On the Effective Use of Data Dependency for Reliable Cloud Service Monitoring

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vom Fachbereich Informatik
der Technischen Universität Darmstadt

genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt von

Heng Zhang, M.Sc.

aus Hubei, China

Referenten:
Prof. Thomas Schneider, Ph.D.
Prof. Neeraj Suri, Ph.D.

Tag der Einreichung: 30.09.2019
Tag der mündlichen Prüfung: 18.11.2019

Darmstadt, 2019
D17

Heng Zhang: *On the Effective Use of Data Dependency for Reliable Cloud Service Monitoring*
Darmstadt, Technische Universität Darmstadt
Tag der mündlichen Prüfung: 18.11.2019

Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
URN: urn:nbn:de:tuda-tuprints-94644

On the Effective Use of Data Dependency for Reliable Cloud Service Monitoring

by

Heng Zhang

Erklärung

Hiermit versichere ich, die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 18. November 2019

Heng Zhang

Abstract

Cloud computing is a large-scale distributed computing paradigm that aims at providing powerful computing and storage capability by dynamically sharing a pool of system resources (e.g., network bandwidth, storage space, or virtualized devices) in a multi-tenant environment. With the support of the computing technology, a plethora of cloud services have been developed for meeting the different requirements of cloud service customers (CSCs). While cloud service has many attractive advantages (e.g., rapid service deployment, reliable service availability, elastic service reconfiguration, or economic service billing), the security assurance of cloud services is a key concern for the service customers.

Cloud monitoring is an essential mechanism for managing the security assurance of cloud services. Over the last few years, a large number of monitoring mechanisms have been proposed. The mechanisms are developed for monitoring varied security problems in the cloud with the common assumption that all the monitoring information is directly available. These mechanisms can achieve satisfactory monitoring performance only if the assumption can be satisfied (e.g., protecting cloud services from distributed denial of service (DDoS) attacks by checking the traffic information collected from network monitors). However, the existing mechanisms are unfortunately incapable of dealing with the security threats that are subtly crafted by malicious attackers without producing evident attack traces. Due to that the useful information related to the attacks is difficult to collect, the attacks can silently bypass the existing monitoring mechanisms and secretly undermine the victim services. As a result, to develop an effective monitoring mechanism for securing cloud services becomes a compelling demand.

For motivating the issue, this thesis initially investigates a typical cloud security attack that can gradually drain system resources in a target cloud without triggering any alarms for highlighting the realistic demand of performing effective security monitoring in cloud systems. To combat the attack, a pragmatic security countermeasure is proposed for securing the cloud. To meet the demand, the thesis focuses on achieving effective security assurance management of cloud services by addressing the common shortcoming of existing monitoring mechanisms. Using the data relation (i.e., data dependency) existing in the collected monitoring data sets, the thesis demonstrates the possibility of leveraging the available information and the existing data relation to indirectly monitor cloud security problems with a novel inference-based security mechanism. Furthermore, the thesis also demonstrates the feasibility of taking advantage of data dependency to obtain the input information for running the inference mechanism by developing a practical data ascertaining technique. Finally, this thesis targets addressing potential data errors that can undermine the reliability of the proposed monitoring mechanism. Consequently, a reliability assessment mechanism is developed to select suitable data used by the proposed mechanism for generating reliable monitoring results.

Zusammenfassung

Cloud-Computing ist ein Paradigma des groß angelegten, verteilten Rechnens, das darauf abzielt, leistungsstarke Rechen- und Speicherkapazitäten bereitzustellen, indem es einen Pool von Systemressourcen (z.B. Netzwerkbandbreite, Speicherplatz oder virtualisierte Geräte) in einer von mehreren Nutzern geteilten Umgebung dynamisch zuteilt. Es wurde eine Vielzahl von Cloud-Services entwickelt, um den unterschiedlichen Anforderungen von Kuden (Cloud Service Customers oder CSCs) gerecht zu werden. Während Cloud-Services viele attraktive Vorteile bieten (z.B. schnelle Servicebereitstellung, zuverlässige Serviceverfügbarkeit, elastische Service-Rekonfiguration oder wirtschaftliche Serviceabrechnung), ist die Sicherheit von Cloud-Services für die Servicekunden ein zentrales Anliegen.

Das sogenannte Cloud-Monitoring ist ein wesentlicher Mechanismus für die Verwaltung der Sicherheit von Cloud-Services. In den letzten Jahren wurde eine Vielzahl von Überwachungsmechanismen entwickelt. Diese Mechanismen wurden für die Überwachung verschiedener Sicherheitsprobleme in der Cloud entwickelt, mit der gemeinsamen Annahme, dass alle zur Überwachung erforderlichen Informationen direkt verfügbar sind. Folglich können solche Mechanismen nur dann eine zufriedenstellende Überwachungsleistung erzielen, wenn diese Annahme erfüllt werden kann (z.B. Schutz von Cloud-Diensten vor Distributed Denial of Service (DDoS) Angriffen durch Überprüfung der von Netzwerk-Monitoren gesammelten Verkehrsinformationen). Bestehende Mechanismen sind jedoch leider nicht in der Lage, mit Sicherheitsbedrohungen umzugehen, die von Angreifern subtil inszeniert werden, ohne offensichtliche Angriffsspuren zu hinterlassen. Da nützliche Informationen über die Angriffe schwer zu sammeln sind, können die Angriffe die bestehenden Überwachungsmechanismen umgehen und die angegriffenen Dienste heimlich untergraben. Infolgedessen wird die Entwicklung eines effektiven Überwachungsmechanismus zur Sicherung von Cloud-Diensten zu einer zwingenden Notwendigkeit.

Zur Motivation des Problems untersucht diese Arbeit zunächst einen typischen Cloud-Sicherheitsangriff, der Systemressourcen in einer Ziel-Cloud schrittweise aufzehren kann, ohne Alarme auszulösen, um die realistische Anforderung einer effektiven Sicherheitsüberwachung in Cloud-Systemen hervorzuheben. Um den Angriff zu bekämpfen, wird eine pragmatische Maßnahme zur Sicherung der Cloud vorgeschlagen. Die Arbeit konzentriert sich auf das Erreichen einer effektiven Sicherheitsverwaltung von Cloud-Services, indem sie den gemeinsamen Mangel an bestehenden Überwachungsmechanismen behebt. Unter Verwendung der in den gesammelten Monitoring-Datensätzen vorhandenen Datenrelation (d.h. Datenabhängigkeit) demonstriert die Arbeit die Möglichkeit, die verfügbaren Informationen und die bestehende Datenrelation zu nutzen, um indirekt Cloud-Sicherheitsprobleme mit einem neuartigen inferenzbasierten Sicherheitsmechanismus zu überwachen. Darüber hinaus zeigt die Arbeit auch die Machbarkeit der Nutzung der Datenabhängigkeit, um die Input-Informationen für den Betrieb des Inferenzmechanismus durch die Entwicklung einer praktischen Datenerhebungstechnik zu erlangen. Abschließend befasst sich diese Arbeit mit potenziellen Datenfehlern, die die Zuverlässigkeit des vorgeschlagenen Überwachungsmechanismus beeinträchtigen können. Daher wird ein Mechanismus zur Zuverlässigkeitsbewertung entwickelt, um geeignete Daten auszuwählen, die der vorgeschlagenen Mechanismus zur Erzeugung zuverlässiger Überwachungsergebnisse verwenden kann.

Acknowledgments

I consider myself so lucky that I have the opportunity to work toward the PhD in my life. It is my pleasure to acknowledge the important people who give me a lot of support on this exciting academic journey in Technische Universität Darmstadt.

First and foremost, I would like to show my gratitude to my advisor Prof. Dr. Neeraj Suri, who provides me many insightful research advice and shares a lot of precious professional experience with me. Thank you for building such a great research group where every member can get promptly support from your side and devote himself to pursuing his own research interest.

Besides, I want to show my special appreciation to Prof. Dr. Thomas Schneider for acting as the referent of the defense and Prof. Dr. Guido Salvaneschi for being the chair of the committee. I would like to also thank Prof. Dr. Max Mühlhäuser and Prof. Dr. Reiner Hähnle for being the members of the committee.

Moreover, I am very grateful to all colleagues for the intensive interaction and productive cooperation over the past years. I would like to thank Salman and Tsvetoslava for the insightful discussions about the topic of cloud security monitoring. I also would like to thank Oliver, Nicolas, Yiqun, Stefan and Habib from the system security team for sharing their knowledge with me. A big thank to both Sabine and Ute for their great job to support the entire group. In addition, I also want to show my thanks to former members Jesus, Ruben, Kubilay, Mohammadreza, Ahmed, Hatem, Daniel, Thorsten and Patrick.

Last but not least, I would like to show my deep gratitude to my family and friends at this exciting moment. Without your support, I cannot accomplish my PhD journey alone.

Contents

1 Introduction

Cloud computing is a novel technology paradigm that targets achieving on-demand service provisioning through flexibly configuring and dynamically sharing a pool of software, hardware and virtualized resources (e.g., CPU, memory, storage, or network) among multiple tenants in a large-scale distributed computing system [MG11]. Empowered by the novel computing technology, cloud services are proliferating across a wide range of business fields for satisfying the varied demand of the people. For example, Amazon Inc. that is a worldwide cloud service provider (CSP) offers more than 165 services (e.g., elastic computing, deep learning, big data storage, or mobile application development) to serve different cloud service customers (CSCs) such as companies, individuals, organizations, or governments through its proprietary computing platform termed Amazon Web Service (AWS) [Ama19]. Other CSPs like Google or Microsoft also provide similar all-in-one products like Google Cloud [Goo19] or Microsoft Azure [Mic19]. Apart from the leading providers, a plethora of small and medium size CSPs target rendering services with distinctive features. For instance, Dropbox offers superior file hosting service on top of Amazon AWS with the attractive feature of synchronizing and sharing files among different users [Dro19], while GitHub provides version controlled repositories in the cloud to facilitate software development, code review, or team collaboration for both the open-source community and business companies [Git19]. According to the report from Forbes, the global cloud market is projected to more than 200 billion US dollars [Col18]. Likewise, Gartner has also forecast that the overall revenue of cloud service market keeps increasing over the next few years [Gar19]. Briefly, cloud services have deeply integrated into modern business activities as well as people's daily lives.

The far-reaching impacts introduced by cloud services has been observed in many aspects. One of the most crucial aspects is with respect to the security assurance of cloud services. In practice, plenty of cloud attacks that focus on secretly compromising victim services have been reported over the past years. Different from traditional security threats (e.g., brute-force attacks or denial-of-service (DoS) attacks) that generally produce an enormous amount of attack traces, cloud attacks tend to carefully cloak their existence in the victim systems and thus pose an immense challenge to security administrators. For example, Heartbleed attack is a notorious security threat that has successfully compromised millions of thousands of cloud services worldwide [NVD14]. The attack exploits the careless program bug of the OpenSSL library that is widely used for performing transport layer security (TLS) authentication. In consequence, the attacker can successfully steal very sensitive information from the CSCs, such as user login credentials, bank account details, or personal medical records [Die99]. For another example, iCloud, as a frequently used cloud service for Apple customers, has also suffered from a very serious attack whereby security attackers can obtain an unauthorized copy of user data containing highly private photos and other sensitive information of the victim CSCs [App14]. Both examples explicitly demonstrate the severity of security threats in the cloud. As a matter of fact, the security vendor McAfee has reported that up to 80% of Infrastructure as a Service (IaaS) and Software as a Service (SaaS) users have suffered from at least one security incident [LLC19]. To address the cloud threats, effective security mechanisms are demanded.

Implementing security monitoring on cloud services is an essential approach for improving security assurance. Nowadays, many CSPs make use of existing security mechanisms for monitoring their systems. The existing mechanisms are initially proposed for dealing with security threats like network traffic attacks or brutal-force password attacks in traditional information systems. However, the proposed mechanisms are unfortunately incapable of achieving satisfactory security performance in a cloud scenario. Specifically, the mechanisms usually adopt a similar design architecture that relies on a central node to manage the information collected from all client nodes and generate monitoring results. The inherent drawback of the mechanisms is rooted in the scalability of the architecture.

While a large number of monitoring mechanisms have been proposed for securing cloud services. The mechanisms have rarely considered the difficulty of collecting relevant monitoring data in the realistic scenario. Unlike performing security monitoring on local information systems where the information used by monitoring mechanisms is accessible, monitoring cloud services often encounter the information collection problem for due to realistic constraints. As an example, all CSPs provisioning their services to EU citizens are obligated to manage the cloud in compliance with GDPR [PE19]. As a result, some sensitive information is strictly protected and impossible to collect by cloud monitors. The information collection problem essentially undermines the viability of the existing monitoring mechanisms for securing cloud services.

To achieve service security monitoring, the prerequisite is to have an insightful understanding on cloud systems especially regarding the peculiarities that are distinct from other information systems. One distinctive observation is that cloud systems widely apply virtualization technologies to fulfill resource sharing among a set of running services [plc15][IBM19][VMw19a][VMw19b]. The technology supports flexibly isolating and partitioning relevant system resources based on different service demand [JC16], while it also inflicts hardship for collecting relevant monitoring information that is strictly isolated from external access. Without the information, many existing monitoring mechanisms are no longer viable. Another noteworthy observation is that a cloud system consists of a large number of coordinated servers (a.k.a server farms) that could be located anywhere of the world. The distributed servers are cooperating with each others to deliver services in compliance with respective requirements contracted between CSPs and CSCs. Considering the physical scale and the structural complexity of the cloud system, massive information could be continuously collected by the security monitors that are preconfigured in different locations of the system. Hence, the collected information is not only too excessive to manage in a prompt manner but also too heterogeneous to perceive its value for monitoring security problems.

As collecting monitoring information in cloud systems is sometimes a formidable obstacle, the question arises if it is possible to develop a workaround by leveraging existing information to indirectly monitor security problems. Cloud attacks are often subtly crafted to avoid precipitating obvious attack traces. Securing cloud services against such threats is an arduous task, while it might be possible to accomplish the task by carefully investigating the collected information and the existing data relations. Generally, monitoring information (e.g., system call events, file system operation violations, or user activity logs) reflects the status of the target system from a very specific perspective. Any operation or behavior, regardless of malicious or not, taken place in the system can exert an influence on the current status. Such influences that represent the correlation with the occurred operations or behaviors are usually implicitly contained in the collected information. Namely, implementing cloud security attacks can trigger a sequence of system status changes but the collected monitoring information is unable to provide a straightforward clue for spotting this attack. If prudently examining the monitoring information together with the implicitly existing data relations (e.g., from a dependency perspective), it might be possible to discover profound security hints. In other words, a good understanding of monitoring data set supports to infer the obscure cloud security threats. As a result, achieving an in-depth understanding of monitoring information is a key factor for enhancing monitoring performance. Unfortunately, existing security mechanisms often overlook the implicit clues and predominantly rely on the explicit information to perform monitoring tasks.

From a security perspective, some information collected from particular software components or hardware is non-trivial than others for monitoring a specific security problem. Nonetheless, ascertaining the information valuable for pinpointing security problems is still an open issue. For instance, CPU utilization rate is a key reference for determining whether a process with extremely high CPU-load has been exploited by attackers. By contrast, some file system operation (like mounting a storage devices for reading files via executing *"/bin/mount"*) is insignificant for determining the potential process exploitation. The reason behind such a difference is high CPU workload has been rarely observed in a normal cloud system, while the file operations that are repetitively executed in the system is less useful for monitoring security threats. The example presents a very simple scenario where just two pieces of monitoring information are considered. In a real cloud system, the collected monitoring information is highly heterogeneous. Unfortunately, the heterogeneity results in that effectively ascertaining the valuable information very burdensome, even without considering the existing data relations. As a result, it is desirable to develop a systematic methodology for successfully distinguishing the valuable information from the raw data set.

Reliability, as a critical concern for assessing distributed systems, imposes limitation on the achievable monitoring performance of cloud systems [How+88] [Xin+03][Ses+04]. The results generated by the indirect monitoring methodology are based on the inferences made from relevant information together with the corresponding data relations (i.e., dependencies). Theoretically, the reliability of monitoring results, especially derived by applying some inference-based methodology, can be significantly undermined. The overall reliability of monitoring results keeps degrading as an increasing number of cloud nodes deployed into a cloud system. It is challenging to timely collect reliable monitoring data in such a large scale and high dynamic scenario. The latency is likely to exist between the occurrence of security problems and the collection of monitoring information. As a result, many transient security problems are probably overlooked. In spite of considering the time latency, collecting monitoring information in a reliable way is still a hard task. The existing monitoring schemas are commonly proposed with the default assumption that all information collected by cloud monitors is reliable. As a matter of fact, unpredictable problems can appear in any time when performing cloud monitoring tasks. For example, the possibility of software component malfunction is parallel to the complexity degree of the cloud system. In addition, the program bug might produce incorrect information under particular conditions that are contrary to theoretical expectations. If the erroneous information is used as the input by monitoring mechanisms, the output results are thus unreliable. All factors create the bottleneck of achieving reliable security monitoring. However, complementary solutions are unable to successfully address the problem.

Based on the general background, this thesis focuses on

- performing a case study on the state of the art cloud attack for revealing the characteristics of modern security threats in the cloud and highlighting the significance of developing effective indirect monitoring mechanisms to address the major drawback of existing security mechanisms.
- developing a pragmatic inference-based security mechanism for indirectly monitoring cloud problems by taking advantage of the information collected by security monitors and relevant data relations existing in the monitoring data set (i.e., data dependency).
- designing a practical data analysis technique that leverages the dependency between some monitoring data and the security threats to identify the valuable input information for enabling the indirect monitoring mechanism.
- devising an effective optimization approach for obtaining reliable monitoring information involving the monitoring process so as to improve the reliability of the indirect monitoring mechanism.

The developed mechanisms and techniques target improving the security monitoring performance of cloud services. The relevant studies and evaluations have been conducted for assessing the effectiveness of the corresponding propositions.

The reminder of this chapter is structured as follows. Section 1.1 presents the basics of contemporary cloud systems and summaries the characteristics. Section 1.2 presents a brief discussion of data dependency from a security perspective.

1.1 A Brief Overview of Modern Cloud Systems

This section presents the background of modern cloud system to provide the common base for the discussions in this thesis. It explains the noteworthy features and reviewing the cloud from a utility perspective.

Cloud system is a novel utility for rapidly provisioning on-demand services for massive CSCs with minimal management cost and optimal resource multiplexing. Compared to traditional IT systems, cloud systems have several distinctive advantages. First, the system supports flexible pay-as-you-go billing mechanism that helps CSCs to reduce the expenditure for using cloud services. Furthermore, the system is pooling system sources that are dynamically allocated for executing relevant services and promptly collected back to the pool after the execution. Moreover, the CSCs can perform on-demand reconfiguration (scale up/scale up) on system resources without the intervention from other parties. In addition, the system offers alternative interfaces that can be accessed by diverse devices such as smart phones, tablets, or computers. Last but not least, the system also underpins elastic service provision according to the rapid change of customer demand.

Cloud systems refer to different concepts according to the specific contexts [Fox+09]. Precisely, from a hardware perspective, it refers to an enormous amount of hardware servers and other peripherals (e.g., network connection or storage devices) that are the infrastructural foundation for running cloud services. From a software perspective, it refers to a computing system that runs remotely for providing particular services operated by CSPs. It is worth mentioning that this thesis puts emphasis on the exact software aspect of the cloud system.

Figure 1.1 depicts a combined view of cloud systems that is composed of the service model and the deployment model. Based on the different types of cloud services provisioned to customers, the service model consists of three different components as shown in the top-left corner of the Figure. Specifically, 1) Infrastructure as a Service (IaaS) targets provisioning necessary hardware resources (e.g., CPU, memory, storage or network) for enabling CSCs to install relevant virtual machines and operation systems as well as the ancillary applications. The CSCs have the full control on the installed programs, while they do not involve the management of the underlying infrastructure. 2) Platform as a Service (PaaS) focuses on providing the environment where CSCs can create and host customized platforms (e.g., programming development environment, database systems, or software frameworks) by leveraging the programs, software libraries or relevant services from the underlying IaaS. Likewise, the CSCs can manage their own platform without the necessity of dealing with the underlying infrastructure. 3) Software as a Service (SaaS) aims at directly offering CSCs specific services that are deployed on top of PaaS. A variety of SaaS examples can be found, such as managing business by SAP, checking email

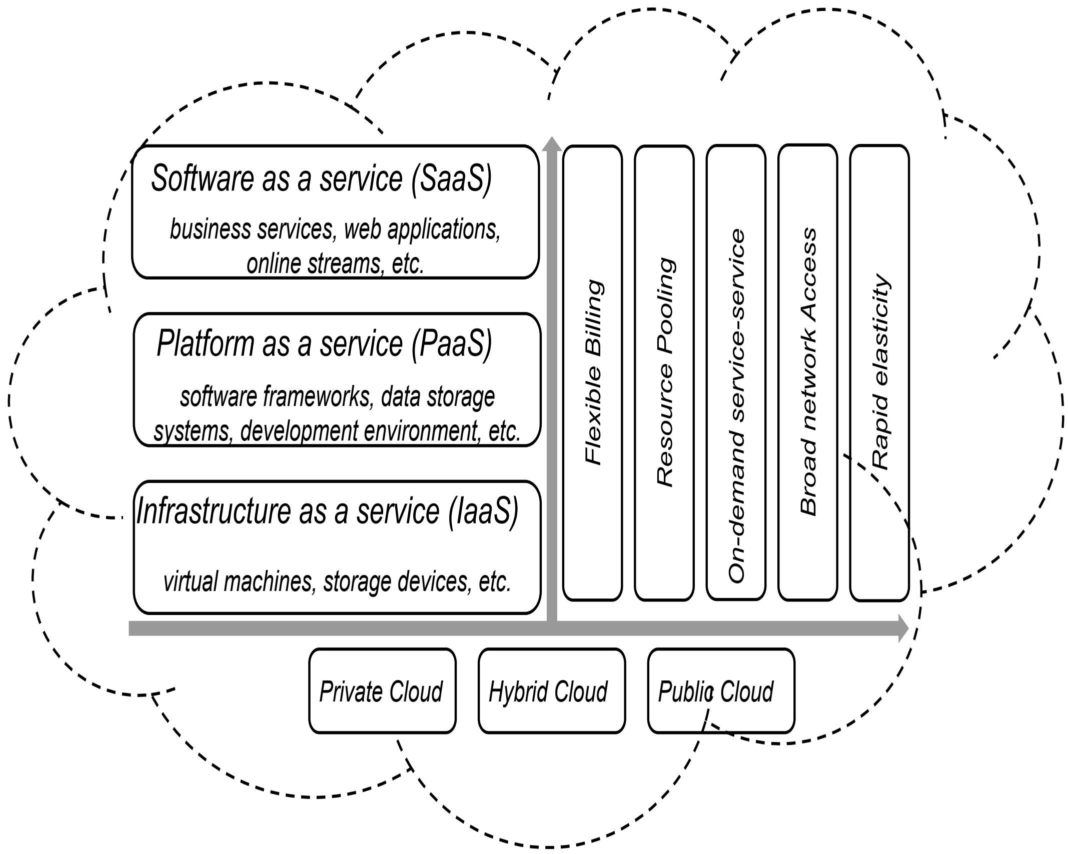

Software as a service (SaaS)
business services, web applications, online streams, etc.
Platform as a service (PaaS)
software frameworks, data storage systems, development environment, etc.
Infrastructure as a service (IaaS)
virtual machines, storage devices, etc.
Flexible Billing
Resource Pooling
On-demand service-service
Broad network Access
Rapid elasticity
Private Cloud
Hybrid Cloud
Public Cloud

Figure 1.1: An overview of cloud system that consists of the service model, the deployment model, and the advantage model

with gmail, or watching online streams via Netflix. The CSCs can use the services without considering other issues.

Taking the user role into consideration, the deployment model of cloud systems comprises three components. The first one is public cloud, like Amazon AWS or Microsoft Azure, that is constructed and operated the cloud system on the premises of some business companies (or other organizations) for provisioning services to the public. On the contrary, the private cloud is created for serving exclusive users who generally belong to the same companies (or organizations). Different from the public cloud, the private cloud is not necessarily built on the CSP's proprietary premises but possible to be hosted in some public cloud. The hybrid cloud is composed of both types of cloud mentioned above. However, it requires specialized configuration to guarantee data interoperability and contextual information sharing among the different types of cloud.

Cloud systems pose some obstacles that introduce tremendous difficulties for achieving effective service monitoring. First of all, cloud systems are commonly advocated for the high availability of service provisioning. Without timely discovering the security threats that might result in serious service availability deterioration, CSPs would suffer from both financial loss and reputation damage. As a large scale distributed system, the cloud system is very likely to suffer from unknown software bugs that can unexpectedly trigger serious service interruption. The unforeseeable bugs are generally not reproducible and extremely hard to monitor. It poses a significant challenge for using cloud monitoring to manage service availability. Besides, cloud systems incorporate plenty of commercial software components (or libraries) that are completely proprietary so that the software APIs are kept as the business secret for preventing potential external access to internal details. It is a formidable obstacle to collect useful information for performing service monitoring tasks. In the end, cloud systems take advantage of the virtualization technology to multiplex system resources for serving multiple CSCs in a parallel manner [JC16]. Nonetheless, virtualized environments are not completely immune to security attacks. For instance, improper virtualization on network might enable the attackers to retrieve the sensitive information of CSP's infrastructure details [Arm+15]. It is worth noticing that the scheduling mechanism of virtual machines makes it difficult to understand the relevant program details with respect to batch processing scenarios. Hence, virtualization mechanisms add extra difficulty to implement effective monitoring on cloud systems. Therefore, it is dispensable to develop an effective mechanism that can overcome the aforementioned difficulties.

1.2 A Monitoring Perspective on Dependencies

From a monitoring perspective, it is vital to deal with the dependencies existing in a target system. The dependencies refer to the effect of an object's attribute change (e.g., data value variance, program status alterations, or control flow redirection) that is caused by another object under a given condition in a given system. For example, the return-to-libc attack is a serious security threat for cloud systems. To perform the attack, the attackers are depending on writing the crafted content at the particular return address on the call stack. The content aims at replacing the original addressing information with a selected subroutine that is existing in the executable memory of a target process. Consequently, the program control flow is tampered by the attackers. As a result, security mechanisms need to carefully investigate the dependencies for improving the monitoring performance.

During monitoring a target system, data dependency is the most commonly observed data relation in the collected monitoring data sets. If some data dependency is exceptionally introduced by some vulnerabilities, security attackers might take advantage of it for compromising cloud services as reported in [Kru+05]. For ease of discussion, we succinctly present the exploitation process with a simplified automaton in Figure 1.2. Specifically, the service is designed to execute privileged

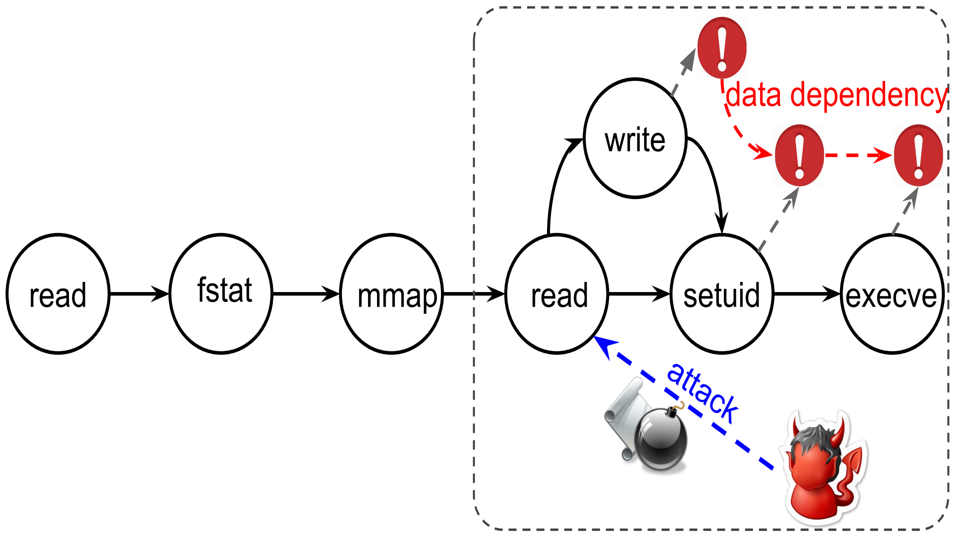


Figure 1.2: An example program automaton for highlighting the influence of data dependency

operations (via system call *execve*) after successfully passing user identity authentication (via system call *read*). Besides, it also records the history of issued operations together with the relevant information of the user (via the system calls in the dashed box). While the authentication function unexpectedly contains code flaw that overlooks necessary boundary check of the relevant parameters before executing the operations. Hence, security attackers can exploit the flaw by passing crafted parameters (the system call *read* in the dashed box) for running malicious codes. It is hard to monitor the exploitation of the unexpected flaw, while monitoring relevant system call invocations can exhibit some eccentricities (depicted by exclamation marks) by virtue of data dependency. In cloud systems, the dependency normally originates from the resource allocation mechanism (sharing/contention) of co-located virtual machines and the function invocation mechanism of relevant software components in different tiers [Wen+18]. Being subject to the data dependency, perceptible data deviation that is introduced by passing crafted parameters into the service can accordingly appear in other dependent data. Hence, data dependency offers a pragmatic leverage for underpinning security monitoring.

1.3 Research Questions and Contributions

This thesis adopts a problem driven approach to state a set of research questions, for which the investigation processes and corresponding results are reported. The commonality of all the questions is to improve the security assurance of cloud services with the information that is feasible to collect by cloud monitors. To that end, the first research question investigates an exemplar attack for unveiling the characteristics of cloud threats. An effective security approach for mitigating the exemplar has also proposed. The second research question studies the feasibility of developing an indirect mechanism to infer the inaccessible information by leveraging data relations (e.g., dependency) existing between the inaccessible information and the information that can be collected by security monitors. The third research question explores an effective mechanism to ascertain valuable information from the raw monitoring data set for performing indirect cloud monitoring tasks. The last research question discusses the reliability issue of the collected information that is often taken it for grant as the correct input for generating monitoring results.

Research Question 1 *(RQ*1*): How difficult is it to effectively secure cloud services against security threats with existing monitoring solutions?*

Many cloud systems rely on existing monitoring solutions, which are originally developed for addressing traditional security threats (e.g., flood attacks or computer worms) for securing the hosted cloud services. Compared with the traditional threats that often generate evident attack traces, cloud threats that have been observed in recent years expose different characteristics. The recent threats are not only subtly crafted to avoid generating apparent attack traces but also requiring sufficient expertise to identify the existence. However, existing monitoring solutions neglect to consider the scenario where no evident attack traces can be observed and thus incapable of dealing with the novel security threats in the cloud.

Contribution 1 *(C* 1*): Revealing the characteristics of the recent security threats in cloud systems and proposing a novel security mechanism for addressing the threat*

While the existing solutions make use of the collected monitoring information for securing the cloud, they are rarely paying attention to discerning the characteristics of security threats that are vital for developing effective monitoring solutions. To that end, this thesis performs a systematic analysis on a typical cloud attack for revealing the characteristics of cloud threats in Chapter 2. A thorough investigation is conducted on an application layer distributed denial of service attack that targets stealthily compromising victim cloud services. By analyzing the pattern of service requests submitted by remote users (i.e., cloud attackers), it is possible to extract the characteristics of the attack by gaining an in-depth understanding of monitoring information. By leveraging the identified characteristics, this thesis proposes a challenge-response mechanism based security approach to address the threat. The evaluation is implemented on a large scale testing platform where a set of different service scenarios are configured for assessing the efficacy of the proposed approach. The evaluation results demonstrate benign security performance. The contribution has been reported in the publication "Sentry: A novel approach for mitigating application layer DDoS threats." in TrustCom 2016.

Research Question 2 *(RQ* 2*): What is the impediment to improve the security assurance of cloud services when key monitoring information is inaccessible?*

Due to massive technical difficulties (like the virtualization technology) and legal constraints (like the EU GDPR law [PE19]) imposing on cloud systems, it is challenging to access some information that is indispensable for monitoring security problems. Without acquiring the critical monitoring information, it can result in the deployed security mechanisms unable to function normally. However, monitoring information is often subject to data dependencies in many cloud systems. It is possible to develop a workaround to monitor the inaccessible information, while using the dependencies for monitoring inaccessible information in cloud systems is a novel topic that has rarely been discussed.

Contribution 2 *(C* 2*): A reliable schema for implementing indirect cloud monitoring by taking advantage of service dependencies in cloud systems*

This thesis investigates the feasibility of inferring inaccessible information with data dependency and also assessing the reliable degree of the inference results in Chapter 3. Moreover, a **de**pendency-based **Q**uantitative **A**ggregation **M**ethodology termed *deQAM* has been developed for inferring the value of the inaccessible information by parameterizing the dependencies. Besides, a bi-directional quantification mapping model has been introduced to bridge the dependency with the value of monitoring information. For minimizing the uncertainty of the inferred results, an assessment approach has also been proposed for quantifying the reliable degree of the results with the 95% confidence interval. The evaluation is conducted through a case study, by which the evaluation results demonstrate the viability of the proposition in a cloud scenario. The contribution has been reported in the publication "deQAM: A dependency based indirect monitoring approach for cloud services" at SCC 2017.

