ADVANCING SECURITY SERVICES FOR CLOUD APPLICATIONS

by

Marwa Abdelhafiz Mohamed Afifi Elsayed

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen’s University
Kingston, Ontario, Canada

September 2018

Copyright © Marwa Abdelhafiz Mohamed Afifi Elsayed, 2018
Dedication

To

My beloved mother, Laila,
The soul of my beloved father, Abdelhafiz,
My beautiful daughters, Sama and Layan,
My lovely brothers, Mohamed and Ahmed.

...with sincere love and gratitude.
Abstract

With cloud computing taking roots, Software as a Service (SaaS) is transforming the future of Information Technology (IT). SaaS is a modern pervasive software delivery model in the Cloud in which software providers host applications and provide them to consumers over the Internet. The Cloud brings operational and analytical applications together to empower software innovation. SaaS has become indispensable to the advancement of applications spanning different domains such as business, banking, and health. Although the glory of SaaS grows in popularity, an important question remains: how secure are cloud SaaS applications? Cloud SaaS applications are highly accessible, and the vast amount of sensitive data they manipulate makes them an attractive target by attackers. Nevertheless, software providers and consumers overlook critical security measures as they move to the Cloud, obliterating any gains made.

In this thesis, we address these concerns and aim to advance security services for cloud SaaS applications. Cloud Security as a Service (SecaaS) model expands the horizon to deliver security solutions over the Internet. Motivated by the rise of the SecaaS model, this research proposes a platform introducing Information Flow Control as a Service (IFCaaS) notion. The platform embeds robust and effective IFC-based security services in different phases of the software development lifecycle (SDLC) to govern end-to-end protection of cloud applications. Further, the platform
is augmented by the Cloud capabilities to provide efficient and scalable security services. Data breaches due to security vulnerabilities, insecure APIs and interfaces as well as insecure computations and unauthorized access are prevalent security issues to cloud applications. Hence, this research expands on targeting two different types of applications in the Cloud: operational and analytical. It presents two different security services and builds a framework for each service. They aim at mitigating the aforesaid security issues regarding each application type. Extensive evaluation of the proposed frameworks is conducted over benchmark applications in real-world settings. The experimental results reveal that the presented frameworks provide robust, effective, and yet efficient protection for cloud applications against prevalent security breaches. They offer significant improvement in terms of detection accuracy, performance, scalability, and resource consumption.
Statement of Originality

I hereby certify that all of the work described within this thesis is the original work of the author. Any ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

Marwa Elsayed

August 2018
Co-Authorship


Acknowledgements

First and foremost, I thank God (Allah) the most Gracious and Merciful for blessing me with guidance, strength, patience and perseverance to complete this work.

I would like to express my sincere gratitude and special appreciation to my advisor, *Prof. Mohammad Zulkernine*. I do not know how to thank you enough for being always there and being a real supporter, not only a great mentor and advisor. Thank you for your outstanding guidance, insightful discussions, and fruitful feedbacks throughout my research. I would like to extend my deep gratitude and appreciation to include my supervisory committee members, *Prof. Patrick Martin and Prof. Hossam Hassanein*. I would never have been able to complete this work without the support, trust, encouragement, understanding, advices and patience from all of you, *Prof. Zulkernine, Prof. Martin, and Prof. Hassanein*. I have been honored to be your student. I really cannot find words to express my gratitude to you. Not only have I learnt from your knowledge and experience, but I also have been inspired by your exceptional personalities. This boosted me professionally and personally.

I was so fortune for being part of Queen’s School of Computing. Special appreciation is due to *Mrs. Debby Robertson*, our graduate assistant. Thank you for being always there and for your endless encouragement. Special thanks go to *Mrs. Wendy Powley* for advocating and inspiring women in computing. I would like also to
thank my current and former colleagues in the Queen’s Reliable Software Technology (QRST) lab for providing positive work environment.

This thesis has not been possible without the unconditional real support, care, and love of my parents throughout my life. Dad and mom, thank you for all that you have been for me, all that you have done for me, and all that you are. My dearest dad, Abdelhafiz, my backbone, the best father ever, I am sincerely grateful to you for believing in me and leading me to the success way. I am missing you so much at this moment, however you are always in my heart and mind. My fantastic mother, Laila, my best role model, my best friend, I am deeply grateful to you for all the sacrifices and efforts you have made for us. Your unlimited support and sincere prayers gave me the encouragement throughout every step of the way and the strength to overcome all the hardships we have faced. My brothers, Mohamed and Ahmed, thank you for being the best supportive brothers ever. Thank you for being my source of enthusiasm. My lovely daughters, Sama and Layan, my sweetheart, thank you for your unwavering support, love and faith in me. Your smiles are the source of light and joy throughout my PhD journey. My extended family, uncles, aunts, and cousins, you are just amazing, and I am so lucky being part of you. Thank you for your encouragement and honest love. I owe all of you, parents, brothers, daughters, and extended family, the success I achieve in my life. You all are the best precious blessings God gave to me.
Special appreciation is also dedicated to all my true friends, connections, teachers who were a source of stimulation and encouragement to strive for reaching my goal.

Last but not least, I would like to acknowledge with appreciation the funding provided by Queen's University, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and the Schumberger Foundation. During my study, I was lucky for being qualified to receive the Schamberger Fellowship for women in science, technology, engineering and mathematics (STEM).

Finally, I would like to express my love and gratitude to my home country Egypt for everything it offered to enrich me till this point. I would like also to extend my love and gratitude to Canada, my second home, for welcoming me with open arms and continuing to grow and develop me in the way that I appreciate.
Contents

Dedication ......................................................................................................................... i
Abstract ............................................................................................................................. ii
Statement of Originality ................................................................................................. iv
Co-Authorship .................................................................................................................... v
Acknowledgements ......................................................................................................... vi
Contents ............................................................................................................................ ix
List of Tables .................................................................................................................... xiii
List of Figures .................................................................................................................. xiv
List of Acronyms .............................................................................................................. xvi
Chapter 1: Introduction ..................................................................................................... 1
  1.1 Motivations ................................................................................................................ 3
  1.2 Research Statement ................................................................................................... 8
  1.3 Thesis Contributions ................................................................................................. 12
  1.4 Thesis Outline ........................................................................................................... 14
Chapter 2: Background & Related Work ........................................................................ 16
  2.1 Background ............................................................................................................... 16
  2.1.1 Anatomy of Cloud Stack ...................................................................................... 16
  2.1.2 Cloud Applications: The Big Picture .................................................................. 20
  2.1.3 Need for Security ................................................................................................. 25
  2.1.4 Information Flow Control .................................................................................... 34
  2.2 Literature Survey ...................................................................................................... 35
  2.2.1 Security-aware Cloud Service Development ....................................................... 35
Chapter 3: A Taxonomy of Security as a Service ...........................................48
3.1 Introduction ..................................................................................................48
3.2 Taxonomy of SecaaS ................................................................................51
3.2.1 Service Operation ................................................................................54
3.2.2 Security Solution ....................................................................................55
3.2.3 Threat ......................................................................................................58
3.2.4 Some Observed Trends ..........................................................................58
3.3 Classification of SecaaS ............................................................................62
3.4 Comparative Analysis ...............................................................................79
3.5 Summary ....................................................................................................82

Chapter 4: Integrating Security in Cloud Application Development Cycle ....83
4.1 Introduction ..................................................................................................83
4.2 Warm-up Discussion ...................................................................................86
4.3 The Platform ...............................................................................................88
4.3.1 The Platform Architecture ....................................................................88
4.3.2 Responsibility Zones .............................................................................90
4.3.3 Provisioning Security into the SaaS Development Cycle ....................91
4.3.4 The Platform Principles ........................................................................98
4.4 Summary ...................................................................................................99

Chapter 5: Security Diagnosis as a Service for Cloud Applications ..........101
5.1 Introduction ...............................................................................................101
5.2 Overview and Assumption .........................................................................106
5.2.1 The SDaaS Operational Overview ....................................................107
5.2.2 Threat Model ........................................................................................108
The Visualization Manager ................................................................. 167
6.4 Experimental Evaluation ............................................................ 168
6.4.1 Anomaly Detection .................................................................. 169
A. Data Leakage .............................................................................. 169
B. Data Tampering ........................................................................... 170
C. Access Violations ........................................................................ 170
D. Misconfigurations ....................................................................... 171
E. Insecure Computations ............................................................... 172
6.4.2 Performance and Scalability .................................................... 172
A. Component-level Performance .................................................... 172
B. System-level Performance .......................................................... 176
6.5 Summary ..................................................................................... 178

Chapter 7: Conclusions and Future Directions ................................. 181
7.1 Summary and Concluding Remarks ............................................. 182
7.2 Limitations ................................................................................ 186
7.3 Future Directions ...................................................................... 188

Bibliography ..................................................................................... 191
List of Tables

3.1 Categorization of existing SecaaS work .......................................................... 52
5.1 Tool information.................................................................................................. 134
5.2 Specifications of source, sink, declassification methods .................................. 135
5.3 The impact of different tasks to SDaaS response time ...................................... 142
List of Figures

2.1 The Cloud stack .................................................................................................................. 17
2.2 Worldwide public cloud service revenue forecast ............................................................ 20
2.3 The types of cloud SaaS applications .............................................................................. 21
2.4 Multi-tenancy approaches in the Cloud .......................................................................... 24
2.5 Cloud operational application architecture ...................................................................... 29
2.6 Example of NoSQL injection vulnerability ..................................................................... 30
2.7 Cloud analytical application architecture ....................................................................... 32
2.8 Code snippets of security breaches in an analytical application ..................................... 33
3.1 Taxonomy of Security as a Service in the Cloud .............................................................. 52
3.2 The trends in existing SecaaS work .................................................................................. 60
4.1 The SaaS Development Lifecycle (SaaSDLC) ................................................................. 86
4.2 The platform to interweave security in the SaaS Development Cycle ......................... 89
4.3 Responsibility distribution w.r.t the platform’s security services and activities ............ 90
5.1 The SDaaS framework operational overview ................................................................. 107
5.2 The SDaaS framework components ............................................................................... 109
5.3 An example of inter-service invocation .......................................................................... 117
5.4 Examples of pointwise representation of sample statements ......................................... 119
5.5 The workflow of our diagnosis approach ....................................................................... 127
5.6 Illustration of information flow paths used in computing diagnosis metrics ................. 128
5.7 SDaaS vulnerability detection evaluation over various vulnerability types 137
5.8 SDaaS cloud environments in Amazon .......................................................................... 138
5.9 SDaaS average Response Time in Seconds for five Runs ................................................ 139
5.10 SDaaS scalability evaluation for different request loads ................................................. 143
5.11 SDaaS CPU utilization evaluation for different request loads ....................................... 145
5.12 SDaaS memory consumption for different benchmark applications ........................... 147
6.1 The SMaaS framework operational overview ............................................ 155
6.2 The SMaaS framework components .......................................................... 159
6.3 The SMaaS data pipeline ........................................................................... 164
6.4 The JSON-Schema of Information Flow Profile ......................................... 166
6.5 I/O performance evaluation over various data size of the SMaaS components ........................................................................................................... 174
6.6 Execution time evaluation over various data size of the SMaaS components ........................................................................................................... 175
6.7 The SMaaS CPU utilization evaluation ......................................................... 178
6.8 The SMaaS memory consumption evaluation .............................................. 178
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SecaaS</td>
<td>Security as a Service</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>4Vs</td>
<td>Velocity, Volume, Variety, and Veracity</td>
</tr>
<tr>
<td>SQLI</td>
<td>SQL injection</td>
</tr>
<tr>
<td>XPathI</td>
<td>XML Path Injection</td>
</tr>
<tr>
<td>XSS</td>
<td>Cross Site Scripting</td>
</tr>
<tr>
<td>CMDI</td>
<td>Command Injection</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>SIEM</td>
<td>Security Information and Event Management</td>
</tr>
<tr>
<td>IFC</td>
<td>Information Flow Control</td>
</tr>
<tr>
<td>IFCaaS</td>
<td>Information Flow Control as a Service</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Lifecycle</td>
</tr>
<tr>
<td>SDaaS</td>
<td>Security Diagnosis as a Service</td>
</tr>
<tr>
<td>SDG</td>
<td>System Dependence Graph</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Software Application Security Project</td>
</tr>
<tr>
<td>SMaaS</td>
<td>Security Monitoring as a Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>DaaS</td>
<td>Data-storage as a Service</td>
</tr>
<tr>
<td>CaaS</td>
<td>Communication as a Service</td>
</tr>
<tr>
<td>VMM</td>
<td>Virtual Machine Monitor</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>HaaS</td>
<td>Hardware as a Service</td>
</tr>
<tr>
<td>CAGR</td>
<td>Compound Annual Growth Rate</td>
</tr>
<tr>
<td>BI</td>
<td>Business Intelligence</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Not only SQL</td>
</tr>
<tr>
<td>CSA</td>
<td>Cloud Security Alliance</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CIA</td>
<td>Confidentiality, Integrity, and Availability</td>
</tr>
<tr>
<td>PbD</td>
<td>Privacy by Design</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity and Access management</td>
</tr>
<tr>
<td>DLP</td>
<td>Data Loss Prevention</td>
</tr>
<tr>
<td>BCDR</td>
<td>Business Continuity and Disaster Recovery</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>VMI</td>
<td>Virtual Machine Introspection</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Development Network</td>
</tr>
<tr>
<td>SDL</td>
<td>Security Development Lifecycle</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>DevOps</td>
<td>Development and Operations</td>
</tr>
<tr>
<td>IDE</td>
<td>Inter-procedural Distributed Environment</td>
</tr>
<tr>
<td>CFG</td>
<td>Control Flow Graph</td>
</tr>
<tr>
<td>PDG</td>
<td>Program Dependence Graph</td>
</tr>
<tr>
<td>NiFi</td>
<td>Niagara Files</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>AMS</td>
<td>Ambari Metric Service</td>
</tr>
<tr>
<td>CoT</td>
<td>Cloud-assisted Internet of Things</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

Cloud computing is a model for computing that revolutionizes Information Technology (IT) landscape. It represents a paradigm shift for delivering a myriad of IT resources ranging from infrastructure, platform, software, to networking as cloud services via the Internet or a private network. It promises impressive gains in providing IT services such as rapid elasticity, reliability, cost reduction, and quality of services [1]. Pioneering companies such as Microsoft, Amazon Web Service, Salesforce, IBM, Google, and Oracle, among others, have empowered "cloud computing as a utility" to go mainstream in IT.

According to a recent study about the future of cloud computing [2], cloud Software as a Service (SaaS) has dominant adoption that is expected to accelerate rapidly by 2026, with a presence of $55B in 2018 [3] and a compound annual growth rate of 19% [2]. SaaS has become a key enabler for a multitude of applications in different domains such as business, marketing, banking, health, government, and social media, to name a few. It has also become an integral part in everyday life.
Applications such as Microsoft Office 365, Google Analytics, Gmail, Dropbox, Facebook, and Netflix have implemented SaaS into daily routines of informing, learning, analyzing, communicating, sharing, commuting, and entertaining. SaaS guarantees software services for everyone, everywhere, with cost and efficiency gains. However, the impressive gains of such a technology shift may come at the cost of security [4, 5].

Application attacks are the prevalent type targeting cloud environments. Data security is the major roadblock to the adoption of cloud SaaS applications in security-critical domains. Cloud applications are prone to new attack venues as well as existing ones. This security risk is further magnified by the loss of control and lack of security enforcement over sensitive data manipulated by cloud applications. There is a gap between moving to the Cloud and implementing security initiatives. Cloud providers and consumers are still sacrificing security in the process as providers focus on meeting the rapid time-to-market pace, while consumers forget their cybersecurity due diligence.

This thesis addresses these concerns and aims to advance security services for cloud applications. Cloud Security as a Service (SecaaS) model expands the horizon to deliver security solutions over the Internet ranging from authentication, anti-virus, anti-malware, intrusion detection, and security event management services. This model supports many of the Cloud’s major gains such as cost reduction and quality of service. Motivated by the rise of the SecaaS model, the objective of this research
1.1. MOTIVATION

is to develop robust, effective, and efficient techniques and architectures to provide security services, protecting cloud applications from prevalent security threats.

1.1 Motivation

Recent years have witnessed an increasing momentum for cloud Software as a Service (SaaS) adoption. Modern technologies make it quite easy to develop and enrich SaaS applications for a mobile, Internet of Things (IoT), big data analytics, or Web workforce. These technologies capitalize cloud application agility, availability, and performance scalability.

With the ever-growing SaaS adoption rates, a number of critical questions related to security arise. From the consumers’ perspective, how can consumers verify the end-to-end security of their information that flow through SaaS applications, and how can consumers trust the security of such applications that are hosted and maintained by providers separate from their physical control? From the providers’ perspective, how can providers protect their SaaS applications, which may be exposed to various consumers (tenants), from prevalent security threats? This thesis attempts to elevate the trust in the security of SaaS applications and answer these questions.

Several research reports [8-10] have shown that application attacks are the prevalent type targeting cloud environments. Data breaches due to security vulnerabilities, insecure APIs (Application Program Interfaces) and interfaces as well as insecure computations and unauthorized access are considered the top critical
issues in cloud applications security [6, 7]. Technologies used in developing cloud applications leave them prone to novel attack venues as well as existing ones. In this respect, data security is exposed to serious threats due to malicious intents or inadvertent vulnerabilities. This security risk is further magnified by the loss of control and lack of security enforcement over sensitive data manipulated by cloud applications. Furthermore, several challenges originating in cloud applications may render traditional security technologies and regulations ineffective.

