. Massey, R. L. Rutledge, A. I. Antón, J. D. Hemmings, and P. P. Swire, "A Strategy for Addressing Ambiguity in Regulatory Requirements," Georgia Institute of Technology, Technical Report, 2015.

[94] "170.302: General certification criteria for complete ehrs or ehr modules," Code of Federal Regulations. [Online]. Available: https://www.govinfo.gov/content/pkg/CFR-2014-title45-vol1/xml/CFR-2014-title45-vol1-sec170-302.xml

[95] "Chef InSpec - Audit and Automated Testing Framework." [Online]. Available: https://www.inspec.io/

[96] "FedRAMP Security Assessment Framework, ver. 2.4," Nov. 2017. [Online]. Available: https://www.fedramp.gov/assets/resources/documents/FedRAMP_Security_Assessment_Framework.pdf

[97] "RSA Archer." [Online]. Available: https://www.rsa.com/en-us/products/integrated-risk-management/audit-management

[98] K. W. Ullah, A. S. Ahmed, and J. Ylitalo, "Towards Building an Automated Security Compliance Tool for the Cloud," in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 1587–1593.

[99] S. Majumdar, T. Madi, Y. Wang, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Security Compliance Auditing of Identity and Access Management in the Cloud: Application to OpenStack," in *IEEE 7th International Conference on Cloud Computing Technology and Science*, Nov. 2015, pp. 58–65.

[100] P. Stephanow and C. Banse, "Evaluating the Performance of Continuous Test-Based Cloud Service Certification," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 1117–1126.

[101] U.S. House, 111th Congress, "*H.R. 1, American Recovery and Reinvestment Act of 2009*," Feb. 2009.

[102] A. K. Massey, B. Smith, P. N. Otto, and A. I. Antón, "Assessing the accuracy of legal implementation readiness decisions," in *IEEE 19th International Requirements Engineering Conference*, Aug. 2011, pp. 207–216.

[103] T. Breaux, M. Vail, and A. Anton, "Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations," in *14th IEEE International Requirements Engineering Conference*, Minneapolis/St. Paul, MN, Sep. 2006, pp. 49–58.

[104] T. D. Breaux, "A method to acquire compliance monitors from regulations," in *Third International Workshop on Requirements Engineering and Law*, Sydney, Australia, Sep. 2010, pp. 17–26.

[105] 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, Tech. Rep. UILU-ENG-14-2202, July 2014. [Online]. Available: https://www.ideals.illinois.edu/handle/2142/90443

[106] U. Thakore, G. A. Weaver, and W. H. Sanders, "An Actor-centric, Asset-Based Monitor Deployment Model for Cloud Computing," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing*, Dresden, Germany, Dec. 2013, pp. 311–312.

[107] "Amazon CloudWatch – Application and Infrastructure Monitoring." [Online]. Available: https://aws.amazon.com/cloudwatch/

[108] "Home - xen project." [Online]. Available: https://xenproject.org/

[109] "Red Hat OpenShift, Built on Kubernetes." [Online]. Available: https://www.openshift.com

[110] "Get Authorized: Agency Authorization | FedRAMP.gov," library Catalog: www.fedramp.gov. [Online]. Available: https://fedramp.gov/agency-authorization/

[111] "The Federal Risk And Management Program Dashboard." [Online]. Available: https://marketplace.fedramp.gov/

[112] "Get Authorized: Joint Authorization Board | FedRAMP.gov," library Catalog: www.fedramp.gov. [Online]. Available: https://fedramp.gov/jab-authorization/

[113] "Documents | FedRAMP.gov." [Online]. Available: https://fedramp.gov/documents/

[114] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust Log-based Anomaly Detection on Unstable Log Data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: ACM, 2019, event-place: Tallinn, Estonia. [Online]. Available: http://doi.acm.org/10.1145/3338906.3338931 pp. 807–817.

[115] C. Bertero, M. Roy, C. Sauvanaud, and G. Tredan, "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection," in *IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2017, pp. 351–360.

[116] N. Aussel, Y. Petetin, and S. Chabridon, "Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2018, iSSN: 2375-0227. pp. 237–243.

[117] "Time-series data simplified." [Online]. Available: https://www.timescale.com/

[118] "Apache Spark™ - Unified Analytics Engine for Big Data." [Online]. Available: https://spark.apache.org/

[119] A. Mueen, S. Nath, and J. Liu, *Fast Approximate Correlation for Massive Time-series Data*.

[120] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining Invariants from Console Logs for System Problem Detection," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855864 pp. 24–24.

[121] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining Program Workflow from Interleaved Traces," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, NY, USA: ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1835804.1835883 pp. 613–622.

[122] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis," in *Proceedings 2020 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24270.pdf

[123] Q. Fu, J.-G. Lou, J. Li, and Y. Wang, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," in *2009 Ninth IEEE International Conference on Data Mining (ICDM 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 149–158.

[124] U. Thakore, A. Fawaz, and W. H. Sanders, "Detecting Monitor Compromise Using Evidential Reasoning: Poster," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS '18. New York, NY, USA: ACM, 2018. [Online]. Available: http://doi.acm.org/10.1145/3190619.3191693 pp. 16–16.