Research Question 3 *(RQ* 3*): Why is indispensable to ascertain the information that exerts an influence on the inaccessible information for enabling indirect cloud monitoring?*

For protecting cloud services against malicious attacks, some selected information is used as the input by security mechanisms for monitoring the potential attacks. While the input information, in some cases, is evident for monitoring some threats (like using network throughput as the input to monitor DDoS attacks), the input is often obscure to discern for enabling indirect security monitoring on stealthy attacks. As a matter of fact, the influence imposed by the attacks can be implicitly reflected by some correlated information in the monitoring data set. Unfortunately, it is a complex task to ascertain the valuable input of the important security implications for conducting indirect monitoring in the cloud.

Contribution 3 *(C* 3*): A monitoring path identification mechanism for ascertaining the key input information for supporting indirect security monitoring on cloud services*

This thesis investigates the potential for making use of data relation (i.e., dependencies) to identify a set of monitoring data termed "monitoring path" that is used as the input for performing indirect service monitoring in Chapter 4. The thesis first presents a novel reduction technique, which targets removing irrelevant monitoring data regarding the specific security threat, to simplify the identification process. Additionally, this thesis proposes a data ascertaining mechanism to identify a special set of monitoring data that highlighted as the monitoring path for pinpointing security threats by leveraging data relations as well as the attributes of involving data. Moreover, an informative property graph has been presented to underpin large scale fine-grain monitoring path identification in a convenient manner. The evaluation is based on the case studies of two realistic cloud security threats. The results demonstrate the high efficiency for identifying monitoring paths to underpin indirect cloud monitoring tasks. The contribution has been reported in the publication "Flashlight: a novel monitoring path identification schema for securing cloud services" at ARES 2018.

Research Question 4 *(RQ* 4*): How to improve the reliability of indirect monitoring performance that can be undermined by many practical factors in a cloud scenario?*

During conducting indirect security monitoring on cloud services, the information collected by security monitors involves the generation of the final results. In practice, the collected information is frequently suffering from many serious problems such as security monitor malfunctioning, unforeseeable data corruptions, or malicious data tampering. Due to these problems, a reliability concern with respect to the generated results has arisen. A possible way to obtain reliable monitoring results is to improve the reliable degree of the input information used by the indirect monitoring mechanism. However, determining suitable input information for supporting reliably indirect cloud monitoring has not been discussed.

Contribution 4 *(C* 4*): A weighted optimization-based approach to obtain reliable information for improving the overall monitoring performance*

This thesis investigates the feasibility for obtaining reliable monitoring information that can be used by the monitoring mechanism in Chapter 5. For effectively ascertaining the reliable information, the thesis first introduces a technique for cleansing the erroneous data whose value is far-deviated from the ground truth in monitoring data set through analyzing the statistical property of the collected information. Besides, an optimization approach has been proposed for quantifying the inverse relationship between the reliability and the value deviation of a given piece of collected data. Furthermore, a weighted aggregation approach has been developed for determining the reliable monitoring information. The evaluation of the proposed approach has carried out on diverse experimental configurations. The results demonstrate the efficacy of the approach by producing the reliable value for the given information. The contribution has been reported in the publication "Whetstone: Reliable Monitoring of Cloud Services" at SMARTCOMP 2018.

1.4 Publications

The following publications have, in parts verbatim, been included in this thesis.

[Zha+16b] Heng Zhang, Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. "Sentry: A novel approach for mitigating application layer DDoS threats". In: *2016 IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2016, pp. 465–472

[Zha+17] Heng Zhang, Ruben Trapero, Jesus Luna, and Neeraj Suri. "deQAM: A Dependency Based Indirect Monitoring Approach for Cloud Services". In: *2017 IEEE International Conference on Services Computing*. IEEE. 2017, pp. 27–34

[Zha+18a] Heng Zhang, Jesus Luna, Neeraj Suri, and Ruben Trapero. "Flashlight: a novel monitoring path identification schema for securing cloud services". In: *2018 Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM. 2018, pp. 5–14

[Zha+18b] Heng Zhang, Jesus Luna, Ruben Trapero, and Neeraj Suri. "Whetstone: Reliable Monitoring of Cloud Services". In: *2018 IEEE International Conference on Smart Computing*. IEEE. 2018, pp. 115–122

The following publications are related to different aspects covered in this thesis, but have not been included.

[ZMS18] Heng Zhang, Salman Manzoor, and Neeraj Suri. "Monitoring Path Discovery for Supporting Indirect Monitoring of Cloud Services". In: *2018 IEEE International Conference on Cloud Engineering*. IEEE. 2018, pp. 274–277

[MZS18] Salman Manzoor, Heng Zhang, and Neeraj Suri. "Threat Modeling and Analysis for the Cloud Ecosystem". In: *2018 IEEE International Conference on Cloud Engineering*. IEEE. 2018, pp. 278–281

[Alb+17] Soha Alboghdady, Stefan Winter, Ahmed Taha, Heng Zhang, and Neeraj Suri. "C'mon: Monitoring the compliance of cloud services to contracted properties". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM. 2017, p. 36

1.5 Organization

The reminder of this thesis is organized as follows.

Chapter 2 initiates the discussion of the first research question. It presents a novel security approach that leverages the characteristics of the collected monitoring data for unearthing a subtly crafted cloud threat. The evaluation results are presented to demonstrate the efficiency of the proposition that can help to improve the security assurance of cloud services.

Chapter 3 discusses the second research question and proposes a systemic methodology to infer the inaccessible information for realizing indirect monitoring on cloud problems by making use of the data dependency and data quantification models extracted from the collected monitoring data set. This chapter also presents the evaluation of the proposed methodology and its effectiveness.

Chapter 4 proceeds with the discussion of the third research question and studies data dependency at an in-depth level for identifying the non trivial monitoring data to enable the proposed inference process. Apart from that, it proposes an effective solution to ascertain the monitoring data required by the inference process. It also demonstrates the details and results of the evaluation process.

Chapter 5 continues with the discussion of the fourth research question and presents a pragmatic algorithm to improve the reliability of the generated monitoring results by taking advantage of the inverse relation between data error and data reliability. This chapter also specifies the viability of the proposed algorithm which is supported by the obtained evaluation results.

Finally, Chapter 6 concludes this thesis with a succinct summary that highlights the contributions and insights of the thesis.

2 Towards Security Monitoring in Modern Cloud Systems

Security monitoring is an essential approach to secure modern cloud services that are appealing to CSCs with the advocated technical and economic advantages such as transparent resource access, scalability, elasticity and multiple others. Implementing effective service monitoring is a key concern for many CSCs. Therefore, many cloud systems employ existing security mechanisms for performing security monitoring tasks. However, different than the traditional threats that can be managed by the existing monitoring mechanisms, cloud services often suffer from novel security threats that cannot be effectively dealt with by the existing mechanisms. To reveal the characteristics of cloud threats, we conduct a case study on an application-layer distributed denial of service (DDoS) attack that is one of the difficult threats to mitigate in cloud systems. The attack typically interrupts target systems by stealthily draining the available resources and resulting in serious performance degradation and availability problem. With the existing security mechanisms (e.g., intrusion detection/protection), it is impossible to achieve effective monitoring on the evolving application layer DDoS attacks. To address the problem, we perform an in-depth investigation on the security attack and extract the characteristics of the threats. Based on the characteristics, we propose a novel and efficient methodology termed *SENTRY* that specifically aims at securing cloud services against the threat of application-layer DDoS attacks. We propose *SENTRY* as a run time challenge-response based approach. We assess our proposition with different experimental settings. The evaluation results demonstrate that it manages to dynamically adapt to varied service load in the cloud and successfully thwart suspicious service requests from malicious clients.

2.1 Introduction

Distributed denial of service (DDoS) attacks constitute a non-trivial security threat for cloud service providers where the attacks overload the victim server systems to result in degraded services. Most DDoS attacks target the easy-to-attack transport layer (layer 3 of OSI TCP/IP stack) and network layer (layer 4 of OSI TCP/IP stack) of a communications system. The attacks directed at these layers are designed to flood a network interface with attack traffic in order to overwhelm its resources and deny its ability to respond to legitimate traffic.

While network attacks are still a significant challenge due to their scale, the DDoS attacks targeting the application layer may prove to be a more vexing long-term challenge [Aka15]. This challenge arises due to the increasing number and complexity of web applications along with the large network bandwidths of the systems hosting these applications [RKK04] where the attack progressively depletes (versus typical flooding in classical DDoS) the resources from a web or application server. For example, an attack incident occurred at Bitbucket Data Center[1], where the data center was intermittently out of service for over 12 hours [Gle11]. Furthermore, the increasing number of web applications and the shortage of techniques to mitigate DDoS attacks makes them highly attractive targets. Typically, the application layer DDoS attack (a) produces less network traffic than traditional DDoS attack in network channels making their detection hard, (b) causes higher system overhead with the same amount of attacking requests traffic than the traditional DDoS attack in the server side, and (c) displays higher possibility to bypass intrusion and detection systems than the traditional DDoS attack.

In order to address such application layer DDoS attacks, the thesis proposes a novel security mitigation scheme called *SENTRY*. *SENTRY* takes advantage of the remote user's local uplink bandwidth to (a) interactively examine the legitimacy of the request in order to dynamically mitigate the resource flooding caused by the application layer DDoS attacks, and (b) to dynamically restrict resource exhaustion effects. Fundamentally, an uplink bandwidth based challenge-response process is imposed on predefined types of service requests. Overall, the schema for mitigation of application layer DDoS attacks makes the following contributions:

- *SENTRY* works at the middleware/protocol level to alleviate the configuration workload caused by dealing with lower-level network details, and allows add-on production line deployment for cloud service providers.
- *SENTRY* is adaptable to support servers handling varied workload scenarios.
- *SENTRY* aims to defeat the potential dishonest attempts by launching a physical bandwidth based challenge-response process to thwart "smart" adversaries intending to cheat. Consequently, it blocks suspicious service requests from dishonest clients.

[1]A mainstream code-hosting software-as-a-service (SaaS) provider [Atl19].

The evaluation shows that the *SENTRY* can effectively mitigate application layer DDoS attacks in practice as demonstrated with four different use cases.

The reminder of this chapter is organized as follows. Section 2.2 presents the basic characteristics of application layer DDoS attacks. Section 2.3 details the attacker and victim models on which we quantify the performance impact of these attacks. In Section 2.4, we detail the design of the attack mitigation scheme. Section 2.5 presents the experimental evaluation to validate the effectiveness of *SENTRY*. In Section 2.6, we overview related works on mitigating application layer DDoS attacks.

2.2 Background

The application layer DDoS attack is a sophisticated DDoS attack that stealthily depletes the available resources on victim servers. Compared to the traditional networking layer oriented DDoS attacks, the application layer DDoS attacks present three main characteristics as follows.

Firstly, the application layer DDoS attack is a workload-enhancing attack that manifests the denial of service via resource-starving performance degradation where the resources commonly consist of CPU cycles, I/O, physical memory and network bandwidth. Although cloud server systems possess massive system resources, a specific type of resource could still become the bottleneck of the overall system performance in some cases. For instance, while the Amazon cloud service has huge network traffic handling capability, an XML and HTTP protocol based application layer DDoS attack targeting Amazon EC2 resources [VS12] resulted in a complete saturation of the EC2 resources.

Secondly, the application layer DDoS attack is an asymmetric DDoS attack [Ran+06]. The application layer DDoS attack targets very specific application protocols, which entail characteristically high overhead services. The attacker sends a few but selective high overhead service requests to target servers from multiple exploited client hosts resulting in excessive system overhead for the target servers to process them. As a result, the application layer DDoS attack can keep deteriorating the system performance until the target servers are completely out of service.

Thirdly, the application layer DDoS attack is a stealthy type of attack that initiates "normal" service requests that then bypass the "anomalous" behavior focused intrusion detection systems. For example, the authentication service is a necessary application service in many cloud service systems. But it is vulnerable to the masquerading signature attack that consumes considerable system resources to run the verification process[MR04]. By distributing the masquerading service requests across multiple attacking sources, the application layer DDoS attack produces "minor" traffic changes that elude the (high) traffic analysis based intrusion detection systems.

Given such characteristics, the need is to develop a mitigation solution that can block attacking service requests from dishonest clients. To achieve this purpose, the proposed scheme designs a resource based challenge-response scheme for mitigating application layer DDoS attacks. The proposed scheme interactively challenges and validates the service requests from the remote clients in order to block the suspicious attacking requests.

2.3 Models

In this section, we (a) describe the attacker model used for performing the application layer DDoS attacks and (b) present the victim model on which we measure the performance impact of these attacks.

2.3.1 Attack Model

The goal of the attacker is to overwhelm one or more server resources so that the legitimate clients suffer from high service latency and low throughput. This goal can be made by decreasing the quality of the service provided to their clients. Hence, the first step needed for designing an effective mitigation approach is to characterize the potential behavior of the attackers. To this end, one of the possible methods is to identify the high overhead operations associated with the victim services as depicted in Figure 2.2 which shows the system workload state in different cases as follows:

Mode A. This is the normal operational case (without attacks) where a server becomes overloaded while processing a high amount of different user service requests within a short period. These heterogeneous service requests swarm into the server continuously as shown in *Mode A* in Figure 2.1. Different types of service requests present different appearance ratios in the requests flow and cause different processing overhead in the server. The salient observation being that a high-rate of high-overhead services can result in an overload.

Mode B. This is the application layer DDoS attack case where the attackers take advantage of selective high-overhead service requests. The attackers manipulate the targeted client hosts to send high-overhead types of service requests. Although the aggregated number of service requests is not necessarily large, a victim server is overwhelmed, for a specific resource, for processing all the incoming service requests. This *Mode B* represents the workload caused by application layer DDoS attacks in Figure 2.1. Note that the high-overhead types of service requests characteristically appear very often in application layer DDoS attack cases than in normal workload cases.

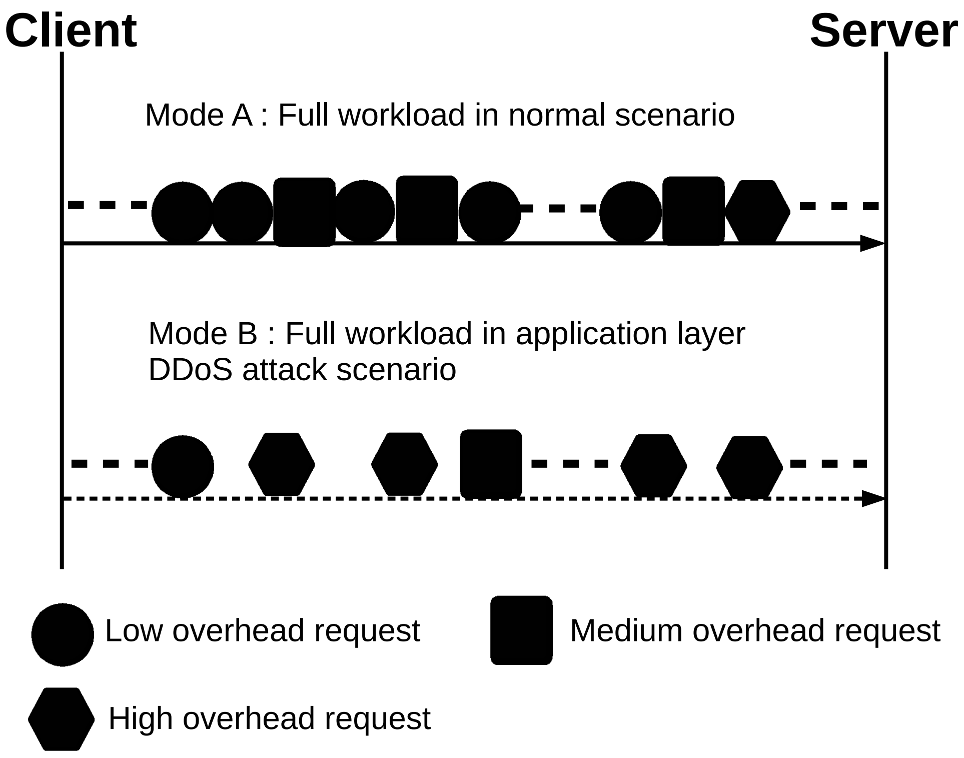

Client
Server
Mode A : Full workload in normal scenario
Mode B : Full workload in application layer DDoS attack scenario
Low overhead request
Medium overhead request
High overhead request

Figure 2.1: The comparison of high system workload scenarios: normal case vs application layer DDoS attack case

2.3.2 Victim Model

Cloud server systems are designed to simultaneously service high volumes of clients requesting varied services. The victim model focuses on those services which are vulnerable to application layer DDoS attacks. In this thesis, we use the example proposed in [Ran+06] as the victim model, which presents the different system overhead caused by processing different types of service requests in an online server system. It can be used to categorize service requests into different classes according to different levels of processing overhead. Based on this example, we consider an online bookstore hosted on multi-tiered architecture as an example of e-commerce application. Figure 2.2 shows the variation in processing times for different service requests in a bookstore application [Ran+06].

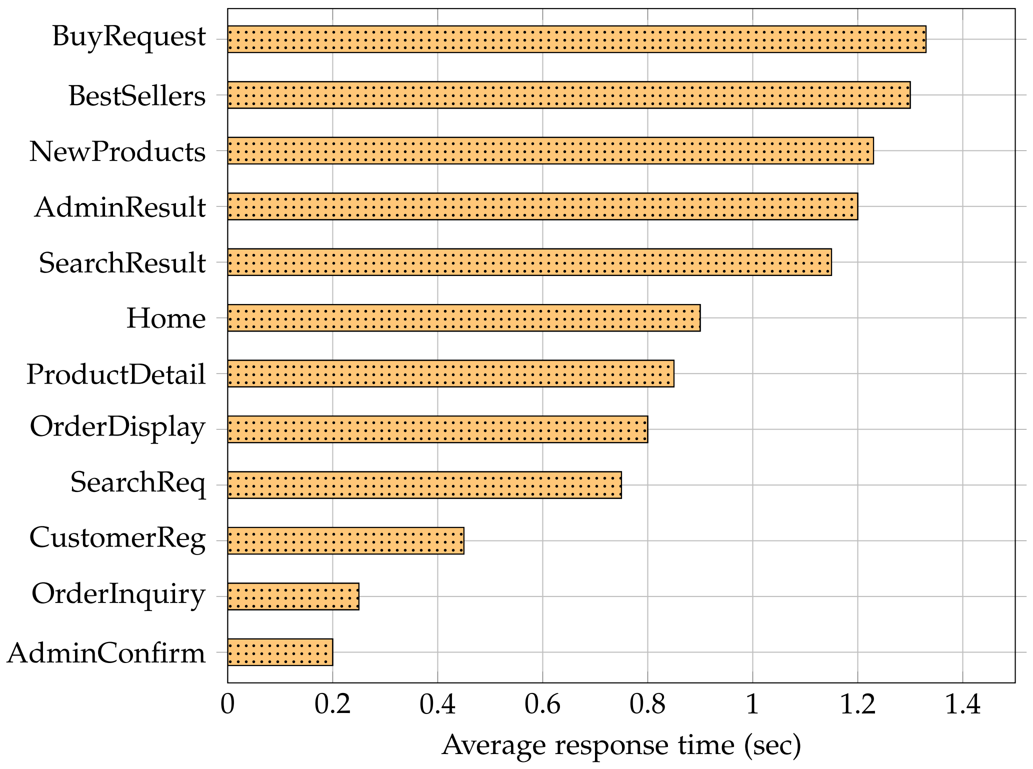


Figure 2.2: Processing times for different dynamic contents requests in an online service [Ran+06]

As shown in Figure 2.2, the “BestSellers” service request causes remarkably higher processing overhead than the “AdminConfirm” service request. The reason for such difference is the different amount of system resources needed to perform these requests. For example, the “BestSellers” request involves high resource demanding operations such as inquiring the database, sorting related results and returning the final result to the user.

In order to facilitate subsequent discussions in this thesis, we make the following assumptions:

I) We assume that the cloud service provider can conduct surveillance on processing the incoming service requests at the server side. This assumption has been put in practice by some cloud service providers. For example, Amazon offers a monitoring product called "Amazon CloudWatch"[Ama15] that can check the AWS resources situation in an approximate real-time mode.

II) We assume that attackers have full control of the exploited hosts including manipulating the local system resources of the hosts.

2.4 Proposed Mitigation Scheme

In this section, we propose a resource based challenge-response scheme for mitigating application layer DDoS attacks. The mitigation scheme (a) actively challenges the request senders validity, and (b) filters out suspicious requests by verifying the responses from the senders. In order to launch an application layer DDoS attack, attacking participants have to send a large number of attacking requests to overwhelm a target server with enough attack strength which refers to the aggregated sending rate of attack requests to a target server per second. Therefore, we assume that attacking participants will make full use of the local bandwidth resources by sending high overhead service requests more frequently than normal users whose service requests present a uniform arrival rate [XY09]. In consequence, such high overhead service requests result in excessive system resources consumption. Thus it is a critical task for a security mitigation solution to minimize the attack strength for reducing the system overhead. Hence it is necessary to identify and discard the high overhead attacking requests from the service request flow by examining the request responses to specially generated challenge messages. We explain the proposed mitigation scheme by first describing the system overview and then detailing the moderator component.

2.4.1 System Overview

The system consists of a cloud client (or remote client), a cloud server and a novel mitigation component called "Moderator" as depicted in Figure 2.3. The Moderator is placed at the server side and is responsible for conducting challenge-response processes against incoming service requests in order to mitigate application layer DDoS attacks.

The mitigation scheme, as depicted in Figure 2.3, comprises the following steps:

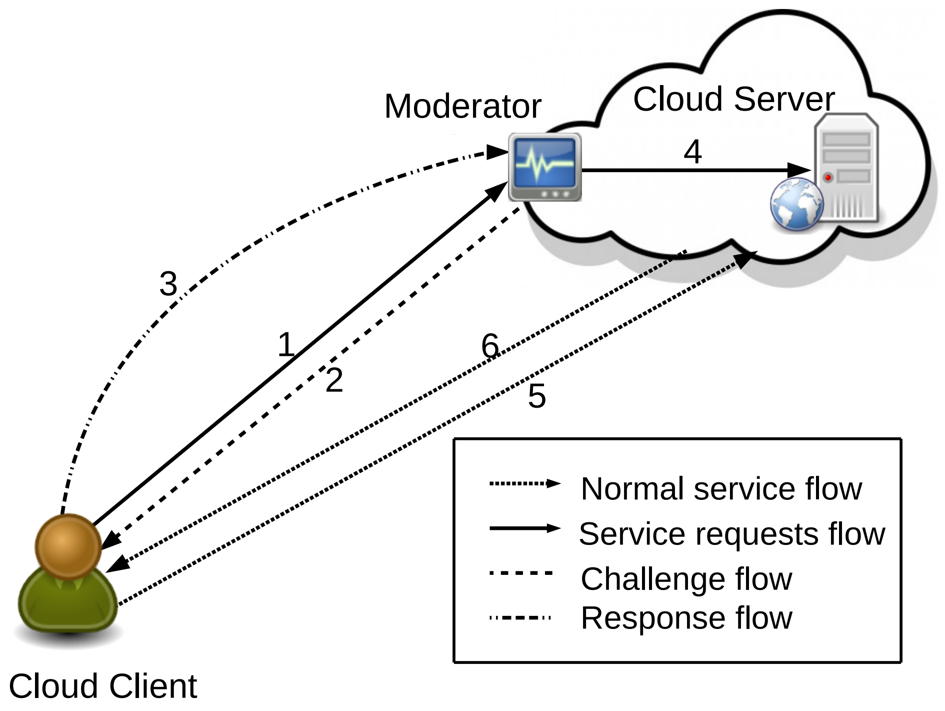


Figure 2.3: An overview of the system framework that consists of the service interactions between a cloud system and remote CSCs

1. **Step 1.** A remote service client sends a high overhead service request flow to the server system
2. **Step 2.** The moderator component samples the incoming service requests. Once the high overhead service request is sampled, a challenge message is issued and sent to the client.
3. **Step 3.** The client responds to the challenge message with local bandwidth resources (This is completely explained in Section 2.4.2).
4. **Step 4.** The client's response received by the moderator is verified to check whether it is valid or not. The moderator will drop that sampled service request if it is invalid. Otherwise, it will forward this service request to the server.
5. **Step 5 & 6.** The requests that are not sampled by the moderator are served normally by the cloud server.

Typically, challenges are data structures obtained by considering parameters from the client, such as CPU cycles, memory or bandwidth resources. In this work, we have chosen the client side physical uplink bandwidth as the base for designing the challenges used in *SENTRY*. The reason for choosing this are:

I. Most network applications offer their services to remote clients using the downlink bandwidth resources[Fra+03] (except for few network applications as peer to peer transmission). This means that using the uplink bandwidth for the proposed mitigation scheme resource causes a limited performance influence on these services.

II. The client bandwidth is strictly managed by his/her local Internet service provider (ISP) and cannot be modified by DDoS attackers. Therefore, client bandwidth resource becomes a good base to design the challenges due to its strong speculation-proof property.

2.4.2 Moderator Description

The moderator component manages the challenge and response processes. It works as an intermediate component for challenging the selected high overhead service requests and verifying the corresponding responses. Each challenge message encloses the expected size of the response to be sent back by the senders. At the same time, it is independent from the server which facilitates the deployment.

The moderator workflow is depicted in Figure 2.4. It consists of several internal modules, namely Probing Module, Challenge Module, Receiving Module, Relay Module and Failure Handling Module.

Probing Module

The probing module (PM) is responsible for sampling the incoming service requests from clients. It works as a flexible sampler with different possible configurations that can be used to adapt the sampling rate. These configurations are adjusted (by the server administrator) by modifying the following two parameters:

- **Sampling target (S_{Target})**: S_{Target} specifies the targeted type of service requests sampled by the PM. In this thesis, we set the high overhead type service requests to be the S_{Target}, as these are the type of requests used in the attacker model (cf., Section 2.3.1). For example, we assume that the service requests of type "BestSellers" specified in the victim model (cf., Section 2.3.2) are the sampling targets from all the other request types specified in the victim

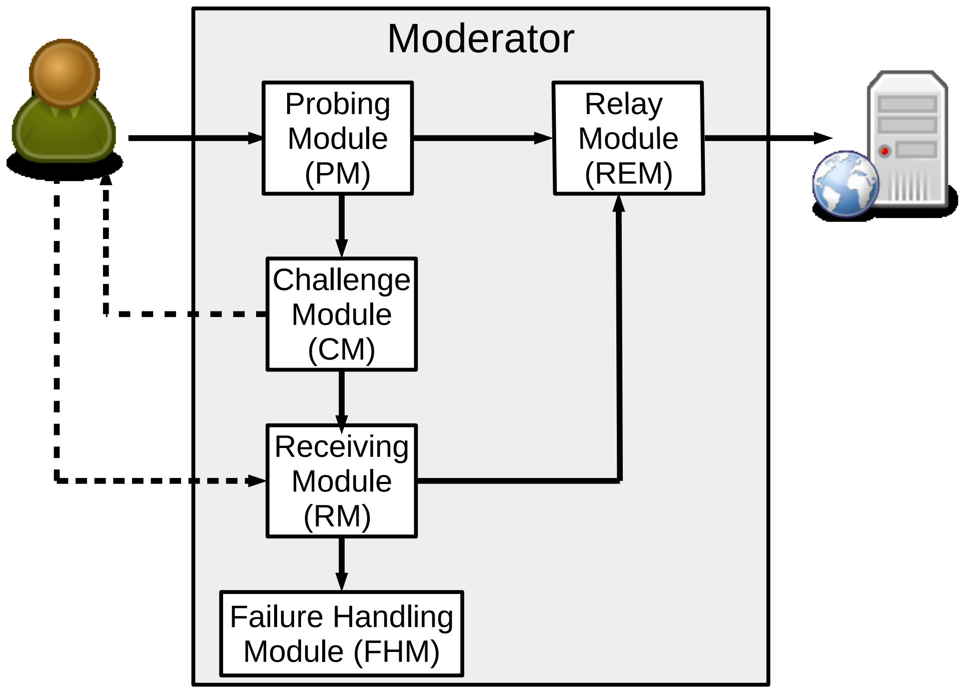


Figure 2.4: Internal design and process diagram of moderator with respect to a complete challenge-response process

model. This means, only "BestSellers" requests are sampled from all types of service requests received by the server.

For example, in Figure 2.5 the session of Client N contains all service requests with different overhead types. The light dark blocks refer to those low overhead service requests submitted by Client N. The medium dark blocks refer to medium overhead service requests from Client N and the deep dark blocks refer to the high overhead ones. The specific configuration of S_{Target} depends on how much system resources will be allocated for the moderator component. The more system resources are available for the moderator, the more types of service request can be added into S_{Target}.

- **Sampling Probability (S_{Prob}):** S_{Prob} specifies the percentage of sampled requests (S_{Prob}). For example, if S_{Prob} is 20% then one out of five target service requests is sampled for subsequent challenge-response process. Thus, according to the previous example 20% of "BestSellers" type requests are sampled and sent to the next module.

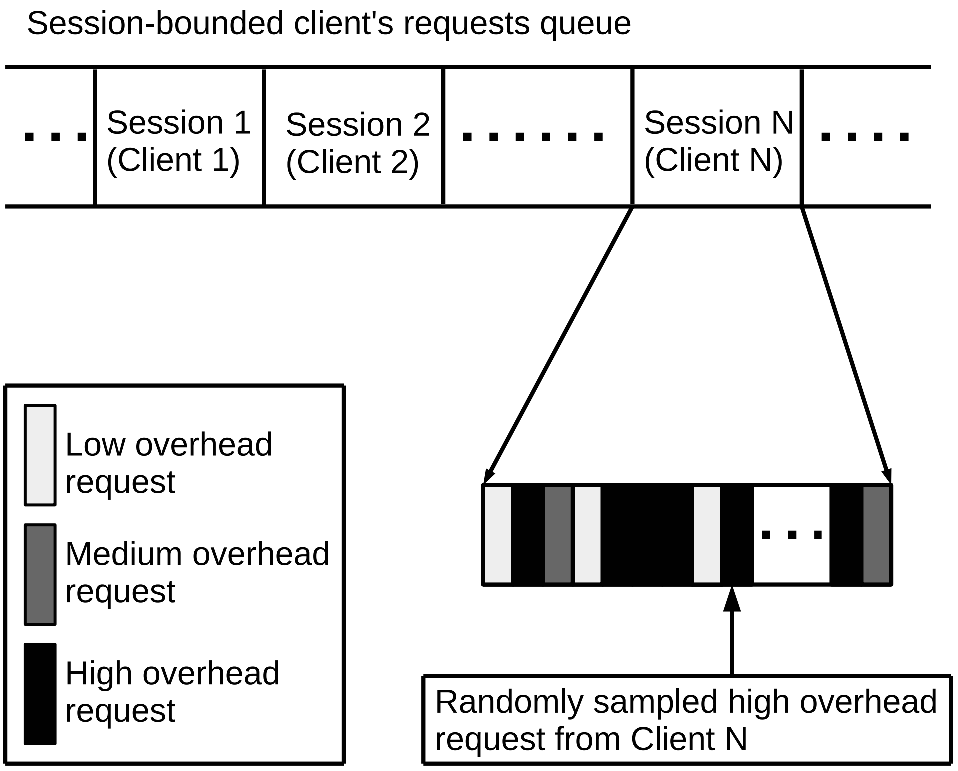


Figure 2.5: A user session-based random service request sampling diagram

Once these parameters are configured, the PM is ready to execute the sampling task for the moderator. At the end of this process, the successful sampled target service request (denoted as *Req*) is sent to the Challenge Module as shown in Figure 2.4.

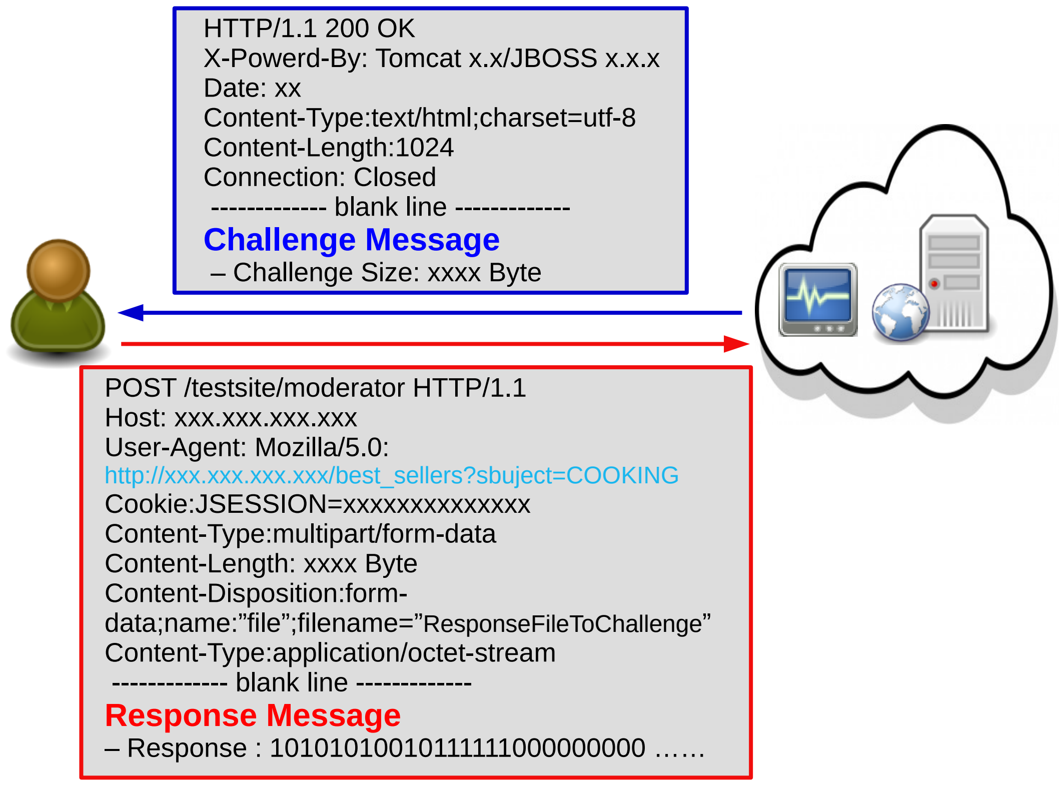
Challenge Module

The challenge module (CM) issues challenge messages for every sampled request received from the PM. The challenge message is a standard HTTP/1.1 response message with the challenge information embedded in the message body. As there are different type of service requests, we introduce a weighted challenge algorithm (based on the algorithms specified in [SW11]) to classify the sampled service request *Req* into three different main groups according to their type. Namely, Group G_{low} contains requests with low overhead. Group G_{medium} contains requests with medium overhead. Group G_{high} contains requests with high overhead.