Web service APIs are the most preferred way of exposing SaaS (e.g., Google, Amazon, Yahoo, Facebook, and Twitter). Cloud consumers leverage cloud-based (e.g., RESTful) APIs through an Internet browser or in an aggregated fashion with other services (called a mash-up) to enrich their operational applications. These web services represent a front-end interface opening the door for web-based attacks. A study on thousands of applications run on public, private, and hybrid cloud has revealed that 96% of tested applications have one or more serious vulnerabilities [10]. Successful exploitation of such vulnerabilities not only result in a breach of the integrity and confidentiality of one tenant’s data, but also extend to cross-tenant violations [6, 7]. In this sense, information flow vulnerabilities, including SQL injection (SQLI), XML Path injection (XPathI), Cross Site Scripting (XSS), and command injection (CMDI), to name a few, are prevalent attack venues towards such cloud applications [8]. The common reason for these vulnerabilities is the insecure information flow due to the improper validation and sanitization of untrusted inputs.
used directly in performing sensitive operations. In a similar way, sensitive data insecurely flows to unauthorized operations or external storage without proper anonymization or encryption measures. Thus, an effective security analysis is significantly needed to identify these critical vulnerabilities in operational applications well before production, which, in turn, can mitigate the potential risk as well as save money and time needed for remediation efforts.

Cloud analytic applications are prone to data breaches due to insecure computations and unauthorized access as a result of vulnerable, malicious, or misconfigured nodes or tasks [7]. Analytic applications are usually deployed in clusters that provide distributed large-scale heterogeneous computing environments. Such computing environments leave analytic applications subject to weak authentication, misconfigurations, and infrastructure attacks leading to data confidentiality and integrity violations. The risk is further expanded from the coupling of data analytics with the Cloud. Nonetheless, there is still a lack of effective security monitoring and analysis measures embedded in cloud analytic clusters to detect such malicious and anomalous activities. Faced with such emerging security threats, there is a pressing necessity to gather – in near real time – incident information. Security Information and Event Management (SIEM) [155] systems have been the longstanding traditional approach for handling compliance reporting efforts for detected security incidents. However, these systems focus mainly on finding known patterns of threats, demanding skilled analysts in order to operate
effectively. SIEM systems, in turn, are difficult to maintain and handle the unique complexity of analytic clusters and the sophistication of the threat landscape. Therefore, real-time security monitoring is especially important for analytic applications to keep providers attentive, secure consumers’ assets (sensitive data), and get ahead of the aforesaid threats. An effective real-time monitoring system needs to be onboard with advanced automated capabilities to facilitate cohesive analytics of security data, provide comprehensive visibility across analytic clusters, and improve detection and response time.

Despite the research efforts that have targeted detecting and mitigating data breaches, many limitations with respect to cloud environment still exist. Researchers over the past years have focused on security-type and taint-analysis based systems. Type-based systems [98-101] offer restricted programming languages and require annotations that hinder their practical use. Taint-based systems [102-108] focus on statically vetting insecure explicit flows and ignore implicit flows. This hampers their effectiveness in settings where strong guarantee of information security is a must.

More recent approaches [55, 92-97] focused on tracking data flow at runtime. They pinpoint security breaches after an application is deployed or even after damages may have already happened. Such approaches usually entail modification to the underlying environment, which hamper their potential adoption in the Cloud.
1.1. MOTIVATION

They cause runtime performance overhead and are prone to evasion by attacks that may change their behavior during the analysis.

Other approaches advocated different techniques to detect data violations in the Cloud ranging from intrusion detection [33, 109, 110], penetration testing [111, 112], differential privacy [123], integrity verification [124-127], policy enforcement [128-131], data provenance [132-135], honeypot-based [136], to encryption-based [137] mechanisms. Intrusion detection mechanisms do not guarantee end-to-end security verification. Penetration testing treats an analyzed application as a black box; thus, it provides little assurance that an application is immune to attacks. The effectiveness of differential privacy as a widespread solution is still not proven. The integrity verification approaches require intercepting the application execution for verifying result integrity, which comes at the cost of performance penalty. Access control policies cannot prevent misuse activities breaching data security, after an access had been granted. Provenance mechanism incurs overhead due to collecting, storing, and analyzing provenance data, which can lead to impracticability.

Nevertheless, existing approaches have mainly tried to tailor traditional security techniques to fit in the Cloud environment. They still have limitations in providing scalable and effective security solutions, due to the unique requirements induced by the Cloud. With these limitations, there is a need for robust and efficient frameworks, which consider such requirements, to protect against prevalent security breaches.
1.2 Research Statement

As the momentum of cloud applications grows, security concerns grow even more. Our ultimate goal is to elevate the trust in the security of cloud applications. This thesis aims to advance security services with novel techniques and architectures to protect cloud applications. Towards achieving this goal, we propose a platform that introduces the notion of Information Flow Control as a Service (IFCaaS). We argue that information flow control (IFC) mechanisms can be embedded in different phases of the software development lifecycle (SDLC) to enhance the security of cloud applications. IFC is a classical well-known mechanism, which can verify that the information flow in a software system complies with specified security policies, to govern data integrity and confidentiality. The proposed platform delineates the groundwork for cloud-delivered IFC-based security activities and services. We propose building different frameworks for these services to provide different styles of defense against advanced threats for cloud applications. The core idea behind this thesis is summarized in the following statement:

*We believe that the incorporation of information flow analysis in providing cloud-based security services can achieve advanced in-depth defense solutions for cloud applications against prevalent security breaches. Such solutions can elevate the trust of not only cloud consumers but also providers in the security of cloud applications.*
1.2. RESEARCH STATEMENT

A key primary step towards robust security frameworks for cloud applications is to acquire knowledge of the Cloud stack, understand prevalent threats that are directed to cloud SaaS applications, and categorize existing research work to better understand what security mechanisms and features need to be in place to help mitigate those threats. Thus, in this thesis research, we also propose a taxonomy of research work in security as a service (SecaaS) in the Cloud. The taxonomy explores the current state-of-the-art with respect to three main dimensions: service operation, security solution, and threat. We further present a comparative analysis of existing research work to assess their effectiveness in terms of three key properties: visibility, control, and robustness. The study takes into consideration the implications of the service operation and the security solution on the effectiveness of the surveyed studies in mitigating the target threats.

Leveraging IFC in providing security solutions as services brings effectiveness for end-to-end security protection. However, an effective solution needs to further adhere to the requirements that are induced by the Cloud, involving: 1) adaptability to the unique characteristics (i.e., challenges) of cloud applications; 2) ease of deployment; 3) high scalability and performance; and 4) convenience and ease of management. Towards reaching these objectives, this research work spans two directions, more specifically; it proposes developing two different frameworks that adopt IFCaaS to protect cloud applications against data integrity and confidentiality violations. This is achieved by putting a trusted party in charge to
certify and inspect the security of cloud applications. These frameworks target two different types of applications in the Cloud: operational and analytical systems. They target mitigating different security issues relevant to these applications.

The first framework features cloud-delivered Security Diagnosis as a Service (SDaaS) to reveal security vulnerabilities in SaaS applications that handle operational data. It aims at mitigating the risk early in the development phase. It mainly relies on static information flow control analysis. Several challenges, which are undecidable problems hindering static analysis, originate in SaaS applications due to their dynamic and complex nature. These challenges are summarized as follows: 1) approximating an application’s runtime lifecycle; 2) capturing the control and data flow stem from inter-service interactions; and 3) handling external library invocations. As a vital part of SDaaS, we propose several modeling strategies to boost the framework in solving the aforesaid problems. SDaaS extracts data and control dependencies from an application's bytecode. These dependences are modeled in System Dependence Graph (SDG), upon which we employ information flow control and program slicing techniques for security inspection. The security inspection reveals vulnerabilities violating data integrity and confidentiality and derives several quantitative metrics about the inspected slices. These metrics are used as a basis of our rule-based approach to diagnose an application’s behavior (i.e., trustworthy, vulnerable, or malicious). Grounded on the diagnosis results, the trusted party decides whether to certify a candidate application or not. SDaaS also provides a
comprehensive report about an analyzed application’s security status in terms of potential risk, behavior diagnosis, and security risk rating. The report is beneficial to prioritize remediation efforts by focusing on the most significant vulnerabilities and critical issues.

The second proposed framework features real-time Security Monitoring as a Service (SMaaS) that employs information flow-based log analysis for detecting security anomalies in cloud analytic applications. The distinct characteristics of computations and data in the distributed large-scale analytic systems invite several challenges to develop an effective log analysis for anomaly detection solution. These challenges are summarized as follows: 1) handling log data that is characterized by the 4Vs (velocity, volume, variety, and veracity) and collected across the cluster nodes; 2) involving the complex data and control flows enclosed among the cluster nodes to execute analytic applications; 3) considering the different roles of core daemons responsible for running such analytic applications; and 4) mining for tangible evidence of security anomalies from log data. SMaaS specifically leverages streaming big data analytics and cloud technologies to overcome these challenges. This is done through data pipeline mixing advanced analytic technologies to automate the collection, management, processing, analysis, and visualization of log data from multiple sources, making it valuable, comprehensive, and cohesive for security inspection. SMaaS works towards extracting information flow profile from log data to model the execution of a candidate application. Upon the information
flow profile, SMaaS employs several techniques to detect security anomalies that indicate data integrity and confidentiality violations.

1.3 Thesis Contributions

The importance of this thesis is twofold: a) it enables SaaS consumers to verify the security of cloud applications handling their sensitive data; and b) it also helps service providers to discover and mitigate attack venues that may threaten the security of their offered applications. The main contributions of this thesis are summarized as follows:

1- A Taxonomy of Security as Service solutions in the Cloud: This thesis introduces a comprehensive taxonomy that classifies research work in SecaaS through a three-dimensional perspective: the operation features of the security service, the security solution involved in the service, and the security threat targeted to mitigate. This study provides a guideline to consider the implications of the service operation and the security solution on the effectiveness of the surveyed studies to mitigate the target threats in terms of three key properties: visibility, control, and robustness [153].

2- Integrating Security in Cloud Application Development Cycle Platform: We propose a platform to introduce the notion of Information Flow Control as a Service (IFCaaS). The proposed platform delineates the groundwork to interweave security from inception through deployment and beyond for SaaS applications. The platform specifically envisions IFC-based activities and
services to integrate security into every phase of the development lifecycle of SaaS applications. Such services provide different styles of defense against advanced threats to the Cloud. The platform aims to provide effective security solutions that adhere to the following requirements: a) adaptability to the unique features (i.e., challenges) of cloud applications; b) ease of deployment; c) high scalability and performance; and d) convenience and ease of management [149].

3- *Security Diagnosis as a Service (SDaaS) Framework:* We propose and develop SDaaS, exemplifying static analysis service offered in the development phase. The SDaaS framework aims to reveal security vulnerabilities in cloud applications, which handle operational data. It mainly relies on static information flow analysis to protect against various types of vulnerabilities and mitigate the risk early in the development phase. SDaaS helps in diagnosing an application’s security status in terms of potential risk, behavior diagnosis, and risk rating. This is beneficial to prioritize remediation efforts by focusing on the most significant vulnerabilities and critical issues. The importance of SDaaS is twofold: a) it promotes SaaS consumers to choose a SaaS application to process their sensitive data with a trust in its security; and b) it also helps service providers to discover and mitigate attack venues that may threaten the security of their offered services [150, 151].

4- *Security Monitoring as a Service (SMaaS) Framework:* We propose and develop SMaaS as an adoption of the runtime monitoring service provided in the
1.4. THESIS OUTLINE

development phase. The SMaaS framework aims to reveal security anomalies in cloud applications that perform big data analytics. It employs information flow-based log analysis to detect different types of anomalies. SMaaS leverages streaming big data analytics to automate log data ingestion, processing, analysis, and visualization for real-time security inspection and monitoring. SMaaS has two benefits: 1) it hardens the security of analytic clusters (e.g., Hadoop) by inspecting applications running over them and 2) it protects big data processed by analytic applications by detecting anomalies that breach its security [152].

1.4 Thesis Outline

The remainder of this document is organized as follows. In Chapter 2, we present background on cloud computing and introduce core aspects of cloud SaaS applications and the prevalent security threats against them. Chapter 2 then ends with discussing the research work related to Chapter 4, 5, and 6. Chapter 3 proposes a taxonomy of SecaaS, highlights the research trends in this area, and provides a comparative analysis detailing the common strengths and weaknesses of existing SecaaS work. Chapter 4 presents a platform for provisioning IFC-based activities and services to embed security into the development cycle phases of cloud applications. It describes the platform details including its architecture, security zones, and principles. Chapter 5 presents a framework for Security Diagnosis as a Service (SDaaS) to detect vulnerabilities in cloud operational applications. This chapter introduces the SDaaS operational overview and the threat model. It next presents the
details of the SDaaS framework components and the experimental validation. Chapter 6 presents a framework for Security Monitoring as a Service (SMaaS) to reveal anomalies in cloud analytical applications. This chapter starts with highlighting the SMaaS operational overview and the threat model. Afterwards, the chapter provides the details of the SMaaS framework as well as the experimental evaluation. Finally, we conclude this thesis, address the limitations of our work, and present some future research directions in Chapter 7.
CHAPTER 2

Background and Related Work

In this chapter, we provide background on Cloud ecosystem and SaaS applications. We study the prevalent security threats against the Cloud, giving more emphasis on threats against cloud applications. We elaborate how these threats can lead to various information flow violations in the context of each application type (operational and analytical) targeted in this thesis. Then, we discuss how information flow control is relevant to address these threats. Finally, we survey the literature and point out the major limitations in the related research directions covering security-aware cloud service development, vulnerability vetting, and anomaly detection. Yet, the research work related to offering security solutions as services will be discussed further in Chapter 3.

2.1 Background

2.1.1 Anatomy of the Cloud Ecosystem
Cloud computing represents a paradigm shift that refactors the IT landscape for delivering resources, systems and applications as services [1, 4].

The Cloud promises impressive gains in delivering IT services such as the rapid elasticity, reliability, cost reduction, and quality of services. These advantages are increasingly attracting governments, business organizations, and individuals to migrate their applications and data to the Cloud.

The idea of cloud computing revolves from the NIST (National Institute of Standards and Technology) definition [11]: “It is a model for enabling ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly
2.1. BACKGROUND

**provisioned and released with minimal management effort or service provider interaction.**

The Cloud computing model has four deployment models: *Private Cloud*, a Cloud platform dedicated for specific organization; *Public Cloud*, a Cloud platform available to public users to use the Cloud infrastructure; *Hybrid Cloud*, a private Cloud that can be extended to use resources in public clouds; and *Community Cloud*, a Cloud platform supports a community of several organizations having shared concerns where the Cloud can be private or public, may be managed by these organizations or a third party, and may exist on or off premise.

As depicted in Figure 2.1, the Cloud stack is logically organized into five layers [12]. Each layer covers one or more cloud services. The Cloud stack layers are as follows:

**Application Layer:** This layer provides software as a service (SaaS). It targets the end users who traditionally access the software services or applications provided by this layer as online, on-demand services over the internet. Google Apps [13] (e.g., Google Docs and Sheets) are examples of SaaS.

**Software Environment Layer:** This layer provides a platform as a service (PaaS). It supplies high-level platforms, programming tools, and application servers to accelerate the development, deployment, and management of cloud applications without installing any tools or platforms on the developers’ local machines. Google’s App Engine [14] is an example of PaaS.
2.1. BACKGROUND

**Software Infrastructure Layer:** This layer is a software layer composed of three distinct types of resources: computational, storage, and communications. These resources are delivered by cloud providers as internet-based services relying on the virtualization technology.

- **Computational resources:** They offer infrastructure as a service (IaaS). They are provided to cloud users in the form of virtual machines (VMs). Amazon EC2 [15] is an example of IaaS.
- **Storage resources:** They offer data storage as a service (DaaS). They are provided to Cloud users to store their data at remote disks and access them anytime from any place. Amazon’s S3 [16] is an example of commercial cloud DaaS systems.
- **Communication resources:** They offer communication as a service (CaaS). Open VSwitch [17] is an example of CaaS.

**Virtualization Layer:** This layer provides the basic software management for the shared physical hardware resources in the Cloud. Virtual Machine Monitor (VMM) is the primary software behind virtualization. It facilitates the creation of VMs, each with separate operating systems (OSs) and applications. It also manages the operation of these VMs by allocating necessary resources such as CPU, memory, and storage. KVM [18] and Xen [19] are the examples of the VMM.

**Physical Layer:** This layer is the backbone of the Cloud including actual physical hardware and switches. In this regard, users of this layer are normally big enterprises
2.1. BACKGROUND

with large IT requirements in need of subleasing hardware as a service (HaaS). For this, a HaaS provider operates, manages, and upgrades the hardware on behalf of its consumers for the lifetime of the sublease.

2.1.2 Cloud Applications: The Big Picture

As SaaS continues to be a powerful evolution in the software industry, its adoption grows fast in the Cloud. The SaaS market is projected to remain the largest segment in the public Cloud, as shown in Figure 2.2, with a compound annual growth rate (CAGR) of 18 percent on average, reaching over $117 billion by 2021 [20].

SaaS is causing an evolution in the way software is developed, delivered, managed, and maintained. This model enables cloud consumers to leverage software applications as services on-demand, while managing and maintaining the applications and their underlying infrastructure are centralized at the Cloud. It promotes software service providers to benefit from economies of scale.

Figure 2.2: Worldwide public cloud service revenue forecast.
Towards attaining this benefit, the software providers need to support scalability, by efficiently sharing the Cloud resources among the various consumers (a.k.a. tenants) on-demand, and flexibility, by applying different-level of feature customization as required for each consumer (tenant), in order to balance the operational and upfront development costs of providing their software services.

Cloud SaaS applications are typically complex and decomposed into various distributed components. There are different architectural styles to build cloud applications. The applications can be architected in monolithic traditional multi-tiers, microservices, or big data analytic, among other styles chosen according to application type (e.g., operational or analytical). The application type reflects the
data processed by the target application. In this thesis, we target two different application types in the Cloud: operational and analytical, defined as follows:

**Operational applications:** This category refers to applications designed to retrieve and store operational data to perform their ordinary functions. Examples of such applications include a healthcare system handling patient information, a web commerce application processing shopping data, and a stock-trading application processing investment data. Shopify\(^1\) is a Canadian e-commerce platform that recently empowered by the Cloud for developing and hosting online stores.