In this challenge message, the CM asks the client for a specific amount of binary data (specified in the challenge message as shown in Figure 2.6). The client sends a response message containing binary data with the specified size as shown in Figure 2.6. The weighted challenge algorithm generates the challenge size (CZ) according to the type of Req such that:

$$CZ = \begin{cases} \delta, & \text{if } Req \in G_{low} \\ (\alpha+1)\delta, & \text{if } Req \in G_{medium} \\ \delta^{\beta} + (\alpha+1)\delta, & \text{if } Req \in G_{high} \end{cases} \tag{2.1}$$

where α, β and δ are positive integers that can be configured to customize the size of the challenge depending on the Req group (low, medium or high). More details about the algorithm complexity analysis are given in [SW11]. In order to thwart potential guessing attempts from advanced attackers, all these three variables' values are randomly generated within a set of specified ranges. Once CZ is calculated and generated, it is added to the challenge message as depicted in Figure 2.6.


HTTP/1.1 200 OK
X-Powerd-By: Tomcat x.x/JBOSS x.x.x
Date: xx
Content-Type:text/html;charset=utf-8
Content-Length:1024
Connection: Closed
------------- blank line -------------
Challenge Message
– Challenge Size: xxxx Byte

POST /testsite/moderator HTTP/1.1
Host: xxx.xxx.xxx.xxx
User-Agent: Mozilla/5.0:
http://xxx.xxx.xxx.xxx/best_sellers?sbuject=COOKING
Cookie:JSESSION=xxxxxxxxxxxxxx
Content-Type:multipart/form-data
Content-Length: xxxx Byte
Content-Disposition:form-data;name:"file";filename="ResponseFileToChallenge"
Content-Type:application/octet-stream
------------- blank line -------------
Response Message
– Response : 10101010010111111000000000 ……

Figure 2.6: An example challenge-response message pair regarding a given challenge size (CZ)

Receiving Module

The receiving module (RM) receives and validates the response of the challenge from the remote client. Its main responsibility is, firstly, to check the actual size of the challenge response. Then the RM verifies whether the received response size matches with the challenge response size specified by CM. The RM also uses the client's Session ID (denoted by SID_{Req}) to identify the sender of *Req* and forward or discard the client's service request *Req*. More specifically, the RM receives the HTTP request message with a POST method, which is the response to the challenge message issued by the CM. This response message contains the required data submitted by the remote client. The RM retrieves the client's session information SID_{Req} from the message header and checks the size of the binary data in the message body. Then, it compares the retrieved response information from the clients and the issued challenge information from the CM. If the response matches the issued challenge message, *Req* is assumed to be sent from a honest client for correctly making an uplink bandwidth response to specified challenge size. Therefore, RM will route this request *Req* together with its session information SID_{Req} to the Relay Module (REM) for further process. On the other side, if the response mismatches the issued challenge message, *Req* is marked as a suspicious attacking request. Therefore, the RM sends this request *Req* and its session information SID_{Req} to the Failure Handling Module (FHM) as depicted in Figure 2.4.

It is worth highlighting that not all the failures in this challenge-response mitigation process are from attackers. Some legal clients might also occasionally suffer from some transient connection congestion or hardware failures. In this case, it is expected that the clients will resubmit their service requests and make correct responses when challenged again by the moderator. However, attackers can either not make correct responses to the challenge messages or only a limited number of attacking service requests can be processed by server systems if they are "smart enough" to mimic all behaviors of a normal client. For the former case, all attacking requests are filtered out thoroughly. For the latter case, the attacking strength is significantly minimized as only a limited amount of attacking requests get processed at the server and most of the attacking requests are blocked by unsuccessful uplink bandwidth resource responses.

Relay Module

The relay module (REM) acts as moderator's output interface and its main responsibility is to forward the sampled service request *Req* to the server system. Obviously, any non-sampled service requests are directly relayed to server systems by REM as shown in Figure 2.4.

Failure Handling Module

The failure handling module (FHM) is an optional module in the design. Once a sampled service request *Req* failed to make a correct response to corresponding challenge message, it will be handled by the FHM. The FHM is responsible to execute some post-challenge processes, which comprise intrusive IP banning, request redirecting, user information logging and so on.

2.5 Evaluation & Discussion

In this section, we evaluate the performance of the moderator component for several configurations. We discuss the results corresponding to each setting, and subsequently outline the comparisons with contemporary works to illustrate the advantages of the design.

2.5.1 Experiment

SENTRY consists of three elements as specified in Section 2.4.1 (cf., Figure 2.3): a web server, a moderator and a cloud client. The web server is implemented using a Jboss application server and Mysql to offer the database services. We used the victim model shown in Section 2.3.2 as the web server. The moderator consists of a set of developed JSP files deployed on the Jboss application server. The cloud clients are modeled as emulated browsers which are used to emulate human clients' operations by sending different type of service requests to the web server. Emulated browsers send different service requests to the online bookstore application scenario such as searching books, inquiring Best Sellers, registering new accounts, confirming orders and so on. Furthermore, we deploy a group of emulated browsers to act as attack participants in order to frequently submit high overhead attack requests.

In the experiment, we deployed more than 600 concurrent emulated browsers, where 100 of them were specially configured as attacking participants to deliberately

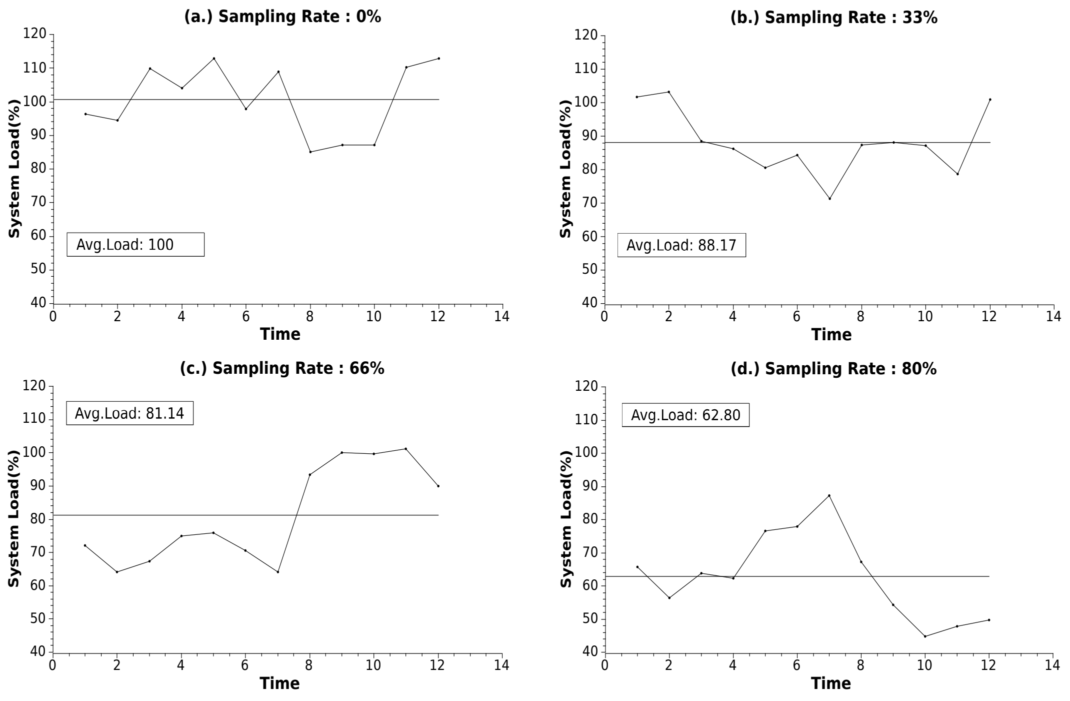


Figure 2.7: System overhead graph with four different sampling rates (SR): $SR = 0$, $SR = 33\%$, $SR = 66\%$, $SR = 80\%$

submit attacking service requests specified by S_{Target}. These attacking service requests took up to 25% amount of the overall service requests. We configured the sampling parameter S_{Target} to be modeled as "BestSellers" type request for referring to high overhead service request (cf., Section 2.3.2), and "Search" type request for medium overhead service request as specified in Equation 2.1. In addition, we also configured S_{Prob} with different sampling rates (0%, 33%, 66%, and 80%) to investigate the behavior of the moderator in different situations. Thus, we measured the system workload at each of the defined four sampling rates. To perform this, we collected 600 seconds of system workload data after the web server entered the stable status. The workload graph of the tests is presented in Figure 2.7. It consists of four sub-graphs for plotting each test. In these sub-graphs, the X-axis is the testing time (in seconds) and each time unit represents 50 seconds. The Y-axis shows the server workload (in percentage).

Attack Scenario with $S_{Prob} = 0$

The first experiment presents the server working at full load with the moderator deactivated. In this case, the overall emulated browsers are connected to the web server to request various kinds of services simultaneously. As shown in Figure 2.7 (a.), the web server's average work load is full. The server received up to 79097 service requests from the emulated browsers. Therefore, the online bookstore server is overloaded and the service is likely suffering a denial of service attack.

Attack Scenario with $S_{Prob} = 33\%$

In the second experiment, the moderator is activated and the sampling rate is configured at 33%, which means 1/3 of submitted search and bestsellers requests are sent to the moderator. As shown in Figure 2.7 (b.), web server's average work load is reduced to 88.17%. The server system received 78787 service requests from emulated browsers, while 5599 service requests failed to get served due to incorrect responses to challenge messages.

Attack Scenario with $S_{Prob} = 66\%$

In the third experiment, the moderator is activated and the sampling rate is configured with 66%, which means that 2/3 of the submitted search and bestsellers requests are sent to the moderator. As shown in Figure 2.7 (c.), web server's average work load decreased to 81.14%. The server system received 79157 service requests from emulated browsers. While, 10518 service requests failed to get served due to their incorrect responses to challenge messages.

Attack Scenario with $S_{Prob} = 80\%$

In the fourth experiment, the moderator is activated and the sampling rate is configured with 80%. As shown in Figure 2.7 (d.), web server's average workload further decreased to 62.80%. The server system received 79281 service requests from emulated browsers. While, 12310 service requests failed to get served due to their incorrect responses to challenge messages.

2.5.2 Discussion

From these experiments, three specific type of data have been collected and presented in Table 2.1, namely mean system load, blocking rate and false negative rate.

The mean system load reflects the performance of the moderator component and the effectiveness of *SENTRY*. As the first row in Table 2.1 shows a monotonic system

Table 2.1: Experimental Results Assessment Table
(N.B. the evaluation value of the sample rate (SR) is respectively assigned as 0, 33%, 66%, and 80%. Meanwhile, the mean system load, blocking rate, and false negative rate are reported.)

Data	**Moderator Sampling Rate**			
	0	33%	66%	80%
Mean System Load	100%	88.17%	81.14%	62.80%
Blocking Rate	0	7.11%	13.29%	15.53%
False Negative Rate	N.A	13.86%	19.47%	22.36%

load decrease from 100% (when moderator is deactivated, $S_{Prob} = 0$) to 88.17% (with $S_{Prob} = 33\%$) and further dropped to 81.14% when the sampling rate raised to 66%. The system load decreases to 62.80%, when S_{Prob} is 80%. From the above data, the application layer DDoS attack threat is successfully mitigated. The reason for the system load drop is that attack participants are controlled by attacking scripts. Those scripts are not able to decode moderator's challenge messages and make correct bandwidth responses accordingly. In case any attacker is "smart enough" to mimic human behavior, the limited client side bandwidth resource imposes rigid restriction on attacking scripts, which can not send too many attacking requests to achieve the service denial purpose. The blocking rate refers to the percentage of blocked attacking requests in the entire service request flow when the moderator is activated. From the second row in Table 2.1, the blocking rate raises from 7.11% (with 33% sampling rate) to 13.29% (with 66% sampling rate) and then to 15.53% (with 80% sampling rate). Considering that 25% of the service flows are attacking requests, these blocking rates can still be maintained at high level. For example, it can still block up to 77.64% attacking requests in the overall attacking service flow even when the sampling rate is equal to 80%. The false negative rate shows the mitigation performance of the moderator component. It refers to the rate of those attacking requests which *SENTRY* failed to block. As in the third row in Table 2.1, the false negative rate increases gradually as long as we increase the sampling rate. It shows a false negative rate of 13.86% with 33% sampling rate and 19.47% with 66% sampling rate and 22.36% with 80% sampling rate. As a result, the higher the sampling rate is, the higher the false negative rate is.

2.6 Related Work

This section presents a survey on the state of art regarding to addressing application layer DDoS attacks.

Many researchers have attempted to mitigate the security threat of application layer DDoS attacks using a variety of techniques. For example, Stavrou et al.[Sta+04] proposed a heterogeneous countermeasure against DoS attacks based on a graphical Turing test which is an enhanced version of the classical CAPTCHA method[Von+03]. Since the classical CAPTCHA method was breached by the work from Mori and Malik[MM03]. This graphical Turing test method consumed a considerable amount of server resources to generate graphical CAPTCHAs for remote users.

Yen and Lee[YL05] introduced a statistical technique to mitigate the threat of random querying based application layer DDoS attacks. A potential problem of their technique is that attacking sources are assumed to generate service requests in Round-robin mode, which mismatches with the real attacking scenarios. While Xie et al.[Xie+13] presented an improved semi-Markov model to mitigate application layer DDoS attack threats by profiling the dynamic access behaviors of aggregated proxy traffic from remote clients. However, the semi-Markov algorithm complexity depends on its parameters which are very challenging to choose.

Seufert and O'Brien[SO07] presented a machine learning based defense mechanism to mitigate the DDoS threat. They collected data from the OSI TCP/IP layer 3, layer 4 and layer 7 from all incoming user requests and employed an artificial neural networks (ANN) to categorize the user types for the isolation purpose. Unfortunately, it is an expensive computational task to run the ANN algorithm especially when the traffic is at a very large scale. In contrast, Yu et al.[Yu+09] developed a light weight application layer DDoS attack mitigation solution by leveraging the trust property to differentiate attackers from the normal user group. The trust is derived from a user's visiting history and encrypted for the storage. While it can be exploited by attackers to trigger a significant amount of workload for computing the trust.

Khor and Nakao[KN11] designed a self-verifying Proof-of-Work (sPoW) mitigation solution against application layer DDoS attack. The sPoW mechanism grants normal users to access different services by solving different difficulty-level puzzles. However, it is incapable of defeating high intelligent adversaries who can pass the puzzle tests.

Furthermore, Wang et al.[Wan+15a] designed a graphical inference model based mitigation solution in Software Defined Network(SDN) which is an emerging network architecture decoupling data plane and control plane in traditional network architectures. However, it is an experimental solution and not feasible to deploy in contemporary network architectures.

Few works addressed the DDoS problem by applying resource-based schemes. The resources are rigidly constrained by physical capacity limitations and difficult to be speculated by malicious attackers. For instance, Abadi et al.[Aba+05] proposed a memory resource based scheme to defeat adversarial clients by equipping a memory-bounded hard functions. While Walfish et al.[Wal+10] proposed a bandwidth resource based scheme to make normal clients get served with more bandwidth resource than abnormal ones by requesting all connected clients to join a bandwidth resource auction. Moreover, Khanna et al.[Kha+12] proposed another bandwidth resource based scheme on the shared communication channels. However, it is a relatively heavy-weight solution for deployments in practice.

This review highlights that existing approaches encounter validity of assumptions or high cost of implementation as briefly summarized in Table 2.2. Since all these contemporary security solutions do not satisfactorily or efficiently mitigate the application layer DDoS attack threat, it forms the premise for *SENTRY*.

2.7 Conclusion

In this chapter, we investigated the threat of application layer DDoS attacks against server systems. In order to mitigate this threat, we propose an uplink bandwidth based challenge-response mitigation scheme (called *SENTRY*) with flexible mitigating capability and strong speculation-proof property. In addition to evaluate the performance of the proposed mitigation scheme, we implemented *SENTRY* in a software component called "Moderator" that is deployed at the server side. The experimental results collected from the evaluation process demonstrate the effectiveness of the proposed challenge-response mitigation mechanism. Therefore, the proposed mitigation scheme can assist servers to thwart the application layer DDoS attacks by reducing unnecessary system overhead which caused by processing the attacking service requests. In future work, we will focus on addressing the tunability of the moderator's sampling parameters to improve performance.

Table 2.2: A comparison table for characterizing existing application layer DDoS attack mitigation schemes from different perspectives

Mitigation Approach	**Characteristics**	**Comments**
Turing test based mitigation scheme [Sta+04]	• Graphic Turing Tests • High execution overhead	• High service latency
Statistic based mitigation scheme [YL05]	• Statistics based • Statistical model dependent	• High false negative rate
Trust based mitigation scheme [Yu+09]	• Trust analysis based • Analyzing browsing behaviors	• False negative rate depending on attacking profile • Huge amount of log files required • Vulnerable to human behavior mimic attacks
Machine learning based mitigation scheme [SO07]	• Machine learning based • Sample collection • Feature extraction algorithm needs training	• High execution overhead
Software Defined Network based mitigation scheme [Wan+15a]	• SDN based • Control network flow with separate planes	• Communication overhead depends on SDN structure
Hidden semi-Markov model based mitigation scheme [Xie+13]	• Statistic based • Legal users get higher service possibility	• Difficulty in model parameter selection
Resource based mitigation scheme [Wal+10]	• Bandwidth resource based • Legal users get higher service possibility	• Client status information required to maintain at the server side

3 Implementing Indirect Service Monitoring Using Dependencies

In Chapter 2, we investigate the characteristics of cloud threats and reveal the challenge for developing effective security mechanisms to address the evolving threats. To manage the security assurance, cloud service monitoring is a critical need for both providers and customers to assess the state of resources and the level of delivery of services. However, existing cloud monitoring methods are often inapplicable in case the targeted service parameters are inaccessible, e.g., CSPs do not allow external access to some service parameters for varied reasons (e.g., proprietary IPR, privacy issues, security concerns or technical access difficulties). As a result, alternate cloud service monitoring approaches are needed to address this issue. In fact, cloud services are not provisioned in isolation and very often share common resources with other services that naturally introduces service dependencies between different services. In this chapter, we propose a novel cloud monitoring approach that targets the dependencies to enable indirect monitoring of inaccessible service parameters by using the information collected from other cloud services. We develop an assessment approach to quantify the reliability of the results generated by the proposed monitoring approach. In the end, we present a case study to evaluate the applicability of our approach.

3.1 Introduction

Cloud service monitoring is an important quality assurance mechanism for the actors involved in the provisioning of cloud services. For example, cloud service monitoring mechanisms are used by Cloud Service Providers to manage cloud services such as getting up-to-date service information, planning maintenance tasks or checking the fulfillment of the service agreements also for cloud customers. Cloud service monitoring mechanisms are also used by third parties, such as by certification auditors which rely on the monitoring information to issue certificates.

Although cloud service monitoring is crucial for the actors involved in the cloud, existing cloud service monitoring mechanisms are effective only if the related parameter information is directly accessible on the provider's side [Lei+12][Men+12] [ML13][FH15][LLL15]. However, many providers do not allow external parties to access specific service parameters for many reasons, such as privacy reasons, security concerns or lack of proper access interfaces. In these cases, the existing monitoring mechanisms become inapplicable. As a result, it is necessary to design a new cloud monitoring mechanism to address this gap.

Given that cloud services often share common resources and work on top of the same platform or infrastructure[1], it is inevitable that dependencies exist between some cloud services. As a result, these service dependencies can be used for developing the new cloud service monitoring mechanism which can monitor those inaccessible parameters of the cloud service by using the parameter information collected from other cloud services.

For example, a cloud secure storage service does not allow external parties to directly monitor parameters related to the *data security level* which is an important aspect of a SecSLA[2]. To help cloud service customers validate the SecSLA compliance, third parties such as security auditors can use service dependencies to indirectly monitor these parameters. In the case of a cloud secure storage service we can suppose that it works as the combination of three different services, namely an authentication service (S_{Auth}), an encryption service (S_{Encypt}), and a storage service (S_{Store}). Therefore, the security auditor can check parameter information from S_{Auth} to see whether the access is from a legitimate user or it can also access to information from S_{Encypt} to see which type of encryption applied on the user data. However, two major challenges need to be solved for designing an indirect monitoring mechanism for cloud services: (1) how to quantitatively use service dependencies for monitoring service parameters, and (2) how to evaluate the reliability of the result from the indirect monitoring.

We address these challenges by proposing a novel indirect monitoring scheme. The proposed approach takes advantage of service dependencies for which the value of service parameters can be quantitatively formulated with the provisioning level of cloud services, and indirectly monitored by aggregating the relevant provisioning levels. To the best of our knowledge, our work is the first attempt to address the indirect monitoring issue in cloud. The contributions are:

[1]An example is the cloud storage service Dropbox is working on top of AWS S3. cf. https://aws.amazon.com/customerapps/1955

[2]SecSLA is a legal agreement negotiated and contracted between the Cloud Service Provider (CSP) and the Cloud Service Customer (CSC) to guarantee cloud services delivered with the specific security assurance.

1. An indirect cloud service monitoring approach termed **de**pendency-based **Q**uantitative **A**ggregation **M**ethodology (*deQAM*) which parameterizes service dependencies to monitor the target parameter's value.
2. A reliability assessment approach which evaluates the confidence level of the monitoring result.
3. A case study to validate the applicability of the proposed monitoring approach.

The reminder of this chapter is organized as follows. Section 3.2 presents some basic considerations for designing the indirect monitoring approach. Section 3.3 elaborates the proposed monitoring approach. Section 3.4 shows the applicability of the approach on a case study for *deQAM*. Section 3.5 discusses the case study results. Section 3.6 presents the related work, and Section 3.7 concludes the chapter.

3.2 Basic Concepts

This section reviews several basic issues related to designing an indirect cloud service monitoring approach.

3.2.1 Cloud Service Provisioning & Indirect Monitoring

The provision of cloud services typically involves multiple supporting services and also multiple service providers where many services or service providers also share common resources or infrastructure. Taking a cloud secure storage service for example, it offers the service of encrypting and storing client's data in cloud using a service hierarchy such as the one depicted in Fig. 3.1.

With the above service provisioning hierarchy, some parameters of the cloud secure storage service could be indirectly monitored by third parties without requiring direct access to these parameters. For example, the *data security level* is an important security parameter for CSCs using this cloud secure storage service (S_{Target}). However, this security parameter is impossible to be directly monitored due to privacy protection concerns. Other parameters can be monitored by third parties from other cloud services residing in the same hierarchy. The possible examples could be the supported *encryption algorithm list* in the cloud encryption service (S_{Encypt}), *authentication notification* messages in the cloud authentication service (S_{Auth}) or the *bug report of the security module* in Microsoft Azure (MSA).

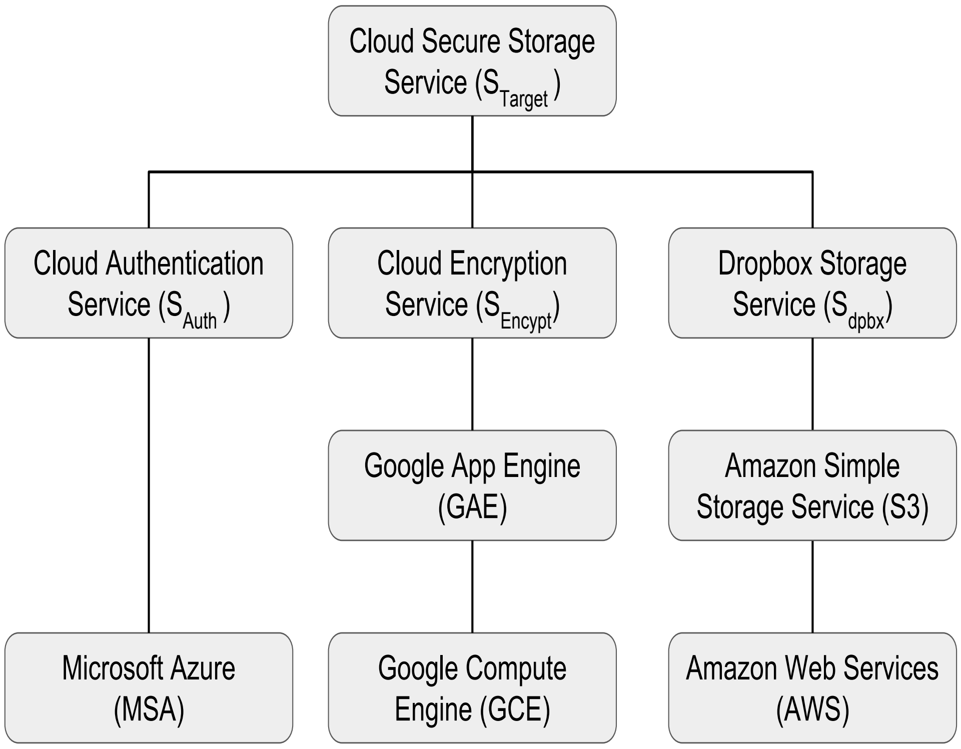

Cloud Secure Storage Service (S_{Target})
Cloud Authentication Service (S_{Auth})
Cloud Encryption Service (S_{Encypt})
Dropbox Storage Service (S_{dpbx})
Google App Engine (GAE)
Amazon Simple Storage Service (S3)
Microsoft Azure (MSA)
Google Compute Engine (GCE)
Amazon Web Services (AWS)

Figure 3.1: The provisioning hierarchy of an example secure storage service in the cloud

For example, by studying the related parameters collected from other cloud services in Fig. 3.1, it is possible for third parties to indirectly monitor the parameter *data security level*.

3.2.2 Service Dependency & Characterization

Service dependency is the directed relation between different services, which refers to one service (a.k.a dependent service) subjected to the persistent constrains from another or multiple services (a.k.a. antecedent service) [Win+09]. Service dependency specifies the provisioning of services that depends on its antecedent services, which might affect the provisioning of the dependent service (e.g., degradations or disruptions or failures) [Zho+07]. For the aforementioned example, the provisioning of the cloud secure storage service (dependent service) depends on its antecedent services, namely the cloud authentication service and the cloud encryption service in order to meet the security requirements of the storage service. Therefore, the service dependency has deep influence on the service provisioning of cloud services.

From the monitoring perspective, the following characteristics of service dependency are worth highlighting:

Direction: The dependency direction states the directional information that identifies the dependent and the antecedent in the service dependency. It can assist in trimming the service dependencies by removing the irrelevant services.

Type: The dependency type reflects antecedent services imposing different types of influence on dependent services. The influence is either the decisive type (enable/disable) or the indecisive type (altering to some extent), affecting the dependent service provisioning.

Strength: The dependency strength represents the degree of dependent services relying on their antecedent services. The higher dependency strength implies the closer service provisioning of dependent services relying on their antecedent services.

3.2.3 Uncertainty & Data Estimation

Uncertainty is a key issue in designing an indirect monitoring approach. Uncertainty, which is caused by the incompleteness of knowledge about the monitoring target, deviates the value of the monitoring result from the real value [DM00].

However, the indirect monitoring approach takes the parameter information from antecedent services without the complete knowledge of the target parameter in dependent services. Therefore, it is nontrivial to address the uncertainty problem for the indirect monitoring approach. In practice, a common solution to address the uncertainty issue is to determine a confidence level (typically 95%) for evaluating the reliability of the monitoring result.

3.3 The Proposed Methodology

This section presents the details of the dependency-based Quantitative Aggregation Methodology (*deQAM*) as the proposed indirect cloud monitoring approach.

3.3.1 System Overview

The dependency-based quantitative aggregation methodology (*deQAM*) is the indirect monitoring approach proposed to monitor the parameters of cloud services. *deQAM* uses the related parameter information collected from antecedent services to monitor the dependent service parameter which is difficult to be monitored directly. *deQAM* adopts a multi-step process to achieve the indirect monitoring as shown in the bottom boxed part of Fig. 3.2. Briefly, the "utility mapping" step converts the parameter value to a utility level as explained in Section 3.3.2. The "dependency parameterization" step generates the trimmed parameterized service graph with service dependencies. The "monitoring result inference" step computes the utility of the target parameter. The "monitoring result generation" step generates the indirect monitoring result by mapping the target parameter utility back to the parameter value. The "result reliability evaluation" step addresses the reliability issue by estimating the monitoring result's reliability at a 95% confidence level. These steps are detailed in Section 3.3.3.

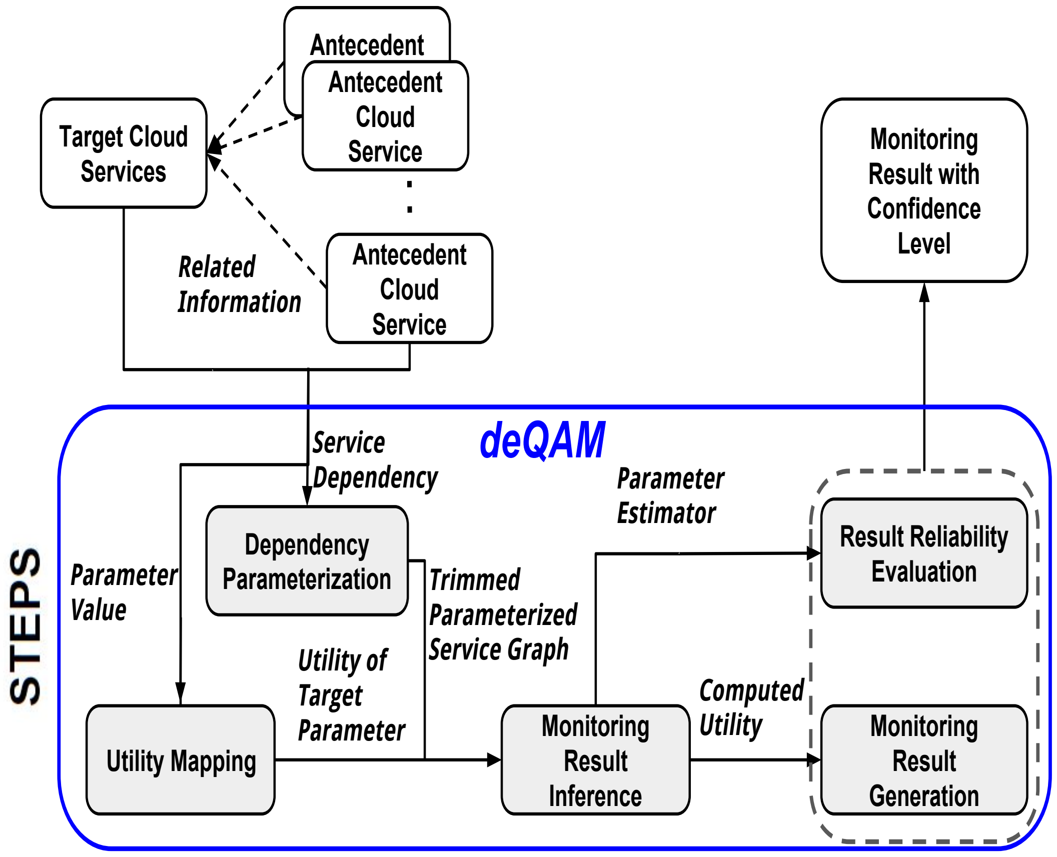

Antecedent
Antecedent Cloud Service
Target Cloud Services
Monitoring Result with Confidence Level
Related Information
Antecedent Cloud Service
Service Dependency
deQAM
Parameter Estimator
STEPS
Parameter Value
Dependency Parameterization
Trimmed Parameterized Service Graph
Result Reliability Evaluation
Utility of Target Parameter
Computed Utility
Utility Mapping
Monitoring Result Inference
Monitoring Result Generation

Figure 3.2: The main framework of the proposed indirect monitoring methodology (*deQAM*)

3.3.2 Transformation of Dependency

deQAM treats the service dependency as quantitative constraints on the cloud service provisioning. It uses *utility* as the quantitative unit for representing the influence that the service parameter value has on the service provisioning. *deQAM* categorizes the utility into two classes.

Eigen utility: Eigen utility represents the basic provisioning level of the dependent service affected by the value of other parameters in the same service. If the eigen utility is exceptional (i.e. lower than some thresholds), a serious problem might have occurred with the service provisioning, such as service dysfunctional, interrupted or completely down.

Contributed utility: Contributed utility represents the dependent service's provisioning level affected by the value of parameters of antecedent services. According to the different types of influence, it is further categorized into two types, namely the *mandatory-type contributed utility* and the *optional-type contributed utility*. Specifically, the mandatory-type contributed utility represents the decisive influence on the provisioning of the dependent service which decides whether the dependent service is provisioned or not. The optional-type contributed utility represents the indecisive influence affected on the provisioning level of the dependent service which increases the service provisioning level based on the basic provisioning level.

As a result, the utility of the service parameter value is the sum of the eigen utility and the contributed utility.