**Analytical applications:** This category represents applications designed to retrieve, analyze, transform, and report data for business intelligence (BI). Such applications often process analytical data which has been created from previously stored operational data over time. For example, shopping data in a web commerce system can be historically recorded and then ingested into an analytical application to learn about market trends. Sisense\(^2\) is a BI platform to run various business analytics.

The application architecture is affected by two technology choices: compute and storage. The former is the hosting model for the computing resources that run on the application. Cloud software service providers can deploy their applications over PaaS (Platform as a Service) or IaaS (Infrastructure as a Service) owned by them or offered by other cloud hosting providers. The latter is the storage model used by an application to persist and access the processed data. The storage model broadly falls

---

1 Shopify. https://www.shopify.ca/
2 Sisense: https://www.sisense.com/
into two categories: SQL (Structured Query Language) and NoSQL (Not only SQL). Each category provides various technologies to work with different types of applications (operational and analytical), as depicted in Figure 2.3.

There are different SQL technologies (e.g., relational database) and NoSQL technologies (e.g., key/value datastore, column family datastore, document datastore, and graph datastore) available to support operational applications. There are also other SQL technologies (e.g., relational reporting and analytic) and NoSQL technologies (e.g., big data analytics) available to support analytical applications.

SQL and NoSQL storage technologies provide two different scalability choices to enable cloud applications to deal with the increase in concurrent consumers and the amount of data. The former supports a scale-up option which indicates a default centralized approach that relies on large and complex servers. The latter supports a scale-out option which indicates a distributed approach that leverages many commodity servers.

Multi-tenancy, meaning that different consumers (tenants) may share software and hardware resources, is a key enabler of economics of scale. In principle, multi-tenancy can be achieved via various approaches at the hypervisor, operating system, platform, and application levels [21]. As illustrated in Figure 2.4, each multi-tenancy approach introduces a trade-off between three prime factors: scalability, flexibility, and cost-reduction.
2.1. BACKGROUND

In the case of hypervisor-level, the software service provider hosts each tenant’s application in dedicated virtual machine (VM) having its own operating system (OS) and platform, sharing only the underlying infrastructure. In the case of operating system-level, each tenant has a separate application with a dedicated platform (e.g., middleware and application server), while the operating system and infrastructure are shared. In the case of platform-level, the platform, operating system, and infrastructure layers are shared to varying degrees across tenants, whereas multiple instances of the application are deployed in separate application spaces (i.e., containers), each instance serves a target tenant, and all instances may possibly also share the backend data storage. Last but not least, in the case of application-level, all tenants share everything starting from the application instance.
and the associated data storage to the underlying software and hardware stack. In this case, tenant separation is mainly realized and handled in the application code. Thus, the application-level multi-tenancy approach achieves the highest degree of resource sharing driven by scalability while cutting operational costs to an economical level.

2.1.3 Need for Security

As cloud adoption continues to surge, security concerns continue to rise on even more [22]. The unique nature of the Cloud, including its dynamic distribution, multi-tenancy, virtualization, and on-demand scalability, introduces major security concerns that can hinder the acceleration of its adoption, as follows:

Loss of control: Outsourced data, applications, and IT assets in the Cloud are typically hosted and maintained by cloud providers separate from a consumer’s physical control.

Lack of isolation: The more that cloud providers leverage shared resources, the more complexity is added to ensure isolation across their tenants, and the more security risks arise against outsourced data, applications and IT assets.

Lack of regulation enforcement: There is uncertainty in the ability to enforce security regulations and assess the security compliance of cloud providers. This, in turn, leads to hesitancy in migrating sensitive assets to the Cloud.

The Cloud Security Alliance (CSA) identified twelve top security threats against the Cloud [6]. In this section, we focus on discussing the five most prevalent severe threats (out of the twelve) that make cloud SaaS applications an attractive
2.1. BACKGROUND

target for attacks. The risk from these threats arises due to overlooking security practices during the engineering process of cloud applications as the software providers scramble to keep up with the time-to-market pace. In what follows, the threats are further discussed.

**Insecure interfaces and APIs:** Web service APIs are the most preferred way of exposing cloud services to provide consumers with value-added features in various ways. Protecting such front-end interfaces and APIs is crucial to the security of associated cloud services as a whole. A security incident is recently reported about exploiting the API of the US Internal Revenue Cloud Service to profit from taxpayers’ personal information approximately financial loss of $6 billion [23].

**System and application vulnerabilities:** Cloud computing inherits most of the core technologies used in the Web, Internet, virtualization, and other technologies. The integration of these technologies makes the Cloud environment prone to vulnerabilities that can lead to attacks of diverse complexities and consequences. A successful exploitation of vulnerability can compromise the integrity, confidentiality, and availability of data, applications, and other virtual IT assets shared across cloud consumers (tenants). In a study applied on thousands of applications running on public, private, and hybrid cloud by CENZIC, it is reported that 96 percent of tested applications are vulnerable to attacks [10].

**Malicious insiders and anomalous activities:** The Cloud as a distributed computing and storage environment is exposed to malicious activities either from insider party
or insider node. A malicious party or node can have access to data, application, network, and other IT assets in the Cloud and misuse intentionally this privilege in a way violating confidentiality, integrity, or availability of such assets. Skyhigh’s study reveals that 85 percent of anomalous activities in the Cloud are indicative of insider threats [24].

**Shared technology risks:** Vulnerabilities in shared cloud infrastructure, platform, or application can easily negate the isolation across consumers (tenants). These vulnerabilities can enable an attacker to gain access to assets outside the scope of an exploited component, compromising the entire cloud. According to well-known vulnerability databases [25], VMM vulnerabilities, in the virtualization layer, are considered a dangerous attack surface that could result in a single point of compromise for the security of all hosted components (VMs).

**Data breaches:** They represent security incidents in which unauthorized parties can access, modify, or disclose sensitive data outsourced in the Cloud. An incident can take place as a result of inadvertent flaw or malicious intent including, but not limited to, application vulnerabilities, human errors, security misconfigurations, rogue networks, fraudulent activities, and malicious insiders. A recent incident is reported about the disclosure of the credentials of an antivirus company’s consumers due to exploiting a security vulnerability in its cloud application hosted on Amazon Web Services (AWS) public cloud [26].
By examining the aforesaid security threats in the scope of cloud applications, they can be casted as information flow violations in terms of exposing data confidentiality, integrity, and availability (CIA). In this respect, we frame these threats with respect to the classical CIA-triad into the following three main pillars:

**Confidentiality:** It ensures that data can only be accessible by intended authorized recipients (e.g., users, processes, devices, etc.) to maintain its secrecy.

**Integrity:** It asserts that data cannot be viewed or modified by untrusted source (e.g., user inputs, processes, etc.) to maintain its trustworthiness, correctness, and consistency over its lifetime (in process, transit, and store).

**Availability:** It assures the timely and reliable access of data upon request.

In what follows, we will dig deeper into explaining some examples of potential information flow violations in the Cloud applications in question: operational and analytical, respectively.

**Information flow violations in operational applications**

Cloud operational applications are exposed to novel attack venues as well as existing ones as a result of exploiting application vulnerabilities and insecure APIs and interfaces. These threats can expand to cause data breaches and shared technology violations. In this regard, information flow vulnerabilities such as SQL injection (SQLI), XML Path injection (XPathI), Cross Site Scripting (XSS), and command injection (CMDI), among others, are reported as the prevalent attacks towards such applications [8].
The majority of SaaS applications follow the shared everything flavor of application-level multi-tenancy as the most recommended choice to achieve high economies of scale [21], albeit at the cost of security. As depicted in Figure 2.5, the data layer consists of shared backend datastore with either shared or separate tables, where the data isolation is logically regulated in the application code.

Figure 2.6 shows a web service with a NoSQL injection vulnerability. The service uses NoSQL technology in the form of MongoDB, a document-oriented database, for storing unstructured data as key-value pairs, aggregating them into documents using the JSON (JavaScript Object Notation) format, and supporting JSON and JavaScript as handy query languages.
2.1. BACKGROUND

Insecure Service

```java
GET
@Path("/serviceURL/{patientId}")
public void retrieveMedicalRecord (@PathParam("patientId") String patientId)
{
    // ....
    MongoClient dbClient = new MongoClient( hostName , portNum );
    MongoDatabase db = dbClient.getDatabase(dbName);
    MongoCollection<Document> collection = db.getCollection(collectionName);
    String jsonDocument = "{tenant_id': 'Tenant1', 'patient_id': 'patientId'}";
    DBObject query = BasicDBObjectBuilder.start().add("$where", jsonDocument).get();
    MongoCursor<Document> cursor = collection.find(query).iterator();
    //...
```

Potential Violation

```
{"ne": ""}, 'tenant_id':{" ne": " "}
```

Figure 2.6: Example of NoSQL injection vulnerability

In this example, Lines (8-9) intend to retrieve medical records based on the key `patient_Id` whose value is obtained from the user input submitted as a parameter with the HTTP Get request to the service and the key `tenant_id` whose value is fixed restricting the access to the authorized tenant “Tenant1”. Line 8 deserializes the values of these keys as JSON document without being properly validated or sanitized. Thus, an attacker can manipulate the values of these keys by injecting arbitrary values (e.g., {"ne": ""}, 'tenant_id':{" ne": " "}). Since Line 9 uses the JSON document to manipulate the structure of the query, the crafted input supersedes the condition that enforces the manipulation of “Tenant1” data. The resulting query interprets the crafted input containing $ne operators and returns all records whose `patient_Id` is not equal to empty string for all tenants whose `tenant_id` is not equal to
empty string, instead of comparing the patient_Id key for equality with the patientId value. As a result, the medical records of not only other patients of the authorized tenant “Tenant1” but also other patients of all tenants sharing the same collection will be retrieved. In this respect, this vulnerability is considered violation of data integrity as a result of allowing untrusted input to manipulate critical computations retrieving sensitive data from the backend data store. More sophisticated exploitations may extend this technique by using different MongoDB operators to execute arbitrary JSON or JavaScript code [148].

**Information flow violations in analytical applications**

Analytical applications are prone to data breaches due to insecure computations, misconfiguration, and unauthorized access as a result of vulnerable, malicious, or misconfigured nodes or tasks [7].

These applications mainly rely on analytic technologies running over clusters of distributed commodity machines for massive data storage and parallel processing. With the latest advances in analytic technology (e.g., Hadoop), multiple applications or workloads (e.g., batch, machine learning, and stream processing) can run on the same cluster, accessing a common data pool along with hardware and software resources. These resources are shared to varying degrees according to the multi-tenancy flavor adopted when building the analytic cluster. Here, we provide some examples of security violations in batch-oriented applications or workloads (e.g., MapReduce jobs).
As shown in Figure 2.7, these applications usually split input data into small splits and spread multiple instances of the application across a distributed cluster of nodes to achieve parallel processing. Specifically, each instance runs over the node that stores the data split processed and analyzed by this instance and then the final outcome is produced by aggregating the resultant data from all instances according to the conducted analytic task.

Figure 2.8, part (a) and (b), respectively, illustrates snippets of various malicious activities performed in a MapReduce application including a) tampering with input data by using wrong data leading to untrustworthy analytic results, breaching data integrity; and b) violating the access permissions of the output opening it to the world and making it prone to security compromise, breaking its CIA. Figure 2.8 part (c) shows a misconfigured application changing the cluster settings (cache and other directories) in a way that exposes sensitive data produced throughout the application’s execution to CIA violations.
2.1. BACKGROUND

Figure 2.8: Code snippets of security breaches in an analytical application: (a) data tampering, (b) access violation, and (c) misconfiguration, respectively. Lines in red refer to the anomalous activity.
2.1.4 Information Flow Control

Information flow control (IFC) is a well-known technique that asserts the end-to-end security of sensitive information processed by a software system. It governs the flow of information through the software system. This is achieved by verifying that the information flow in the system complies with specified security policies, ensuring data integrity and confidentiality. An information flow policy represents the security rules for a given application that specifies the rights or the restrictions of how data may be processed. The non-interference is the most prominent example of a security policy. To ensure confidentiality, it requires that sensitive information does not flow to unauthorized destinations. Similarly, to ensure integrity, it requires that untrusted sources do not modify sensitive information.

Other classical techniques involving access control (e.g., Bell and LaPadula [91] model), encryption, firewalls, among others, have tackled information security as well from an access perspective. Thus, these mechanisms can only assert security at certain points in data processing, not during the whole processing path. In contrast, the IFC technique has the advantage of tackling the problem from a processing perspective by considering the application's semantics and providing end-to-end enforcement.

In this thesis, we are motivated to leverage IFC as a foundation to interweave different security activities and services into the SaaS application development cycle, in order to address the aforesaid security threats against such applications (e.g.,
operational and analytical) and protect the processed data from information flow violations in terms of confidentiality, integrity, and availability.

2.2 Literature Survey

As inferred from the previous sections, the Cloud is attractive to software service providers for offering their applications. However, being part of the Cloud, such applications are more accessible and exposed to threats with varying severity. The risk is further amplified from sacrificing security in the engineering process of these applications.

In this section, we shed light on current research work proposed to address these concerns. The section starts with covering existing work related to securing cloud service development. Then, we focus on discussing related research work targeting the detection of vulnerabilities and anomalies leading to CIA violations.

2.2.1 Security-aware Cloud Service Development

Promoting for security-aware cloud service development and adoption has attracted IT industry as well as research community.

Microsoft embarked on the security development lifecycle (SDL) [75] which plays a vital role to guide the development of secure software applications in a classic way adopting the Waterfall model. The SDL is built around seven phases of a long development cycle: training, requirements, design, implementation, verification, release, and response. Microsoft has then adapted the SDL by proposing the Security
Development Lifecycle for Agile Development (SDL-Agile) [76]. The SDL-Agile reorganizes the SDL security activities and recommendations into three lean categories that differ in the frequency of completion throughout a short development cycle needed for the Agile model. Both Microsoft initiatives aim to reduce the number and severity of vulnerabilities in the developed applications.

Privacy by Design (PbD) [154] revolves around seven main principles: proactive not reactive; privacy as the default setting; privacy embedded into design; full functionality; end-to-end security; visibility and transparency; and respect for user privacy. It is embraced globally for considering privacy and data protection compliance from the start when building new software systems.

Coping with the evolution of the Cloud, the Cloud Security Alliance (CSA), the National Institute of Standards and Technology (NIST), and the Software Assurance Forum for Excellence in Code (SAFECode) are among the leading organizations providing various guidelines and standards for secure cloud environments. The CSA and the SAFECode [77] propose best security practices for designing and implementing cloud platform services. Another study by the NIST [78] provides guidance for consumers about the key security safeguards to put in place when outsourcing their data, applications, and infrastructure to a public cloud.

Researchers build over industry guidelines proposing various insights towards secure cloud development and adoption. However, they do not provide a comprehensive insight for cloud application development. They target either specific
2.2. LITERATURE SURVEY

security practice (e.g., security requirements, threat modeling, and risk management) [79, 80, 83] or particular phase (e.g., development, deployment, and integration) in the development cycle [81-88].

In contrast, we build our platform, introduced in Chapter 4, from a generic standpoint to support different development methodologies. Specifically, the platform integrates IFC-based activities and services throughout the entire development cycle to help in covering comprehensive practices and building more secure applications for the Cloud environment. It provides different styles of defense for not only protecting the developed applications, but also governing the integrity, confidentiality, and availability of the processed data. This is achieved through various principles to stand up to several prevalent security threats. In this sense, embedding security into the engineering process as a first priority can serve as one of the means that ensures privacy as well.

Kao et al. [79] provide a framework that supports a Secure System Development Life Cycle (SSDLC) to assess the security governance of cloud services from government and industry standpoints. The framework provides guidance to audit the compliance with data privacy-related regulations and laws through the proposed SSDLC phases: initiation, development, implementation, operation, and destruction.

Aljawarneh et al. [80] present a generic framework suggesting high-level guidelines for service providers. These guidelines focus on the security requirements
needed to protect five assets in the Cloud including storage, service, hypervisor, record, and datacenters. However, a practical insight of engineering these requirements in the SDLC phases is missing.

Alruwaili et al. [81] introduce SecSDLC based on Waterfall SDLC model consisting of six phases: initiation and investigation, requirements analysis, development and provisioning, implementation, operation and maintenance, and de-provisioning and retirement. Chou et al. [82] propose a model for securely integrating SaaS applications with exiting enterprise applications. The model provides guidance about design concepts and countermeasures to address specific security risks arising at the integration phase.

Almorsy et al. [83] introduce a different insight to decouple the security requirement engineering from the application development lifecycle to be incorporated as an ad-hoc process at the deployment phase rather than throughout the early phases. Torkura et al. [84] extend this research line and provide high-level overview to fit security engineering late into the deployment phases for particularly DevOps pipelines.

Other approaches [85, 86] focus on applying security practices at the development phase in DevOps pipeline. Weber et al. blend traditional SDLC and Data Security Lifecycle to form SaaS security life cycle (SSLC) [85]. Thanh et al. [86] introduce a framework to integrate virtualization-based isolation techniques in DevOps. However, the proposed framework does not provide comprehensive insight
to apply security practices through the SDLC. Only few approaches [87, 88], relevant to DevOps pipelines, call for applying continuous security practices.

### 2.2.2 Vulnerability Detection

This section discusses mainly the most relevant approaches inspired by IFC for detecting information security violations in the context of cloud and non-cloud environments.

In 1976, Denning [89] originated a centralized lattice model for secure information flow in procedural languages. After that in 1999, Myer [90] introduced decentralized IFC model and found the notion of security label. IFC models, as classical well-known mechanisms that govern the flow of information through a software system, have received quite attention in the area of software security.