3.3.3 Concrete Design

deQAM is a multi-step indirect cloud service monitoring approach that quantitatively analyzes the utility as follows.

Step 1. Utility Mapping

deQAM starts the monitoring process by mapping the parameter value V_p, collected from antecedent services, onto utility U_p. Based on the discussion in Section 3.3.2, we propose two different conversion rules (depicted as $f_{\_B}$ for *Boolean* and $f_{\_N}$ for *Numerical*) for mapping the parameter value V_p onto utility U_p and the inverse conversion rules (depicted as $f_{\_B}^{-1}$ and $f_{\_N}^{-1}$) for the back-mapping process.

$$f_{\_B} : V_p \mapsto U_p = \begin{cases} 1 & V_p \in S_{en} \\ 0 & V_p \in S_{dis} \end{cases}$$

$$f_{\_B}^{-1} : U_p \mapsto V_p = \begin{cases} V_p \in S_{en} & U_p = 1 \\ V_p \in S_{dis} & U_p = 0 \end{cases}$$

The parameter value V_p having the dominant influence on the dependent service's provisioning is mapped to the *boolean* value by using $f_{\_B}$, in which S_{en} is the set of the value for enabling the service provisioning (e.g., *yes, enable or activate*) and S_{dis} is the set of the value for disabling the service provisioning (e.g., *no, disable or inactive*). For example, the parameter value of *login failure message* is $V_1 = true/false$, its utility is mapped onto $U_1 = 0/1$ by using $f_{\_B}$ when the service provisioning is *disabled/enabled*.

$$f_{\_N} : V_p \mapsto U_p = \frac{V_p - V_{p}\_min}{V_{p}\_max - V_{p}\_min}$$

$$f_{\_N}^{-1} : U_p \mapsto V_p = \frac{U_p - U_{p}\_min}{U_{p}\_max - U_{p}\_min}$$

The parameter value V_p having the influence of increasing the provisioning level of the dependent service is mapped to the *numerical* value in the range of $[0, 1]$ by using $f_{\_N}$, in which V_{p}_max and V_{p}_min are the upper and the lower boundary of V_p's varying range. For example, the parameter value of *CPU usage rate* varies in the range of $[0, 100\%]$. If the collected *CPU usage rate* is $V_2 = 62\%$, then its utility U_2 is mapped onto 0.62 by using $f_{\_N}$. In a similar way, the parameter value of *client storage quota* is an element of the set $\{250GB, 500GB, 750GB, 1000GB\}$ and can be regarded as varying in the range of $[0, 1]$ which is divided into four parts. Then the utility of corresponding elements are mapped onto 0.25, 0.5, 0.75, 1 by using $f_{\_N}$.

Step 2. Service Dependency Parameterization

$deQAM$ characterizes service dependencies with the parameterized service dependency graph which consists of a set of vertexes representing cloud services and a set of edges representing parameterized dependencies between different cloud services. The parameterized service dependency graph is defined as:

Definition. A parameterized service dependency graph is a multi-parameter directed graph denoted with $G = (V, E)$, in which:

• V are the vertexes of Graph G which represents a finite set of cloud services denoted by $V = \{S_1, S_2, ..., S_i\}$. S_i means the *i-th* service.

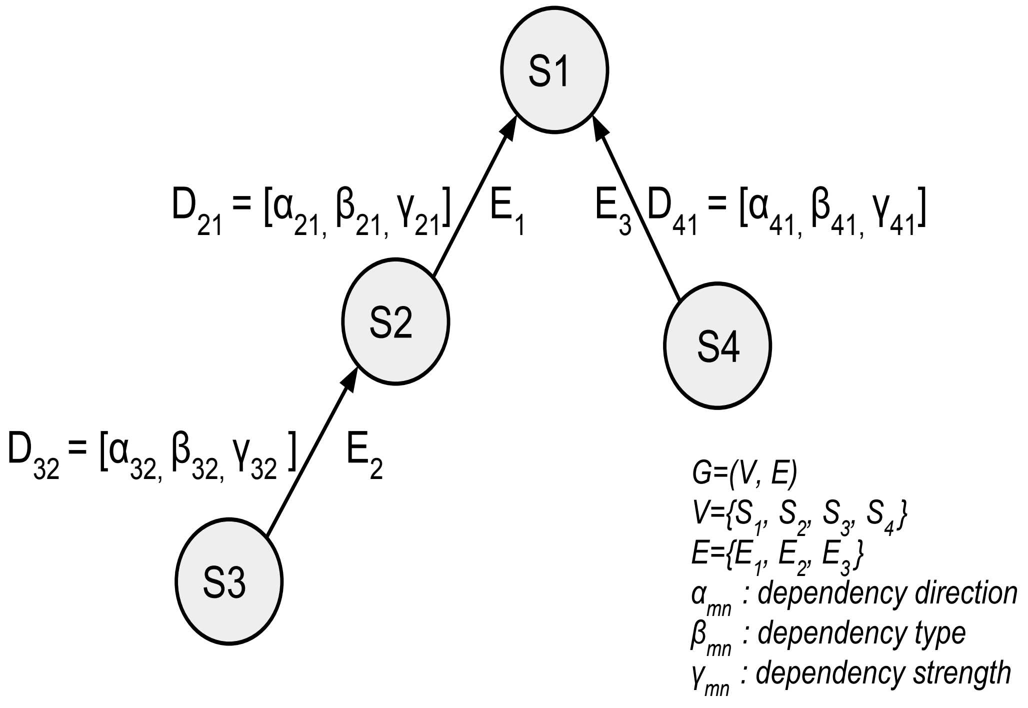

S1
S2
S3
S4
$D_{21} = [\alpha_{21}, \beta_{21}, \gamma_{21}]$
E_1
E_3
$D_{41} = [\alpha_{41}, \beta_{41}, \gamma_{41}]$
$D_{32} = [\alpha_{32}, \beta_{32}, \gamma_{32}]$
E_2
G=(V, E)
V={S_1, S_2, S_3, S_4}
E={E_1, E_2, E_3}
α_{mn} : dependency direction
β_{mn} : dependency type
γ_{mn} : dependency strength

Figure 3.3: An example service dependency graph where the dependency are parameterized as a three-parameter vector

• E are the directed edges of Graph G which represents a finite set of dependencies between different services denoted by $E = \{E_1, E_2, ..., E_j\}$, E_j means the *j-th* service dependency between two cloud services. E_j is characterized with a three-parameter vector $\vec{D_{mn}} = [\alpha_{mn}, \beta_{mn}, \gamma_{mn}]$ to parameterize service dependencies obtained from the service provisioning hierarchy as illustrated in Section 3.2.2.

α_{mn}**:** It is the parameter specifying the target-oriented dependency direction between two different cloud services. Specifically, if the antecedent service is S_m and the dependent service is S_n ($S_m, S_n \in V$) and both services are directly or indirectly affect a given target service, its value is 1. Otherwise, if S_m and S_n does not affect the given target service, the value of dependency direction is -1.

β_{mn}**:** It is the parameter representing the different types of influence of the S_m's parameter value on the S_n's service provisioning. Its value is assigned to 0 to indicate the decisive influence dependency, while its value is assigned to 1 to indicate the indecisive influence dependency.

γ_{mn}: It is the parameter defining the rate that the utility mapped from S_m's parameter value contributes onto S_n's service provisioning level caused by the service dependency. It is a real number in the range of $[0, 1]$.

Based on the parameterized service dependency graph G, *deQAM* adopts a target-centric service dependency trimming method which examines dependencies with regard to the target service (S_{Target}) and recursively checks the corresponding antecedent services to remove irrelevant service dependencies (i.e. service dependencies do not affect the monitoring result) as specified in Algorithm 1. The trimming algorithm takes advantage of a set of services (denoted by S_x) and a set of corresponding parameterized dependency vectors (denoted by E_y) derived from these services regarding to the target service. By recursively comparing the dependency direction α_{xt} of every service in S_x, the trimming algorithm can finally generate the minimized service dependency graph (denoted by $G^{'}$) with regard to the target service. In addition, the trimming algorithm traverses all nodes (services) of the graph to do the dependency direction (α_{xt}) comparison. Therefore, the complexity for the trimming algorithm is $\mathcal{O}(n)$, where n is the node number.

Algorithm 1: Target-Centric Service Dependency Trimming Algorithm

```
Require: parameterized service dependency graph $G = (V, E)$ of $i$ vertexes and $j$ edges.
Ensure: trimmed parameterized service dependency graph $G^{'} = (V^{'}, E^{'})$.
 1: set $S_x \in V = \{S_1, S_2, ..., S_i\}, E_y \in E = \{E_1, E_2, ..., E_j\}$;
 2: set $V^{'} = \{S_{Target}\}$ and $E^{'}$;
 3: while before the element number of $V^{'}$ no longer changes do
 4:   for each element $S^{'} \in V^{'}$ do
 5:     set $S_t = S^{'}$;
 6:     for each element $S_x \in V \bigcap S_x \neq S_t$ do
 7:       if $E_y \in E \bigcap \alpha_{xt} == 1$ then
 8:         add $S_x$ and $E_y$'s dependency $\vec{D_{xt}}$ into $V^{'}$ and $E^{'}$ respectively;
 9:       end if
10:     end for
11:   end for
12: end while
13: return Trimmed parameterized service dependency graph $G^{'} = (V^{'}, E^{'})$
```

Step 3. Monitoring Result Inference

Based on the trimmed parameterized service dependency graph G', utility U_φ of the inaccessible parameter φ of the target service S_{Target} is inferred by using parameterized dependencies. The utility computation process consists of three different parts as follows:

Eigen utility: The eigen utility U_{E_φ} of the inaccessible parameter φ of S_{Target} is decided by the baseline of the service provisioning. It is the threshold of the service provisioning without considering any dependency from antecedent services.

Mandatory-type contributed utility: The mandatory-type contributed utility U_{M_φ} of the inaccessible parameter φ of S_{Target} is computed by using the dependencies which impose the dominant influence on the provisioning of S_{Target}. Due to the dominant influence of the dependency type, the mandatory-type contributed utility U_{M_φ} is the product of all boolean utilities mapped from the antecedent services' parameter value with decisive influence on the service provisioning of S_{Target} (i.e. when $\beta_{mn} = 0$). It can be computed as follows:

$$U_{M\_\varphi} = \prod_{1}^{l} U_l \tag{3.1}$$

In Equation (3.1), l is the number of service dependencies with $\beta_{mn} = 0$ in $G^{'}$ and U_l is the utility mapped from the antecedent service S_l' parameter value V_l.

Optional-type contributed utility: The optional-type contributed utility U_{O_φ} of the inaccessible parameter φ of S_{Target} aggregates the utility computed by all optional-type service dependencies. However, it requires to compute the utility caused by the ripple-effect of service dependency which is the phenomenon of the influence (either quantitative or qualitative) propagation from antecedent services to dependent services [Koz11]. Therefore, the optional-type utility computation needs to include both the direct and the ripple-effect optional-type contributed utility computation. As a result, *deQAM* computes the optional-type contributed utility U_{O_φ} as:

• *Ripple-effect optional-type contributed utility:* The ripple-effect optional-type contributed utility $U_{mn}^{(o.r)}$ is the optional utility indirectly contributed by the parameter of antecedent service S_m to the parameter of dependent service S_n. Given a three-node dependency chain in the trimmed parameterized dependency graph $G^{'}$, it can be represented as $\{S_m \rightarrow S_k \rightarrow S_n\}$. This dependency chain consists of an antecedent service S_m, an intermediate service S_k and a dependent service S_n. The ripple-effect utility $U_{mn}^{(o.r)}$ propagating from S_m to S_n is computed as:

$$U_{mn}^{(o.r)} = (\gamma_{mk} \cdot U_m) \cdot \gamma_{kn} \tag{3.2}$$

In Equation (3.2), $\gamma_{mk} \cdot U_m$ is the optional utility mapped from S_m's parameter value V_m and contributed to S_k. Multiplying the coefficient γ_{kn}, it results in the final optional utility contributed to the inaccessible parameter's utility of S_n.

• *Direct optional-type contributed utility:* Based on the ripple-effect optional-type utility computation method, the direct optional-type contributed utility $U_{mn}^{(o.d)}$ is computed as the simplified form, in which the dependency chain contains only two nodes as $\{S_m \to S_n\}$. Accordingly, the direct optional-type contributed utility is computed as:

$$U_{mn}^{(o.d)} = \gamma_{mn} \cdot U_m \tag{3.3}$$

The output of Equation (3.3) is the optional utility mapped from the parameter value of S_m and contributed to the inaccessible parameter's utility of S_n.

Based on Equation (3.2) and (3.3), the optional-type contributed utility U_{O_φ} of the inaccessible parameter φ of S_{Target} is aggregated as:

$$U_{O\_\varphi} = \sum_{1}^{p} \sum_{1}^{q} U_{mn}^{(o.r)} + \sum_{1}^{r} U_{mn}^{(o.d)} \tag{3.4}$$

In Equation (3.4), p is the number of dependency chains related to S_{Target}, q is the number of all non-adjacent antecedent services of S_{Target} in each dependency chain, and r is the number of direct antecedent services of S_{Target}.

Inaccessible service parameter utility inference: As a result, utility U_φ of the inaccessible parameter φ of S_{Target} is computed with the results from the three different parts as:

$$U_\varphi = U_{M\_\varphi} \cdot (U_{E\_\varphi} + U_{O\_\varphi}) \tag{3.5}$$

In Equation (3.5), the mandatory-type contributed utility U_{M_φ} is the dominance over the utility U_φ of the inaccessible parameter φ of S_{Target} like the switch.

Step 4. Monitoring Result Generation

The monitoring result generation step maps the derived utility U_φ back to the inaccessible parameter's value V_φ. To do so, we apply the inverse conversion rules $f_{\_B}^{-1}$ and $f_{\_N}^{-1}$ in **Step 1** for the back-mapping process. However, the special attention is required to manage the case of mapping the parameter value V_φ back to a given set, because the computed utility may not exactly match any element in the set. Therefore, we propose to assign the derived utility U_φ to the nearest utility mapped from some element value in the set. For example, if the derived utility U_φ is 0.32 which does not match any utility in the set like $\{0.25, 0.5, 0.75, 1\}$. However, its two adjacent elements are 0.25 and 0.5. As a result, utility U_φ is replaced by 0.25 which has the minimum deviation to 0.32 than others.

Step 5. Result Reliability Evaluation

Because the monitoring result is inferred by *deQAM* indirectly, it suffers from uncertainty which has to be estimated. Therefore, it is necessary to address the uncertainty issue by evaluating the reliability of the generated monitoring result V_φ of the inaccessible parameter φ of S_{Target}.

The *deQAM*'s inference process relies on the service dependency parameters α_{mn}, β_{mn}, and γ_{mn} from **Step 2**. As the value of γ_{mn} is generally derived by studying the empirical experimental data or the knowledge of the experts [ZN08], the incomplete knowledge of γ_{mn} is the source of uncertainty for the indirect monitoring result V_φ. Therefore, we estimate γ_{mn} based on a statistic evaluation approach to assess the confidence level of the generated monitoring result V_φ.

The evaluation approach is designed with the assumption that γ_{mn} is derived with the best knowledge of the experts or the best-effort study on empirical experimental data. In other words, the value of γ_{mn} applied in *deQAM*'s utility computing process is the most likely approximation of the real strength of the dependency. Consequently, uncertainty ξ_{mn} is defined as the deviation between the approximation value and the true value of γ_{mn}. As a result, the distribution of uncertainty can reflect the reliability of γ_{mn}. Specifically, the variance of the uncertainty distribution σ_{mn} is inversely proportional to the reliability of γ_{mn}. Apart from the variance σ_{mn}, the mean of the uncertainty distribution μ_{mn} is set to 0 due to no intentional bias introduced for deriving γ_{mn} according to the assumption. Meanwhile, the Gaussian distribution is commonly used for modeling the uncertainty problem. Therefore, the distribution of γ_{mn}'s uncertainty follows the Gaussian distribution $N(0, \sigma_{mn})$. As the value of γ_{mn} is independently obtained, the uncertainty $\xi(V_\varphi)$ of the generated monitoring result V_φ is the aggregation of all γ_{mn}'s uncertainty. Thus, $\xi(V_\varphi)$ follows the Gaussian distribution as:

$$\xi(V_\varphi) \sim N(0, \sum \sigma_{mn}) \tag{3.6}$$

However the true value of σ_{mn} is generally unknown. Hence, we adopt the mean of the square products of the uncertainty differences to estimate the true value of $\hat{\sigma}^2_{\xi(V_\varphi)}$ obtained from Equation (3.6) as:

$$\hat{\sigma}^2_{\xi(V_\varphi)} = \frac{1}{t}\sum_{1}^{t}(\gamma_{mn} - \gamma_{mean})^2 \tag{3.7}$$

In Equation (3.7), γ_{mean} is the mean value of all γ_{mn} participated in the utility computation process. While t is the total number of γ_{mn} participated in the utility computation process. Accordingly, a χ^2-distribution can be obtained based on Equation (3.7) as:

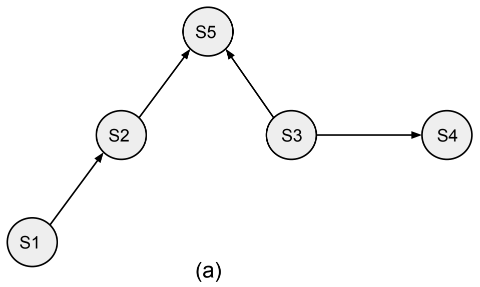
$$\chi^2(t) \sim \frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\sigma^2_{\xi(V_\varphi)}} = \frac{t \cdot \hat{\sigma}^2_{\xi(V_\varphi)}}{\sigma^2_{\xi(V_\varphi)}} \tag{3.8}$$

Therefore, the χ^2-distribution statistic characteristic can be used to determine the confidence level $(1 - c)$ to assess the reliability of the generated monitoring result V_φ [Li+14a]. Based on the distribution (3.8), the confidence level $1 - c$ of $\sigma^2_{\xi(V_\varphi)}$ is derived to represent the generated monitoring result's reliability [Sch12] as:

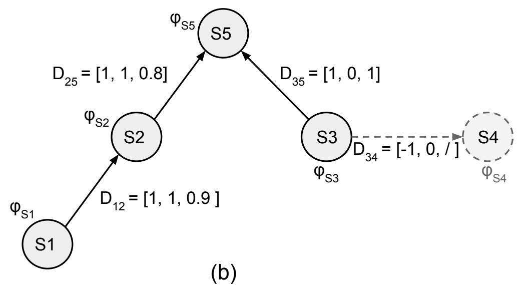
$$\left(\frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(1-\frac{c}{2},t)}}, \frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(\frac{c}{2},t)}}\right) \tag{3.9}$$

3.4 Case Study

deQAM is applied to an example cloud service to infer the inaccessible service parameter as an initial effort to validate the proposed approach. To the best of our knowledge, it is the first study for conducting indirect monitoring on the inaccessible parameter of the cloud service.



(a) The raw service provisioning hierarchy for cloud service $S5$



(b) The parameterized service dependency graph for cloud service $S5$

Figure 3.4a depicts the service provisioning hierarchy of a cloud service with other services. Without loss of generality, we assign the parameters φ_{s_1} and φ_{s_2} of services $S1$ and $S2$ (respectively) holding the optional-type dependencies indirectly and directly to the parameter φ_{s_5} of the service $S5$, the parameter φ_{s_3} of the service $S3$ holding the mandatory-type dependencies to the parameter φ_{s_4} of the service $S4$ and the parameter φ_{s_5} of the service $S5$. The target monitoring parameter φ_{s_5}, with the possible value varying range of $[0, 100\%]$, is unable to be monitored directly. However, the other information of the parameter from the other services can be collected as in Table 3.1. Then, *deQAM* conducts the indirect monitoring process as follows:

Table 3.1: Case study: information collected for performing indirect parameter monitoring (N.B. five different services are considered in the evaluation, while the monitoring information and dependency are parameterized.)

Parameter	Dependent cloud services				
Location	Service 1	Service 2	Service 3	Service 4	Service 5
Name	φ_{s_1}	φ_{s_2}	φ_{s_3}	φ_{s_4}	φ_{s_5}(Target)
Value Type	Numerical	Numerical	Boolean	Boolean	Numerical
Variation Range	[0,100%]	Lv 1/2/3/4/5	(in)activated	(un)available	[0,100%]
Collected Value	30%	Lv 1	activated	available	N.A
Trimming Status	Keep	Keep	Keep	Removed	Keep
DepDirection (α_{mn})	$\alpha_{12} = 1$	$\alpha_{25} = 1$	$\alpha_{35} = 1$	Removed	N.A
DepType (β_{mn})	$\beta_{12} = 1$	$\beta_{25} = 1$	$\beta_{35} = 0$	Removed	N.A
DepStrength (γ_{mn})	$\gamma_{12} = 0.9$	$\gamma_{25} = 0.8$	$\gamma_{35} = 1$	Removed	N.A

• ***Step 1. Utility Mapping*** To monitor parameter φ_{s_5}, *deQAM* starts by mapping the value of the parameters collected from the antecedent services onto utility. By using the proposed conversion rules f_B and f_N, the utility of the value of the collected parameters ($\varphi_{s_1}, \varphi_{s_2}, \varphi_{s_3}$) are converted as:

$$U_{\varphi_{s_1}} = \frac{30\%}{100\% - 0} = 0.3$$

$$U_{\varphi_{s_2}} = \frac{Lv1}{\{Lv1, Lv2, Lv3, Lv4, Lv5\}} = \frac{0.2}{1.0 - 0} = 0.2$$

$$U_{\varphi_{s_3}} = 1$$

It is worth noticing that φ_{s_3} is a Boolean and $V_{\varphi_{s_3}}$ is *activated*.

• ***Step 2. Service Dependency Parameterization*** *deQAM* constructs the parameterized service dependency graph G for conducting the utility analysis of φ_{s_5}. As in Figure 3.4b, G contains 5 vertexes denoting the 5 different cloud services and

4 edges specifying the dependency with the three-parameter (α_{mn}, β_{mn} and γ_{mn}) vectors. Then, the target-centric dependency trimming algorithm is executed to remove the irrelevant services based on the inputs of G and α_{mn}. The output of the algorithm is the trimmed parameterized service dependency graph $G^{'}$ which removes $S4$ as denoted in dotted form. Since the service dependency $\vec{D_{34}}$ is irrelevant to the utility analysis for monitoring φ_{s_5} and its directional parameter α_{34} sets to -1 in the figure.

• ***Step 3. Monitoring Result Inference*** *deQAM* computes the utility of the parameters collected from $S1$, $S2$ and $S3$.

$*$ *Eigen utility:* φ_{s_5}'s eigen utility $U_{E\_\varphi_{s_5}}$ is decided by the basic provisioning level of $S5$. In this case study, we set $U_{E\_\varphi_{s_5}}$ to 0.5.

$*$ *Mandatory-type contributed utility:* *deQAM* computes the mandatory-type contributed utility $U_{M\_\varphi_{s_5}}$ by using the monitoring information collected from $S3$. As $U_{\varphi_{s_3}} = 1$, the mandatory-type contributed utility is computed as:

$$U_{M\_\varphi_{s_5}} = \prod_{1}^{1} U_{\varphi_{s_3}} = 1$$

$*$ *Optional-type contributed utility:* *deQAM* computes the optional-type contributed utility $U_{O\_\varphi_{s_5}}$ by using the monitoring information collected from $S1$ and $S2$. According to Equation (3.2) and (3.3), $S1$'s ripple-effect optional-type utility $U_{15}^{(o.r)}$ and $S2$'s direct optional-type contributed utility $U_{25}^{(o.d)}$ are computed respectively as:

$$U_{15}^{(o.r)} = (\gamma_{12} \cdot U_{\varphi_{s_1}}) \cdot \gamma_{25} = 0.216$$
$$U_{25}^{(o.d)} = \gamma_{25} \cdot U_{\varphi_{s_2}} = 0.16$$

By Equation (3.4), φ_{s_5}'s optional-type contributed utility $U_{O\_\varphi_{s_5}}$ is:

$$U_{O\_\varphi_{s_5}} = U_{15}^{(o.r)} + U_{25}^{(o.d)} = 0.376$$

$*$ *Inaccessible service parameter utility inference:* Based on the above computation, utility $U_{\varphi_{s_5}}$ of the inaccessible service parameter φ_{s_5} is derived by Equation (3.5) as:

$$U_{\varphi_{s_5}} = U_{M\_\varphi_{s_5}} \cdot (U_{E\_\varphi_{s_5}} + U_{O\_\varphi_{s_5}}) = 0.876$$

• ***Step 4. Monitoring Result Generation*** *deQAM* maps utility $U_{\varphi_{s_5}}$ back to φ_{s_5}'s value to generate the monitoring result $V_{\varphi_{s_5}}$. By applying proposed inverse conversion rule $f_{\_N}^{-1}$, the monitoring result $V_{\varphi_{s_5}}$ of the inaccessible parameter φ_{s_5} is generated as follows. Because $U_{\varphi_{s_5}}$ is 0.876 and the varying range of $V_{\varphi_{s_5}}$ is $[0, 100\%]$, the monitoring result $V_{\varphi_{s_5}}$ is derived as:

$$V_{\varphi_{s_5}} = \frac{U_{\varphi_{s_5}}}{U_{\varphi_{s_5\_max}} - U_{\varphi_{s_5\_min}}} \cdot 100\% = \frac{0.876}{1-0} \cdot 100\% = 87.6\%$$

• ***Step 5. Result Reliability Evaluation*** *deQAM* determines the confidence level of the generated monitoring result by using Equation (3.9) and the value of γ_{mn} specified in Table 3.1. In this case study, we assess the reliability of the monitoring result $V\varphi_{s_5}$ with the confidence level $(1-c)$ of 95% (i.e. $1 - c = 0.95$). By checking the χ^2 distribution table [Sch12], we find $\chi^2_{(\frac{0.05}{2},3)} = 0.216$ and $\chi^2_{(1-\frac{0.05}{2},3)} = 9.348$. According to Equation (3.7) and (3.9) and Table 3.1,

$$\gamma_{12} = 0.9, \gamma_{25} = 0.8, \gamma_{35} = 1, \gamma_{mean} = (\gamma_{12} + \gamma_{25} + \gamma_{35})/3 = 0.9$$

$$\hat{\sigma}^2_{\xi(V_{\varphi_{s_5}})} = \frac{1}{3}\sum_{1}^{3}(\gamma_{mn} - \gamma_{mean})^2 = \frac{0.1^2 + 0.1^2}{3} = 0.0067$$

$$\frac{\sum_1^3(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(1-\frac{0.05}{2},3)}} = \frac{0.1^2 + 0.1^2}{0.226} = 0.0926$$

$$\frac{\sum_1^3(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(\frac{0.05}{2},3)}} = \frac{0.1^2 + 0.1^2}{9.348} = 0.0021$$

As a result, *deQAM* determines the uncertainty variance $\hat{\sigma}^2_{\xi(V_{\varphi_{s_5}})}$ of the generated monitoring result $V_{\varphi_{s_5}}$ of inaccessible parameter φ_{s_5} is 0.0067 and the 95% confidence level is derived as the interval of $(0.0021, 0.0926)$.

3.5 Discussion

Our case study shows that *deQAM* can successfully infer the monitoring result by using the dependencies across cloud services. The study also shows that *deQAM* can evaluate the reliability of the monitoring result by giving a 95% confidence level of the uncertainty estimation. The variance of uncertainty is as small as 0.0067. This implies that the monitoring result is unlikely to unexpectedly deviate from the true value of $V_{\varphi_{s_5}}$. Meanwhile, the 95% confidence level is based on the interval of $(0.0021, 0.0926)$. This narrow confidence interval constrains the randomness of the uncertainty variance. The narrow confidence interval disables the potential big deviation of the real uncertainty variance. As the uncertainty variance 0.0067 falls within the confidence interval, the reliability of the monitoring result is statistically sound.

Our case study also utilizes different types of parameters commonly existing in cloud services. For instance, the parameter with a varying range represents the service parameter with variable status like CPU usage rate as in IaaS. Similarly, the parameter selected from a set represents the service parameters with several possible levels like different encryption levels of cloud encryption services. Moreover, the parameter with Boolean value represents the service parameter with the switch-effect like authentication parameters deciding the access to cloud services. Therefore, the case study demonstrates *deQAM*'s capability of handling most of the common service parameter types used in the cloud.

3.6 Related Work

The indirect monitoring for cloud service is a novel research topic that is only beginning to garner attention. Thus, some works that likely have potential value for design the indirect monitoring approach are surveyed below.

One related field is about service dependency though most works in this field focus on describing the notion of "dependency" from varied perspectives. For example, Winkler et al. [Win+09] classified the service dependencies into discrete types for creating the service dependency model. Eppinger et al. [EB12] presented the design structure matrix (DSM) for refining the service dependency. Likewise, Qi et al. [Qi+12] proposed a dependency graph to analyze the services by trust. However, Garvey et al. [GP09][GPS14] proposed a functional dependency network analysis framework (FDNA) by studying the characteristics of service dependency. Similarly, Guariniello et al. [GD13][GD14][GD17] proposed the system behavior modeling methodologies by improving the FNDA.

Another related field is about uncertainty, as the indirect monitoring approaches have to address the reliability issue by assessing the uncertainty of the generated monitoring results. For example, Wang et al. [Wan+14] proposed an adapted backward cloud generator algorithm to address the uncertainty of service quality. Tang et al. [TT14] proposed a Bayesian based methodology to address the prediction uncertainty of service level agreement violations in cloud. Furthermore, Huynh et al. [HBB12] proposed a state-based policy framework to assess the monitoring quality caused the uncertain. Moreover, Galland et al. [Gal+10] proposed the different data value estimation algorithms to address the uncertainty. Additionally, Li et al. [Li+14a] proposed a weighting optimization algorithm to address the uncertainty concern by assigning different weights to data sources. As contemporary work, Yin et al. [YT11] proposed a semi-supervised learning method to reduce the uncertainty.

3.7 Conclusion

Indirect monitoring for cloud services is an emerging challenge introduced by the inaccessibility of multiple cloud service parameters. However, the existing monitoring approaches typically are inapplicable without direct access to those service parameters. Therefore, we propose *deQAM* as a dependency-based quantitative aggregation indirect-monitoring methodology to specifically address this gap.

deQAM introduces the *utility* to correlate the inaccessible parameter of the dependent cloud service with the related accessible parameters of antecedent cloud services by using the service dependency. Furthermore, *deQAM* also adopts a multi-step utility computing procedure to infer the inaccessible parameter's value as the indirect monitoring result. Additionally, *deQAM* evaluates the reliability of the monitoring result by specifying its confidence level.

The advantages of our approach are the capability of monitoring the inaccessible cloud service parameters and the adaptability of working in many other indirectly monitoring scenarios. Currently, *deQAM* can deal with the static indirect monitoring case. In the future, we plan to improve *deQAM* for dealing with the dynamic monitoring scenario, in which the value of relevant parameters change over time. Overall, this thesis provides a valuable starting point for exploring indirect monitoring in cloud.

4 Performing In-Depth Analysis on Monitoring Information

In Chapter 3, we propose a novel indirect monitoring mechanism for managing the security assurance of cloud services. The proposed mechanism outperforms a plethora of monitoring schemas that have been proposed over the recent years with the unique merit of working in the scenario where the key monitoring information is not directly accessible. To conduct the indirect security monitoring in cloud systems, a key prerequisite is to obtain a set of selected monitoring data termed "*monitoring path*" that is with respect to pinpointing a particular security problem. However, how to ascertain the monitoring path is still an open issue. In this chapter, we propose *Flashlight* as a novel monitoring path identification mechanism to address the gap where the information of monitoring targets is inaccessible. For this purpose, we first introduce a novel data reduction technique to filter unnecessary monitoring information in *Flashlight* . We also develop a data association approach for enabling *Flashlight* to identify the monitoring path by utilizing data relations and data attributes. Additionally, we present a *monitoring property graph* to support fine-grain monitoring path identification and represent relevant identified paths as well other valuable information. Finally, we evaluate the efficacy of the proposed mechanism by conducting two case studies on realistic security threats in the cloud. The evaluation results demonstrate that *Flashlight* can successfully identify monitoring paths for underpinning indirect security monitoring on cloud services.

4.1 Introduction

Cloud monitoring is a critical mechanism to help secure cloud services, and at the same time validating the compliance of security requirements proposed by cloud service customers (CSCs). It utilizes collected monitoring information for ensuring data security, managing service performance, protecting user privacy, and other objectives. Consequently, a variety of cloud monitoring schemas have been proposed in recent years [DGY10][Bow+11][ML13][Wu+13][Den+14][Sae+14][Wan+15b]. The proposed monitoring schemas are predominantly developed based on an assumption where the monitoring information is directly accessible. In reality, only a very few of monitoring information can be accessed by CSCs. For example, CloudWatch that targets monitoring applications and resources in Amazon Web Services (AWS) only provides the CSCs with a small number of monitoring metrics, such as CPU utilization, disk read/write behaviors, network traffic, and status check reports [Ama17]. In many cloud services, some monitoring information is also restricted to be accessed by the CSCs. For instance, the version information of OpenSSL library, which is adopted by a Software-as-a-Service (SaaS) cloud service provider (CSP) for securing network communications, is important for monitoring the Heartbleed attack [NVD14] but cannot be directly accessed for security reasons. Unfortunately, the proposed schemas cannot deal with this situation where particular monitoring information is inaccessible. The situation obstructs CSCs to check the security compliance of the subscribed cloud services.

The challenge of lacking direct access to particular monitoring information has recently been addressed by an indirect cloud monitoring mechanism [Zha+17]. This indirect monitoring mechanism is developed based on the observation that cloud services typically do not function on a standalone basis but could share system resources in some situation. Resource sharing could introduce data associations which can be exploited by attackers [HJS01][WXW15][Zha+14]. For example, a study [Ris+09] reports that the AES [NIS01] and RSA [RSA78] secret keys can be stolen by exploiting the data association introduced by memory resource sharing in virtualized environments, while such data associations can also be utilized to monitor cloud services. The proposed monitoring mechanism particularly takes advantage of the data association termed "*monitoring path*", which is formed by a set of accessible monitoring data and related data relations, to indirectly infer the inaccessible monitoring information. For example, the encryption overhead of the cloud service depends on the adopted encryption method and the size of the target file to encrypt. By aggregating the monitoring path that is formed by the information of the target file size, the adopted encryption method, and the dependency, the indirect mechanism can infer the information of encryption overhead which cannot be directly monitored. While the proposed indirect monitoring mechanism exhibits its unique advantage of monitoring the particular information that is inaccessible for a variety of reasons (e.g., technical difficulties, intellectual property protection issues, or privacy concerns), the key problem of ascertaining monitoring paths to enable the indirect cloud monitoring has not been addressed.

The monitoring path is difficult to ascertain due to several reasons. First, it is hard to discern the valuable monitoring information that is relevant to help identify monitoring paths. Massive monitoring data is continuously collected by cloud security monitors, while only a small portion of monitoring data is useful for spotting potential data associations utilized as the monitoring paths for performing indirect cloud monitoring tasks. An effective technique for filtering the irrelevant monitoring data from the collected data is demanded. Second, there is a lack of quantitative techniques to associate useful monitoring data for generating the monitoring path. Given a set of monitoring data and a monitoring target, a valid quantitative technique is required to identify the data association that underpins the indirect monitoring on the inaccessible information of the monitoring target. Third, there is an absence of an effective schema that can not only represent the identified monitoring paths but also further support identifying monitoring paths in a complex scenario. It is difficult to properly represent the identified monitoring paths with existing representation mechanisms. Especially, the monitoring data involved complex data relations (e.g., service dependency, time causality, or data consistency) introduces additional challenges for identifying monitoring paths. Without properly ascertaining monitoring paths , the efficacy of the proposed indirect monitoring mechanism is substantially jeopardized.

In this thesis, we follow the research direction of the emerging indirect cloud monitoring and thus propose *Flashlight* as a novel monitoring path identification mechanism to address the aforementioned gap. Given a set of collected monitoring data and the related information (i.e., data attributes and data relations), our proposed mechanism can select useful monitoring data from the collected data set, identify particular data associations as the monitoring paths, and represent the identified monitoring paths together with other useful information. To this end, *Flashlight* first makes use of the data attributes to obtain valuable monitoring data by filtering a significant amount of trivial monitoring data. Next, *Flashlight* takes advantage of the data relations to identify data associations (i.e., monitoring paths) by utilizing the statistical characteristics of the collected monitoring data. Finally, *Flashlight* proposes a monitoring property graph to represent the identified monitoring paths and other complex information of the collected monitoring data (i.e., values, attributes, relations, and associations) which underpins the fine-grain monitoring path identification. By employing *Flashlight*, security professionals can conveniently obtain monitoring paths for performing indirect cloud monitoring rather than manually conducting a cumbersome monitoring path analysis which is not only requiring a profound knowledge of all the collected monitoring data but also strictly subject to the scale of the collected data set.

To the best of our knowledge, our work is the first monitoring path identification mechanism for facilitating indirect cloud monitoring. In summary, we make the following contributions:

1. We introduce a novel monitoring path identification mechanism that can discern valuable monitoring data and ascertain monitoring paths for performing indirect cloud monitoring.

2. We propose a novel monitoring property graph with the advantage of representing complex information (i.e., data value, data attributes, data relations, data associations, and monitoring paths) in order to improve the efficacy of the indirect cloud monitoring methodology.

3. We evaluate the efficacy of the proposed mechanism with identifying monitoring paths that help indirectly monitor different classes of practical security threats in cloud.

The remainder of this chapter is organized as follows. Section 4.2 outlines the issues behind the monitoring path identification. Section 4.3 formulates the monitoring path identification problem. Section 4.4 details the design of the proposed mechanism. The effectiveness of the proposition is evaluated in Section 4.5 by performing case studies on real cloud threats. Section 4.6 reviews related work.

4.2 Background

To motivate the issues, we first present two prominently reported security threats to highlight the significance of monitoring path identification for achieving effective service security compliance monitoring by using the indirect cloud monitoring mechanism. Subsequently, we analyze the reported threats to provide insights on the monitoring data underpinning our design.

4.2.1 Existing Threats

Our work is motivated by the CSC's pragmatic requirement of monitoring the security compliance of cloud services. Effectively monitoring security compliance requires in-depth understanding of the links among monitoring data collected from cloud services whose security compliance might be violated by security threats.

The access-driven cache attack

A typical security requirement from the CSC is about the encryption key management which targets keeping the applied encryption key in safe. However, it is difficult to monitor the compliance of this security requirement without perceiving the valid monitoring path. For example, an access-driven cache attack (ADCA) aims to exploit the timing behaviors of cache accesses so as to compromise the confidentiality of cloud services. Gullasch et al. [GBK11] present an access-driven cache attack which enables an unprivileged spying process to recover the secret key of a running encryption process (AES-128). To conduct such an attack, the attacker exploits the knowledge of the *cache specification*, namely the *cache access status (hits/misses)*, *CPU cycles*, and *AES*-128 *encryption protocol*. It is challenging to monitor the security compliance which might be violated by this particular attack, if the interlinked data associations across data elements are not fully understood.

The resource-freeing attack

The CSC also puts emphasis on the business continuity management to assure the service availability varying within the specified range. Nonetheless, it is laborious to monitor the compliance of this security requirement without deeply understanding the monitoring data. For instance, a resource-free attack (RFA) targets the imperfect isolation mechanism of the virtual machine (VM) hypervisor. The attacker introduces crafted interference on a victim VM to trigger a performance bottleneck which leads the victim VM to free up the system resource for the beneficiary VM. Varadarajan et al. [Var+12] present the RFA case to exploit the hypervisor isolation policy (i.e., the fair-sharing policy) to free up the network resource from one web service to another web service, even though both web services should equally share the network resources. In this case, the attacker implements the attack by exploiting the knowledge of, *CPU utilization*, *high CPU overhead service request*, and *the crafted attack process*. Likewise, it is also critical to discover the compliance violation caused by the RFA attack with effectively understanding collective data associations.

4.2.2 Example Interpretation

These examples highlight the challenge of monitoring the security compliance of cloud services without gaining an insight on the collective monitoring data. The key point for addressing this challenge is to identify the "useful" associations across the monitoring data. A data association can be regarded as the *monitoring path*, which is formed by a sequence of monitoring data and data relations, used for monitoring the security compliance. To identify the monitoring path, two dominant issues need to be properly considered.

First, the monitoring data collected from the cloud carries much more information than just the data value. Considering to monitor Distributed Denial of Service (DDoS) attacks, a lot of monitoring data is collected by security monitors. The value of monitoring data (e.g., the amount of incoming user requests) can provide important information for monitoring DDoS attacks. Besides, some special types of monitoring data might have higher criticality than other collected data (e.g., the file accessing data or the process context data) in the context of the DDoS attack monitoring. Moreover, monitoring data might be subject to various relations such as time causality or service dependency. Without a collective understanding of the monitoring data, it is difficult to identify the monitoring path.

Second, an effective data association technique for identifying the monitoring path is missing. While a large number of cloud monitoring techniques have been proposed by researchers [Sha+10][Bow+11][Den+14][Nin+15], these techniques target monitoring the specific individual data (e.g., parameters, values, or formats). Given the paucity of data association mechanisms, these approaches cannot properly monitor the security compliance that may be violated by particular security threats (e.g., ADCA or RFA) mentioned in Section 4.2.1. Determining such data associations underlies our proposed approach.

4.2.3 Data Relation

Relations across the monitoring data is indispensable for identifying data associations in the raw monitoring data set [Vim+14]. Data relations encompass a wide range of aspects, such as causality, consistency, or dependency. For example, data dependency is a common type of data relation existing among many cloud services. Data dependency is observed between the Heartbleed attack and the OpenSSL library which is a widely used cryptography library for encrypting the network connection between a cloud server and its clients. Namely, if the OpenSSL library supporting the Heartbeat mechanism has the version number of "1.0.1" (excluding "1.0.1*g*") [NVD14], the Heartbleed attack could take effect. Therefore, the indispensable knowledge on such dependency helps identify data associations.

Besides, the attribute of monitoring data is also the critical knowledge that can be used to facilitate identifying data associations. In this thesis, we consider the collected data with two key attributes, namely *reducibility* and *criticality*.

Reducibility is used to capture the necessity of the monitoring data. For example, when Dropbox conducts the directory scanning operation (by running the "pcscd" daemon), this operation can produce a lot of repetitive monitoring information (i.e., *reading/accessing* operations recursively conducted in the specified path) [Xu+16]. From a monitoring perspective, the repetitive data does not convey any additional useful information. As a result, the repetitive data can be reasonably reduced by leveraging data reducibility.

Criticality is used to capture the significance of the monitoring data. For every monitoring target, the collected monitoring data has different levels of importance. As the ADCA example shows, the monitoring data of *cache access status (hits/misses)* has more importance than *file accessing status* with respect to monitoring the security compliance that may be violated by this particular security threats. Accordingly, the collected data can be weighted by taking advantage of the data criticality.

4.3 Problem Formulation

On this background, we now present a formalization of the monitoring path identification problem.

In this thesis, we consider the scenario that contains a set of monitoring data $\mathcal{D} = \{d_1, d_2, \cdots, d_k\}$ (for some $k \in \mathbb{N}$) collected by deployed security monitors and a set of data relations $\mathcal{R} = \{R_1, R_2, \cdots, R_l\}$ (for some $l \in \mathbb{N}$) existing among the collected data. The data relation R_i is a function $R_i : \{\mathcal{D}\} \to \mathcal{P}(\mathcal{D})$. The output of the function is a subset $D_i = \{d_{i1}, d_{i2}, \cdots, d_{ij}\}$ (for some $j \in \mathbb{N}, 1 \leqslant i \leqslant l$) where d_{ij} is collected monitoring data represented by a 3-tuple $d_{ij} = (v_{ij}, r_{ij}, c_{ij})$. For monitoring data d_{ij}, v_{ij} represents its value, $r_{ij} \in \{reducible, irreducible\}$ represents its data reducibility, and $c_{ij} \in [0, 1]$ represents its data criticality. That is to say, when the relation $\mathcal{R}_i$ functions on the data set $\mathcal{D}$, a corresponding subset of monitoring data D_i can be obtained.

This scenario targets the monitoring data collected by taking advantage of data relations (e.g., causality, consistency, or dependency) and data attributes (e.g., criticality and reducibility). Notably, data relations and attributes of monitoring data can be derived by utilizing the expert knowledge or applying existing approaches [Bia+10][ZYR14][Pap+15]. For the sake of simplicity, we assume that one collected monitoring data is subject to only one type of relation. As a matter of fact, even though the collected monitoring data may be subject to multiple types of relations, it is necessary to consider one specific type of relation for monitoring the information of a specific target.

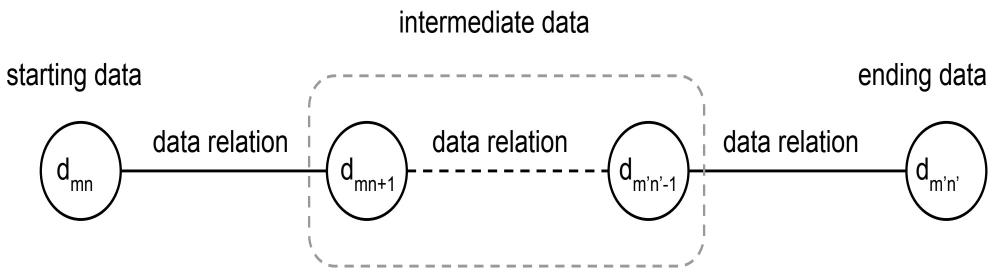


Figure 4.1: An example monitoring path that is composed of starting data, a set of intermediate data, and ending data

With the above definitions, we define a monitoring path p as a structure that begins from a monitoring data $d_{mn} \in \mathcal{D}(m, n \in \mathbb{N})$ via a sequence of intermediate monitoring data and ends at another monitoring data $d_{m'n'} \in \mathcal{D}(m', n' \in \mathbb{N})$. For each intermediate data of monitoring path p, it associates a predecessor with a successor. The association between adjacent monitoring data in the monitoring path reflects relations between the data. As depicted in Figure 4.1, the monitoring path p is composed of starting data d_{mn}, ending data $d_{m'n'}$, and a set of intermediate data (e.g., d_{mn+1} or $d_{m'n'-1}$). Two adjacent data d_{mn}, d_{mn+1} are required to belong to a common data relation: $d_{mn}, d_{mn+1} \in R_i(\mathcal{D})$ for some $1 \leqslant i \leqslant l$. By synthesizing such data associations, monitoring path p bridges the accessible monitoring data and the inaccessible monitoring target so as to enable the indirect monitoring task.

In this thesis, we propose to identify the monitoring data association by utilizing the data attributes and the data relations. The monitoring path identification problem is defined as follows.

- Given a relation set $\mathcal{R}$,
- Given a monitoring data set $\mathcal{D}$, where the collected data d_{ij} is equipped with two data attributes : data reducibility r_{ij} and data criticality c_{ij},

▷ The monitoring path identification problem is to identify a set of monitoring paths $\mathcal{P} = \{p_s; s \in \mathbb{N}\}$ where p_s is the s^{th} monitoring path in $\mathcal{P}$.

4.4 Proposed Methodology

In this section, we detail the proposed methodology. We first provide an overview of the framework of the proposed mechanism. Based on the framework, we explain the specific design step by step.

4.4.1 Methodology Overview

The key point of the monitoring path identification is to identify the relevant monitoring data associations. We take advantage of data relations and data attributes as discussed in Section 4.2.3 to identify data associations. As a result, we propose a multi-step monitoring path identification mechanism as depicted in Figure 4.2. The

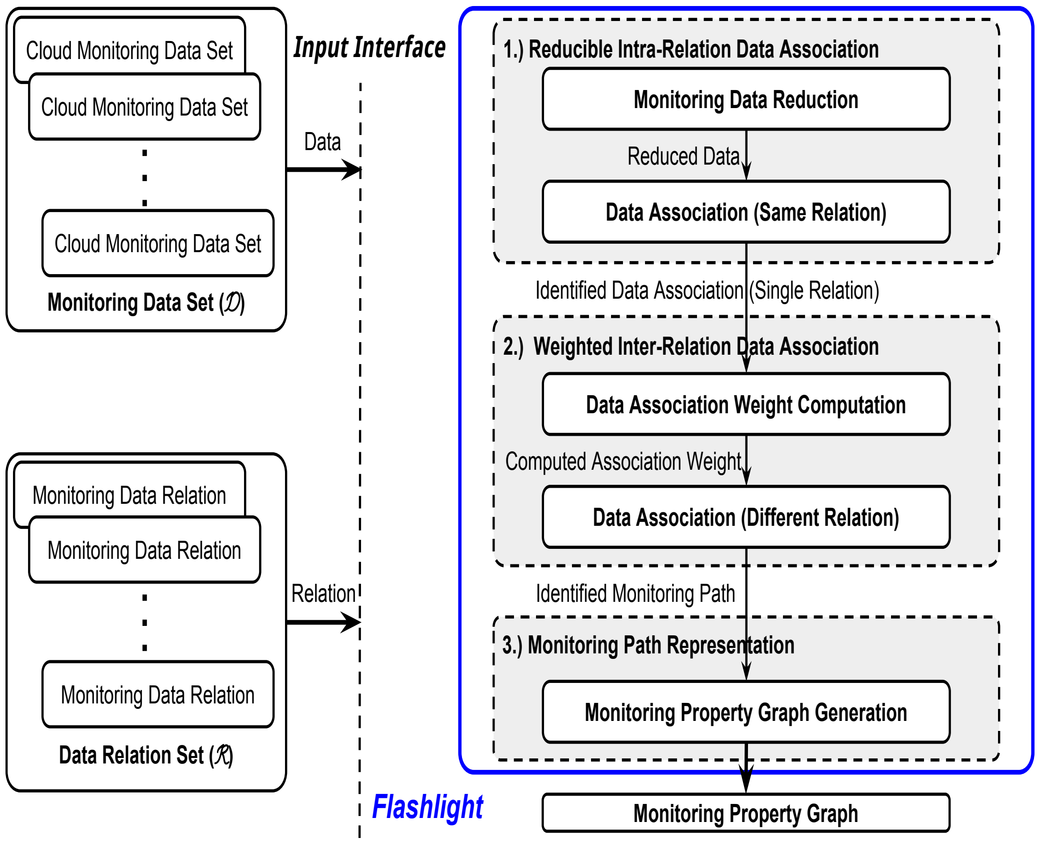

Cloud Monitoring Data Set
Cloud Monitoring Data Set
Cloud Monitoring Data Set
Monitoring Data Set ($\mathcal{D}$)
Input Interface
Data
Monitoring Data Relation
Monitoring Data Relation
Monitoring Data Relation
Data Relation Set ($\mathcal{R}$)
Relation
Flashlight
1.) Reducible Intra-Relation Data Association
Monitoring Data Reduction
Reduced Data
Data Association (Same Relation)
Identified Data Association (Single Relation)
2.) Weighted Inter-Relation Data Association
Data Association Weight Computation
Computed Association Weight
Data Association (Different Relation)
Identified Monitoring Path
3.) Monitoring Path Representation
Monitoring Property Graph Generation
Monitoring Property Graph

Figure 4.2: The framework of the proposed multi-process monitoring path identification methodology

proposed mechanism contains three steps as follows.

1. *Reducible intra-relation data association* takes a set of monitoring data $\mathcal{D}$ and a set of data relations $\mathcal{R}$ as the input to execute the association process for deriving the data association with the same relation.

2. *Weighted inter-relation data association* takes data associations derived from different relations (Step 1) as input to associate them as a monitoring path.

3. *Monitoring path representation* takes the relevant information from the previous steps as the input for representing the identified monitoring paths in the proposed monitoring property graph where the represented monitoring information (i.e., data values, data attributes, data relations, and data associations) facilitates monitoring path identification when the monitoring data involves with more complex relations.

4.4.2 Methodology Design

We now detail the proposed multi-step methodology as demonstrated in the framework that is depicted in Figure 4.2.

Reducible intra-relation data association

To identify the data associations in the monitoring data set, *Flashlight* first reduces the input data set by utilizing data attributes (i.e., *reducibility*) and then ascertains data associations with the identical relation therein.

In practice, the collected monitoring data set is not necessarily completely required for conducting the specific monitoring task. Considering the Dropbox example in Section 4.2.3, we notice that even though a large amount of monitoring data is collected (i.e., the repetitive low-level file system accessing data), the meaningful monitoring information is about the scanning operation that triggered the “pcscd" process. In other words, the repetitive low-level data does not provide much meaningful monitoring information.

Without loss of generality, we explain the data reduction process in Figure 4.3 which consists of five nodes (monitoring data) and six edges (data associations). In this figure, we use full circles to represent *irreducible monitoring data* (e.g., d_{ii} can denote the monitoring data of “pcscd" process), while dashed circles denote *reducible monitoring data* (e.g., d_{ii+1} is the monitoring data of the low level file system accessing operation occurred in hard disks). The solid edge represents the irreducible association between a pair of the irreducible monitoring data. In contrast to the

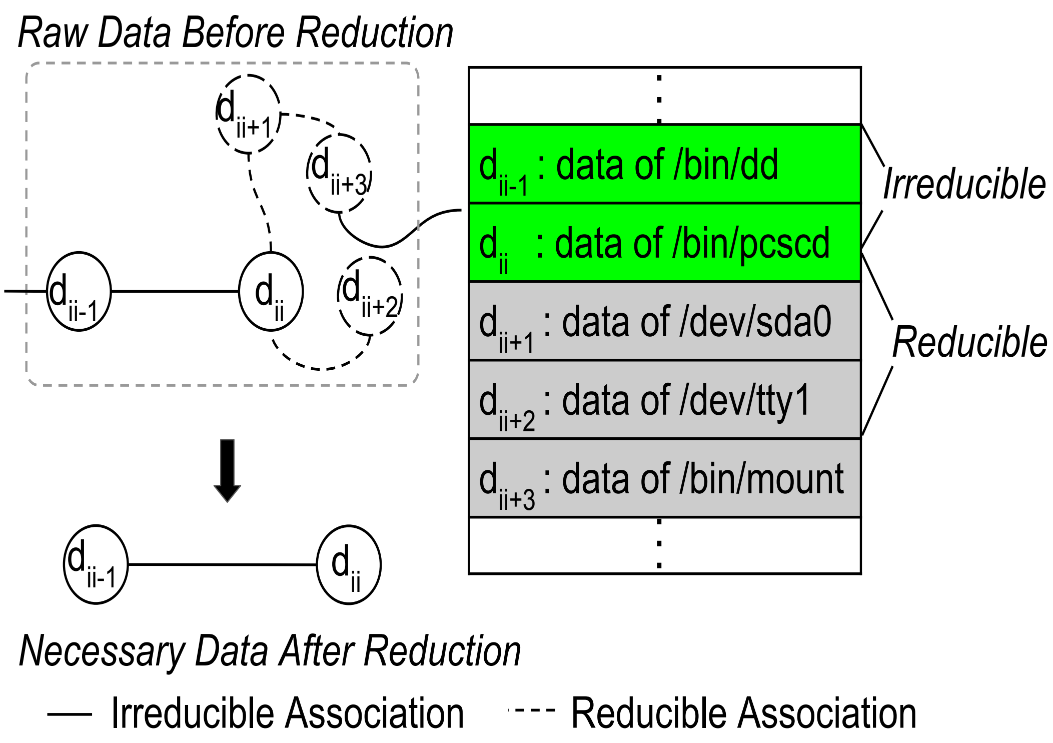

Raw Data Before Reduction
d_{ii+1}
d_{ii+3}
d_{ii-1}
d_{ii}
d_{ii+2}
d_{ii-1} : data of /bin/dd
d_{ii} : data of /bin/pcscd
d_{ii+1} : data of /dev/sda0
d_{ii+2} : data of /dev/tty1
d_{ii+3} : data of /bin/mount
Irreducible
Reducible
d_{ii-1}
d_{ii}
Necessary Data After Reduction
— Irreducible Association
--- Reducible Association

Figure 4.3: A data reduction example that demonstrates the feasibility of filtering irrelevant monitoring information

solid edge, the dotted edge represents the reducible association which is introduced by the reducible monitoring data. In this example, the "pcscd" process is executed by calling */bin/dd* command. Then, the "pcscd" process repetitively performs a series low level of operations like mounting storage devices (*/bin/mount*), reading hard disks (*/dev/sda*0), or accessing TTY devices (*/dev/tty*1). From a monitoring perspective, monitoring information of those repetitive operations does not give more help and thus can be properly reduced. Therefore, by removing the reducible monitoring data, the original graph can be reduced to a simpler structure as a two-node association between d_{ii-1} and d_{ii}. This data reduction process is conducted by examining the value of r_{ij} in the triple d_{ij} as defined in Section 4.3. To be specific, if r_{ij} equals to "reducible", d_{ij} will be removed from the input data set. Otherwise, d_{ij} will be kept in the data set.

After filtering the unnecessary data, *Flashlight* proceeds to an intra-relation data association process. By running a relation-based monitoring process (e.g., a monitoring process collects several data in terms of the service dependency), certain monitoring data combinations frequently appear in the collected monitoring data set. For example, Transport Layer Security (TLS) handshake request and response messages frequently appear on Port 443 (i.e., the HTTPS port) during monitoring the Heartbleed attack. Such data co-appearance indicates the existence of an association among the monitoring data. The frequency of the co-appearance also reflects the association degree of the data. With the above observation, the intra-relation monitoring data association is defined as follows.

Definition 1. *Let (d_{ij-1}, d_{ij}) be a pair of monitoring data subject to the same relation R_i. The monitoring data association between d_{ij-1} and d_{ij} is denoted as $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$: $d_{ij-1} \uplus d_{ij}$.*

In this thesis, $\uplus$ is the symbol used for representing the association between two different objects, such as two different monitoring data or two different monitoring data associations.

Flashlight adopts two metrics (i.e., the *support* and the *confidence*) to characterize the data association $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$ inspired by work [AS94] that utilizes the two metrics to quantify statistic properties of different objects to discover existing associations among the objects.

The *support (sup)* of the data association $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$ denotes the proportion of the collected monitoring data sets which include both d_{ij-1} and d_{ij}. It captures the co-appearance rate of two monitoring data by representing with the probability of the data co-appearance. Hence, the formula for computing *support* is as follows.

$$sup(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}) = \sigma(d_{ij-1} \bigcap d_{ij}) \tag{4.1}$$

In Formula (4.1), $\sigma(d_{ij-1} \bigcap d_{ij})$ presents the proportion of the monitoring data sets containing both d_{ij-1} and d_{ij}. These data sets are collected by the monitoring process with utilizing relation R_i.

The *confidence (conf)* of the data association $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$ denotes the ratio of d_{ij} in the collected monitoring data sets which contain both d_{ij-1} and d_{ij}. It captures the conditional probability of d_{ij}'s appearance given d_{ij-1}'s appearance. Accordingly, the formula for computing *confidence* is given as follows.

$$conf(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}) = \frac{\sigma(d_{ij-1} \bigcap d_{ij})}{\sigma(d_{ij-1})} \tag{4.2}$$

In Formula (4.2), $\sigma(d_{ij-1})$ presents the proportion of the monitoring data sets containing d_{ij-1}. Likewise, these data sets are collected by the relation R_i based monitoring process.

As the monitoring data appearance is regarded as a random variable, the correlation between different data also needs to be considered. A common method to quantify the random variables' correlation is to compute the Pearson correlation coefficient of the given variables. However, the Pearson correlation coefficient is generally hard to compute [TKS00]. To effectively capture the correlation between monitoring data, *Flashlight* introduces the *ambiguity* which can not only represent monitoring data correlations but also be easier to determine by applying an optimized solution as follows.

$$amb(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}) = \sqrt{\frac{p(d_{ij-1}, d_{ij})}{p(d_{ij-1}) \cdot p(d_{ij})} \cdot \sigma(d_{ij-1} \bigcap d_{ij})} \tag{4.3}$$

In Formula (4.3), $\sigma(d_{ij-1} \bigcap d_{ij})$ is $sup(\mathcal{A}|_{d_{ij}}^{d_{ij-1}})$, $p(d_{ij-1})$ is the probability of d_{ij-1}, and $p(d_{ij})$ is the probability of d_{ij}. $p(d_{ij-1}, d_{ij})$ is the joint probability of both data.

By applying Formulas (4.1)(4.2)(4.3), the intra-relation data associations can be identified over data set D_i and the identification results can be presented by an association vector $I(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}) = (sup(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}), conf(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}), amb(\mathcal{A}|_{d_{ij}}^{d_{ij-1}}))$.

With this data association process, *Flashlight* can apply it in a step-wise mode for associating more data subject to the same relation. For example, to associate the next monitoring data d_{ij+1}, *Flashlight* regards the derived data association $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$ as a whole (i.e., like one "virtual data") and applies the intra-relation data association process to identify the association vector $I(\mathcal{A}|_{d_{ij+1}}^{d_{ij}})$. The data association process keeps executing until no more data can be associated according to the predefined threshold or baseline.

Weighted inter-relation data association

In many scenarios, the monitoring path is a combination of multiple data associations derived from different data sets subject to different types of relations. Therefore, the *inter-relation data association* technique is proposed based on the following idea: In Step 4.4.2, a set of data associations are derived by leveraging different data relations. To properly select and combine the derived data associations, the inter-relation data association can be achieved.

To perform the inter-relation data association, we consider two different components: the *precedent* and *subsequent*. The *precedent* is the data association that is about to be combined with another data association. The *subsequent* is the data association selected for the combination. The combination of both components can be also regarded as a random event where both *precedent* and *subsequent* appear at the same time with respect to monitoring a specific target.

Different from the intra-relation data association where the data is subject to the same relation, the inter-relation data association needs to select the *subsequent* for the *precedent*. One common practice for making selections is to consider the criticality of the selection object. As a result, we propose a weighting method for selecting the *subsequent* based on considering the criticality.

As the *subsequent* is a data association that contains a set of monitoring data, we propose to adopt the mean value of the data criticality in the *subsequent* as its *weight*. Formally, the *weight* of the data association is defined as follows.

Definition 2. *Weight W of a data association is the mean value of the data criticality of the association which can be computed as*

$$W(\mathcal{A}|_{d_{io}}^{d_{ih}}) = W(d_{ih} \biguplus \cdots \biguplus d_{io}) = \frac{1}{o-h} \sum_{j=h}^{j=o} c_{ij} \tag{4.4}$$

Where, $\mathcal{A}|_{d_{io}}^{d_{ih}} (h \leqslant o; h, o \in \mathbb{N})$ represents the data association of $d_{ih} \biguplus \cdots \biguplus d_{io}$ and c_{ij} is the data criticality of d_{ij}.

By comparing weight W, the data association of the maximum value W in all the identified intra-relation associations is selected as the *subsequent* for a given *precedent*. The association process of *precedent* and *subsequent* can be carried out similarly to the intra-relation data association by regarding the identified associations as random variables. As a result, the identified inter-relation data association is regarded as the identified monitoring path which contains a set of monitoring data subject to different data relations.

Monitoring result representation

After executing the data association processes, *Flashlight* represents the output of the monitoring path identification process with a graph. Moreover, the graph needs to be able to represent complex information, such as the collected monitoring data, the corresponding data value, the data attribute, the data relation, and the identified data association so as to support the proposed data association process to deal with more complex monitoring scenario. Unfortunately, common graphical methods cannot meet these requirements.

To address this problem, *Flashlight* introduces the *Monitoring Property Graph* inspired by [RN12]. Compared to other graphical representations, the proposed monitoring property graph is able to represent a variety of information as depicted in Figure 4.4. The definition of the monitoring property graph is given as follows.

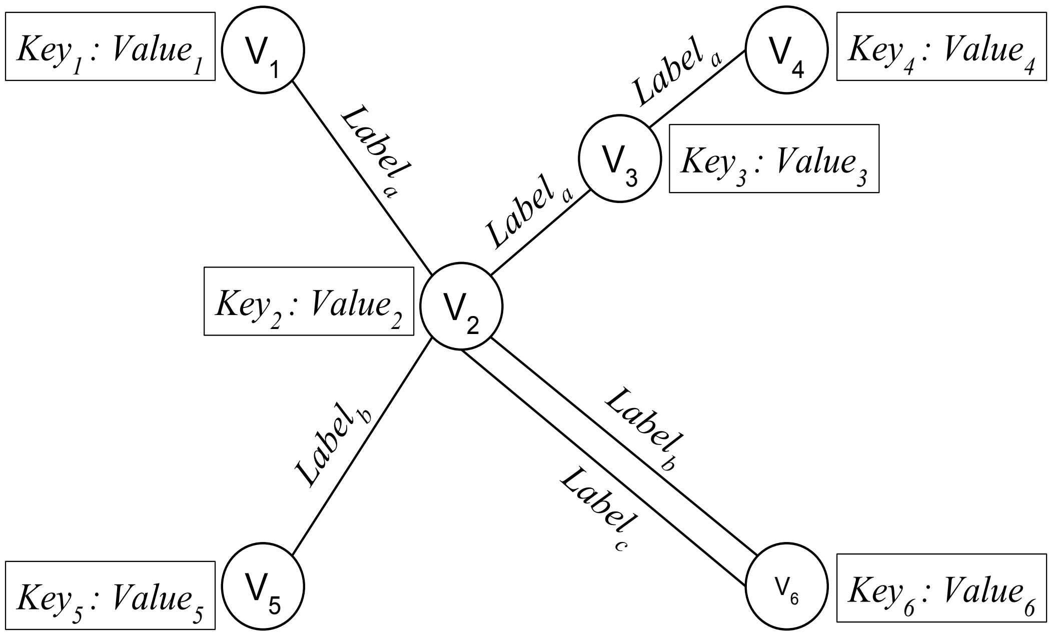


Figure 4.4: An example monitoring property graph with 6 vertices, 6 edges, 3 labels, and 6 key-value pairs.

Definition 3. *A monitoring property graph $G = (V, E, \lambda, \mu)$ is an edge-attributed multi-graph, where V is a finite set of vertices which represent the monitoring data d_{ij}. $E \subseteq (V \times V)$ is a finite set of edges which refer to the data associations $\mathcal{A}|_{d_{ij}}^{d_{ij-1}}$. $\lambda : E \rightarrow \Sigma$ is a labeling function that specifies the edges E with the selected labels Σ for stating the association vector $I(\mathcal{A}|_{d_{io}}^{d_{ih}})$, and $\mu : V \rightarrow K \times S$ annotates vertices with key-value pairs, where $k \in K$ (K is the key set) represents the key (e.g., name) and $s \in S$ (S is the value set) represents the value.*

Figure 4.4 shows a monitoring property graph consisting of six vertices $\{V_1, V_2, V_3, V_4, V_5, V_6\}$ and six edges with three different kinds of labels $\{label_a, label_b, label_c\}$. Every vertex is annotated with the key-value pair (e.g., V_1 is annotated with Key_1 and $Value_1$). Notably, the monitoring property graph is a multi-graph that allows multiple edges between two vertices. For example, two edges exist between V_2 and V_6 while each edge has a different label (i.e., $Label_b$ and $Label_c$). Therefore, the monitoring property graph can address the challenge that monitoring data involves multiple relations.