Our goal of elevating cloud consumer trust in cloud providers’ security is also shared by research work on dynamic data flow tracking. These approaches [41, 92-97] aim to pinpoint security breaches after an application is deployed or even after damages may have already happened. These approaches implement IFC at different layers in the Cloud stack. They differ in the granularity of monitoring data flow, ranging from byte-level [41] to message-level [93]. Some approaches need installation of monitoring agents at the Cloud side [95] or additionally at the consumer side [55]. Some approaches require modification to an application code and the underlying platform [41, 92, 93] or the virtual machine monitoring system [94]. CamFlow [96] and Pileus [97] incorporates IFC at the OS-level to offer cloud
platforms with runtime data flow tracking capability. Dynamic taint analysis is simple in contrast to static analysis but at the cost of performance penalty. In distinction, our SDaaS framework does not impose runtime overhead over an analyzed application, augments the detection of security vulnerabilities during development rather than after production, does not require modifications at any level, does not involve specific software or hardware installation, and cannot be evaded by malicious applications.

Similar to our work, CloudFence [41] and IDSaaS [33] take some steps in the research line of introducing their security solutions as a service. CloudFence is offered by the Cloud provider as PaaS in the form of libraries. It is offered to a service provider hosting their application on the provider’s platform. It enables applications’ users tracking their data during runtime. IDSaaS is a VM-based intrusion detection system that employs SNORT\(^3\) to inspect network traffic of virtual machines hosting an application in a public cloud. IDSaaS is offered by cloud providers to their consumers hosting applications on the provider’s infrastructure. Differently, our SDaaS solution is offered as a SaaS by a trusted party (security provider).

Security-type language-based systems [98–101] have originally dominated static approaches proposed to enforce information flow policies in the programming languages through annotations as part of variable types’ declaration. The integration

\(^3\) SNORT: Light Weight Intrusion Detection for Networks http://www.snort.org
of these systems into programming languages usually results in new, separate languages. The analysis of existing programs by these systems may require manually converting the programs into the corresponding languages. This, in turn, may hinder their practical use in real applications written in existing standard languages. However, SDaaS works on the bytecode of an application without any modifications.

Taint analysis has also been applied for statically detecting vulnerabilities in Java web applications [102-104], Android mobile applications [105, 106], and PHP web applications [107, 108]. These approaches differ in their precision and scope as they consider different design dimensions of analysis: flow, context, object, and path-sensitive. They focus mainly on explicit flows and ignore implicit flows through control dependences. These approaches achieve significant steps towards addressing the challenges of static analysis for web and mobile applications. Albeit, they lack a precise model that can consider all the requirements, which originate in cloud SaaS applications, for an effective static analysis solution, like our SDaaS.

Similar in spirit to our work, FlowDroid [106] takes on a semantic model approach to overcome the Android-specific challenges. In contrast, we apply new strategies and algorithms to model the unique features of SaaS applications. Our model strategies are crucial to precisely capture data and control dependences inside SaaS applications. They are independent of the employed analysis. Hence, they can be integrated by other analysis techniques as well. In this work, we choose to leverage information flow analysis based on SDG [114] and program slicing
techniques, which boost SDaaS’s ability to reveal explicit and implicit insecure flows in an application. By nature, IFC-based approaches are not designated to guard against side, covert and timing channel attack vectors. Other approaches \[113\] have been proposed for this purpose.

Other approaches \[109-112\] have advocated different techniques including intrusion detection mechanisms \[33, 109, 110\] and penetration testing \[111, 112\] to detect security violations in the Cloud. Intrusion detection mechanisms do not provide end-to-end security assurance. Interested readers may see the details of further IDS approaches in the Cloud in a recent survey \[74\]. Penetration testing treats an analyzed application as black box. It mainly uncovers security faults by analyzing the responses of tampered and malformed requests sent to the application, with limited visibility of the actual internal behavior of the application. Such a mechanism provides little assurance that an application is immune to attacks. As an enhancement, Nuno Antunes et al. \[112\] propose to combine penetration testing with anomaly detection to unveil SQL injection vulnerabilities in web services. Our experiments verify that SDaaS, as a static analysis solution, outperforms penetration testers, other static analyzers, and an anomaly detector \[112\] in terms of detection coverage, accuracy, and false alarm rate.

Some approaches \[117-122\] incorporate value computation with their analysis for different purposes such as verifying XSS sanitizers \[121\], checking SQL queries \[120\], resolving Android inter-component communications \[117-119\], and vetting
2.2. LITERATURE SURVEY

API misuse [122]. Similarly, our solution reduces value computation to constant propagation analysis. While commonly inspired by the IDE analysis [115], our IDE problem formulation and analysis design differ significantly from these previous works [117–122]. IC3 [119] specifies the semantics of the relevant APIs via a declarative language. In contrast, we embodied the specifications of the client API, which is used to access Web resources, in an ad-hoc manner. We propose a generic light-weight IDE analysis on demand-driven basis in the course of building the call graph. In contrast, IC3 applies it as a preliminary processing technique while R-Droid [122] is a post-processing approach. Hence, our approach is more relevant, practical, and efficient to solve the inter-service invocations. Similar to IC3 and R-Droid, our approach provides concrete values of modeled objects. In contrast, other approaches (e.g., [118]) provide coarse-grained evaluations that result in conservative approximation for runtime values.

2.2.3 Anomaly Detection

Several research efforts, proposed to fortify analytic world against security threats, have span different research directions, ranging from differential privacy [123], integrity verification [124-127], policy enforcement [128-131], data provenance [132-135], honeypot-based [136], to encryption-based [137] mechanisms, among others.

Airavat [123] applied differential privacy to protect data from malicious MapReduce jobs. Despite the fact that differential privacy mechanism recently
attracted researchers as an effective solution in specific problem contexts, its effectiveness as a widespread solution is still not proven. Integrity verification is a mechanism applied for decades, which recently appears in the light of MapReduce to examine the security of results produced by MapReduce jobs.

SecureMR [124] and TrustMR [125] rely mainly on replication-based computations, while VIAF [126] extends the mechanism to incorporate query-based approach to further detect colluding attacks. IntegrityMR [127] performed the integrity checks at application layer along with MapReduce task layer. These approaches differ in the scope of integrity assurance, the logical layer of operation, and the mode of checking. In general, they require intercepting the computation of MapReduce tasks for verifying result integrity, which comes at the cost of performance penalty.

Some approaches enforce security policies for access control such as GuardMR [128] and Vigiles [129] at different granularities by modifying the underlying platform [129] or adding an extra access control layer [128]. Access control policies cannot prevent misuse activities breaching data security, after an access had been granted. Another approach [130] pays attention to propose IFC-based access control model that supports multi-tenancy in SaaS systems. IFC endorses advancement over access control as it can provide end-to-end protection. As an alternative enhancement, accountability mechanisms are proposed to harden access control policies. AccountableMR [131] incorporates such enhancement at MapReduce. The
accountability is achieved by verifying that data access happened after authorization is in compliance with the security policies, governing the data security.

Data provenance (or lineage) mechanism is typically used to keep history about data for the purpose of reproducibility. Recently, few approaches embrace such mechanism for big data security [132-135]. One approach [136] proposed formal perception about provenance mechanism to enable forward and backward tracing of data during the execution of MapReduce tasks. Other approaches [133-135] perform data provenance by analyzing metadata information and system log files to collect traces about data processing for the purpose of detecting anomalies. Such mechanism faces several challenges that may hinder its practicability such as the volume of captured provenance data, the storage and integration required to effectively analyze these data, and the most important factor is the overhead incurred from collecting these data during the execution of distributed analytic tasks.

Other approach [136] takes on honeypot-based mechanism to detect unauthorized access in MapReduce. A different approach [137] leverages encryption to protect data stored in analytic clusters (e.g., Spark⁴). Encryption mechanism may disrupt the typical operations within the system when data is being processed. Furthermore, encryption and decryption are costly operations that may impose performance burden and reduction of system operations too.

---

In contrast our SMaaS solution overcomes these limitations by solving the anomaly detection as IFC-based streaming data analytic problem. This advanced approach enables SMaaS to yield the right information in the right context at the right time. In another word, SMaaS can derive cohesive and comprehensive information for security inspection from raw log data at real-time. This is done with scalability, efficiency, and without introducing modification or imposing performance overhead to the monitored cluster.

2.3 Summary

This chapter starts with background knowledge about the Cloud ecosystem focusing on SaaS applications and the prevalent security threats against them. We present how these threats in different types of cloud applications (operational and analytical) lead to various information flow violations.

This chapter also introduces information flow control mechanism as one of the main foundations of this thesis. However, we dedicate Chapter 3 to address security as a service (SecaaS) model which is the other main foundation of this thesis. Despite the steps taken towards the realization of security services, the security threats targeting cloud applications have not attracted adequate attention in this research community.

We thus further explore the literature and survey existing research work tackling security-aware cloud service development, vulnerability vetting, and anomaly detection in this chapter. Existing work has mainly tried to tailor traditional
security techniques to fit in the Cloud environment. They still have limitations in providing scalable and effective security solutions due to the unique requirements that originate in cloud applications. With these limitations, there are open research challenges that need further investigation which are addressed in Chapters 4, 5, and 6. Prior to that, we provide a taxonomy of security as a service work in the next chapter.
CHAPTER 3

A Taxonomy of Security as a Service

This chapter explores security as a service (SecaaS) through a tri-dimensional taxonomy: service operation, security solution, and threat. The taxonomy helps to draw some observations on the recent research trends. The chapter also discusses existing SecaaS work according to their security mechanisms and contrasts the effectiveness of the surveyed work in terms of visibility, control, and robustness. Then, it presents a comparative analysis that details the common strengths and weaknesses of existing SecaaS work. The analysis takes into consideration the implications of the service operation and the security solution as the central dimensions affecting the effectiveness of the surveyed work in mitigating the target threats.

3.1 Introduction

On the heels of the evolving expansion of cybersecurity threat landscape (i.e., internal and external) and the growing shortage of cybersecurity resources (i.e., tools
and skills) [6, 27], Security as a Service (SecaaS) rises as a promising model in the
spot to fill this pressing gap. SecaaS simply revolves around provisioning delivered,
maintained, and managed security solutions as services over the Cloud through
subscription or on-demand basis to protect the security of cloud or on-premise
systems [28]. Inheriting the Cloud’s gains, the SecaaS model promises cost, time,
and maintenance effort reduction while providing a high-level of protection.

Gartner research expects that public cloud will deliver over 60 percent of
security solutions and the SecaaS market will continue to shine, reaching up to $9
billion by 2020 [29]. The attention of the research community in the importance of
SecaaS grew up in recent years [64-71]. These research efforts span from just
proposing primitive concepts that have not been implemented or practically validated
[64-67] till introducing better perceived guidelines in this research area [68-71].

Despite this growing attention, there is a lack of studies providing a taxonomy
of SecaaS work to draw deep and comprehensive insights of these offerings as well
as understand their common characteristics and limitations. Wang and Yongchareon
[72] address surveying the SecaaS work from a high-level perspective. The survey
mainly maps the SecaaS categories identified by Cloud Security Alliance (CSA) [28]
into three main categories: protective, detective, and reactive. However, it is still a
crucial defy for SecaaS consumers, who are overwhelmed by the diversity of these
offerings, to assess and select the proper solution which fulfill their security
requirements.
Towards addressing these limitations, this section aims to propose a taxonomy of SecaaS research work. The taxonomy allows for understanding the evolution and trends in this research area as well as reasoning about existing work with respect to three main dimensions: service operation, security solution, and threat.

In this respect, the proposed taxonomy reinforces the SecaaS consumers and researchers to gain clear and comprehensive insight about the operation and design of each work including its delivery model, deployment model, and the responsible entities to operate the proposed service. It also helps consumers perceive the security mechanism and the defense plan employed by each approach to achieve certain security requirements. Last but not least, the taxonomy also identifies the attack surface and methods of the security threats targeted to be mitigated by each work. Then, we present a discussion of the existing SecaaS work by classifying them according to their security mechanism. The study assesses the effectiveness of the surveyed work in terms of three main properties: visibility, control, and robustness. We envision that these properties are particularly influential in promoting the trust and adoption of SecaaS. We thus revolve our survey study per se around these properties.

- **Visibility:** This property reflects the depth of the visibility given to the SecaaS solution into the internal semantics or state of the protected assets or systems.
3.2. **TAXONOMY OF SECAAS**

- **Control**: This property reflects the ability to provide either consumers or their delegated parties the control to customize the proposed service according to the consumer’s requirements.

- **Robustness**: This property represents the ability of the SecaaS solution to self-defend against attack and evasion\(^5\) attempts.

Then, we wrap up with a comparative analysis detailing the common strengths and weaknesses of surveyed studies from a panorama standpoint related to our proposed taxonomy. Specifically, the implications of the different aspects of the service operation and the security solution dimensions are considered when contrasting the effectiveness of the current SecaaS work in terms of the aforesaid properties.

### 3.2 Taxonomy of SecaaS

This section explains the taxonomy and the current trends in this research area. Figure 3.1 portrays our taxonomy that aims to explore and identify each addressed SecaaS work through a three-dimensional perspective: the operational features of the security service, the security solution involved in the service, and the security threat to be mitigated. These three main dimensions seem orthogonal, albeit correlated, to reason about the addressed approaches.

---

\(^5\) Evasion of a SECaaS solution is meant when the solution fails to recognize the attack. This happens because the attack succeeds to hide its malicious activity, or the solution lacks the ability to identify the security violation. While attacking a SECaaS solution involves disabling the solution or tampering with the security information to prevent the solution from detecting the security violation.
3.2. TAXONOMY OF SECAAS

Figure 3.1: Taxonomy of Security as a Service in the Cloud. The category acronyms in the rightmost box are used as the column headings in Table 3.1 and Figure 3.2.
3.2. TAXONOMY OF SECAAS

As depicted in the figure, each dimension comprises of different aspects which are further classified into finer categories representing the lower-levels of the taxonomy. In what follows, the taxonomy details are further explained in the light of the aforesaid dimensions.

3.2.1 Service Operation

This dimension covers the aspects related to the service operation which include the service delivery model, the deployment model, and the responsible entity as defined below:

The Delivery Model. It identifies the IT resources utilized to deploy and deliver the SecaaS solution in the Cloud. A SecaaS solution can be delivered in the form of cloud application (i.e., Software as a Service “SaaS”) or as an add-on security service in cloud infrastructure (e.g., Infrastructure as a Service “IaaS”, Data Storage as a Service “DaaS”, or Communication as a Service “CaaS”), platform (i.e., Platform as a Service), or virtualization (e.g., Virtual Machine Monitor “VMM” a.k.a. hypervisor).

The Deployment Model. It represents the Cloud datacenter used to deploy and run the SecaaS solution ranging from public, private, or hybrid which is a combination of the earlier two. A SecaaS solution may support multiple deployment models.

The Responsible Entity. It refers to the main entity responsible to deliver, maintain, and manage the SecaaS solution. This entity can have either a solo or shared responsibility. In case of a solo responsibility, the responsible entity can be one of
three actors: The Cloud or service provider offering the Cloud resources or IT service, the Cloud consumer utilizing cloud resources or IT service, or dedicated trusted third party offering the SecaaS solution. SDaaS [151] is an example where the trusted party is in charge to provide the security service. In case of a shared responsibility, the responsible entity can be any combination of the aforementioned three actors such as SMaaS [152] where the responsibility is held by both the Cloud analytic provider as well as the trusted party.

3.2.2 Security Solution

This dimension includes the aspects related to the employed security solution which involve the security mechanism, the defense tactic, and the targeted security requirements.

The Security Mechanism. The SecaaS solutions expand the horizon employing various security mechanisms that can be categorized according to their core functionalities. In our taxonomy, we utilize the twelve categories of SecaaS recently defined by the CSA [28]. A SecaaS solution can adopt one of these mechanisms. For the sake of completeness of this chapter, here we present these categories in brief. They are explained in depth in the CSA study [28].

Identity and access management (IAM): This mechanism provides control over identity administration to govern that authenticated users access resources or assets in conformance with the specified authorization rules.
3.2. TAXONOMY OF SECAAS

*Data loss prevention (DLP)*: This mechanism provides strategy to monitor the security of data in move, at rest, or in use to protect against potential data leakage and unauthorized data access.

*Security information and event management (SIEM)*: This mechanism provides strategy to correlate, aggregate, and analyze log and event information together for detecting and responding against real-time security incidents.

*Business continuity and disaster recovery (BCDR)*: This mechanism covers planning and implementing strategies to maintain system’s operation while facing disruptions due to cyberattacks, natural disasters, or human faults.

*Encryption*: This mechanism applies cryptographic techniques to convert data into encoded format recognized only by authorized users who are able to decrypt it back.

*Web security*: This mechanism provides strategy to intercept, filter, and limit web access by a software (e.g., security gateway) to protect against online threats (e.g., malware, spyware, and virus) breaking through web browsing.

*Email security*: This mechanism works on enforcing email access rules and inspecting email contents to protect against online threats (e.g., phishing, spam, virus, etc.).

*Network security*: This mechanism examines the network traffic to maintain its usability and integrity secured against various threats (e.g., distributed denial of service “DDoS”, malware, and virus) through hardware or software technologies (e.g., security gateway, firewall, and virtual appliance).
3.2. TAXONOMY OF SECAAS

_Vulnerability scanning:_ This mechanism inspects a target application, network, system, or infrastructure to detect potential security holes that can be exploited by attackers.

_Security assessments:_ This mechanism reviews the security of an intended system from an extensive technical and conceptual standpoint to help in identifying risk and defining remediation required to maintain the system’s compliance with industry standards.

_Intrusion management:_ This mechanism inspects network traffic, application, software system, or virtual machine (VM) activities to detect and prevent external or internal intrusions. A comprehensive study about different techniques applied by intrusion detection systems in the Cloud are out of the scope of this work. Still, more details can be found in our earlier study [74].