The proposed monitoring property graph enhances the monitoring path identification, as it supports the data association process functioning in complex data relation scenario with fine granularity. Namely, the data association process can be conducted on selected monitoring information (i.e., data value, data attribute, and data relation) of the monitoring property graph. As an example, if monitoring data involves more than one data relation, it can be regarded as a compound of several "virtual data" where each "virtual data" only involves one single data relation. Therefore, the data association process can be conducted with finer granularity so as to effectively identify more monitoring paths.

4.5 Evaluation

We evaluate the effectiveness of the proposed methodology for helping secure cloud services by performing case studies on two different classes of security threats. The evaluation tasks focus on addressing two critical questions as follows.

- Can *Flashlight* identify the monitoring path for supporting indirect cloud monitoring on the known security threats while some key monitoring information is inaccessible?
- Can *Flashlight* identify the monitoring path for supporting indirect monitoring on the unknown cloud security threats with existing monitoring information?

In the following parts, we present the details of the case studies as well as the related discussions.

4.5.1 Experimental Settings

We evaluate *Flashlight*'s efficacy of ascertaining monitoring paths to help monitor real security threats by simulated experiments. To this end, we set up an Apache web server for simulating a cloud service where the storage functionality is supported by the installed Samba server (version 4.5.9). Besides, we adopt the HTTPS protocol to secure the communication between the web server and simulated users. We adopt the vulnerable OpenSSL library (version 1.0.1) to implement the TLS protocol for the HTTPS connections. To simulate the situation that only limited amount of monitoring information is available in real cloud, we deploy a network monitor to capture the amount of incoming TCP requests (d_{11}), the amount of port 443 requests (d_{12}), the amount of HTTPS responses (d_{13}), and the amount

Table 4.1: Case Study: An Excerpt of The Collected Monitoring Data Set for Conducting Monitoring Path Identification
(N.B. $Rq_{\_tcp}$: inbound TCP requests, $Rq_{\_443}$: inbound port 443 requests,$Rsp_{\_https}$: outbound https responses, $Login_{\_odd}$: login records from odd places, $Op_{\_Root}$: executed privileged operations, $O_{\_Proc.}$: process overhead, $O_{\_NW}$: network overhead)

The Information of the Collected Monitoring Data Set							
ID	d_{11}	d_{12}	d_{13}	d_{21}	d_{22}	d_{31}	d_{32}
Name	$Rq_{\_tcp}$	$Rq_{\_443}$	$Rsp_{\_https}$	$Login_{\_odd}$	$Op_{\_Root}$	$O_{\_Proc.}$	$O_{\_NW}$
Relation	R_1	R_1	R_1	R_2	R_2	R_3	R_3
Value	v_{11}	v_{12}	v_{13}	v_{21}	v_{22}	v_{31}	v_{32}
Reducibility	+	-	-	-	-	-	-
Criticality	0.30	0.53	0.41	0.87	0.98	0.79	0.76
Appearance	N.A	0.83	0.62	0.34	0.27	0.25	0.18
Appearance	N.A	0.45		0.22		0.17	
Appearance	N.A	0.20				N.A	
Appearance	N.A	N.A		0.10			

of odd-place logins (d_{21}). Additionally, we deploy a system activity monitor to capture the amount of privilege-required operations (d_{22}), process overhead (d_{31}), and network traffic overhead (d_{32}). Some relevant data is also required to be collected by leveraging expert knowledge or statistics analysis on security databases like CVSS[1], NVD[2], or exploit-DB[3]. In our experiments, the obtained relevant data consists of data relations (i.e., time causality R_1 and service dependency R_2), data reducibility (reducible / irreducible), data criticality, and monitoring data appearance probabilities. We list the collected information in Table 4.1 which contains three data relations, two data attributes, and four different kinds of normalized data appearance probabilities.

4.5.2 Case Study

Case I : The Heartbleed Attack

For many cloud services, some security threats are difficult to monitor as the particular monitoring information for characterizing the threats is inaccessible. For example, the Heartbleed attack is a notorious security threat that can compromise the CSC's security requirements by exploiting the code flaw of OpenSSL cryptography library (version 1.0.1). By sending crafted packets, attackers are able to retrieve

[1]http://www.cvedetails.com
[2]https://nvd.nist.gov
[3]https://www.exploit-db.com

memory content that may contain confidential information (e.g., user passwords, credit card numbers, or other sensitive information) from the cloud service adopting the vulnerable OpenSSL library for encrypting service communications over port 443 [Die99]. However, the OpenSSL version information that can characterize if the cloud service prone to the Heartbleed attack cannot be directly accessed by the CSC, as it is restricted by the underlying CSP (i.e., a PaaS provider deploys the OpenSSL library to offer the encryption function). To highlight *Flashlight*'s virtue of ascertaining the monitoring path for indirectly monitoring security threats without the access to the key information, we perform a case study on identifying the monitoring path of the abstracted Heartbleed attack.

Flashlight performs the monitoring path identification process by utilizing the collected information as follows.

Step 1. Reducible intra-relation data association: *Flashlight* first starts the reducible intra-relation data association process over the data subject to same relation. Table 4.1 lists three different types of relations: $R_1(causality), R_2(dependency)$, and $R_3(abnormality)$. Accordingly, three different data sets can be derived : $D_1 = \{d_{11}, d_{12}, d_{13}\}$, $D_2 = \{d_{21}, d_{22}\}$, and $D_3 = \{d_{31}, d_{32}\}$. As the value of the collected monitoring data d_{ij} does not affect the identification process, Table 4.1 simply denotes the real data value as v_{ij}. The listed data is used for performing association processes.

Prior to running the association process, *Flashlight* filters the less valuable monitoring information from the raw data set. Namely, *Flashlight* starts with removing the reducible data which is listed in Table 4.1. In this case, the data reduction procedure is only applicable on D_1 where d_{11} is marked as reducible. As a result, the intra-relation data association can be applied on three different data pairs as $d_{12} \uplus d_{13}$, $d_{21} \uplus d_{22}$ and $d_{31} \uplus d_{32}$.

By (4.1)(4.2)(4.3), the association vector of $\mathcal{A}|_{d_{13}}^{d_{12}}$ is computed as:

$$sup(\mathcal{A}|_{d_{13}}^{d_{12}}) = \sigma(d_{12} \bigcap d_{13}) = 0.83$$

$$conf(\mathcal{A}|_{d_{13}}^{d_{12}}) = \frac{\sigma(d_{12} \bigcap d_{13})}{\sigma(d_{12})} = \frac{0.45}{0.83} = 0.54$$

$$amb(\mathcal{A}|_{d_{13}}^{d_{12}}) = \sqrt{\frac{0.45}{0.83 \cdot 0.62} \cdot 0.83} = 0.85$$

Therefore, the relation R_1-based data association can be identified as $I_1 = (0.83, 0.54, 0.85)$. Similarly, the association vector of $\mathcal{A}|_{d_{22}}^{d_{21}}$ computes as: $sup(\mathcal{A}|_{d_{22}}^{d_{21}}) = 0.34, conf(\mathcal{A}|_{d_{22}}^{d_{21}}) = 0.65$, and $amb(\mathcal{A}|_{d_{22}}^{d_{21}}) = 0.90$. Hence, the relation R_2-based data association can be identified as $I_3 = (0.34, 0.65, 0.90)$. Additionally, $\mathcal{A}|_{d_{32}}^{d_{31}}$ is computed as: $sup(\mathcal{A}|_{d_{32}}^{d_{31}}) = 0.25, conf(\mathcal{A}|_{d_{32}}^{d_{31}}) = 0.68$, and $amb(\mathcal{A}|_{d_{32}}^{d_{31}}) = 0.97$. Thus, the relation R_3-based association can be identified as $I_5 = (0.25, 0.68, 0.97)$.

Step 2. Weighted inter-relation data association: Based on the identified intra-relation data associations, *Flashlight* proceeds to execute the weighted inter-relation data association. As discussed in Section 4.4.2, the key point for performing inter-relation data association is to select the proper *subsequent* by considering the data association weight which is decided by the mean value of the data criticality of a data association. As a result, the weight of association $\mathcal{A}|_{d_{13}}^{d_{12}}$ can be computed by applying Formula (4.4) as

$$W(\mathcal{A}|_{d_{13}}^{d_{12}}) = (c_{12} + c_{13})/2 = (0.53 + 0.41)/2 = 0.470$$

Similarly, the weights of association $\mathcal{A}|_{d_{22}}^{d_{21}}$ is $W(\mathcal{A}|_{d_{22}}^{d_{21}}) = 0.925$ and $W(\mathcal{A}|_{d_{32}}^{d_{31}}) = 0.775$ respectively. In terms of the computed weights, the selected *subsequent* for the *precedent* $\mathcal{A}|_{d_{13}}^{d_{12}}$ is $\mathcal{A}|_{d_{22}}^{d_{21}}$ rather than $\mathcal{A}|_{d_{32}}^{d_{31}}$, as $W(\mathcal{A}|_{d_{22}}^{d_{21}}) > W(\mathcal{A}|_{d_{32}}^{d_{31}})$. Therefore, the vector of the inter-relation data association of $\mathcal{A}|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}$ can be computed as: $sup(\mathcal{A}|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}) = 0.45, conf(\mathcal{A}|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}) = 0.44$, and $amb(\mathcal{A}|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}) = 0.95$. The inter-relation ($R_1$ and R_2) data association $\mathcal{A}|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}$ is identified by $I_2 = (0.45, 0.44, 0.95)$ as monitoring path p_1 for monitoring Heartbleed attacks.

The obtained monitoring path p_1, which consists of four monitoring data as the Port 443 request amount (d_{12}), the HTTPS request amount (d_{13}), the odd-place login amount (d_{21}), and the privilege-required operation amount (d_{22}), can be interpreted as follows. By taking advantage of causality (i.e., relation R_1), the association between the HTTPS requests and the OpenSSL responses is identified as $I_1 = (0.83, 0.54, 0.95)$ where the high *sup* and *conf* value indicates the evident co-appearance between HTTPS requests and the corresponding responses observed on port 443 (the well-known port for HTTPS protocol). In addition, the *amb* value is 0.95 which refers to a very strong correlation between the two behaviors. By leveraging the service dependency (i.e., relation R_2), the association between the odd-place logins and the unexpected privilege escalations is identified as $I_3 = (0.34, 0.65, 0.90)$ that indicates a strong correlation (*amb* = 0.90) existing between the two behaviors at a noticeable rate (*sup* = 0.34, *conf* = 0.65). It implies the privilege-required operations (e.g., service subscription alteration or user password modification) performed by the "CSC" logged in from odd places with correct password. Moreover, the inter-relation association of the two identified associations is computed as $I_2 = (0.45, 0.44, 0.95)$ which reveals that a portion (*sup* = 0.45, *conf* = 0.44) of the user requests (i.e., the port 443 activities) are very likely to result in these privileged operations (*amb*=0.95).

Case II: The SambaCry Attack.

In practice, many undisclosed threats that can stealthily undermine the security compliance of cloud services are hard to be monitored as no knowledge about the threats is available. For instance, the SambaCry attack is a recently exposed security threat that can violate the cloud service's security compliance [NVD17]. The attack exploits the vulnerability of Samba services (version $3.5.x \sim 4.6.4$) that is widely deployed in cloud for offering file and printing services. By uploading a crafted library encapsulated with malicious codes to a writable shared folder, an attacker can remotely subvert the cloud systems and perform arbitrary operations (e.g., privilege escalations). Nevertheless, it is hard to monitor such a threat in cloud due to a lack of the specific attack details. To demonstrate *Flashlight*'s merit of identifying the monitoring path for helping to monitor unknown security threats with existing monitoring information, we conduct a case study on identifying the monitoring path regarding the abstracted SambaCry attack that simulates an unknown cloud security threat.

To support indirectly monitoring the threat, *Flashlight* identifies the monitoring path with the collected information as follows.

Step 1. Reducible intra-relation data association: With the obtained association information (i.e., I_1, I_3, and I_5) derived in the previous case study, *Flashlight* does not need to rerun the intra-relation association process and thus can directly proceed to carry out the inter-relation data association.

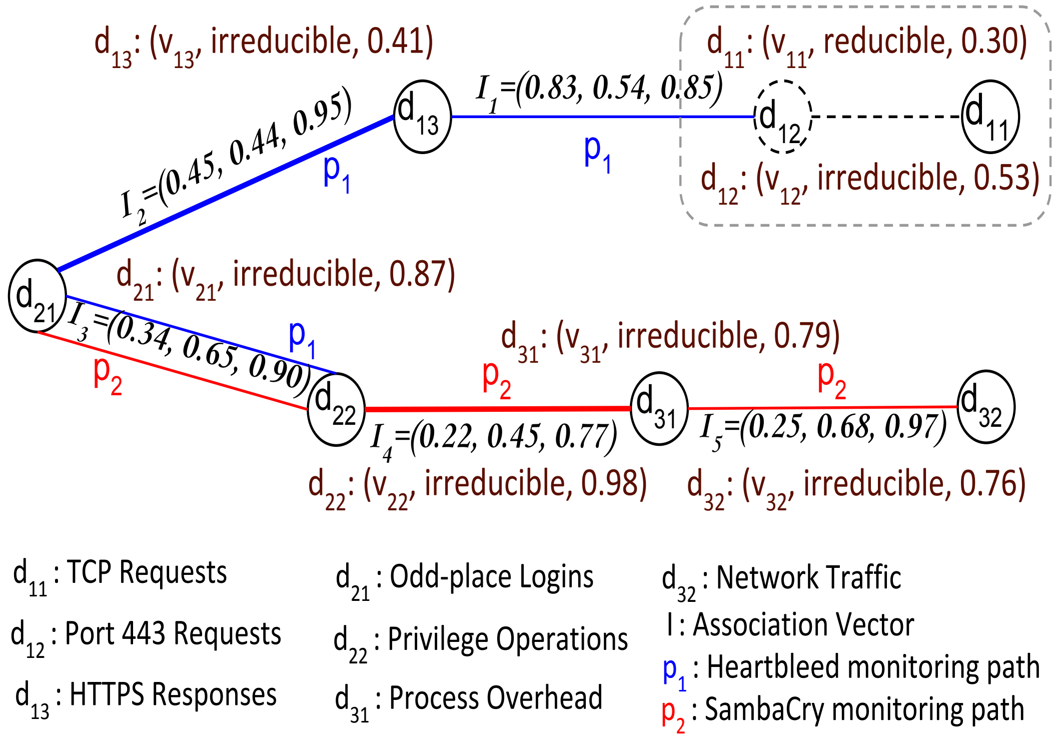
Step 2. Weighted inter-relation data association: *Flashlight* performs weighted inter-relation data association by considering $\mathcal{A}|_{d_{32}}^{d_{31}}$ as the *precedent*, as a strong data correlation between the process overhead (d_{31}) and the network traffic (d_{32}) is indicated by a high $amb(\mathcal{A}|_{d_{32}}^{d_{31}})$ value (0.97) in the computed I_5.

By comparing the identified intra-relation association weights, *Flashlight* selects $\mathcal{A}|_{d_{22}}^{d_{21}}$ as the *subsequent* for the $\mathcal{A}|_{d_{32}}^{d_{31}}$ based on the inequality of $W(\mathcal{A}|_{d_{22}}^{d_{21}}) = 0.925 > W(\mathcal{A}|_{d_{13}}^{d_{12}}) = 0.470$. Therefore, the weighted inter-relation data association vector for $\mathcal{A}|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}$ is computed as: $sup(\mathcal{A}|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}) = 0.22, conf(\mathcal{A}|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}) = 0.45$, and $amb(\mathcal{A}|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}) = 0.77$. Thus, the inter-relation (R_2 and R_3) data association $\mathcal{A}|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}$ is identified as $I_4 = (0.22, 0.45, 0.77)$ which is regarded as the monitoring path p_2 for supporting indirectly monitoring the SambaCry attack.

The identified monitoring path p_2, which contains monitoring data as the odd-place login amount (d_{21}), the privilege-required operation amount (d_{22}), the process overhead (d_{31}), and the network traffic (d_{32}), can also be explained as follows. By examining the abnormality (i.e., relation R_3), the association between the high overhead of a particular process and the high network traffic is identified as $I_5 = (0.25, 0.68, 0.97)$ where the high *amb* value 0.97 refers to a very strong correlation between them at an unnoticeable rate (*sup*=0.25, *conf*=0.68). As aforementioned, the association between the odd-place logins and the privilege-required escalation is identified by leveraging the service dependency (i.e., relation R_2) as $I_3 = (0.34, 0.65, 0.90)$ that reveals the privilege-required operations caused by the users log in from odd places. The inter-relation association between the two data associations is identified as $I_4 = (0.22, 0.45, 0.77)$ which indicates the abnormal high traffic closely correlating with the odd-place login user's behavior (*amb*=0.77) of initiating the particular brute-force attack process secretly (*sup*=0.22, *conf*=0.65). Specifically, the attack is stealthily mounted by subverting the Samba service with uploading a crafted library file of attack codes to the user's writable shared folders.

Monitoring path representation

Flashlight introduces a novel schema termed monitoring property graph for representing the identified monitoring paths p_1 and p_2 together with related monitoring information over the given monitoring data set in Figure 4.5. With the monitoring property graph, related monitoring information can be comprehensively characterized in a quantitative way. Every collected monitoring data is represented by a vertex annotated with a key-value pair where the key is the data ID and the value is a vector representing the data value and data attributes (reducibility and criticality). For example, the monitoring data of an HTTPS response message is denoted by a key-value pair as $d_{11} : (v_{11}, reducible, 0.30)$, where d_{11} is the key and $(v_{11}, reducible, 0.30)$ is the value key representing the data value (v_{11}), the data attributes (reducibility = *reducible* and criticality = 0.30). Noticeably, the data



d13: (v13, irreducible, 0.41)
d11: (v11, reducible, 0.30)
I1=(0.83, 0.54, 0.85)
p1
d12: (v12, irreducible, 0.53)
I2=(0.45, 0.44, 0.95)
p1
d21: (v21, irreducible, 0.87)
I3=(0.34, 0.65, 0.90)
p1
p2
d31: (v31, irreducible, 0.79)
p2
p2
I4=(0.22, 0.45, 0.77)
I5=(0.25, 0.68, 0.97)
d22: (v22, irreducible, 0.98)
d32: (v32, irreducible, 0.76)
d11: TCP Requests
d12: Port 443 Requests
d13: HTTPS Responses
d21: Odd-place Logins
d22: Privilege Operations
d31: Process Overhead
d32: Network Traffic
I: Association Vector
p1: Heartbleed monitoring path
p2: SambaCry monitoring path

Figure 4.5: The generated monitoring property graph regarding monitoring two stealthy cloud security attacks

reduction is shown as a dotted line representing the reducible data d_{11} and its association with d_{12} in Figure 4.5. Besides, every data association is represented by an edge annotated with an association vector. For instance, data association between d_{12} and d_{13} (i.e., $A|_{d_{13}}^{d_{12}}$) is denoted by an edge annotated by the identified association vector $I_1 = (0.83, 0.54, 0.85)$ where each value respectively represents the value of *support (sup)*, *confidence (conf)*, and *ambiguity (amb)*. Moreover, the identified monitoring paths can also be represented in this graph, namely monitoring path $p_1 : A|_{d_{13}}^{d_{12}} \uplus \mathcal{A}|_{d_{22}}^{d_{21}}$ (highlighted with blue color) and $p_2 : A|_{d_{22}}^{d_{21}} \uplus \mathcal{A}|_{d_{32}}^{d_{31}}$ (highlighted with red color). Both monitoring paths contain four different monitoring data.

Discussions The case studies demonstrate *Flashlight*'s advantages of identifying monitoring paths in different situations. The details of running the indirect cloud monitoring approach with the identified monitoring paths (p_1 and p_2) follow the descriptions in [Zha+17] and are not further detailed here. Besides, *Flashlight* also has several characteristics which are worth mentioning as follows.

Flashlight takes advantage of the basic knowledge (i.e., data relations and data attributes) to identify monitoring paths which are challenging to be achieved by pure manual analysis. As security experts require to have very in-depth knowledge for performing the manual analysis [NCR02], it is very likely for the experts to overlook some important monitoring paths without sufficient expertise.

While machine learning approaches (like principal components analysis) can also be used for identifying data associations, the learning process is not only incurring tremendous computational overhead but also subject to the scale of the monitoring data set. As the scale/complexity increment of cloud systems, its applicability is substantially exacerbated. By contrast, *Flashlight* adopts a statistical-based approach for identifying monitoring paths to circumvent the aforementioned challenges. In fact, *Flashlight* supports executing the intra-relation association processes in a parallel mode which is particularly suitable for applying in cloud scenarios.

Besides, *Flashlight* can even support fine-grain monitoring path identifications by leveraging the monitoring property graph. Specifically, when monitoring data involves multiple data relations, the multi-graph property of monitoring property graph enables to replace the vertex representing that multi-relation data with several *virtual vertices* where each *virtual vertex* represents the monitoring data involving one single data relation. Then, the monitoring path identification process can be performed as usual.

Overall, *Flashlight* is proposed for working in a static scenario where all monitoring data stays unchanged during the path identification process. However, it can be adapted for managing the dynamic scenario as the static scenario can be regarded as a snapshot of the monitoring status in a dynamic cloud system at a specific time instance. The compiled set of such snapshots (at predefined intervals) can help capture transitional behaviors.

Threats to validity While *Flashlight* can theoretically manage to identify monitoring paths at any length, proper thresholds on the *support/confidence/ambiguity* should be predefined for obtaining meaningful monitoring paths. As the increment of the path length, the co-appearance rate of monitoring data keeps dropping down. Without the appropriate threshold settings, the performance of *Flashlight* might be undermined as it costs excessive overhead to identify an unnecessary overlong monitoring path.

4.6 Related Work

Monitoring path identification for securing cloud services is a novel topic with little specific coverage and with existing related work focused on general cloud monitoring and the generic path discovery.

The existing cloud monitoring approaches developed by taking advantage of different factors (e.g., data relations, timing characteristics, or topology properties) are applicable only when the monitoring parameter/data is directly accessible. For example, Deng et al. [Den+14] proposed an access policy-based security tracing framework called *LACT* to monitor leaked access credentials with respect to cloud storage services. However, this framework requires all the relevant monitoring data to be accessible. Du et al. [DGY10] proposed the *ROSIA* framework to monitor malicious service providers within a multi-tenant cloud system by examining the consistency of data flow. Nevertheless, *ROSIA* requires the whole data flow to be accessible. A large number of monitoring methodologies (e.g., *RAFT*[Bow+11], *MaaS*[ML13]), *PhisEye*[HKB16] are proposed for monitoring different aspects of cloud services (e.g., service elasticity, transient violations, or data robustness), while these methodologies become invalid when losing the direct access to the related monitoring data. Many other monitoring methodologies (e.g., [Wu+13], [Sae+14], [Agr+07]) are developed by adopting a distributed model to monitor cloud services by running a series of local cloud monitoring tasks. Unfortunately, the proposed methodologies also require direct access to all needed monitoring information.

Even though not directly related to the topic, we found some research works which might be helpful for addressing the monitoring path identification question. A small number of approaches are proposed for dealing with the path discovery task on an ad-hoc basis. For instance, *CloRExPa* [DLS14] is proposed to trace the malicious data modifications inside cloud systems by making use of the proposed virtual machine state/action graph. In a cloud scenario, it is challenging to construct the state/action graphs when necessary information is unavailable. Ravindranath et al. [Rav+12] proposed a discovery technique to capture the path of the mobile user transaction by examining the time latency. Yet, this technique is developed for mobile systems rather than cloud systems. Several discovery approaches (e.g., [NCR02], [DY14], [Liu+14]) are proposed by analyzing relevant statistical properties for locating obfuscated sequences, searching demanded content, or reconstructing attack scenarios, respectively. While, these approaches are inapplicable for the cloud scenario where the analysis process takes a lot of effort. Besides, some discovery techniques are developed by adopting taint-based methods. Naderi et al. [Nad+15] proposed a hybrid taint-based approach to discover SQL injections, while Yamaguchi et al. [Yam+15] proposed a taint-based approach to discover the patterns of unknown vulnerabilities. However, it is hard to perform the taint process on the monitoring data collected by security monitors from a monitoring path identification perspective. Various inference-based discovery methodologies are also proposed. Olivo et al. [ODL15] proposed an inference methodology to infer a new type of Denial of Service (DoS) attacks by using particular database attributes. Unfortunately, the methodology is unsuitable for working in a complex cloud scenario, as it is hard to obtain all the required information. Additionally, Yadwadkar et al. [YAK14] proposed an approach to infer victim cloud tasks by using support vector machine to study system features, but the approach incurs expensive system overhead for performing the machine learning process and the feature selection process.

4.7 Conclusion

Indirect cloud monitoring is an emerging and promising security approach to effectively monitor cloud services where particular monitoring information is inaccessible. To indirectly monitor cloud services, the prerequisite is to perform monitoring path identification which is a novel topic and still in an incipient stage in the research community. In this thesis, *Flashlight* is proposed as a systematic approach for addressing the monitoring path identification problem by synthetically leveraging the complex relations and attributes of collected monitoring data. By performing the case studies on real security threats, *Flashlight* demonstrates its efficacy of identifying monitoring paths for security professionals indirectly monitoring cloud services without directly accessing to particular monitoring information.

The advantages of our proposed approach are the capability of managing different types of data relations and the possibility to deal with various path identification scenarios. In the future, we plan to automate *Flashlight*'s monitoring path identification process. Overall, this thesis offers a good starting point for investigating the monitoring path identification question for securing cloud services.

5 Optimizing Data Reliability for Sound Monitoring Performance

In Chapter 4, we propose a monitoring path identification mechanism to ascertain proper input information to enable performing indirect security monitoring on cloud services. In practice, the monitoring information that is collected by various cloud monitors and used by the indirect monitoring mechanism is often susceptible to varied reliability issues (e.g., monitor malfunctions, data corruptions, or data tampering). Nowadays, cloud services have become powerful enablers for a variety of smart computing solutions supporting multimedia, social networking, e-commerce and critical infrastructures among others. Consequently, as we increasingly depend on the cloud, the need exists to ensure its effective role as a trustworthy services platform. Towards this objective, cloud monitoring mechanisms cannot simply assume that the collected information is completely reliable and correct. Hence, how to obtain reliable information for implementing cloud monitoring is still an open issue. In this chapter, we propose *Whetstone* as a novel approach to address the gap where an efficient approach for ascertaining reliable values from monitoring data sets is required. Specifically, we introduce a statistical approach to filter defective data from the collected data set. Besides, we develop an optimization approach to quantify the reliability of the collected data by leveraging the value deviation of the collected data. Finally, we propose a weighted aggregation approach to generate the reliable value based on the obtained information. We evaluate the proposed approach with different experimental configurations. The experiment results demonstrate the efficacy of our approach for successfully generating the maximum likelihood reliable information for improving the overall reliability of the indirect cloud monitoring mechanism.

5.1 Introduction

Cloud services, by virtue of providing transparent access to back-end distributed resources, are increasingly underpinning a variety of smart computing projects. For example, a large-scale Internet of Things (IoT) network may utilize cloud services to process the massive data collected by the attached IoT devices which lack the requisite computational resources to locally process the data [KLS14]. Similarly, a smart community may desire an intelligent carpooling service to reduce carbon dioxide (CO_2) emissions for protecting the environment. To this end, the carpooling service takes advantage of a cloud service to compute optimized taxi dispatching plans [Zha+16a]. Moreover, a smart grid achieves to offer sustainable and economic electricity distribution by utilizing cloud services to manage the communication of heterogeneous information [Bae+15]. As the cloud enabled applications proliferate, the increasingly dependency on the cloud also portends the need to ensure the cloud as a dependable services platform.

In this context, the monitoring of the cloud operations (for functionality, resource optimization and especially the detection of anomalous behaviors) forms an essential basis for securing cloud services. As a result, a large number of monitoring mechanisms have recently been proposed [DGY10][Bow+11][Wu+13][Mol+13] [Den+14] [Zha+17]. However, virtually all existing monitoring schemas assume that the information collected by security monitors is reliably correct. However, this assumption can be fallacious for a variety of reasons, e.g., security monitor malfunctions, unpredictable data/network corruptions, or malicious data tampering [Li+14b]. For instance, security monitors deployed for collecting monitored data might encounter the problem of transient malfunctions or failures. As a result, the collected data is unreliable or even completely flawed. Additionally, the monitored data might be corrupted during recording/transmission phases by unpredictable factors such as communication channel congestion or noise. Moreover, the monitored data might be intentionally tampered by attackers for bypassing security mechanisms or for fabricating necessary preconditions for performing subsequent attacks [Ger+15]. Therefore, to ensure that existing monitoring mechanisms generate correct cloud services monitoring results, the key point is to ascertain the reliability of the collected data.

In order to improve data reliability, existing methodologies target obtaining reliable data value in two steps [KOS11][RY12]. The first step is to measure the value of a given target multiple times. The second step is to collate the multiple recorded values to generate a reference representative value. The commonly adopted aggregation technique is termed *Majority Voting* that procures reliable values by taking advantage of a voting process [Li+16]. Specifically, the value with more votes (i.e., occurrence frequency) contributes more to the final procured value in the aggregation process. The value generated by using this technique is the reference “reliable" value with respect to the monitored target. Naturally, the value occurrence directly affects the reliability of the generated mean value. However, a major drawback of this technique is in overlooking the important fact that the reliability of the collected values is distinctive [Yu+14]. Supposing that an unreliable value (e.g., erroneous data) occurs in the collected value set for many times, the occurrence-based technique fails to generate reliable value. In other words, this technique is only applicable when every collected value has the same level of reliability. Therefore, an approach that is able to generate reliable values in the presence of raw values with differing reliability degrees is needed.

To address this gap, we propose a novel methodology termed *Whetstone* for obtaining the reliable monitoring information from the collected data set in this thesis. To this end, we first adopt a data cleansing approach to filter the unreliable data by making use of statistical properties of the monitored data. Then, we propose a data reliability quantification approach by leveraging the relationship of data reliability and value deviation. Finally, we develop a novel weighted aggregation approach to generate reliable values based on the reliability of collected values.

To the best of our knowledge, our approach is the first work proposed for deriving reliable data to support monitoring cloud services. In summary, we make the following contributions:

1. We propose *Whetstone* as a novel methodology to generate reliable monitoring values from collected data by considering the reliability degree of collected data individually.

2. We propose a quantification approach for ascertaining the reliability of the monitored data based on an optimization model and theoretically prove the correctness of the determined reliability results.

3. Our experimental results demonstrate the effectiveness of the proposed approach to generate the reliable value via a tunable optimization coefficient.

The remainder of this chapter is organized as follows. Section 5.2 describes the considerations of obtaining reliable value from collected data. Section 5.3 models the reliable value generation problem. Section 5.4 details the design of our proposed approach. Section 5.5 evaluates the effectiveness of our proposition with experiments and makes discussions. Section 5.6 reviews the related work.

5.2 Background

We first review the challenges of developing an effective methodology to obtain reliable monitoring values. Next, we present the main observations that underpin the development of our proposed methodology.

5.2.1 Challenges

Monitored data collected by security monitors contains a variety of useful information (e.g., abnormal workload variation, unusual service customer logins, or occasional virtual machine exceptions) for monitoring cloud services. With reliable monitoring data, cloud monitoring mechanisms are expected to generate correct monitoring results. In reality, the monitored data suffers from various reliability problems, such as failures of security hardware (e.g., monitoring devices), errors of data flows (e.g., communication channels), or manipulations of data sources (e.g., system log tampering). To address these problems, a common solution is to repeatedly measure the value of a given target and aggregate the values for deriving a reliable value. As mentioned in the introduction, a widely adopted aggregation technique is developed based on the *majority voting* principle of the value receiving more votes also more important for the aggregation [Li+16]. In practice, the occurrence of the collected value is generally utilized as the vote and the mean of collected values is generally regarded as the reliable value derived by this technique. The major problem of the technique is that every collected value is considered as uniformly reliable in the aggregation process. However, erroneous values and normal values have completely different significance from a reliability perspective. As a result, it is questionable to simply aggregate the collected values for the reliable value generation by merely considering the value occurrence.

The reliability of monitoring data is the value that represents the degree of the collected data free from errors during a monitoring process. From a reliability perspective, the distance from the collected values to the true value varies. As depicted in Figure 5.1, the collected values of a monitored data (denoted by green square / red parallelogram / diamond / triangle) are all deviated from the real value (denoted by blue point) to some extent. If the collected values have tiny deviation (e.g., locating inside the small radius (r_1) dashed circle), the aggregated value (denoted by magenta point v_1) is closer to the real value. While the collected values have greater deviation (e.g., locating inside the greater radius (r_2) dashed circle), the aggregated value (denoted by magenta point v_2) is less closer to the real value. These two cases highlight that the reliability of the generated value is directly affected by the reliability of the collected data. As a result, two main problems need to be solved for deriving a reliable value from the collected data.

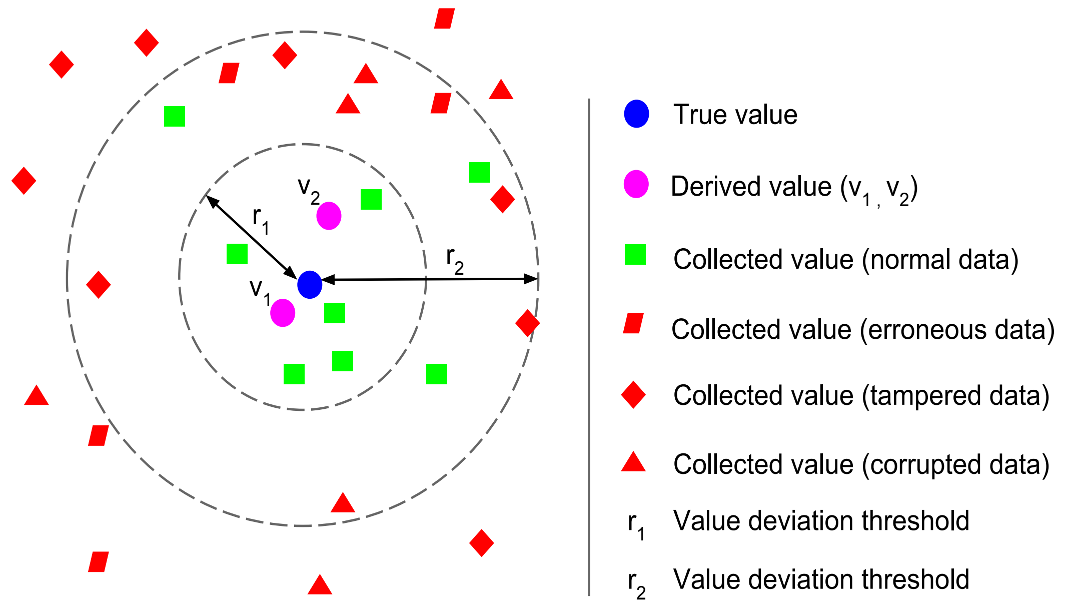

v_2
r_1
v_1
r_2
True value
Derived value (v_1, v_2)
Collected value (normal data)
Collected value (erroneous data)
Collected value (tampered data)
Collected value (corrupted data)
r_1 Value deviation threshold
r_2 Value deviation threshold

Figure 5.1: A reliability view of the data value collected by cloud monitors

The first problem is that the collected data might contain defective values which involve various reliability issues as aforementioned. Hence, a data cleansing approach that supports filtering defective data from the raw data set is needed for obtaining a cleansed data set. Even with the cleansed data, another challenge is to develop a proper aggregation approach that can utilize the cleansed data for generating reliable values.

5.2.2 Observations

Two issues are frequently observed in cloud monitoring activities which also constitute the basis for the proposed methodology.

The first observation is that the population of the monitored data value approximately follows a Gaussian distribution given that the number of collected values is sufficiently large [Guo+06]. This high volume data collection is typical for cloud security monitors and thus the relevance of Gaussian distributions. According to the statistical property of Gaussian distribution, if a value is significantly deviated from the mean value (i.e., the true value), it has a high likelihood to be defective.

The second observation is that there is an inverse relationship between the data reliability and the value deviation regarding a monitored data [Li+14b]. Namely, the value of reliable monitored data is closer to the true value while the value of unreliable monitored data is distant from the real value. As an example, system event logs contain valuable information for securing cloud services. In normal situation, security auditors/experts can utilize the reliable information extracted from the event logs (e.g., the log of failed login events) to discover potential attack behaviors. However, if the log has been tampered by attackers, the critical information indicating brute-force password attacks against customer accounts can be deliberately removed. Thus, the tampered data contains the information that is quite deviated from the real situation reflected by the reliable data.

These observations highlight the typical characteristics of the monitored data. In the next section, we introduce the proposed approach for generating the reliable value of the monitored data by taking advantage of these observations.

5.3 Problem Statement

We now present the problem model of generating the reliable values of monitored data, and outline the relevant notations and terminologies adopted in the thesis.

5.3.1 Problem Model

In this thesis, we particularly consider the problem of generating reliable value of monitored data for performing security monitoring on cloud services. Structurally, we describe the problem with an input-output problem model as follows:

- *Input*
  To obtain a reliable value with respect to a monitored target T in a cloud service, a set of monitored data $D^T = \{d_1^T, d_2^T, \cdots, d_m^T\}$ ($m \in \mathbb{N}$) is collected by a deployed security monitor for m times. The collected data d_m^T denotes the m^{th} measured value of target T. The size of the collected data set D^T is m.

- *Output*
  Based on the obtained data set D^T, the output is data $\widehat{d^T}$ that represents the reliable value of target T by properly aggregating all collected data $d_1^T, d_2^T, \cdots, d_m^T$.

5.3.2 Solution Approach

To obtain the reliable value of monitored data $\widehat{d^T}$, our methodology needs to:

- Obtain a set of cleansed data $D^{'T}$ by precluding defective data (e.g., corrupted data, erroneous data, or tampered data) from the raw data set D^T.
- Measure the cleansed data's value deviation for taking advantage of the inverse relationship between the value deviation and the data reliability.
- Determine the reliability degree of every cleansed data in $D^{'T}$ as a weight for generating $\widehat{d^T}$.
- Aggregate the cleansed data and its corresponding weights to generate the reliable value $\widehat{d^T}$.

5.4 Proposed Methodology: Whetstone

We first present an overview on the *Whetstone*'s framework before detailing the design of the proposed methodology.

5.4.1 System Overview

To obtain reliable monitored data, we propose a multi-step methodology termed *Whetstone*. Prior to discussing the design details, we present the framework of *Whetstone* in Figure 5.2. The framework of our proposed methodology consists of four major steps summarized as follows.

1. *Cleanse defective data:* The proposed methodology starts by conducting a statistical preprocessing process (i.e., Grubbs' test) to filter defective data from the raw monitored data set in order to obtain a cleansed data set as the foundation for supporting subsequent procedures (Section 5.4.2).
2. *Measure data deviation:* The methodology measures the cleansed data so as to obtain the value deviation from a reference base (Section 5.4.2).
3. *Quantify data reliability:* The methodology introduces an optimization approach to quantify the reliability of every cleansed data by taking advantage of the obtained deviations (Section 5.4.2).

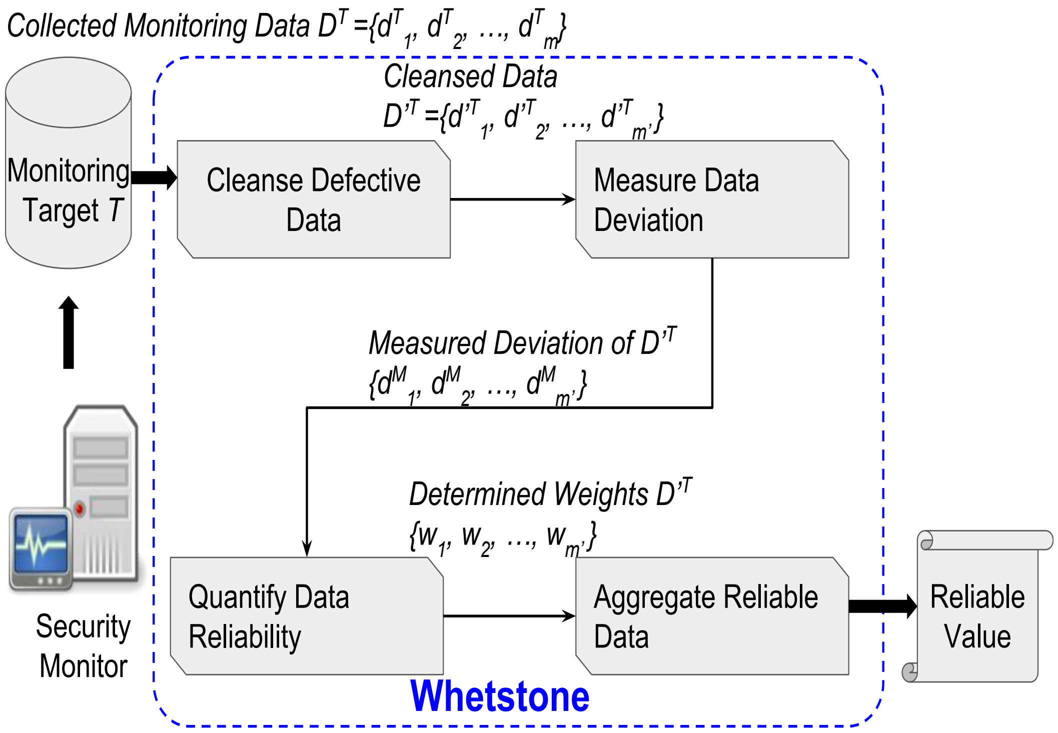

Collected Monitoring Data $D^T = \{d^T_1, d^T_2, \ldots, d^T_m\}$
Monitoring Target T
Cleansed Data $D'^T = \{d'^T_1, d'^T_2, \ldots, d'^T_{m'}\}$
Cleanse Defective Data
Measure Data Deviation
Measured Deviation of D'^T $\{d^M_1, d^M_2, \ldots, d^M_{m'}\}$
Determined Weights D'^T $\{w_1, w_2, \ldots, w_{m'}\}$
Quantify Data Reliability
Aggregate Reliable Data
Reliable Value
Security Monitor
Whetstone

Figure 5.2: The framework of the proposed methodology with four different data processing phases

4. *Aggregate reliable data: Whetstone* finally aggregates the obtained information (i.e., the cleansed data and its reliability) to generate a reliable value with respect to a specific monitored target (Section 5.4.2).

5.4.2 Design Methodology

In this part, we explain the design of the proposed methodology that enables us to generate a reliable value from the collected data regarding a monitored target.

Cleanse defective data

To obtain a reliable value, *Whetstone* first requires to cleanse all collected value so as to obtain a cleansed set for aggregating a reliable value. Considering collected values follows a Gaussian distribution [Guo+06], we propose to cleanse the collected values by utilizing the Grubbs' test which is a statistical method for identifying far-deviated data in a data set complying with the Gaussian distribution [Gru69].

For a set of collected data, the defective data denotes the most deviated data (i.e., either the greatest or the smallest one) in the m-element data population D^T. Hence, *Whetstone* first proposes two hypotheses of the data state as follows.

- H_0 : *There is no defective data in the collected data set* D^T.
- H_a : *There is exactly one defective data in the collected data set* D^T.

To test the above hypotheses, *Whetstone* applies the following two-sided Grubbs' test formula [Gru69].

$$G = \frac{\max\limits_{i=1,2,\cdots,m} \left| d_i^T - \bar{d^T} \right|}{s} = \frac{\max\limits_{i=1,2,\cdots,m} \left| d_i^T - \bar{d^T} \right|}{\sqrt{\frac{\sum_{i=1}^{m} \left(d_i^T - \bar{d^T} \right)^2}{m-1}}} \tag{5.1}$$

where $i \in \{1, 2, \cdots, m\}$, G denotes the value of the Grubbs' test, $\bar{d^T}$ represents the mean value of all collected data in D_T, and s is the standard deviation of D^T.

After determining the value of G by Formula (5.1), *Whetstone* can test the hypothesis as follows. Specifically, H_0 is rejected at the significance level α if

$$G > \frac{m-1}{\sqrt{m}} \sqrt{\frac{(t_{\alpha/(2m),m-2})^2}{m-2+(t_{\alpha/(2m),m-2})^2}} \tag{5.2}$$

In (5.2), $t_{\alpha/(2m),m-2})$ represents the t-distribution with $m-2$ degrees of freedom at the $\alpha/(2m)$ significance level. We set the value of α to 0.05 in the thesis.

Whetstone executes Grubbs' testing processes in an iterative manner until H_0 is no longer rejected at the specified confidence level α by Formula (5.2). In consequence, the data that results in the rejections of H_0 in the testing process will be removed from the raw data set. We denote the cleansed data set by $D^{'T} = \{d_1^{'T}, d_2^{'T}, \cdots, d_{m'}^{'T}\}$ ($m' \leqslant m; m, m' \in \mathbb{N}$) which is a subset of the raw data set D^T (i.e., $D^{'T} \subseteq D^T$). Accordingly, the size of the cleansed set $D^{'T}$ is m' and the number of removed data is $m - m'$.

Measure data deviation

After obtaining the cleansed data set $D^{'T}$, *Whetstone* proceeds to quantify the data reliability. As discussed in Section 5.2.2, high reliability data demonstrates the small deviation from the true value, while low reliability data demonstrates the high deviation from the true value. Therefore, the inverse relationship between data reliability and value deviation can be utilized as an effective leverage for ascertaining data reliability. *Whetstone* takes advantage of the inverse relationship to quantify the reliability of cleansed data in $D^{'T}$.

To this end, *Whetstone* needs to tackle a major problem of obtaining the value deviation, as it is challenging to make use of the inverse relationship for data reliability quantification without knowing the deviation value.

The prerequisite for obtaining a value deviation is to have a reference base that is, in theory, the true value with respect to a monitored target. Ideally, the deviation can be measured by applying distance formulas (e.g., Euclidean distance) with the known reference base. However, the ideal reference base is impossible to acquire due to the fact that the true value is unknown in practice. To deal with this problem, it is necessary to introduce an estimated value which functions as the reference base for value deviation measurement. Considering that the cleansed data in $D^{'T}$ is distributed around the true value, the mean value of $D^{'T}$ is thus meaningful in estimating the true value. Hence, we introduce an approximated reference base $\bar{d^{'}}^{T}$ for measuring value deviations as follows.

$$\bar{d^{'}}^{T} = \frac{\sum_{j=1}^{m^{'}} d_j^{'T}}{m^{'}} \tag{5.3}$$

where $j \in \{1, 2, \cdots, m^{'}\}$, $\bar{d^{'}}^{T}$ denotes the approximated reference base, and $d_j^{'T}$ denotes the data in set $D^{'T}$.

Based on Equation (5.3), the deviation of the monitored data can be measured by computing the distance between $d_j^{'T}$ and $\bar{d^{'}}^{T}$ as follows.

$$d_j^M = |d_j^{'T} - \bar{d^{'}}^{T}| \tag{5.4}$$

where d_j^M denotes the measured value deviation between the monitored data $d_j^{'T}$ and the base $\bar{d^{'}}^{T}$.

Quantify data reliability

Besides the deviation obtained by using Equation (5.4), *Whetstone* also needs a proper method to leverage the inverse relationship for quantifying the reliability of collected data in $D^{'T}$. To facilitate the data reliability quantification, *Whetstone* introduces the weight w for representing the data reliability. The definition of weight is presented as follows.

Definition 1. Weight $w_j \in [0, +\infty]$ is a positive value that is used to proportionally represent the reliability of collected data $d_j^{'T}$ (for some $j = 1, 2, \cdots, m^{'}$).

Noticeably, if the weight is close to the lower bound $w = 0$, it means that the data does not contain much valid monitoring information (i.e., the defective data). If the weight is close to the upper bound $w = +\infty$, it means that the data is absolutely reliable (i.e., the true value).

With the help of weights, the data reliability quantification problem now can be addressed by solving an optimization problem based on the inverse relationship. Specifically, the optimization problem targets finding a particular weight assignment of the cleansed data so as to yield the minimum sum of the product of data weights and data deviations. For this purpose, a data with great given deviation needs to be assigned with the most possible small weight. The correctness of such an weight assignment is supported by Theorem 1 as follows.

Theorem 1. For a finite set of data pairs (d_j^M, w_j) where d_j^M is constant and w_j is bounded (for some $j \in \{1, 2, \cdots, m^{'}\}$), the minimum sum of $d_j^M \cdot w_j$ can only be achieved on condition that a great w_j is paired with the most possible small d_j^M.

Proof. Let the deviation set D be sorted as $d_1^M < \cdots < d_p^M < \cdots < d_q^M < \cdots < d_j^M$ and the weight set W also be sorted as $w_1 < \cdots < w_p < \cdots < w_q < \cdots < w_j$ $(1 < p < q < j)$, the minimum value M is $M = d_1^M w_j + \cdots + d_p^M w_{j+1-p} + \cdots + d_q^M w_{j+1-q} + \cdots + d_j^M w_1$.

Supposing there exists a value $M^{'} = d_1^M w_j + \cdots + d_p^M w_{j+1-q} + \cdots + d_q^M w_{j+1-p} + \cdots + d_j^M w_1$ smaller than M, then it gives the following inequality as

$M - M^{'} = (d_p^M - d_q^M)(w_{j+1-p} - w_{j+1-q}) > 0$

$\because d_p^M < d_q^M$ and $w_{j+1-p} > w_{j+1-q}$

$\therefore (d_p^M - d_q^M)(w_{j+1-p} - w_{j+1-q}) = M - M^{'} < 0$

It contradicts to the given inequality. Therefore, there is no other value smaller than $M = d_1^M w_j + \cdots + d_p^M w_{j+1-p} + \cdots + d_q^M w_{j+1-q} + \cdots + d_j^M w_1$.

□

Based on the above consideration, *Whetstone* proposes the following optimization problem for determining the data reliability.

Definition 2. Given a set of measured value deviations $D = \{d_1^M, d_2^M, \cdots, d_{m'}^M\}$ and a set of weights $w = \{w_1, w_2, \cdots, w_{m'}\}$,

$$\begin{aligned} &\underset{w}{minimize} \quad f(w,d) = \sum_{j=1}^{m'} w_j d_j^M \\ &s.t. \quad f_0(w) = \sum_{j=1}^{m'} \alpha^{-w_j} = C \end{aligned} \tag{5.5}$$

where $\alpha > 1$ and $C \in \mathbb{R}^+$ is a positive coefficient. The optimization problem consists of two proposed functions. Namely, $f(w,d)$ is the proposed objective function that can be optimized by finding the particular weight assignment specified in Theorem 1. $f_0(w)$ is the constraint function that ensures the optimization of $f(w,d)$ is feasible. Given weight w is a variable varying within the range $[0, +\infty]$, we introduce an adjustable coefficient C as the bound of the sum of α^{-w_j} for making the optimization process valid.

To make the objective function optimizable, we introduce a new variable β_j to represent α^{-w_j}. As a result, weight w_j can be represented by,

$$w_j = -log_\alpha^{\beta_j} \tag{5.6}$$

To determine the optimal value, the Lagrange function of the proposed optimization problem thus can be represented based on Equation (5.5)(5.6) as

$$L(\beta_j, \lambda) = \sum_{j=1}^{m'} (-log_\alpha^{\beta_j} \cdot d_j^M) + \lambda(\sum_{j=1}^{m'} \beta_j - C) \tag{5.7}$$

Given that the sum of the equality constraint function is subject to coefficient C, we can compute the Lagrange multiplier λ of the Lagrange function (5.7) on the condition that the partiality derivative of β_j is zero.

$$\lambda = \frac{1}{C} \sum_{j=1}^{m'} d_j^M \tag{5.8}$$

Based on Equation (5.5)−(5.8), we determine the value of weight w_j as follows.

$$w_j = -\log_\alpha \frac{C \cdot d_j^M}{\sum_{j=1}^{m'} d_j^M} \tag{5.9}$$

Aggregate reliable data

With the quantified data reliability, *Whetstone* is able to aggregate the cleansed data with respect to a monitored target. According to the problem model described in Section 5.3.1, the aggregated data can be denoted by $\widehat{d^T}$. To get rid of the potential reliability bias, *Whetstone* proposes a weight-based approach to generate $\widehat{d^T}$. In specific, *Whetstone* determines $\widehat{d^T}$ by aggregating all the data in cleansed set $D^{'T}$ based on the respective reliability degree derived by Equation (5.9) as follows.

$$\widehat{d^T} = \frac{\sum_{j=1}^{m'} w_j d_j^{'T}}{\sum_{j=1}^{m'} w_j} \tag{5.10}$$

In Equation (5.10), the aggregated data d^T considers the reliability contribution of every collected data $d_j^{'T}$ in an uneven manner. Overall, the reliability of $\widehat{d^T}$ is dominated by the high-weight data that has the high possibility to be more approaching to the true value.

5.5 Evaluation

In this section, we evaluate the efficacy of the proposed methodology for generating reliable data that underpins cloud monitoring. The evaluation is conducted in two steps: 1.) We assess the effectiveness of *Whetstone* to generate the primitive result by cleansing defective monitoring data, 2.) We investigate *Whetstone*'s performance towards generating the reliable value by tuning up the value of the optimization coefficient. We first describe the settings of the evaluation. Then, we provide a discussion and interpretations of the experimental results.

5.5.1 Experimental Setting

To evaluate the performance of the proposed methodology, we perform the evaluation in the scenario where a set of collected data values is aggregated to generate a reliable monitoring value. The collected data follows the unknown Gaussian distribution whose mean is the true value with respect to a monitored target. The collected data set may partially contain defective data caused by varied reasons.

To simulate the above scenario, we adopt the following experimental settings for evaluating the proposed methodology. Specifically, we randomly generate a set positive values for simulating the values of system overhead collected by security monitors from a target virtual machine which is used for running cloud services. The generated data set follows the Gaussian distribution where the standard deviation is set to $\sigma = 1$ and the mean value is set to $\mu = 80$ which is used as the unknown true value for benchmarking *Whetstone*'s performance. To simulate the defective data, we manipulate a percentage of data in the data set by adding random offsets while keeping the rest data unchanged. Thus, we create four defective data profiles which respectively contain 5%, 10%, 15%, and 20% manipulated data.

5.5.2 Evaluation on Primitive Results

We first examine the primitive result generated by *Whetstone* in data cleanse process. In order to check *Whetstone*'s capability of generating more reliable values than the existing majority voting method, we carry out four rounds of experiments on the test data set with different numbers of simulated defective data. In each experiment, we first record the mean value of the data set where the simulated data is distant from the mean value of the Gaussian distribution μ. Such mean value is value generated by applying the majority voting method. Next, we execute the proposed methodology to cleanse the defective data and collect the primitive result which is the mean value of the cleansed data set.

Figure 5.3 depicts the primitive results that are obtained from the experiments. In this figure, we present the results of different test sets with different colors. Namely, the test set with 5% defective data is in blue, the test set with 10% defective data is in red, the test set with 15% defective data is in orange, and the test set with 20% defective data is in green. From the figure, we observe that *Whetstone* can successfully filter most defective data. Compared to the mean value of the raw data set, the mean value of the cleansed set becomes closer to the benchmarking line at $\mu = 80$ for every defective data profile. As an example of the test set with 20% defective data (shown in green bar), its mean value gets improved from 78.682 to 79.916 by carrying out the cleansed process. It is worth noticing that *Whetstone* successfully filters 15% defective data which is distant from the benchmarking line in the experiment. The other 5% defective data is overlooked by *Whetstone* as it is not greatly deviated from the benchmarking line even with adding the manipulated offsets. In the rest three experiments, we also observe the similar situation that the defective data is overlooked only when the value deviation of the defective data is quite tiny. It is worth mentioning that the primitive result obtained by *Whetstone*

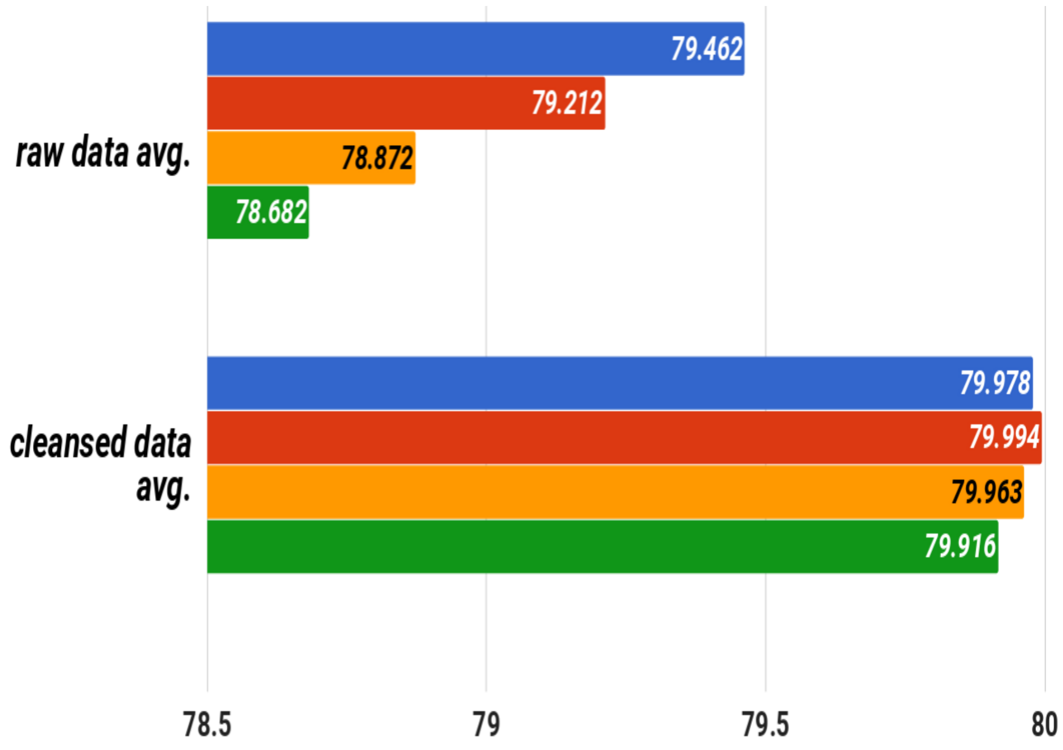


Figure 5.3: The primitive results derived by cleansing the defective data in the raw data set

outperforms the result derived by the existing work that simply aggregate all data for generating the mean of the raw data set.

5.5.3 Evaluation on Final Result

After cleansing defective data, *Whetstone* still needs to deal with the remaining data to generate a final reliable value. The remaining data (including the overlooked defective data) is deviated from the true value to different extents. To investigate the performance of the proposed methodology for generating the reliable value, we perform a series of experiments on the four cleansed test sets by tuning up the coefficient C with different values.

Table 5.1 presents the evaluation results collected from these experiments. In this table, the first column represents the amount of defective data contained in the test set that is used for evaluating the proposed methodology. The rest of the columns record the results by adopting different values of the coefficient C. For each value of C, the generated reliable value (denoted by V_O in the table) is recorded. Apart

Table 5.1: The final results generated by tuning up the coefficient with different values (N.B. $\frac{\%}{CdS}$: the percentage of defective data in the cleansed data set, V_O : Optimized Value, S_W : the symbol of the quantified weight, C : tunable coefficient)

	Coefficient Setting (C)									
$\frac{\%}{CdS}$	C=2		C=5		C=10		C=50		C=100	
	V_O	S_W	V_O	S_W	V_O	S_W	V_O	S_W	V_O	S_W
5%	79.987	+	79.989	+	79.992	+	80.014	-	80.080	-
10%	80.002	+	80.005	+	80.008	+	80.030	-	80.106	-
15%	79.975	+	79.978	+	79.982	+	80.015	-	80.167	-
20%	79.931	+	79.935	+	79.940	+	79.983	-	80.213	-

from that, the symbol of the quantified weight (denoted by S_W in the table) is also recorded in order to check the correctness of the determined weight.

From the experimental results in Table 5.1, we observe that the reliable value generated by *Whetstone* keeps approaching towards the true value before crossing the critical point that denotes the alteration of the weight's symbol from *positive* $(+)$ to *negative* $(-)$. After crossing the critical point, the generated reliable value keeps increasing with assigning greater values to C. For example, we tune up the value of coefficient C to obtain the reliable value for the test set with 20% defective data by using five respective values as $C = \{2, 5, 10, 50, 100\}$. Specifically, *Whetstone* generates a reliable value $V\_O = 79.931$ when $C = 2$. The generated value 79.931 is closer to the true value 80 than the mean value directly obtained by the cleansed data set which still contains 5% defective data as mentioned in Section 5.5.2. Tuning up C with greater values, The reliable value generated by *Whetstone* at the 95% confidence level also keeps increasing (e.g., $V\_O = 79.935$ when $C = 5$ and $V\_O = 79.940$ when $C = 10$). Apart from that, it is worth noticing the alteration of the weight's symbols. We can observe that the symbol of weight is positive when $C = \{2, 5, 10\}$ while the symbol changes to negative when $C = \{50, 100\}$. As a result, we can regard the critical point located in the range $(10, 50)$. The alteration of weight's symbol indicates the validity of the generated value. Considering the weight is used to represent the reliability degree of collected data, the weight has to be a positive value as for its practical significance. If the symbol of weight becomes negative, it indicates that the generated reliable value is invalid.

5.5.4 Discussions

From the collected experimental results, *Whetstone* successfully demonstrates its capability of generating reliable values for supporting cloud monitoring. For instance, *Whetstone* is able to derive a more reliable value than the mean value of the data set where the reliability of collected the data is difficult to known. In addition, it can manage to generate reliable values, even if the data set might contain defective data. Besides, *Whetstone* also possesses several advantages which are worth mentioning as follows.

Whetstone can generate reliable monitored data values without assuming all collected data with equal reliability. Since it is impractical to assume the reliability of collected data without a careful investigation. From a reliability perspective, it is questionable for adopting the mean value as reliable value of the collected data. *Whetstone* addresses this challenge by quantifying the reliability of collected data based on its value deviation.

Whetstone introduces a data cleanse process for further improving the reliability of generated results. By cleansing the defective data from raw data sets, the remaining data is less-deviated from the true value and thus provides a better foundation for quantifying the reliability of collected data (w) with higher precision.

Whetstone supports generating the reliable value from a set of collected monitored data in an automatic way. By keeping tuning up the value of coefficient C, the generated result keeps approaching towards the true value. Once the tuning up process yields any negative weight, the automatic process can be terminated and take the latest result as the most likely reliable value before the symbol alteration.

Whetstone is proposed for generating the reliable monitoring value based on a static data set in the thesis, while it can be adapted to manage the dynamic data flow generated by deployed security monitors in cloud systems. As a dynamic data set can be considered as the compilation of many static data set snapshots sequentially ordered by a temporal order.

5.6 Related Work

The survey work first reveals that the validity of existing monitoring mechanisms commonly suffer from the issue of collected data [Ses+04][Yin+07][DGY10][Bow+11][Mol+13][Wu+13][Den+14][MZX16][Zha+17][GES17]. For example, Seshaderi et al., [Ses+04] proposed a hash-based security technique to monitor remote device state by utilizing the memory information that is assumed to be collected from a noise-free scenario. Practically, it is challenging for security monitors to collect data under the influence of the random noise. Ma et al., [MZX16] proposed *ProTracer* as a monitoring approach to address advanced persistent threat (APT) attacks by making use of the logs of system calls and relate events. While the logs are likely to be tampered by malicious attackers in many cases, the tampered log information leads *ProTracer* to generate incorrect monitoring results. Furthermore, [Yin+07] et al., proposed a flow-analyzing security method for monitoring malwares. The proposed method is developed to function with the default prerequisite that the information captured from data flows is fully reliable. In reality, it is hard to meet such a requirement without an effective mechanism for checking the reliability of captured data. Moreover, Gulmezoglu et al., [GES17] proposed a machine learning based methodology to monitor cloud applications in terms of cache-accessing patterns. Without reliable cache information, it is hard to extract the feature vector that dominates the correctness of monitoring results generated by this methodology.

To the best of our knowledge, few work particularly targets ascertaining reliable monitoring value for achieving reliable cloud monitoring. We observe the work that is relevant to the topic of discovering reliable data for different purposes. For instance, Mahdisoltani et al., [MSS17] proposed a technique to predict reliable data for improving the reliability of storage systems by utilizing training data which requires a lot of effort to prepare. The proposed approach does not require any extra preparation and can directly work with collected data. Additionally, Zheng et al., [Zhe+15] proposed a data reliability improvement technique for assigning task packages in an optimized manner. To elevate data reliability, this technique carries out an iterative updating process that causes significant computational overhead. Likewise, Li et al, [Li+14b] proposed a method to improve data source reliability by adopting a continuous updating process. Compared to both work, the approach does not need to execute iterations and thus can reduce the computational overhead. Furthermore, Fan et al., [Fan+15] proposed a statistical technique to improve data reliability by leveraging the similarity between different topics. Unfortunately, the value of similarity is hard to obtain due to the reason that the constraints of the statistical model are sometimes too difficult to meet. By contrast, the approach does not need to meet strict constraints for applying it. Moreover, Li et al., [Li+15] proposed an optimization method to improve data reliability while it is hard to derive the suitable values of the optimization parameters without expert knowledge. However, the approach supports generating reliable value by using the suitable optimization parameter.

5.7 Conclusion

Acquiring a reliable value of monitoring data is the key factor that affects the correctness of monitoring results generated by cloud monitoring mechanisms. For performing rigorous cloud monitoring, the methodology to obtain the reliable value of the collected data is still a challenge. Therefore, we propose *Whetstone* as a novel multi-step approach to address this gap.

To obtain the reliable monitoring value of monitored data, *Whetstone* starts with cleansing the defective data in collected data set by taking advantage of the significant deviation between the value of defective data and the true value. Afterwards, *Whetstone* quantifies the reliability of the cleansed data by leveraging the relationship of the monitored data reliability inversely proportional to its value deviation from the true value. Finally, *Whetstone* successfully generates the reliable value of the collected data by aggregating the cleansed data with its reliability in a weighted manner.

The merits of the proposed approach are the capability of generating reliable value of the collected data and supporting to ascertain the reliable value by tuning up the value of the constraint coefficient. In the future, we plan to adapt *Whetstone* to manage categorical type cloud monitoring data. Overall, the thesis offers a novel angle for obtaining reliable values of monitored data to support cloud monitoring.