_Continuous monitoring:_ This mechanism provides assessment of the real-time security posture of the target system by constantly monitoring for security vulnerabilities, misconfigurations, and threats.

_The Defense Tactic._ It denotes the defense plan undertaken by the security mechanism employed through the SecaaS solution. This mechanism can be proactive acting before the threat occurrence, detective acting during the threat occurrence, or reactive acting after the threat occurrence. A SecaaS solution can employ a combination of tactics.
The Security Requirements. It identifies the security requirements safeguarded by the SecaaS solution including integrity, confidentiality, availability, privacy, authentication/non-repudiation, and authorization/access control. A SecaaS solution can target protecting a combination of requirements.

3.2.3 Threat
This dimension constitutes the aspects related to the security threat as regards the attack surface and method.

Attack Surface. It refers to the combination of weak points in the target system that the SecaaS solution aims to protect ranging from data, application, infrastructure, network, virtualization, to physical layer.

Threat Method. It recognizes the combination of threat methods targeted by the SecaaS solution to mitigate. In our study, we consider the top twelve treacherous threats identified by the CSA [6]. They include, ordered by severity, data breaches, weak identity & access management, insecure interfaces and APIs, system and application vulnerabilities, account hijacking, malicious insiders, advanced persistent threat, data loss, insufficient due diligence, abuse of service or system, denial of service, and shared technology vulnerabilities.

3.2.4 Some Observed Trends
In this section, we study the trends of the existing research in SECaaS with respect to two aspects: a) the threat methods they target to mitigate and b) the
3.2. TAXONOMY OF SECAA S

security mechanisms they adopt. These aspects are the most important to help security researchers in identifying the gaps in this research area. They are also crucial to differentiate between SecaaS solutions as well as assess if these solutions meet the security needs.

Table 3.1 presents the categorization of the existing SecaaS research work according to our taxonomy presented earlier in this section. Each existing work in our study is denoted in the column “REF” in Table 3.1 as regards its reference number in this thesis. As mentioned before, the other column headings represent the category acronyms as presented in the rightmost box of Figure 3.1. From the table, we derive the following observations about the evolution and trends of the current research in SecaaS.

The top-addressed threats: They are data breaches, weak identity & access management and advanced persistent threat as depicted in Figure 3.2, part (a). However, threats such as insecure interfaces & APIs, system & application vulnerabilities, malicious insiders and anomalous activities, and shared technology threats which are the focus in this thesis, did not receive the same attention.

The top-adopted security mechanisms: They are IAM, network security, security assessment, and intrusion management, as illustrated in Figure 3.2, part (b). However, the least-adopted mechanisms are SIEM, network security, and security assessments.
We infer that there is still necessity to call for new SecaaS initiatives investigating other mechanisms and resolving unaddressed security threats against cloud SaaS applications.
3.2. TAXONOMY OF SECAAS

Similar in spirit to the surveyed approaches, our work proposes SecaaS solutions which have similarities with existing work with respect to the taxonomy’s dimensions. We are motivated to adopt vulnerability scanning and continuous monitoring mechanisms similar to other approaches [e.g., 41, 44, 46] which share our goal to protect the attack surface in applications and data against some of the threats.
(e.g., data breaches) we intend to address, towards achieving the same security requirements (e.g., integrity and confidentiality). Other approaches rely on either proactive [30, 37] or detective [34, 52] defense plans like our work. We design and operate our services on SaaS model similar to some approaches [55, 58, 62, 63] over a public Cloud to be managed by a sole trusted party [58, 62] or shared entities (e.g., cloud provider, trusted party, cloud consumer) [55, 63].

### 3.3 Classification of SecaaS

This section discusses and classifies the current SecaaS work according to the CSA security mechanisms used in our taxonomy. We further study and contrast the approaches in each category in terms of the three main properties: visibility, control, and robustness.

**Identity and Access Management (IAM)**

Carvalho et al. [31] propose secure data storage service bundled with security mechanisms for identity and access management. The Cloud provider specifically delegates a cloud broker to leverage access control list accompanied with broadcast encryption and key rotation to specify and manage consumers' permissions and credentials. In addition, the conducted transactions over stored data are performed by the Cloud provider, audited by the service consumers, and monitored by a trusted party to detect any security violations. The proposed solution’s visibility into the storage service is limited to the managed access control policies. It takes the control
from service consumers and hands it to the broker. The proposed service may be robust against evasion attempts by collusion attacks (between broker and cloud provider) and cover the detection of data leakage and tampering violations (e.g., replay and rollback attacks). However, it is still at great risk against access control, data integrity, and user authentication violations.

Wu et al. [37] proposes access control as a service (RaaS) that allows cloud consumers to configure coarse-grain role-based access control policies and maps them into the IAM policies supported by the Cloud provider (e.g., Amazon IAM). Such service provides consumers with flexible control to express their policies according to their hierarchical role structure to better support separation of duties and privileges between their users. The designated polices are enforceable by the Cloud provider which can provide robustness against unauthorized access or escalated privileges.

Abbasi et al. [38] introduce a very initial idea of attribute-based access control as a service for cloud storage services. The used access control model tries to achieve fine and flexible policies for data protection. The sensitive data are stored in an encrypted form while the access control is delegated to a trusted party. In this regard, the proposed service lacks giving any control to consumers storing their data over the Cloud storage. The authorization model provided by the service has limited high-level visibility that does not consider the granularity of the storage's objects (e.g.,
3.3. CLASSIFICATION OF SECAAS

database tables, columns, etc.). The service may gain robustness due to being isolated from attacks targeting the Cloud storage service.

Lang [39] proposes authorization as a service for cloud applications that applies security and compliance policy automation into the development and runtime platforms. Such service provides consumers with the control needed to generate and configure model-driven security policies through the application development. It also improves the real-time visibility into the application security posture through model-driven incident monitoring, analysis, and auditing automation. This is seamlessly done during the application runtime by collecting application layer alerts and mapping them to the specified policies.

Ghazi et al. [43] introduce database security-as-a-service (DB-SecaaS) for cloud data storage (i.e., NoSQL databases). The proposed service employs authentication, fine-grained authorization, and data encryption to provide a comprehensive solution protecting data in move and at rest. The DB-SecaaS service provides control to the consumers of the Cloud storage to assign, modify, and manage access control policies. It also achieves a fine-grain visibility when making access decisions. This is done by supporting an authorization model that is based on different granularity levels of database objects (e.g., fields, cells, columns, and tables) and further takes into account environmental variables (e.g., time, behavior, and threshold values).
3.3. CLASSIFICATION OF SECAAS

Danwei et al. [53] describe an idea of access control as a security service based on usage-based control (UCON) and negotiation models. The service relies on attributes, obligations and conditions of the UCON model to make authorization decisions before the access of outsourced data can be granted to consumers. It relies on the negotiation model to handle conflicts between access request and access rules. The service also provides flexible control to consumers to adapt the obligations and conditions according to their requirements. However, it still does not consider the least-privilege and horizontal scope principles in its design to support various constraints or heterogeneous requirements of consumers. This, in turn, limits its visibility and robustness to provide wide protection.

Zhang and Chen [54] combine attribute-based authorization with encryption to provide access control as a service for cloud storage services. The authors put the Cloud consumers (data owner) in charge to flexibly control the data encryption and authorization key generation with respect to the access attributes of each user, whereas, the Cloud provider holds the control of enforcing the encryption and authorization policies. The service allows cloud consumer and provider to delegate their tasks to separate trusted third parties. The service also better supports the least-privilege, policy conflicts, and horizontal scale principles. This, in turn, supports the service’s visibility and robustness. However, the service’s visibility is considered limited in terms of supporting different degrees of granularity over the protected data and users. Its protection scope does not also expand vertically.
3.3. CLASSIFICATION OF SECAAS

Data Loss Prevention (DLP)

Papagiannis and Pietzuch [55] introduce CloudFilter as data loss prevention add-on service to protect cloud storage services. The CloudFilter service employs two proxies at the consumer and the Cloud sides. The consumer proxy is responsible to annotate outsourced data with security policies which are enforced by the Cloud proxy to protect data in move. In this regard, CloudFilter’s visibility is limited to the perimeter network to only intercept the HTTP requests sent or received by the storage service. It does not have any visibility into the internal semantics of the storage service. It provides control to cloud consumers to customize and manage the security policies. It also enables consumers to track the propagation of their outsourced data. However, its robustness is endangered from being defeated by disabling its proxies or tampering with the specified policies by a malicious insider (e.g., end user or cloud provider).

Shu and Yao [58] introduce data leakage as a service to detect sensitive data leakage sent outside the local network of its owner. The data owner delegates a trusted party to gain perform offline deep packet inspection in a privacy-preserving manner. The service has visibility to the network packets sent for the inspection while the sensitive data is protected in the form of fuzzy fingerprinting. The service only covers detecting data leakage due to unintentional faults. Thus, it is prone to evasion by more sophisticated attempts (e.g., malware, and malicious insider).
Security Information and Event Management (SIEM)

Kim et al. [56] take initial steps towards proposing SIEM as security service employing deep network packet inspection. The proposed service gains visibility into the virtual network by performing association analysis between network flow stream, session information, and packet traffic to detect security anomalies. This is due to its attachment to the underneath virtualization layer. The service is fully controlled by the Cloud provider managing the virtual network resources. It also gains some robustness from being deployed in isolated VMs.

Roundy et al. [57] provide a SIEM as a service that correlates and analyzes the relationship between known security incidents and overlooked security events based on graph mining to find undiscovered security incidents. The service is fully controlled by the trusted party. Its visibility is limited to the information about the security incidents and events sent from various security tools (e.g., intrusion detection system, monitoring system, etc.). Its robustness is relative to the security measures taken by the trusted party as well as the integrity of the information taken as a basis for the SIEM analysis. However, the proposed approach is based on detecting attacks based on secondary indictors in the collected security information. Thus, the authors claim that their approach can stand against attacks even if they evade primary indicators.
3.3. CLASSIFICATION OF SECAAS

**Business Continuity and Disaster Recovery (BCDR)**

Wood et al. [42] introduce the use of cloud infrastructures as a disaster recovery service for distributed applications. The proposed service replicates the application servers and backend data stores over a remote cloud site to be readily available on the event of a disaster. However, the introduced service does not provide support of business continuation to permit real-time operation for the recovered applications after a failure takes place.

**Encryption**

El Bouchti et al. [30] propose a very initial idea of Cryptographic as a service (CaaS). The CaaS is deployed in separate protected VM to perform the cryptographic (i.e., encryption and decryption) operations controlled by a trusted party while the service management is implemented in the hypervisor hosting the resources provided by the Cloud provider. The operation design and the technical details about the proposed service workflow are missing and vague. Although isolating the service management in the hypervisor can add protection and robustness to the CaaS from network-based attacks, this does not negate that the CaaS is exposed to misconfiguration and cross-VM side channel attacks from co-resident VMs. On the other hand, the degree of the visibility and control over the data and the involved cloud resources (e.g., VMs, network, and storage) that need to be protected are still very limited. A vision about the cryptographic key management and distribution has not been identified either.
3.3. CLASSIFICATION OF SECAAS

Shama et al. [46] introduce an elementary design of encryption and decryption as a service to enable cloud consumers who optionally keen to conceal their data before being uploaded to the Cloud storage. The proposed service employs classical encryption algorithms without any specific adaptation to the Cloud. The service does not have any visibility inside the Cloud storage. Thus, the protection provided by the service is directed for consumers’ sake. Even that the service provides consumers with the control to choose the desired encryption algorithm, its practicability is considered limited as it implicitly assumes that ordinary consumers must be experts to technically differentiate between such algorithms. The robustness of the service against attacks is yet questionable for being exposed in public cloud settings.

Mahalakshmi and Kuppusamy [49] introduce security as a service solution that integrates encryption and decryption capabilities into cloud file-storage services. The proposed service is fully controlled by the Cloud provider. It employs cipher block chaining encryption to improve its robustness against attacks such as brute force.

Web Security

Thomas et al. [32] propose Monarch as URL (uniform resource locator) filtering service to detect spam messages sent to applications such as social networks. The filtering service is fully controlled by a third party and provided to application providers who have the choice to react according to the filtering result. The service is transparent to the target application without any visibility to its internal semantics. It focusses on analyzing URL-based spam messages relying on parallel linear
classification. Thus, it is prone to be evaded by other types of malicious messages such as scams based on phone numbers and XSS worms spreading without attaching to a malicious URL.

Pham et al. [59] introduce Phishing-Aware as anti-phishing service to detect requests to malicious websites. The service combines neural network and fuzzy models to differentiate between benign and malicious websites. The service is offered transparently by Internet and Mobile service providers to their local users. The detection process is pushed into a fog node to be close to the protected users while the training process is kept into the Cloud to benefit from the unlimited data storage and processing power. In this respect, the service does not have any visibility into the internal semantics of the protected network or the contents of the inspected websites. It rather relies on URL and web traffic features. It gains robustness from being isolated into fog and cloud nodes.

Poon and Miri [63] suggest outsourcing anti-virus as a service to scan encrypted data outsourced in cloud storage. The authors assume that the trusted party (anti-virus provider) send the virus signatures to the Cloud provider after encryption by the consumer’s public key. The Cloud provider conducts a shallow analysis and detect any match between the encrypted data and signatures. The analysis results are sent back to the data owner for decryption and then forwarded it to the trusted party for validation. In this regard, the service does not gain any visibility into the contents of the analyzed data to maintain its owner’s privacy. However, the service renders
impractical in reality as anti-virus provider is expected to be reluctant to share their signatures with third parties who may collude with their consumers and reveals the shared signatures. The service robustness remains an open question especially that the effectiveness of the service for processing encrypted data is also subject to practical verification.

**Email Security**

Zawoad et al. [61] introduce spam analyzer as an embedded security service from Internet service providers (ISP) to automatically block phishing websites. The service employs blacklisting technique that maintains the list on demand to avoid any duplications. It specifically inspects URLs collected from spam emails. It does not have any visibility into the semantics of the inspected websites. It rather relies on the hash value of the website’s files fetched from crawling these URLs to decide if it is for duplicate or new discovered malicious site. The service hides its nature by changing the IP addresses of its hosting VMs. This, in turn, may add robustness against reverse blacklisting by attackers. Due to relying on shallow inspection of URLs, the service is vulnerable to evasion from more sophisticated phishing websites not necessarily discoverable through spam emails. The privacy of email owners and users remains as an open subject that needs to be carefully considered.
3.3. CLASSIFICATION OF SECAAS

Network Security

Varadarajan and Tupakula [34] propose a security service that enables cloud providers to protect their infrastructure and provides consumers with control to specify additional security analysis against insider threats, malware (e.g., rootkit), and network-based (e.g., DoS) attacks. The proposed solution is integrated with the VMM at the visualization layer. In this context, the solution can have a good visibility of the network traffic for the hosted VMs. It is also able to acquire a high robustness against attacks due to its isolation from the monitored VMs. This, in turn, ensures that an attacker may not tamper with such service even if the monitored VMs are completely subverted because software running in a VM cannot access or modify software running in the VMM. However, rootkits attaching to unhidden processes can still evade the proposed service.

Guenane et al. [40] propose firewall as a service that leverages network virtualization technologies to protect against network-based (i.e., DDoS) attacks. The Cloud provider has the full control and visibility over the service to filter and manage the network traffic of their consumers as well as allow the traffic transmission and distribution only if it is legitimate. In this respect, consumers' privacy is exposed to the provider. The firewall service is isolated from attacks as it is deployed in VMs separate from the consumers' VMs.

Sun et al. [51] propose security as a service for cloud applications. The proposed service is offered as an integrated feature by cloud (infrastructure)
providers to monitor the network traffic between microservices which make up a candidate application and are hosted over distributed VMs. The security service hands the control to the Cloud consumer (application provider) to specify the policies that enforce their security requirements as well as track any anticipated violations. The service gains visibility to the internal communication and external traffic of the monitored microservices due to its attachment with the hypervisor. It is also deployed in isolation to the hosting VMs, which as a result adds robustness against attacks targeting the candidate application.

Yin et al. [60] integrate anti-virus as a service in the virtualization layer to protect hosted VMs. The service is considered as an add-on feature from providers offering the Cloud infrastructures. It is combined with other services including monitoring network traffic and VM memory to detect abnormal activities. The service employs typical anti-virus engines to scan the target files on demand. Giving its attachment to the virtualization layer, it has visibility to capture the file operations conducted in the host VMs. This also supplies the service with robustness against attacks targeting the VMs. However, its robustness against evasion remains relative to the effectiveness of the employed anti-virus engine and falls short to cover unknown viruses beyond its signature database.

**Vulnerability Scanning**

Pappas et al. [41] propose data flow tracking as a service to track data flow and detect attacks exploiting application vulnerabilities at runtime leading to
3.3. CLASSIFICATION OF SECAAS

Unauthorized access and data leakage. The service is controlled by the Cloud provider to allow consumers, who host their cloud applications over the provider's resources (infrastructure or platform), to taint their users' data. The service, in turn, allows applications' users to gain visibility by directly tracking and monitoring the audit trails of their data manipulated by such applications. The service is prone to be evaded by malwares and attacks causing implicit data flows or change their behavior during the analysis.

Tung et al. [44] propose security testing as a service (S-TaaS) to scan vulnerabilities in web applications. The S-TaaS solution employs typical black-box analysis to simulate intentional and inadvertent attacks searching for potential vulnerabilities. In this context, the S-TaaS service does not have visibility into the application's internal logic and semantic structure. Therefore, it can provide little assurance that an application is immune to attacks and likely prone to evasion by stealth attacks. S-Taas also does not provide control to application's consumers to tune the security testing according to their requirements.

In Chapter 5, we propose vulnerability analysis service, called SDaaS [150, 151], to proactively reveal potential information flow violations in SaaS applications and mitigate the risk early in the development phase, well before deployment. Our solution follows whitebox approach, thus, it has visibility into the semantic structure of the analyzed application. It is fully controlled by the trusted party, while the consumer has the flexibility to identify the scope of the analysis to govern data
integrity, confidentiality, or both. The service is robust against evasion as it considers implicit as well as explicit insecure flow paths. It mainly relies on static analysis, so it is immune from being evaded by malicious applications changing their behavior during the analysis. It is robust against attacks targeting the candidate application as well as other attacks. This is due to its deployment over secured cloud environment.

Security Assessments

Kaliski Jr and Pauley [47] provide a paradigm of risk assessment as a service for the Cloud. Although the proposed paradigm does not provide implementation guideline, it provides recommendations to overcome the challenges originate in the Cloud environments. The authors suggest that the risk assessment service can be offered continuously through different operational methods viz., self-assessment by the Cloud provider, delegation to a trusted party with privilege access to the provider's infrastructure, and consumer without the need of privilege access. The control, visibility and robustness of the assessment service vary as regards the conducted operation design.

Torkura and Meinel [48] propose security assessment as a service for cloud infrastructure management software (e.g., OpenStack). The authors employ the service to alert cloud providers to pay attention to generate security patches for unresolved yet security issues. They discover the unresolved issues by contrasting information from two sources: a) external vulnerability databases managed by the community and b) internal bug trackers maintained by the Cloud provider. In this
3.3. CLASSIFICATION OF SECAAS

regard, the service visibility is limited to the provider's bug reports. It is subject to evasion as the information from the aforesaid sources may suffer from incorrectness or incompleteness which negatively can affect the effectiveness of the security assessment.

Rak et al. [50] provide a different insight about SLA-based (service-level agreement) security assessment service in cloud infrastructure. The assessment service monitors the compliance of the Cloud provider with the security requirement in the SLA. Based on the assessment result, the security services offered by the Cloud provider are either activated or adjusted to avoid any violations. The assessment is fully controlled by either the Cloud provider or a trusted party (broker) according to its operation design. It also has privileged visibility into the Cloud infrastructure to perform the assessment properly.

Gonzales et al. [62] propose Cloud-Trust, a security assessment model that can be used by third parties to audit and quantify the security of cloud infrastructure services. The assessment mainly builds a probabilistic graphical model which considers the potential advanced persistent threat (APT) paths that may target the Cloud infrastructure under assessment, the security mechanisms taken by the Cloud provider to mitigate these potential threats, and the actual features and controls permitted by the provider to the consumers. Based on this model, Cloud-Trust estimate probabilities of APT infiltration and detection. These probabilities provide a
quantitative assessment of the degree of confidentiality and integrity offered by the Cloud provider.

### Intrusion Management

Alharkan and Martin [33] propose intrusion detection system as a service (IDSaaS). The Cloud consumers are responsible to deploy, configure, and manage the IDSaaS to monitor the incoming and ongoing network traffic of the VMs hosting their applications in the Cloud. Hawedi et al. take [35] similar steps to introduce intrusion detection as a service that provides cloud consumers with flexibility to control the IDS rules according to their security requirements. Men et al. [36] extends this research line and introduces privacy-preserving signature-based IDS to guarantee the privacy of the inspected network traffic. The proposed solution enables the Cloud consumer to compute the fingerprints of the IDS as well as the inspected network packets and encrypt them with a secret key before sending them for analysis to the Cloud provider. Although such a solution can prevent cloud provider from breaking the confidentiality of the packet contents, it causes computation overburden.

All these services [33, 35, 36] mainly employ typical network and signature-based IDS (e.g., SNORT). Hence, they may detect only known patterns of intrusions and be prone to evasion attempts from unknown, variant patterns of network-based attacks. They may be also prone to attacks subverting the VMs hosting the proposed
3.3. CLASSIFICATION OF SECAAS

IDSs. These solutions lack any integration with the hypervisor, and thus they cannot ensure full visibility of all traffic on the virtual network layer.

Tupakula et al. [52] rely on hypervisor-based technology (i.e., managed network software) to overcome the aforesaid limitations. In this respect, their proposed service gains visibility and robustness. It also provides control to the consumers to customize the security policies that govern intrusion detection at the infrastructure layer in terms of network, access control, and cross-VM violations.

Continuous Monitoring

Zhou et al. [46] introduce leveraging introspection for security monitoring services to protect VMs hosted on cloud infrastructure from malware and network-based attacks (e.g., rootkits). The service has deep visibility into the monitored VMs due to its deployment in the virtualization layer, managed in a privilege domain, and enabled by the virtual machine introspection (VMI) technology. For the same reasons, the service also gains high robustness for being isolated away from the attack surface targeting the infrastructure layer (VMs).

In Chapter 6, we propose real-time security monitoring as a service (SMaaS) [152] to detect security anomalies in cloud analytical applications. The SMaaS service is offered as an add-on feature from the Cloud (analytic) provider to their consumers, while the trusted party is delegated to operate the security monitoring. The service provides detailed reports and alerts to consumers to keep them updated about the security posture of their data processed by the monitored application. It
gains visibility from modeling the application's execution in terms of information flow based on log data analysis. It is robust against attacks as it is deployed in isolated, secured cluster away from the monitored cluster. The service works in transparent manner; thus, it is robust against evasion from attacks conceal their behavior during the monitoring.

3.4 Comparative Analysis

In this section, we contrast existing SecaaS work from a common point of view detailing their strengths and weaknesses. The analysis considers the implications of the different aspects of the service operations and security solutions when assessing the effectiveness of the current SecaaS work in mitigating the target threats.

The SecaaS work [30, 34, 46, 52, 60] that are designed to deploy or have attachment with the virtualization layer gain unique advantages over other existing approaches due to their deep visibility and high robustness against attack attempts. The acquired visibility offers the capability to reason about low-level information related to the access interactions between software running on the VMs, activities in the underlying physical hardware, and the traffic in the virtual network. However, the semantic gap remains a challenge that needs to be resolved to convert this low-level information into high-level semantic view useful for the security inspection. Thereby, few approaches operate virtual machine introspection (VMI) and semantic construction technologies [34, 46, 52]. Some other approaches [51, 52, 60, 62] have been inspired by the software development network (SDN), an emerging technology
3.4. COMPARATIVE ANALYSIS

simplifying and centralizing network management, to launch their security initiatives.

On the other hand, the robustness of other existing approaches, deployed as SaaS [32, 55, 58, 62, 63, 150-152], IaaS [31, 33, 35-38, 40, 42, 44, 45, 47-49, 51, 53, 54, 56, 57, 59, 61], or PaaS [39, 43, 50, 41], against attacks is more contingent on the security measures taken by the responsible entity when designing and operating their solutions.

Irrespective of how the SecaaS is deployed and delivered, the robustness of all existing approaches [30-61] against evasion attempts as well as successfully mitigating the target security threats is strongly dependent on the employed security solution. Protecting SecaaS solutions against evasion attempts is seldom considered in existing approaches. Only few approaches [31, 57] along with our solutions [150-152] show protection against specific cases of evasion.

Most of existing approaches [32-34, 36-43, 45-47, 49-63] are deployed over public cloud environments. Although this tendency is aligned with recent industry expectations [29], it broadens the threat landscape of such approaches and threatens their robustness against attacks. This, in turn, adds more obligations over responsible entities to carefully consider how security measures can be hardened when designing and operating their SecaaS. However, this is not the case in existing approaches. There is lack of awareness in proposing such measures to secure the proposed SecaaS work, except limited solutions [150-152].
Similarly, the protection of the privacy of consumer’s data either monitored or collected for security inspection has obtained seldom attention. Only few approaches [36, 58, 63] shows preservation measures. In majority, the privacy issue has been managed in careless manner without taking proper measures in the course of designing and operating the SecaaS solutions.

The adoption of offered SecaaS solutions is highly promoted by the control property [73]. We believe that not only the degree of control provided by the SecaaS responsible entity but also the transparency accompanied with such control are together considered the most tangible benefit given to consumers. There is lack of approaches following this concept, as opposed to our solutions [150-152]. Even though there was quite awareness from some of the proposed approaches [34, 35, 37, 39, 43, 45, 51-55, 41, 150-152] to provide consumers with some degree of control, providing transparency to keep consumers aware of the security posture of their protected assets has not been managed with the same rigor. We noticed that the degree of control as well as transparency broadly differ according to the service operation and the security solution of the SecaaS work.

“Prevention is better than cure”, this famous proverb we believe to shape the ultimate goal in cybersecurity. However, only some approaches [30, 37, 43, 45, 49, 53, 54] share the same vision as our solution (SDaaS) [150, 151] and take on proactive defense tactics. No doubt that being proactive and taking security measures such as access control, encryption, and static analysis, among others, before security
threats take place is a good practice. However, in reality, not all of the threats can be mitigated proactively. In case a threat occurs, detective and reactive measures come into the picture to play crucial role. The majority of existing approaches [31-36, 39, 40, 44, 46, 47, 50-52, 55-57, 59, 60, 63, 41] adopt detective defense plans. Few approaches [34, 42, 52] are augmented by reactive along with detective plans like our solution (SMaaS) [152]. Some other approaches [48, 58, 61, 62] work on only reactive tactics, those are considered the least effective solutions. Hence, SecaaS solutions working in harmony to achieve balance between these three defense tactics are highly recommended to dictate a complete security protection.

3.5 Summary

The SecaaS model leverages the great advantages of the Cloud like low-cost and high-scalability to meet the growing cybersecurity demands. This chapter presents the following contributions: (i) it discusses the necessity of a comprehensive taxonomy of SecaaS in the Cloud; (ii) it proposes a taxonomy of existing SecaaS work from a tri-dimensional perspective: service operation, security solution, and threat; (iii) it sheds lights over the trends in the SecaaS research; and (iv) it provides a comprehensive review along with a comparative analysis of the available SecaaS research work.

The knowledge from the taxonomy can be used as a roadmap for SecaaS consumers, especially software service providers, to assess the various offerings and select the proper solution which fulfills their defense plan, meets their security
requirements, and mitigates their identified threats. This is especially important for embedding security throughout the engineering process of SaaS applications as it will be further explained in Chapter 4.
CHAPTER 4

Integrating Security in Cloud Application Development Cycle

4.1 Introduction

There has been a tremendous change in the way organizations and individuals tend to leverage SaaS applications in their cloud business. Cloud SaaS applications enable this evolving business of every shape and size to gain instant access to various world-class IT capabilities on demand through the Internet. The more this business evolves, the more software service providers compete for rapid development to meet the market speed.

This growing trend has reshaped application development landscape through various models, ranging from Waterfall, Agile, and DevOps, among other methodologies, to quickly and cost-effectively accelerate building applications with scalability, elasticity, and adaptability for the Cloud. Due to this trend, software development process is more aligned to functionality-needs, putting security-needs down in the priority list. This also results in delivering applications with potential
vulnerabilities that can lead not only to leakage of consumer's data but also loss of provider's credibility, which adds heavy financial loss for both.

Security concerns are further elevated due to the lack of visibility, control, and regulatory enforcements over consumers' data processed, stored, and accessed by SaaS applications outside the consumer's trusted boundary. These concerns make from the Cloud application layer a soft attractive target of attacks.

Despite the research approaches that have targeted detecting and mitigating security breaches, many limitations with respect to cloud environments still exist. Traditional security measures (e.g., intrusion detection and firewalls) and standard regulations (e.g., ISO "International Organization for Standardization") mainly protect the perimeter of cloud applications against network-based and physical-based attacks. Still, these measures may be circumvented due to the tendency of attacks toward the insecurely developed applications. Core services for authentication and authorization, offered by cloud hosting providers throughout the deployment process of SaaS applications over their resources, are not also suffice to protect against these attacks. All these efforts do not directly address the root cause of the problem which is the failure of software and hosting providers to take a shared security view of SaaS applications from inception through deployment and beyond. Thus, the risk from adopting SaaS has raised the necessity to consider security-needs at the same priority as functionality-needs.
As previously discussed in Chapter 2, research approaches devote for secure SaaS development reflect the lack of widely-accepted standards about SaaS development lifecycle. Most of these approaches [80, 84-88] address certain development methodology model. As opposed to our platform that provides a comprehensive insight, these approaches mainly focus on specific aspect of incorporating security for SaaS development or deployment. They target either specific security practice [79, 80, 83] or particular phase in the development lifecycle [81-88].

For this reason, the research in this chapter attempts to help software and hosting providers to articulate the need of security throughout the engineering process of SaaS. This chapter specifically proposes a novel comprehensive platform to envision how security can be incorporated in different phases of the SaaS development lifecycle (e.g., SaaSDLC) [149]. The platform introduces the notion of information flow control as a security service (IFCaaS). IFCaaS delineates the groundwork to feature cloud-delivered IFC-based security activities and services to provide different style of defenses. Such services provide end-to-end data-oriented protection and harden SaaS applications against prevalent security threats. The platform ties such activities and security services together to achieve three main tactics: administrative, preventive, and detective for improved response and resilience against the threats.
4.2. WARM-UP DISCUSSION

We demonstrate the effectiveness and efficiency of two core services (i.e., static analysis and runtime monitoring) in the platform via proposing two different frameworks. Chapter 5 introduces Security Diagnosis as a Service (SDaaS) framework to represent the static analysis service proposed in the development phase [150, 151]. Chapter 6 presents Security Monitoring as a Service (SMaaS) framework to exemplify the runtime monitoring service proposed in the deployment phase [152].

4.2 Warm-up Discussion

We built our platform based on the SaaSDLC initiated by Microsoft specialists [138]. The SaaSDLC is different than the traditional SDLC as the former associates the principles needed for the Cloud service business model. It also provides a more general standpoint supporting different development methodologies for SaaS applications including the Agile model, the most popular and widely used one. As depicted in Figure 4.1, the SaaSDLC comprises of six stages: envisioning, evaluating, planning, subscribing, developing, and operating. In this section, we present these stages in brief. They are explained in depth in Microsoft specialists' article [138].
**4.2. WARM-UP DISCUSSION**

*Envisioning:* The envisioning of SaaS applications is similar to other typical applications. At this stage, scope, vision, and needs of a candidate SaaS application are defined and refined from a business standpoint. Determining the business plan and potential hosting providers are main deliverable from this stage.

*Evaluating:* This stage focuses on evaluating potential hosting providers by examining their services in terms of budget, reliability, availability, scalability, performance, disaster recovery, and compliance. Refinement of a candidate application's architecture can be needed to suit the selected provider's platform.

*Planning:* At this stage, the requirements of the application are gathered; as a consequence, the design plan of the application considering the functional specifications, schedule, resources, design, technical architecture, and operation monitoring is created. This stage can be iterative where planning needs to focus on not only current iteration but also long-term vision.

*Subscribing:* A product subscription, pricing contracts, and a service level agreement are finalized between the software service provider and the hosting provider at this stage. Backup and recovery strategies can also be delivered throughout this stage. During every iteration this stage can be revised to verify that the design plan is met by the selected hosting provider.

*Developing:* This stage includes developing the application based on the specified requirements, chosen architecture, and selected resource capabilities. It also covers testifying that the application complies with the identified design plan. This stage can
be iterative where the development scope per iteration is identified according to the planning stage based on the specifics of the application.

*Operating:* The operational aspects defined through the former phases (i.e., evaluating and subscribing) are employed in this stage. This stage specifically combines the deployment and maintenance of the application. It covers setting up and testing the backup and disaster recovery as well. It also includes applying frequent updates and security patches as needed.

### 4.3 The platform

#### 4.3.1 The Platform Architecture

The platform introduces Information Flow Control as a Service (IFCaaS) notion. It envisions IFC-based activities and services to interweave security into every phase of SaaS development cycle (e.g., SaaSDLC). It mainly aims to offer different style of defenses as security services that provide end-to-end data-oriented protection and harden cloud applications against prevalent security threats. The security services offered by the platform need to be built on a foundation that manifests the following requirements at a fundamental level in order to receive acceptance and adoption in the Cloud:

- Ease of deployment.
- High scalability and performance.
- Convenience and ease of management.
- Adaptability to the unique characteristics (i.e., challenges) of cloud applications.
4.3. THE PLATFORM

Figure 4.2: The platform to interweave security in the SaaS Development Cycle

Figure 4.2 depicts the platform architecture to meet the following three tactics towards plugging IFC-based security activities and services into the different phases of SaaSDLC.

- **Administrative**: Preliminary activities to address security requirements, threat modeling, and security practices as well as handle responsive measures including risk assessment and threat mitigation.

- **Preventive**: Services to proactively mitigate the security threats in the development phase.

- **Detective**: Services to reveal security threats and govern enforcements at runtime through the deployment phase.

The proposed security activities and services are further detailed in Section 4.3.3.
4.3. THE PLATFORM

### 4.3.2 Responsibility Zones

The Cloud service models considered in this work consists of four main entities: software service provider, hosting provider, trusted party, and SaaS consumers. The hosting provider offers the computing resources either as platform or infrastructure needed to deploy and run SaaS applications in the Cloud. The trusted party here represents a cloud provider that can offer specific IFC-based security services.

Our platform supports the philosophy that security is a shared responsibility between the service provider, hosting provider and the trusted party. It mainly aims to

![Diagram showing responsibility distribution w.r.t the platform security services and activities.]

**Figure 4.3:** Responsibility distribution w.r.t the platform security services and activities.
elevate the trust of not only SaaS consumers but also service providers in the security of cloud applications.

The platform’s benefit is twofold: a) it helps service providers to discover and mitigate attack venues that may threaten the security of their offered applications at different phases of SaaSDLC; and b) it, in turn, enables SaaS consumers to choose a protected application to process their sensitive data with a trust in its security.