6 Summary and Conclusion

Cloud computing is a ubiquitously adopted computing paradigm to facilitate provisioning various online services to both business and individual users for meeting the different user requirements. Underpinning by the novel computing technology, a plethora of cloud systems are developed to simultaneously deliver different services to multiple CSCs with many attractive advantages such as optimal resource utilization, high service efficiency, rigorous virtualized isolation, or elastic service adaptation. Because of the appealing advantages, cloud services have deeply integrated into people's daily lives and inevitably imposed profound influence on the modern world. As the proliferation of cloud services, an important concern that is about the service security assurance has arisen. To address the security concern, effective cloud monitoring mechanisms are required for protecting the running services from realistic security threats. The monitoring mechanisms need to accurately identify security problems (e.g., obvious attack behaviors, suspicious exploitation traces, or vulnerable system configurations) by taking advantage of the information collected by cloud monitors. Hence, how to properly make use of the collected information plays a critical role for achieving sound cloud monitoring.

A large number of monitoring mechanisms have been proposed to improve the security assurance of cloud services. The approaches are commonly making use of the collected information for fulfilling monitoring tasks. A major drawback of the proposed mechanisms is that the key information used by these mechanisms is often difficult, if not impossible, to be collected in a real cloud system. In consequence, the mechanisms are unable to effectively monitor cloud security problems due to missing the access to the key information. A promising solution to address the problem is to draw inferences from available information (i.e., collected monitoring information and existing data relations) to achieve indirectly security monitoring on target services. For the indirect security mechanism, monitoring results that are generated by an inference process suffer from the reliability problem. However, due to the increasing complexity of cloud systems, it is hard to develop an effective indirect security mechanism that can generate reliable monitoring results.

This thesis investigated the topic of achieving indirect security monitoring on cloud services from four points of view: First, from a security practice perspective with the goal of revealing the challenge to directly collect valuable information with respect to monitor subtle security threats in the cloud. Second, from a monitoring perspective with the goal to explore if and how the collected monitoring information and the existing data relation (i.e., data dependencies) can be leveraged to infer the key information that is hard to be collected by security monitors in the real cloud. Third, from a feasibility perspective with the goal to study how can the input information that is required for executing the indirect monitoring be identified in raw monitoring data sets. And fourth, from a reliability perspective with the goal to assess how the information that involves indirectly monitoring on target services can be improved for generating reliable results. In summary, the thesis investigated the following research questions along with the corresponding contributions.

Research Question 1 *(RQ*1*): How difficult is it to effectively secure cloud services against contemporary security threats with existing monitoring solutions?*

Existing security solutions that are originally proposed for monitoring traditional threats are widely adopted for securing cloud services. Unfortunately, cloud threats are different than the traditional threats without incurring evident traces. In other words, the subtly crafted cloud threats demand considerable effort together with extensive expertise to monitor them. Existing solutions cannot fulfill monitoring tasks without the support of useful monitoring information regarding evident attack traces. As a result, they cannot manage the evolving threats in the cloud.

Contribution 1 *(C* 1*): Revealing the characteristics of the recent cloud threat and proposing a novel security mechanism for addressing the threat*

In Chapter 2, we perform an in-depth analysis on the cloud threats that are subtly crated by security attackers for stealthily compromising target services. We carry out a thorough investigation on the hidden security threat (i.e., application layer distributed denial of service (AL-DDoS) attack) and manage to profile the threat characteristics by analyzing the submitted service request pattern. Based on the attack characteristics, we propose a security approach by introducing a challenge-response mechanism to effectively address the threat. We implement the proposed approach on a large scale testing platform with different attack scenarios. In the end, the evaluation results demonstrate the efficacy of our proposed approach for achieving satisfactory security performance in a cloud scenario.

Research Question 2 *(RQ* 2*): What is the impediment to achieve service monitoring for improving the security assurance of cloud services when key information is inaccessible?*

Modern cloud systems are subject to a variety of practical technical difficulties and legal constraints for accessing to some internal information that is critical for performing security monitoring tasks. Due to the lack of the access to the information, the running security mechanisms cannot correctly monitor relevant cloud threats. Based on the fact that monitoring information is often holding data dependencies with other information, a promising workaround can be developed for achieving indirect cloud monitoring. However, the topic of making use of the existing dependencies to monitor inaccessible information in cloud systems has rarely been systematically discussed.

Contribution 2 *(C* 2*): A reliable schema for implementing indirect cloud service monitoring by taking advantage of service dependencies* In Chapter 3, we discuss the feasibility of inferring inaccessible information by leveraging the involving data dependency as well as quantifying the reliability of the results generated by the inference process. We present *deQAM* (**de**pendency-based **Q**uantitative **A**ggregation **M**ethodology) as a novel methodology for inferring the information value with the parameterized dependencies. Besides, we propose a bi-directional quantification model for mapping the quantitative relationship between existing dependency and the inaccessible information. We also propose a reliability assessment approach for specifying the uncertainty of the inferred results. Finally, We perform a case study to evaluate the viability of our proposition in a cloud scenario.

Research Question 3 *(RQ* 3*): Why is indispensable to ascertain the information that exerts an influence on the inaccessible information for enabling indirect cloud monitoring?*

Some information that is used as the input by monitoring mechanisms is important for effectively securing the running services in cloud systems. In some situations, selecting correct input information for monitoring security threats is apparent (e.g., selecting network throughput as the input for monitoring flooding attacks), while it is obscure to ascertain the valuable input information for monitoring on subtle security attacks that do not generate evident monitoring data. However, cloud security attacks inevitably impose either explicit or implicit influence on some correlated information that is collected by security monitors. Unfortunately, to ascertain the information with important security significance is a complex and indispensable task for performing effective indirect monitoring on the hidden cloud threats.

Contribution 3 *(C* 3*): A monitoring path identification mechanism for ascertaining the key input information for supporting indirect security monitoring on cloud services*

In Chapter 4, we investigate the potential to taking advantage of existing data relation like data dependencies for harvesting the relevant monitoring data sets termed “monitoring paths” that are useful for achieving indirect security monitoring on target services. We introduce a novel data reduction approach for removing irrelevant monitoring data regarding the threat analysis process. We present a data ascertaining mechanism for identifying the monitoring data sets as "monitoring paths" to pinpoint secret cloud threats by leveraging both the data relations and the attributes of the monitoring data. We propose a novel monitoring property graph for conveniently underpinning fine-grain monitoring path identification in a large scale scenario. Finally, we conduct two case studies of realistic cloud threats. The evaluation results demonstrate the efficiency of our approaches for harvesting monitoring paths to facilitate indirect security monitoring on cloud threats.

Research Question 4 *(RQ* 4*): How to improve the reliability of indirect cloud monitoring performance that can be seriously undermined by some practical factors?*

To implement security monitoring on cloud services, the reliability of monitoring information plays a dominating role on the overall reliability of the generated monitoring results, particularly for the results of indirect monitoring mechanisms. In practice, the information collected by cloud security monitors often suffer from many unexpected problems such as hardware malfunction, software exception, or data tampering. The realistic problem arises the serious reliability concern on the monitoring results generated by security monitoring mechanisms. A possible solution to address the reliability problem is to improve the reliable degree of the monitoring information used by the monitoring mechanism. However, improving the reliability of the collected monitoring information for underpinning effective cloud monitoring has not been seriously discussed before.

Contribution 4 *(C* 4*): A weighted optimization-based approach to achieve reliable monitoring information for improving cloud monitoring performance*

In Chapter 5, we discuss the possibility of improving the reliability of raw information that is collected by cloud monitors and later used by the proposed indirect monitoring mechanism. We present an efficient data cleansing method for filtering the erroneous data that is far-deviated from the ground truth in raw data set based on the analysis of relevant statistical properties. We also propose a quantitative optimization approach by taking advantage of the inverse relationship between data reliability and data deviation. We develop a weight-based aggregation approach to determine the high reliable monitoring information used by security monitoring mechanisms. At last, we implement the evaluation with different experimental configurations in a simulated testing environment. The evaluation results demonstrate the effectiveness of our approach by improving the overall reliability of the derived monitoring information.

With the proliferation of cloud services, it is of utmost importance for protecting the services against varied security threats with effective monitoring mechanisms. To that end, plenty of security mechanisms have been proposed over the past years for managing service security assurance. The monitoring mechanisms need to function effectively with the continuously evolving attack techniques as well as the increasing complexity of cloud systems. We have considered the realistic challenge of collecting monitoring information, the possibility of implementing indirect cloud monitoring, the feasibility of ascertaining valuable input monitoring information, the reliability of the generated monitoring results, and consequently developed effective approaches to achieve each concern. Such approaches provide the guarantee to the viability of applying dependency-based indirect security techniques to the distributed large scale modern cloud systems, thereby contributing to reliable cloud security monitoring.

List of Figures

List of Tables

Bibliography

[Aba+05] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. "Moderately hard, memory-bound functions". In: *ACM Transactions on Internet Technology* 5.2 (2005), pp. 299–327.

[Agr+07] Shipra Agrawal, Supratim Deb, KVM Naidu, and Rajeev Rastogi. "Efficient detection of distributed constraint violations". In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE. 2007, pp. 1320–1324.

[Aka15] Akamai Technologies. *Akamai state of the internet security report*. https://www.akamai.com/us/en/multimedia/documents/report/q4-2015-state-of-the-internet-security-report.pdf. [Online]. 2015.

[Alb+17] Soha Alboghdady, Stefan Winter, Ahmed Taha, Heng Zhang, and Neeraj Suri. "C'mon: Monitoring the compliance of cloud services to contracted properties". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM. 2017, p. 36.

[Ama15] Amazon Inc. *CloudWatch*. https://aws.amazon.com/cloudwatch/details/?nc2=h_ls. [Online]. 2015.

[Ama17] Amazon. *List the Available CloudWatch Metrics for Your Instances*. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/viewing_metrics_with_cloudwatch.html. [Online]. 2017.

[Ama19] Amazon. *Amazon Web Services*. https://aws.amazon.com. [Online]. 2019.

[App14] Apple. *Apple Media Advisory*. https://www.apple.com/newsroom/2014/09/02Apple-Media-Advisory/. [Online]. 2014.

[Arm+15] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. "Above the Clouds: A View of Cloud Computing". In: *Berkeley Reliable Adaptive Distributed systems Laboratory (RADLab)* (2015).

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. "Fast algorithms for mining association rules". In: *Proceedings of 20th International Conference on Very Large Data Bases*. Vol. 1215. 1994, pp. 487–499.

[Atl19] Atlassian. *Bitbucket Data Center*. https://www.atlassian.com/software/bitbucket/enterprise/data-center. [Online]. 2019.

[Bae+15] Joonsang Baek, Quang Hieu Vu, Joseph K Liu, Xinyi Huang, and Yang Xiang. "A secure cloud computing based framework for big data information management of smart grid". In: *IEEE Transactions on Cloud Computing* 3.2 (2015), pp. 233–244.

[Bia+10] Giuseppe Bianchi, Elisa Boschi, Simone Teofili, and Brian Trammell. "Measurement data reduction through variation rate metering". In: *Proceedings of the IEEE INFOCOM 2010*. IEEE. 2010, pp. 1–9.

[Bow+11] Kevin D Bowers, Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. "How to tell if your cloud files are vulnerable to drive crashes". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM. 2011, pp. 501–514.

[Col18] Louis Columbus. *Roundup Of Cloud Computing Forecasts And Market Estimates, 2018*. https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/. [Online]. 2018.

[Den+14] Hua Deng, Qianhong Wu, Bo Qin, Jian Mao, Xiao Liu, Lei Zhang, and Wenchang Shi. "Who is touching my cloud". In: *European Symposium on Research in Computer Security*. Springer. 2014, pp. 362–379.

[DGY10] Juan Du, Xiaohui Gu, and Ting Yu. "On verifying stateful dataflow processing services in large-scale cloud systems". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. ACM. 2010, pp. 672–674.

[Die99] Dierks T and Allen C. *The TLS Protocol, Version 1.0*. https://www.ietf.org/rfc/rfc2246.txt. [Online]. 1999.

[DLS14] Roberto Di Pietro, Flavio Lombardi, and Matteo Signorini. "CloRExPa: Cloud resilience via execution path analysis". In: *Future Generation Computer Systems* 32 (2014), pp. 168–179.

[DM00] Laurent DeLaurentis and Dimitri Mavris. "Uncertainty modeling and management in multidisciplinary analysis and synthesis". In: *38th Aerospace Sciences Meeting and Exhibit*. AIAA. 2000, p. 422.

[Dro19] Dropbox. *Choose the right Dropbox for you and your business*. https://www.dropbox.com/plans?trigger=nr. [Online]. 2019.

[DY14] Haitao Du and Shanchieh Jay Yang. “Probabilistic inference for obfuscated network attack sequences”. In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2014, pp. 57–67.

[EB12] Steven D Eppinger and Tyson R Browning. *Design structure matrix methods and applications*. MIT press, 2012.

[Fan+15] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. “icrowd: An adaptive crowdsourcing framework”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 1015–1030.

[FH15] Florian Fittkau and Wilhelm Hasselbring. “Elastic application-level monitoring for large software landscapes in the cloud”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2015, pp. 80–94.

[Fox+09] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. “Above the clouds: A berkeley view of cloud computing”. In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28.13 (2009), p. 2009.

[Fra+03] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and Christophe Diot. “Packet-level traffic measurements from the Sprint IP backbone”. In: *IEEE Network* 17.6 (2003), pp. 6–16.

[Gal+10] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. “Corroborating information from disagreeing views”. In: *Proceedings of the third ACM International Conference on Web Search and Data Mining*. ACM. 2010, pp. 131–140.

[Gar19] Gartner. *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019*. https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g. [Online]. 2019.

[GBK11] David Gullasch, Endre Bangerter, and Stephan Krenn. “Cache games–Bringing access-based cache attacks on AES to practice”. In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 490–505.

[GD13] Cesare Guariniello and Daniel DeLaurentis. “Dependency analysis of system-of-systems operational and development networks”. In: *Procedia Computer Science* 16 (2013), pp. 265–274.

[GD14] Cesare Guariniello and Daniel DeLaurentis. “Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis”. In: *Procedia Computer Science* 28 (2014), pp. 720–727.

[GD17] Cesare Guariniello and Daniel DeLaurentis. “Supporting design via the system operational dependency analysis methodology”. In: *Research in Engineering Design* 28.1 (2017), pp. 53–69.

[Ger+15] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. “Tampering with the delivery of blocks and transactions in bitcoin”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 692–705.

[GES17] Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. “Cache-based application detection in the cloud using machine learning”. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017, pp. 288–300.

[Git19] GitHub. *Plans for every developer*. https://github.com/pricing{#}feature-comparison. [Online]. 2019.

[Gle11] Glenn Butcher. *Atlassian subject to Denial Of Service attack*. http://blogs.atlassian.com/2011/06/atlassian_subject_to_denial_of_service_attack. [Online]. 2011.

[Goo19] Google. *Google Cloud Services*. https://cloud.google.com. [Online]. 2019.

[GP09] Paul R Garvey and C Ariel Pinto. “Introduction to functional dependency network analysis”. In: *The Second International Symposium on Engineering Systems*. Vol. 5. MIT World. 2009.

[GPS14] Paul R Garvey, C Ariel Pinto, and Joost Reyes Santos. “Modelling and measuring the operability of interdependent systems and systems of systems: advances in methods and applications”. In: *International Journal of System of Systems Engineering* 5.1 (2014), pp. 1–24.

[Gru69] Frank E Grubbs. “Procedures for detecting outlying observations in samples”. In: *Technometrics* 11.1 (1969), pp. 1–21.

[Guo+06] Zhen Guo, Guofei Jiang, Haifeng Chen, and Kenji Yoshihira. “Tracking probabilistic correlation of monitoring data for fault detection in complex systems”. In: *The 36th IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2006, pp. 259–268.

[HBB12] Khac Tuan Huynh, Anne Barros, and Christophe Bérenguer. “Maintenance decision-making for systems operating under indirect condition monitoring: value of online information and impact of measurement uncertainty”. In: *IEEE Transactions on Reliability* 61.2 (2012), pp. 410–425.

[HJS01] Martin Hiller, Arshad Jhumka, and Neeraj Suri. “An approach for analysing the propagation of data errors in software”. In: *2001 International Conference on Dependable Systems and Networks*. IEEE. 2001, pp. 161–170.

[HKB16] Xiao Han, Nizar Kheir, and Davide Balzarotti. “Phisheye: Live monitoring of sandboxed phishing kits”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 1402–1413.

[How+88] John H Howard, Michael L Kazar, Sherri G Menees, David A Nichols, Mahadev Satyanarayanan, Robert N Sidebotham, and Michael J West. “Scale and performance in a distributed file system”. In: *ACM Transactions on Computer Systems* 6.1 (1988), pp. 51–81.

[IBM19] IBM. *WebSphere Application Platform*. `https://www.ibm.com/cloud/websphere-application-platform`. [Online]. 2019.

[JC16] Nancy Jain and Sakshi Choudhary. “Overview of virtualization in cloud computing”. In: *2016 Symposium on Colossal Data Analysis and Networking*. IEEE. 2016, pp. 1–4.

[Kha+12] Sanjeev Khanna, Santosh S Venkatesh, Omid Fatemieh, Fariba Khan, and Carl A Gunter. “Adaptive selective verification: An efficient adaptive countermeasure to thwart dos attacks”. In: *IEEE/ACM Transactions on Networking* 20.3 (2012), pp. 715–728.

[KLS14] Matthias Kovatsch, Martin Lanter, and Zach Shelby. “Californium: Scalable cloud services for the internet of things with coap”. In: *2014 International Conference on the Internet of Things*. IEEE. 2014, pp. 1–6.

[KN11] Soon Hin Khor and Akihiro Nakao. “DaaS: DDoS mitigation-as-a-service”. In: *2011 IEEE/IPSJ International Symposium on Applications and the Internet*. IEEE. 2011, pp. 160–171.

[KOS11] David R Karger, Sewoong Oh, and Devavrat Shah. “Iterative learning for reliable crowdsourcing systems”. In: *The 24th Annual Conference on Advances in Neural Information Processing Systems*. 2011, pp. 1953–1961.

[Koz11] Heiko Koziolek. “Sustainability evaluation of software architectures: a systematic review”. In: *Proceedings of the joint ACM SIGSOFT Conference on Quality of Software Architectures and ACM SIGSOFT Symposium on Architecting Critical Systems*. ACM. 2011, pp. 3–12.

[Kru+05] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. “Automating mimicry attacks using static binary analysis”. In: *USENIX Security Symposium*. Vol. 14. 2005, pp. 11–11.

[Lei+12] Philipp Leitner, Christian Inzinger, Waldemar Hummer, Benjamin Satzger, and Schahram Dustdar. “Application-level performance monitoring of cloud services based on the complex event processing paradigm”. In: *2012 the Fifth IEEE International Conference on Service-Oriented Computing and Applications*. IEEE. 2012, pp. 1–8.

[Li+14a] Qi Li, Yaliang Li, Jing Gao, Lu Su, Bo Zhao, Murat Demirbas, Wei Fan, and Jiawei Han. “A confidence-aware approach for truth discovery on long-tail data”. In: *Proceedings of the VLDB Endowment* 8.4 (2014), pp. 425–436.

[Li+14b] Qi Li, Yaliang Li, Jing Gao, Bo Zhao, Wei Fan, and Jiawei Han. “Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM. 2014, pp. 1187–1198.

[Li+15] Yaliang Li, Qi Li, Jing Gao, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. “On the discovery of evolving truth”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 675–684.

[Li+16] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. “A survey on truth discovery”. In: *ACM SIGKDD Explorations Newsletter* 17.2 (2016), pp. 1–16.

[Liu+14] Chen Liu, Wesley W Chu, Fred Sabb, D Stott Parker, and Joseph Korpela. "Path knowledge discovery: Association mining based on multi-category lexicons". In: *2014 IEEE International Conference on Big Data*. IEEE. 2014, pp. 1049–1059.

[LLC19] McAfee LLC. *Navigating a Cloudy Sky*. `https://www.mcafee.com/enterprise/en-us/solutions/lp/cloud-security-report-stats.html`. [Online]. 2019.

[LLL15] Bo Li, Jianxin Li, and Lu Liu. "CloudMon: a resource-efficient IaaS cloud monitoring system based on networked intrusion detection system virtual appliances". In: *Concurrency and Computation: Practice and Experience* 27.8 (2015), pp. 1861–1885.

[Men+12] Shicong Meng, Arun K Iyengar, Isabelle M Rouvellou, Ling Liu, Kisung Lee, Balaji Palanisamy, and Yuzhe Tang. "Reliable state monitoring in cloud datacenters". In: *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE. 2012, pp. 951–958.

[MG11] Peter Mell and Tim Grance. "The NIST definition of cloud computing". In: *NIST Special Publication 800-145* (2011).

[Mic19] Microsoft. *Microsoft Azure*. `https://azure.microsoft.com/en-us/`. [Online]. 2019.

[ML13] Shicong Meng and Ling Liu. "Enhanced monitoring-as-a-service for effective cloud management". In: *IEEE Transactions on Computers* 62.9 (2013), pp. 1705–1720.

[MM03] Greg Mori and Jitendra Malik. "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA". In: IEEE. 2003, pp. 127–134.

[Mol+13] Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. "Mela: Monitoring and analyzing elasticity of cloud services". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Vol. 1. IEEE. 2013, pp. 80–87.

[MR04] Jelena Mirkovic and Peter Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms". In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), pp. 39–53.

[MSS17] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. "Proactive error prediction to improve storage system reliability". In: *2017 USENIX Annual Technical Conference*. USENIX. 2017, pp. 391–402.

[MZS18] Salman Manzoor, Heng Zhang, and Neeraj Suri. “Threat Modeling and Analysis for the Cloud Ecosystem”. In: *2018 IEEE International Conference on Cloud Engineering*. IEEE. 2018, pp. 278–281.

[MZX16] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. “ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting.” In: *The Network and Distributed System Security Symposium*. 2016.

[Nad+15] Abbas Naderi-Afooshteh, Anh Nguyen-Tuong, Mandana Bagheri-Marzijarani, Jason D Hiser, and Jack W Davidson. “Joza: Hybrid taint inference for defeating web application sql injection attacks”. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2015, pp. 172–183.

[NCR02] Peng Ning, Yun Cui, and Douglas S Reeves. “Constructing attack scenarios through correlation of intrusion alerts”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM. 2002, pp. 245–254.

[Nin+15] Jianting Ning, Xiaolei Dong, Zhenfu Cao, and Lifei Wei. “Accountable authority ciphertext-policy attribute-based encryption with white-box traceability and public auditing in the cloud”. In: *European Symposium on Research in Computer Security*. Springer. 2015, pp. 270–289.

[NIS01] NIST. “Announcing the advanced encryption standard (AES)”. In: *Federal Information Processing Standards Publication* 197 (2001), pp. 1–51.

[NVD14] NVD. *CVE-2014-0160*. https://nvd.nist.gov/vuln/detail/CVE-2014-0160. [Online]. 2014.

[NVD17] NVD. *CVE-2017-7494*. https://nvd.nist.gov/vuln/detail/CVE-2017-7494. [Online]. 2017.

[ODL15] Oswaldo Olivo, Isil Dillig, and Calvin Lin. “Detecting and exploiting second order denial-of-service vulnerabilities in web applications”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 616–628.

[Pap+15] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. "Functional dependency discovery: An experimental evaluation of seven algorithms". In: *Proceedings of the VLDB Endowment* 8.10 (2015), pp. 1082–1093.

[PE19] European Parliament and Council of the European Union. *General Data Protection Regulation*. https://eur-lex.europa.eu/eli/reg/2016/679/oj. [Online]. 2019.

[plc15] Micro Focus International plc. *PlateSpin Recon*. https://www.netiq.com/documentation/platespin-recon-42/. [Online]. 2015.

[Qi+12] Shanshan Qi, Bixin Li, Cuicui Liu, Xiaona Wu, and Rui Song. "A trust impact analysis model for composite service evolution". In: *2012 19th Asia-Pacific Software Engineering Conference*. Vol. 1. IEEE. 2012, pp. 73–78.

[Ran+06] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, and Edward W Knightly. "DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection". In: *Proceedings of the IEEE INFOCOM 2006*. IEEE. 2006.

[Rav+12] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. "AppInsight: mobile app performance monitoring in the wild". In: *The 10th USENIX Symposium on Operating Systems Design and Implementation*. USENIX. 2012, pp. 107–120.

[Ris+09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM. 2009, pp. 199–212.

[RKK04] Supranamaya Ranjan, Roger Karrer, and E Knightly. "Wide area redirection of dynamic content by internet data centers". In: *IEEE INFOCOM 2004*. Vol. 2. IEEE. 2004, pp. 816–826.

[RN12] Marko A Rodriguez and Peter Neubauer. "The graph traversal pattern". In: *Graph Data Management: Techniques and Applications*. IGI Global, 2012, pp. 29–46.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[RY12] Vikas C Raykar and Shipeng Yu. "Eliminating spammers and ranking annotators for crowdsourced labeling tasks". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 491–518.

[Sae+14] Trausti Saemundsson, Hjortur Bjornsson, Gregory Chockler, and Ymir Vigfusson. "Dynamic performance profiling of cloud caches". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM. 2014, pp. 1–14.

[Sch12] Leopold Schmetterer. *Introduction to mathematical statistics*. Springer Science & Business Media, 2012.

[Ses+04] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. "Swatt: Software-based attestation for embedded devices". In: *Proceedings of the 2004 IEEE Symposium on Security and Privacy*. IEEE. 2004, pp. 272–282.

[Sha+10] Jin Shao, Hao Wei, Qianxiang Wang, and Hong Mei. "A runtime model based monitoring approach for cloud". In: *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE. 2010, pp. 313–320.

[SO07] Stefan Seufert and Darragh O'Brien. "Machine Learning for Automatic Defence Against Distributed Denial of Service Attacks". In: IEEE. 2007, pp. 1217–1222.

[Sta+04] Angelos Stavrou, John Ioannidis, Angelos D Keromytis, Vishal Misra, and Dan Rubenstein. "A pay-per-use DoS protection mechanism for the Web". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2004, pp. 120–134.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. Pearson Education, 2011.

[TKS00] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. "Indirect association: Mining higher order dependencies in data". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2000, pp. 632–637.

[TT14] Bing Tang and Mingdong Tang. "Bayesian model-based prediction of service level agreement violations for cloud services". In: *2014 Theoretical Aspects of Software Engineering Conference*. IEEE. 2014, pp. 170–176.

[Var+12] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. "Resource-freeing attacks: improve your cloud performance (at your neighbor's expense)". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. ACM. 2012, pp. 281–292.

[Vim+14] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. "Fragmentation in presence of data dependencies". In: *IEEE Transactions on Dependable and Secure Computing* 11.6 (2014), pp. 510–523.

[VMw19a] VMware. *VMware Capacity Planner*. https://www.vmware.com/de/products/capacity-planner.html. [Online]. 2019.

[VMw19b] VMware. *VMware vCenter CapacityIQ*. https://my.vmware.com/web/vmware/info/slug/datacenter_downloads/vmware_vcenter_capacityiq/1_0. [Online]. 2019.

[Von+03] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. "CAPTCHA: Using hard AI problems for security". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2003, pp. 294–311.

[VS12] S VivinSandar and Sudhir Shenai. "Economic denial of sustainability (edos) in cloud services using http and xml based ddos attacks". In: *International Journal of Computer Applications* 41.20 (2012), pp. 11–16.

[Wal+10] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. "DDoS defense by offense". In: *ACM Transactions on Computer Systems* 28.1 (2010), p. 3.

[Wan+14] Shangguang Wang, Zhipiao Liu, Qibo Sun, Hua Zou, and Fangchun Yang. "Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing". In: *Journal of Intelligent Manufacturing* 25.2 (2014), pp. 283–291.

[Wan+15a] Bing Wang, Yao Zheng, Wenjing Lou, and YThomas Hou. "DDoS attack protection in the era of cloud computing and software-defined networking". In: *Computer Networks* 81 (2015), pp. 308–319.

[Wan+15b] Jun Wang, Zhiyun Qian, Zhichun Li, Zhenyu Wu, Junghwan Rhee, Xia Ning, Peng Liu, and Guofei Jiang. "Discover and tame long-running idling processes in enterprise systems". In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM. 2015, pp. 543–554.

[Wen+18] Jianping Weng, Jessie Hui Wang, Jiahai Yang, and Yang Yang. "Root cause analysis of anomalies of multitier services in public clouds". In: *IEEE/ACM Transactions on Networking* 26.4 (2018), pp. 1646–1659.

[Win+09] Matthias Winkler, Thomas Springer, Edmundo David Trigos, and Alexander Schill. "Analysing dependencies in service compositions". In: *Service-Oriented Computing. ICSOC / ServiceWave 2009 Workshops*. Springer. 2009, pp. 123–133.

[Wu+13] Yu-Sung Wu, Pei-Keng Sun, Chun-Chi Huang, Sung-Jer Lu, Syu-Fang Lai, and Yi-Yung Chen. "EagleEye: Towards mandatory security monitoring in virtualized datacenter environment". In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2013, pp. 1–12.

[WXW15] Zhenyu Wu, Zhang Xu, and Haining Wang. "Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud". In: *IEEE/ACM Transactions on Networking* 23.2 (2015), pp. 603–615.

[Xie+13] Y Xie, S Tang, X Huang, C Tang, and X Liu. "Detecting latent attack behavior from aggregated Web traffic". In: *Computer Communications* 36.8 (2013), pp. 895–907.

[Xin+03] Qin Xin, Ethan L Miller, Thomas Schwarz, Darrell DE Long, Scott A Brandt, and Witold Litwin. "Reliability mechanisms for very large storage systems". In: *Proceedings of the 2003 IEEE NASA Goddard Conference on Mass Storage Systems and Technologies*. IEEE. 2003, pp. 146–156.

[Xu+16] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. "High fidelity data reduction for big data security dependency analyses". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 504–516.

[XY09] Yi Xie and Shun-Zheng Yu. "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors". In: *IEEE/ACM Transactions on Networking* 17.1 (2009), pp. 54–65.

[YAK14] Neeraja J Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. "Wrangler: Predictable and faster jobs using fewer resources". In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM. 2014, pp. 1–14.

[Yam+15] Fabian Yamaguchi, Alwin Maier, Hugo Gascon, and Konrad Rieck. "Automatic inference of search patterns for taint-style vulnerabilities". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 797–812.

[Yin+07] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. "Panorama: capturing system-wide information flow for malware detection and analysis". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. ACM. 2007, pp. 116–127.

[YL05] Wei Yen and Ming-Fang Lee. "Defending application DDoS with constraint random request attacks". In: *2005 Asia-Pacific Conference on Communications*. IEEE. 2005, pp. 620–624.

[YT11] Xiaoxin Yin and Wenzhao Tan. "Semi-supervised truth discovery". In: *Proceedings of the 20th international Conference on World wide web*. ACM. 2011, pp. 217–226.

[Yu+09] Jie Yu, Chengfang Fang, Liming Lu, and Zhoujun Li. "A lightweight mechanism to mitigate application layer DDoS attacks". In: *International Conference on Scalable Information Systems*. Springer. 2009, pp. 175–191.

[Yu+14] Dian Yu, Hongzhao Huang, Taylor Cassidy, Heng Ji, Chi Wang, Shi Zhi, Jiawei Han, Clare Voss, and Malik Magdon-Ismail. "The wisdom of minority: Unsupervised slot filling validation based on multidimensional truth-finding”. In: *Proceedings of the 25th International Conference on Computational Linguistics*. 2014, pp. 1567–1578.

[Zha+14] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. “Cross-tenant side-channel attacks in PaaS clouds”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 990–1003.

[Zha+16a] Desheng Zhang, Tian He, Fan Zhang, Mingming Lu, Yunhuai Liu, Haengju Lee, and Sang H Son. “Carpooling service for large-scale taxicab networks”. In: *ACM Transactions on Sensor Networks* 12.3 (2016), p. 18.

[Zha+16b] Heng Zhang, Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. “Sentry: A novel approach for mitigating application layer DDoS threats”. In: *2016 IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2016, pp. 465–472.

[Zha+17] Heng Zhang, Ruben Trapero, Jesus Luna, and Neeraj Suri. “deQAM: A Dependency Based Indirect Monitoring Approach for Cloud Services”. In: *2017 IEEE International Conference on Services Computing*. IEEE. 2017, pp. 27–34.

[Zha+18a] Heng Zhang, Jesus Luna, Neeraj Suri, and Ruben Trapero. “Flashlight: a novel monitoring path identification schema for securing cloud services”. In: *2018 Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM. 2018, pp. 5–14.

[Zha+18b] Heng Zhang, Jesus Luna, Ruben Trapero, and Neeraj Suri. “Whetstone: Reliable Monitoring of Cloud Services”. In: *2018 IEEE International Conference on Smart Computing*. IEEE. 2018, pp. 115–122.

[Zhe+15] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. “QASCA: A quality-aware task assignment system for crowdsourcing applications”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 1031–1046.

[Zho+07] Jiehan Zhou, Daniel Pakkala, Jukka Riekki, Mika Ylianttila, et al. “Dependency-aware service oriented architecture and service composition”. In: *IEEE International Conference on Web Services*. IEEE. 2007, pp. 1146–1149.

[ZMS18] Heng Zhang, Salman Manzoor, and Neeraj Suri. "Monitoring Path Discovery for Supporting Indirect Monitoring of Cloud Services". In: *2018 IEEE International Conference on Cloud Engineering*. IEEE. 2018, pp. 274–277.

[ZN08] Thomas Zimmermann and Nachiappan Nagappan. "Predicting defects using network analysis on dependency graphs". In: *2008 ACM / IEEE 30th International Conference on Software Engineering*. IEEE. 2008, pp. 531–540.

[ZYR14] Hao Zhang, Danfeng Daphne Yao, and Naren Ramakrishnan. "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery". In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ACM. 2014, pp. 39–50.