Figure 4.3 shows the responsibility distribution which is identified with respect to the proposed security services and activities through the SaaSDLC. Regardless of the deployment model of the candidate application, the service provider has solo responsibility of all the security activities that should be conducted during the design phase. However, during the subsequent phases, the service provider defines a defense plan in the context of the designated security activities and services. Thus, the service provider responsibility is shaped according to this plan as well as the deployment choice of the application. Further details about the responsibility distribution are discussed in Section 4.3.3.

### 4.3.3 Provisioning Security into the SaaS Development Cycle

In what follows, provisioning the proposed security activities and services are explained in the light of the SaaSDLC phases.

The SaaSDLC originally comprises of the aforesaid six stages. However, we discuss our platform from higher-level perspective by grouping these stages into three core phases: design, development, and deployment, as illustrated in Figure 4.1. The
proposed activities and services are considered inner steps to interweave security into the six stages forming the core phases. They are built around multiple levels of defense throughout the Cloud stack to protect SaaS applications and the processed data. Thus, they can provide redundant protection for better resilience against security threats.

The platform is established around supporting various development methodologies. The activities proposed in the design phase can be engaged in a linear fashion through developing the monolithic applications adopting the Waterfall model; while the security services and supplementary activities employed in the development and deployment phases can be conducted after the software deliverables are all complete. Differently, modern development methodologies (e.g., Agile, DevOps, etc.) break down the software deliverables into functional components to be developed in iterative and incremental fashion. In such a case, the activities and services, in the design and development phases, are performed on a regular basis throughout each iteration; while the activities and services in the deployment phase are carried out after the recent release of the application is delivered.

**DESIGN PHASE**

Starting with clearly identifying the security requirements and threat modeling of the candidate SaaS application, which differ according to application type (e.g., operational and analytics), at the design phase are foremost activities in the platform. Such activities focus mainly on the application security with respect to mitigating
data breaches and governing data integrity, confidentiality, and availability. The software service provider is the main actor behind these activities. These activities are important to assist the service provider in choosing the appropriate tactics as well as embracing the proper measures and services through the subsequent phases in order to achieve the designated requirements and protect against the identified threats. Hence, they play a vital role affecting the service provider's decision when evaluating the Cloud providers (e.g., hosting provider or trusted party) capabilities to meet that purpose. The knowledge from the taxonomy introduced earlier in Chapter 3, can be foremost helpful for the service provider to achieve this goal. It can help the service provider to choose the appropriate SecaaS solutions. Thereafter, the service provider can plan the best security practices to help plot the course of the desired measures and services in the subsequent phases of the SaaSDLC.

**Security Requirements Review:** Gathering the security requirements, oriented around data and control flow through a candidate application, is the main input to the platform. This activity can also include identifying the value and the security levels of data processed by the candidate application in terms of its integrity, confidentiality, and availability.

**Threat Modeling:** This activity includes identifying the potential threats against the candidate application with respect to the gathered requirements. It is a core activity to help the service provider decides the proper measures and services to be employed during the subsequent phases. Such decisions also affect the evaluation and selection
of the Cloud providers (i.e., hosting provider and trusted party) who can fit the security purpose.

**Security Practices Validation:** Once the threat landscape is identified, the resulting activity is deriving a defense plan to plot the course of the appropriate tactics in terms of activities and services through the subsequent phases to achieve the designated requirements and protect against the identified threats. Further, the service provider reviews and validates that the security practices (i.e., services), which are offered by the selected cloud provider (e.g., hosting provider or trusted provider), satisfy the defense plan.

**Service Subscription:** The last step in the design phase is administering the service subscription with respect to the service being planned. This step also includes reviewing the pricing contracts and negotiating the service level agreement.

**DEVELOPMENT PHASE**

The development phase is where preventive and administrative tactics can be employed for proactive mitigation of the identified security threats. The preventive tactics can be achieved through two different IFC-based security services: secure coding and static code analysis. Both services at the end can deliver risk assessment as a final outcome from the security inspection that they conducted. In this sense, an important administrative tactic comes into the picture to help the service provider prioritizes the remediation efforts by focusing on the most significant vulnerabilities and critical violations.
4.3. THE PLATFORM

*Secure Coding:* In this service, IFC is incorporated in the programming languages in the form of libraries. In another word, security-type systems are included in the programming language pool offered by the hosting providers to be used by the service providers in developing their candidate SaaS applications. In case that the candidate application is deployed on infrastructure as a Service (IaaS) model, the full responsibility shifts to the service provider. Such mechanism makes from the security inspection an integral part of implementing the application's code. It specifically facilitates to the service providers to be responsible for annotating the types of variables and expressions in their application programs to express the security policies that govern the use of the annotated types, where these policies become enforceable at the compiler-time to detect potential violations.

*Static Analysis:* In this service, IFC is examined through static code analysis, which is provided as a service by a trusted party, to certify that a candidate application complies with the security policies which govern data integrity and confidentiality. The hosting provider can also initiate the incentive of the service provider to attain such security certificate before the application is deployed on its resources. Such service externalizes the security inspection and sets the service provider free to concentrate on the business logic for developing the candidate application; whilst the trusted party is put in charge to perform the IFC inspection automatically including data annotation, policy specification, and annotation tracking.
4.3. THE PLATFORM

*Risk Assessment:* Automatic risk assessment can be included within the security services conducted in the development phase. Such assessment can mainly rely on the results attained from the IFC security inspection employed by these services. The risk assessment activity here provides an administrative measure to the service provider to prioritize remediation efforts in a way inclined with the actual detected vulnerabilities and violations.

**DEPLOYMENT PHASE**

In the deployment phase, detective and administrative tactics are provided for ongoing runtime protection of a candidate application. There are two different IFC-based services for revealing security violations when an application is executed: runtime enforcement and runtime monitoring. Both services can also support risk mitigation to immediately react against the recognized violations.

*Runtime Enforcement:* In this service, IFC is enforced through runtime analysis. Such a mechanism can be applied at different layers of the Cloud stack ranging from the platform, infrastructure, to the way down to the virtualization layer. As a result, the analysis granularity for tracking the data flows through a candidate application differs from one layer to the other. The service provider is responsible for specifying the security policies and annotating the data if the runtime enforcement service offers a middleware APIs by the hosting provider. On the other hand, the service provider's responsibility diminishes when the IFC is transparently supported in the designated layers.
4.3. THE PLATFORM

**Runtime Monitoring:** This service provides various out-of-the-box monitoring solutions where IFC is incorporated with big data analytics, machine learning, and deep learning techniques to employ a more advanced security inspection. Such service is offered by the hosting provider to promote the incentive of the service provider to monitor the security of its candidate application in post-deployment. The hosting provider is responsible to facilitate the collection of the required auditing, monitoring, and logging data of different granularity from the application's cluster as regards the employed solution; while the security inspection is delegated to be conducted by a trusted party over the collected data. In case of IaaS deployment choice, the service provider holds the responsibility of configuring the application's cluster bundled with the monitoring features by following the trusted party instructions.

**Threat Mitigation:** Threat mitigation is considered an essential administrative measure after detecting security violations. This activity can apply various strategies to limit or stop the extent of detected violations ranging from sending alert notification to taking concrete reactive decisions. It is achieved as an integral part of the services conducted through the deployment phase. Depending on the designated strategy, it can render the attack attempt unsuccessful, or at least prevent further breaching of a compromised application.
4.3.4 The Platform Principles

The CSA identified the top twelve treacherous security threats to cloud security [6]. The proposed platform achieves various principles to help in mitigating and addressing five of these identified threats. Recall that the platform ties risk assessment or threat mitigation with the employed services; it can therefore provide effective protection and improve the response against these threats. The platform's principles are summarized as follows:

Secure Data: The platform provides several services such as the secure coding, static analysis, runtime enforcement, and runtime monitoring, which are equipped with different information flow control mechanisms, to mitigate data breaches. In this sense, the platform is capable of protecting the data processed by SaaS applications not only before but also post deployment.

Secure Interfaces and APIs: The platform provides dedicated services such as static analysis and runtime enforcement to mitigate insecure interfaces and APIs threat. These services are in advantage to inspect and detect insecure flows originating in insecure API communications including inter-service calls in an application. The platform can provide a proactive and reactive protection against such communications with different granularities from various layers in the Cloud stack as regards the employed service.

Reveal System and Application Vulnerability: The platform provides dedicated services such as secure coding, static analysis, and runtime enforcement to reveal
application vulnerabilities. Based on the chosen service, the detection can be achieved before or post deployment.

**Mitigate Malicious Activities:** The platform provides monitoring service to devote for online protection against vulnerable, malicious, and misconfigured tasks or nodes running SaaS applications. This service can correlate the monitoring data from different sources or layers (e.g., compute, network, and application) in the Cloud stack.

**Mitigate Shared Technology risks:** The static analysis, runtime enforcement, and runtime monitoring services allow the platform to enforce and monitor security policies on shared resources and eliminate attacks across different layers in the Cloud stack such as compute, storage, network, and application. These services help in isolating sensitive data and securing its flow across the shared resources more effectively.

**4.4 Summary**

Giving the rising necessity to consider security as a first priority for cloud SaaS applications, this chapter presents a novel platform for weaving security into the engineering process of such applications. This is achieved by integrating IFC-based activities and services through the different phases of the development lifecycle. Such activities and services devote for defense-in-depth via three main tactics: administrative, preventive, and detective for improved response and resilience against prevalent security threats.
4.4. SUMMARY

In this sense, the platform provides a comprehensive insight to bring service providers, hosting providers, and trust security providers together to share the responsibility and consider security throughout the development of cloud applications rather than in isolation as an ad-hoc after deployment. This, in turn, helps develop secured SaaS applications, counteract prevalent security threats upfront, and provision security services for protecting applications and their processed data, pre and post production.
CHAPTER 5

Security Diagnosis as a Service for Cloud SaaS Applications

5.1 Introduction

Recent years have witnessed increasing momentum for Software as a Service (SaaS) adoption. SaaS is a prevalent software delivery model in the Cloud. Web service APIs are the most preferred way of exposing SaaS (e.g., Google, Amazon, Yahoo, Facebook, and Twitter). They become critical components in SaaS applications built over either monolithic multi-tier or microservice architectures. Cloud consumers leverage cloud-based service (e.g., RESTful) APIs through an Internet browser or in an aggregated fashion with other services (called a mash-up) to enrich their client applications for a mobile, Internet of Things (IoT), or Web workforce. Technologies today make it quite easier to develop and provide these services to capitalize SaaS applications agility, availability, and performance scalability. However, the impressive gains of such technology shift may come at the cost of security.
Data breaches due to application vulnerabilities as well as insecure APIs and interfaces are considered the top critical issues to cloud security [6]. Cloud web services represent a front-end interface opening the door for web-based attacks. They can leave SaaS applications prone to novel attack venues as well as existing ones. Information flow vulnerabilities including SQL injection (SQLI), XML Path injection (XPathI), Cross Site Scripting (XSS), and command injection (CMDI), to name a few, are reported as the prevalent attack venues towards cloud services [8]. The common reason of these vulnerabilities is the insecure information flow due to the improper validation and sanitization of data before used directly in performing sensitive operations or sent to other services. Successful exploitations of such vulnerabilities can not only result in a breach of the integrity and confidentiality of one tenant’s data but also extend to cross-tenant violations.

As discussed in Chapter 2, researchers over the past years have focused on security-type and taint-analysis based systems. Type-based systems [98–101] may offer restrictive programming languages and require annotations that hinder their practical use. Taint-based systems [102-104] focus on statically vetting insecure explicit flows and ignore implicit flows. This hampers their effectiveness in settings where strong guarantee of information security is a must. More recent approaches [41, 92-97] focused on tracking data flow at runtime. They pinpoint security breaches after an application is deployed or even after damages may have already happened. Such approaches usually entail modification to the underlying environment, which
hamper their potential adoption. They cause runtime performance overhead and are prone to evasion by attacks that may change their behavior during the analysis. Other approaches advocated different techniques including intrusion detection [33, 109, 110] and penetration testing [111, 112] to detect information security violations in the Cloud. Intrusion detection mechanisms do not guarantee end-to-end security verification. Penetration testing treats an analyzed application as black box; thus, it provides little assurance that an application is immune to attacks.

Towards addressing these limitations, our research work aims to reestablish the trust of cloud service providers in the security of their applications and service consumers in the security of their data. This goal is achieved by putting a trusted party in charge to diagnose and certify the security of cloud SaaS applications. This chapter presents our Security Diagnosis as a Service (SDaaS) framework. It is a novel quantitative static information flow analysis solution for security diagnosis of SaaS applications.

SDaaS mainly relies on static analysis to protect against information flow vulnerabilities. The heterogeneity, complexity, and distributed nature of SaaS applications originate several challenges. These challenges are undecidable problems hindering static analysis, which are summarized as follows:

- **A SaaS application has multiple entry points.** An application’s runtime execution is per request. Mapping incoming HTTP requests to their corresponding entry method is done implicitly by the runtime environment. The
5.1. INTRODUCTION

analysis needs to precisely approximate an application’s lifecycle considering only feasible control flow paths.

- **An analyzed application is composed of web services.** Interactions between services can be inside a single application or across different applications (cloud-to-cloud, cloud-to-fog, cloud-to-client, etc.). Capturing the control and data flow stem from these interactions is crucial for an accurate analysis.

- **An analyzed application typically relies on many other external libraries.**
  
  Analyzing the entire libraries’ code is very expensive.

  We propose several modeling strategies to boost our framework in solving the aforesaid problems. This is crucial to precisely capture data and control dependences inside an application. These dependences are represented in System Dependence Graph (SDG), upon which the framework employs information flow control and program slicing techniques for security inspection.

  The framework derives several quantitative metrics about the inspected slices. These metrics are used as basis of our rule-based approach to diagnose an application’s behavior (i.e., trustworthy, vulnerable, or malicious). Grounded on the diagnosis results, the trusted party decides whether to certify a candidate application or not.

  The framework also provides a comprehensive report about an analyzed application’s security status in terms of potential risk, behavior diagnosis, and security risk rating. The report is available through a convenient online dashboard.
5.1. INTRODUCTION

Such report is beneficial to prioritize remediation efforts by focusing on the most significant vulnerabilities and critical issues.

SDaaS exemplifies the static analysis service offered in the development phase, introduced in Chapter 4. SDaaS is built on a SaaS model, provided on a subscription basis, and offered over the Internet without any requirements for environment modification, software installation, hardware set-up, or special training.

Our main contributions in this work are summarized as follows:

- We propose a novel framework (SDaaS) that leverages quantitative information flow analysis. It aims to diagnose an application’s security status in terms of potential risk, behavior diagnosis, and risk rating.

- We introduce algorithms to model the runtime lifecycle, inter-service interactions, and external-library invocations inside SaaS applications. These algorithms show how to solve these challenges as part of the data flow analysis while building the call graph on the fly. This is crucial to augment static analysis to support SaaS applications and govern the SDG’s construction that precisely captures data and control dependences inside a candidate application.

- We present the formal representation and design of our value computation approach, as part of handling inter-service invocations. We specifically leverage Inter-procedural Distributed Environment (IDE) analysis to compute the value of the relevant fields required to statically reason about the targeted service.
We demonstrate the accuracy, performance, and scalability of our framework through a set of experiments in public cloud on Amazon EC2. We evaluate SDaaS’s detection effectiveness over benchmark applications for assessing vulnerability detection tools and services. We contrast our solution with several tools comprising static code analyzers, penetration testers, and an anomaly detector. We also evaluate various performance aspects including the overall response time of our framework for different requests and the impact of the main tasks of the framework to the response time, system scalability, and resources usage.

The remainder of this chapter is organized as follows. Section 5.2 presents the operational overview of our proposed framework and the threat model it assumes. Section 5.3 presents the details of the framework. The framework implementation and experimental evaluation are presented in Section 5.4. Section 5.5 draws the concluding remarks of the chapter.

### 5.2 Overview and Assumptions

This section outlines the operational overview of our proposed framework and the threat model it assumes.
5.2. OVERVIEW AND ASSUMPTIONS

5.2.1 The SDaaS Operational Overview

The Cloud computing service models considered in this work consists of four main entities: software service provider, hosting provider, trusted party, and service consumers.

As depicted in Figure 5.1, a software service provider initially shows interest to either host or build its SaaS applications relying on a cloud hosting provider’s infrastructure or platform (1). Before such application is deployed and launched, the hosting provider asks the software service provider to attain a security certificate from the trusted party (2). The software service provider subscribes with the trusted party, which offers SDaaS (3). It submits an analysis request to the trusted party along with the bytecode of the application and its exposed web services (3). The
5.2. OVERVIEW AND ASSUMPTIONS

A service provider is given the ability to choose the type of inspection (integrity, confidentiality, or both). The trusted party, in turn, accepts the request and leverages the proposed framework to statically analyze the application to diagnose its security and detect potential vulnerabilities. Detailed analysis results are reported and published in the dashboard to the software provider (4). Based on the results, a security certificate is granted to the analyzed application and a notification is also sent to the hosting provider (5).

The importance of our service is twofold: a) it enables SaaS consumers to choose a certified application with a trust in its security to process their sensitive data; b) it also helps service providers to discover and mitigate attack venues that may threaten the security of their offered services early in the development phase.

5.2.2 Threat Model

The service consumer is untrusted and may send malicious requests to exploit vulnerabilities (e.g., injection, XSS, etc.) in the SaaS application. This misuse can lead to data integrity and confidentiality violations. On the other hand, the software service provider is untrusted, and thus the confidentiality of data manipulated by its offered services may be at risk. The software provider may permit unauthorized access and collude with external uncertified third-party’s services by sending them the sensitive data.
5.3 THE FRAMEWORK

We do not cover protection from side or covert channel attacks (e.g., physical, timing, and termination leaks). We assume the security of the underlying cloud platform and infrastructure, upon which the application will be deployed.

5.3 The Framework

As illustrated in Figure 5.2, the proposed framework comprises the following major components: Lifecycle Model Generator, Dependency Model Constructor, Vulnerability Detector, and Security Diagnosis Practitioner.

The framework takes the bytecode and metadata files of an application as input to the lifecycle model generator component. This component is responsible for applying a set of strategies to resolve entry point invocations and lifecycle callbacks. It provides a synthetic model that semantically simulates the candidate application's lifecycle in terms of entry points and callbacks.

Then, the dependency model constructor component is delegated to compute all dependencies stem from the inter-service interactions and external library invocations.
through data flow analysis. This is essential to resolve all dependences in the application and capture them into a unified SDG.

Upon the SDG, the vulnerability detector component performs information flow control relying on program slicing techniques. For inspected slices, the vulnerability detector pinpoints insecure flow paths that violate data integrity and confidentiality. It also records sanitized paths that are considered safe.

Then, the security diagnosis component derives several quantitative metrics about the detected insecure and sanitized paths. It aims to provide a solid insight about the security status of the analyzed application in terms of potential risk, behavior diagnosis, and security risk rating. Based on the results, our system decides if a security certificate should be granted to the candidate application. The components are further detailed in the following subsections. They also explain how we address the challenges mentioned in Section 5.1.

5.3.1 The Lifecycle Model Generator

The lifecycle model generator component accepts as input the bytecode and the associated configuration metadata files of a candidate application. It takes charge of modeling a candidate application’s lifecycle. It specifically creates a synthetic function that models entry point invocations and lifecycle callbacks. This section presents the details of the steps taken by this component and outlined in Algorithm 1.
As a first step of Algorithm 1 (Line 3), we leverage string analysis to parse the configuration information to identify the Java packages and classes that contain the resources of an application’s web services. The next step (Line 8) is statically analyzing these classes to recognize the entry points for each web service component in an application. These entry points are public lifecycle methods that are triggered by the runtime environment when HTTP request comes in. This is done based on the request URI, the HTTP method (e.g., GET, POST, DELETE, PUT) and the content representation (e.g., XML, JSON, HTML, Text) produced or consumed.

**Algorithm 1.** Generate Lifecycle Model

**Input:** Application Bytecode ($A$) and Configuration Files ($C$)

**Output:** Synthetic Model ($ph_{Env}$)

1: Function $GenerateModel (A, C)$
2: \[\text{declare packages} \leftarrow \emptyset, \text{classes} \leftarrow \emptyset\]
3: \[(\text{packages, classes}) \leftarrow \text{parse}(C)\]
4: syntheticModel $\leftarrow$ createPlaceholders(classes)
5: return syntheticModel
6: procedure createPlaceholders (classes)
7: \[\text{foreach class} \in \text{classes} \text{ do}\]
8: entryPoints $\leftarrow$ analyze(class)
9: $ph_{Env} \leftarrow$ createPlaceholderClass();
10: $ph_{Entry} \leftarrow ph_{Env}.\text{createMain}()$;
11: $ph_{Entry}.\text{Instantiate}(\text{class})$;
12: \[\text{foreach method} \in \text{entryPoints} \text{ do}\]
13: \[\text{params} \leftarrow \text{analyze}(\text{method})\]
14: \[ph_{Entry}.\text{createInvocation}(\text{method}, \text{params})\]
15: end
16: \[\text{syntheticModel} \leftarrow ph_{Env}\]
17: end
18: return syntheticModel
To model the application’s lifecycle, we emulate the runtime environment (i.e., container). Given the statelessness of the RESTful service components where each request occurs in complete isolation. Our strategy introduces a fine granularity service-level model as a synthetic function (Line 9-10). This function is a placeholder method \((ph\_Env)\) (Line 11) that invokes all entry points and callbacks of each service class (Lines 12-15) in proper order driven by the lifecycle specifications of RESTful services [139]. Other callback methods are eventually solved through reachability analysis. The identified entry points are stored in a database each with its associated URL path, HTTP verb, and media type.

SDaaS consequently bases the static analysis to build the SDG over the generated model \((ph\_Env)\), which serves as a main entry method of the analyzed application. In this sense, our model augments any analysis to consider the feasible control flow paths needed to execute the application instead of conservatively assuming all arbitrary orders for possible control flow paths between the application’s components. The benefit of our model is twofold: a) it is useful to reduce false alarms; b) it supports the scalability by reducing the range of the analyzed code without sacrificing the precision of the analysis.

5.3.2 The Dependency Model Constructor

The dependency model constructor component accepts as input the lifecycle model and the application bytecode. This component is responsible of computing all types of dependences: data and control inside a candidate application. It handles modeling
5.3. THE FRAMEWORK

the inter-service and external library invocations. This section starts with presenting the main steps to model these challenges. Then, it dives into the details conducted to resolve each challenge in subsections \(A\) and \(B\), respectively.

Computing the data and control dependences to construct SDG is done through two phases: intra-procedural and inter-procedural [140]. The former phase works on method-level. It computes the control and data dependences based on the control flow graph (CFG) of each method in the application. Specifically, the nodes; involved in at least one dependence, of each CFG are used to build a corresponding program dependence graph (PDG). The PDG edges represent the discovered data and control dependences.

The later phase integrates the results from the former phase and models their effects globally. The SDG is composed by connecting all the PDGs in accordance to the invocation edges of the call graph. This is achieved as follows: a) edges are inserted to connect each call node to the entry node of its corresponding called; b) edges are added between actual-in parameter nodes and their associated formal-in nodes; and c) in analogous way, actual-out and formal-out nodes are connected.

We solve the inter-service and external library invocations while building the call graph in the course of the point-to analysis. The idea is to complement the intra-procedural phase. Our approach applies Algorithm 2 which extends Lhotáček and Hendren approach [116]. It approximates inter-service and external-library invocations and propagates the involved points-to sets. As a result, the dependences
stem from these invocations are incorporated in the inter-procedural phase. The generated call graph represents an abstraction which is essential to consider the invocation structure of the analyzed application to govern the SDG construction. The call graph in our settings is defined according to Definition 5.1.

The call graph takes from the lifecycle model \( ph\_Env \) a main procedure. The graph nodes have two types: entry and call. The former type comprises of entry nodes \( N_{Entry} \) of the main procedure and all services inside an application. The call nodes \( N_{Call} \) encompass the call sites that represent inter-procedure, inter-service, and library invocations. The graph edges connect a) each entry node of a caller procedure to call sites inside it \( Entry_p \rightarrow n_c \); b) call sites appear in any procedure to the entry node of the targeted service represented by its syntetic model \( n_c \rightarrow Entry_p' \).

**Definition 5.1 (Call Graph).** A call graph for an application in our settings is a directed graph \( G = (N, E) \) such that: The graph nodes \( N := \{N_{Entry} + N_{Call}\} \) consist of the entry nodes \( N_{Entry} \) and call site nodes \( N_{Call} \). \( N_{Entry} := \{n_E \in Entry_{ph\_Env} \cup \langle Entry_p \rangle \} \) comprises of entry nodes of the lifecycle model \( ph\_Env \) and each reachable procedure \( p \) in the application. \( N_{Call} := \{n_C \in N_p\} \) covers inter-procedural, inter-service, and external library invocations such that \( (n_C, n_E) \in E_{Call} \). The graph edges \( E := \{E_{Entry} + E_{Call}\} \) comprise of edges to each call sites (either an inter-procedual call or inter-service call) appear in each procedure \( E_{Entry} := \{e_E | e_E = Entry_p \rightarrow n_c\} \), where \( p \) is either caller procedure or service, and edges to the entry of approximated targets such that \( \forall n_C \in N_{Call} \exists E_{call} := \{e_C | e_C = n_C \rightarrow Entry_p'\} \), where \( p' \) is the targeted callee procedure or the syntetic model responsible to invoke the appropriate entry point of the callee service.
Algorithm 2. Resolve dependency model
1: process allocations
2:   repeat
3:     repeat
4:       construct initial points-to relations from assignments
       and statically dispatched method calls
5:       remove first node n from worklist
6:       process assignment, store, and load edges
7:       until worklist is empty
8:       remove node currentNode from worklist
9:       if currentNode is inter-service invocation then
10:          do value computation analysis (currentNode)
11:          identify request params and response object
12:       end
13:       process every store and load edge
14:       process virtual call targets
15:       until worklist is empty
16: end
17: procedure process virtual call targets
18:     for each virtual call o.f. (x₁, ..., xₙ) do
19:       if f is inter-service invocation then
20:          fetch possible entry point from database
21:          create synthetic model as a target for service request
22:       else if f is external-library invocation then
23:          fetch library summary
24:       else
25:          for each allocation node oᵢ ∈ points-to(o) do
26:             decide the target of oᵢ.f. (x₁, ..., xₙ) with static lookup
27:          end do
28:       end if
29:     end for
30:     process resolved call
31:     add all targets of created edges to worklist
32:     if new methods have been added to the graph or
       new connections have been created then
33:       process allocations
34:     end if
35: end
36: procedure process resolved call
37:     add edge from call site to target in the call graph
38:     connect point-to between actual-in & formal-in and
     actual-out & formal-out parameters
39: end
A. Modeling inter-service interactions

Our strategy models the inter-service interactions through several steps: 1) bind the caller service to the appropriate entry point in the called service; and 2) track data flow from the caller service to the called service and vice versa. This means solving for passed parameters used as part of a request and returned variables used as part of a response. We handle these steps while approximating the dynamic binding required to generate the call graph on-the-fly during the points-to analysis.

Algorithm 3. Value Computation Analysis

Input: Node represents inter-service invocation
Output: Final Solution ($S^*$)

1: Function Value Computation Analysis (Node)
2: 
   attest If Node is inter-service invocation instruction Then
   // $G^*$: supergraph, $N$: {Nodes}, $E$: {Edges}, $S$: start node
3:    Construct CFG as $G^* = (N, E, S)$ for p
4:    Define $D$ as a finite set of Symbols
5:    Define $L = (V, \sqcup)$ as join semi-lattice, $V$: {Values}, $\bot$
6:    Define $Env(D, L)$ a set of functions from $D$ to $L$
7:    Define $\Lambda$ to represent the absence of a data flow fact
8:    Define Environment Transformer
    $t: Env(D, L) \rightarrow Env(D, L)$ as distributive function
9:    Associate a Map $M: E^* \rightarrow Env(D, L) \rightarrow Env(D, L)$ to edges
10:   Initialize $env_S \in Env(D, L)$ to $\varphi$
11: repeat
12:    foreach node $n$ in $N$ do
13:       Maintain the values of $D$
14:       Propagate the values along outgoing edges
15:    end do
16: end if
17: return $S^*: D \rightarrow V$
5.3. THE FRAMEWORK

Restful web services can be integrated into an application in various means. Commonly, standard JAX-RS client API or customized client API offered by the SaaS provider is used for this purpose. We model these API functions to derive possible values of the relevant fields of often two main objects: WebTarget and Builder, upon which the runtime system resolves the targeted service. We leverage data flow analysis to solve the value computation problem of the relevant fields. We perform Algorithm 3 as a simple approach that exhibits the essential features of the IDE analysis [115].

The expression shown at Figure 5.3, part (a) is a typical inter-service invocation statement. It invokes chain of methods. This expression creates a new Client instance. Then, it builds WebTarget object and bound its URI field to the provided value. For every invocation to the path method, it creates a new WebTarget instance by appending the provided path value to the URI field of the current target instance. After, it creates a new WebTarget instance by resolving a URI template, representing a request param, with a given name in the URI of the current target instance using a

```java
Response response = 
    ClientBuilder.newClient()
    .target("http://base")
    .path("/rest")
    .path("/get/{id}")
    .resolveTemplate("id", input)
    .request()
    .accept(MediaType.TEXT_PLAIN)
    .get();

01: v4 = javax.ws.rs.client.ClientBuilder.newClient()
02: v7 = v4javax.ws.rs.client.Client.target("http://base")
03: v11 = v7javax.ws.rs.client.WebTarget.path("/rest")
04: v14 = v11javax.ws.rs.client.WebTarget.path("/get/{id}")
05: v17 = java.lang.Integer.valueOf(#11)
06: v19 = v14javax.ws.rs.client.WebTarget.resolveTemplate("id", v17)
07: v21 = v19javax.ws.rs.client.WebTarget.request()
08: v23 = new java.lang.String[]
09: v23[0] = "text/plain"
10: v27 = v21javax.ws.rs.client.Invocation$Builder.accept(v23)
11: v29 = v27javax.ws.rs.client.Invocation$Builder.get()
12: return
```

Figure 5.3: An example of inter-service invocation: (a) an invocation expression; (b) the corresponding SSA statements.
supplied value. Later, it starts building a request to the targeted web resource by creating a Builder object for the request targeted at the URI referenced by this target instance. Next, it updates the Builder object and sets its accepted response media type field. At this point, it invokes the HTTP GET method for the current request, which in turn invokes the corresponding entry point method of the targeted web service at runtime.

Resolving this expression statically requires a precise computation of the string values of relevant fields, specifically the URI of WebTarget object and the accepted media type of the Builder object.

Our approach approximates these values as follows: first, we compute the control flow graph (CFG) of the procedure enclosing the inter-service invocation statement. Second, we design the value computation analysis. Formally, the analysis prepares IDE formulation where $D$ is the set of objects in the procedure representing a finite variable domain, $V$ denotes the value domains, $L = \{2^V, \cup\}$ is the semi-lattice with bottom element $\bot = \emptyset$, $L$ represents the powerset of the set $V$ of values of the modeled objects’ fields with join operation equals the set union. $Env(D, L)$ is the set of environments where an environment $env \in Env(D, L)$ is a map from $D$ to $L$. For any $env \in Env(D, L)$, the value of $env(d)$ at statement $S$ can be computed from domain $V$ along all paths that reach $d$ if $d$ is reachable at $S$. Environment transformers $t: Env(D, L) \rightarrow Env(D, L)$ are functions used to model the effect of procedure’s statements on the fields’ values.
5.3. THE FRAMEWORK

One of the essential features of the IDE algorithm is to extend the graph reachability to solve the expressed problem. Thus, each environment transformer $t$ can be represented as a bigraph with $2(|D| + 1)$ nodes, such that the labels $B$ of each disjoint set of the graph nodes $B = \{b: b = d \in D, or b = \Lambda\}$, where $\Lambda$ denotes the absence of data flow facts.

Each graph edge is labeled by function $f_{d,d'}$ from $L \rightarrow L$. The bigraph reflects the pointwise behavior of the function of the represented environment transformer. The purpose of $f_{d,d'}$ is to reflect the influence of the value of $d$ before a statement on the value of $d'$ after executing the statement. These pointwise representations
5.3. THE FRAMEWORK

simplify the data flow analysis. It allows the meet and composition operations over environment transformers to be implemented inexpensively as union and transitive closure over the bigraph. Figure 5.4 demonstrates examples of pointwise representations in the running example.

Our analysis defines these pointwise representations as transfer functions $\mathcal{F}$ based on the semantics of the modeled API methods. The analysis works on the intermediate statements stem from the SSA\textsuperscript{6} of the analyzed procedure code. Each intermediate statement is modeled by the composition of transformers $\mathcal{I}$ that reflect its effect on the modeled objects. Recall that $L$ is used to model the values of desired m fields, a value is correlated to the relevant field based on its unique position in the lattice. Given, $V = V_1 \times \ldots \times V_m \times \{\omega\}$ comprises of the domains $V_i$ of defined values of each field $i$ and $\omega$ undefined value. For each transfer function $f \in \mathcal{F}$ of an intermediate statement, $f(v) = (\mathcal{I}_1(v_1) \times \ldots \times \mathcal{I}_m(v_m))$, $v \in V$ and $\mathcal{I}_i$ is the transformer of field $i$. A transformer $\mathcal{T}$ over field $i$ with value $S$ can be expressed in terms of GEN/KILL sets, which are identified based on the semantics of the modeled intermediate statement. In general, $\mathcal{T}$ is denoted as follows in (1):

$$
\mathcal{T}(S) = \begin{cases} 
\omega, & S = \omega \\
\varphi, & S = \perp \\
(S - \text{KILL}) \cup \text{GEN}, & S \neq \omega 
\end{cases}
$$

\textsuperscript{6} Static single assignment (SSA) is an intermediate representation in which every variable is assigned exactly once.
At this point, the effect of executing $K$ intermediate statements in single execution path is captured by composing corresponding functions $f$ as indicated in (2). Equation (3) defines the composition. Considering several executions $N$ paths leads to define $Transf: L \rightarrow L$ based on functions in $F$ as denoted in (4).

\[ Transf(v) = \bigcup_{j=1}^{K} f_j(v), \quad v \in V \]  

\[ f_1 \circ \ldots \circ f_K = (T_1^1 \circ T_1^K) \times \ldots \times (T_m^1 \circ T_m^K) \]  

\[ Transf(l) = \bigcup_{j=1}^{N} Transf(v_j), \quad l \in L, l = \{v_1, \ldots, v_N\} \]

The analysis solves data flow equations at each node of the CFG. The dataflow fact sets are transformed along the way according to (5) and (6). Each node has two sets of facts: the set of facts entering the node $IN$ and the set of facts exiting the node $OUT$. The analysis terminates once the iterative application of the data flow equations yields no more changes. The final solution represents the constant values associated with the modeled objects. It captures the effect of the invoked chain of methods on the involved nested objects.

\[ IN[node] = \bigcup_{P \text{ is predecessor of node}} OUT[P] \]  

\[ OUT[node] = Transf_{node}(IN) \]
5.3. THE FRAMEWORK

Returning back to our example at Figure 5.3, we need to reason about the value of URI and accepted media fields. Each object type has its own lattice and transfer functions \( f_{\text{WebTarget}}(v) = T(S_{\text{URI}}) \) and \( f_{\text{Builder}}(v) = T(S_{\text{media}}) \) computed in parallel. For simplicity, we explain the value computation considering the field transformers (the basic blocks of transfer functions) acc. Equation (1). For each IR method (CFG node), we maintain the value of the interesting variables. Initially, the values of the fields \( S_{\text{URI}} \) & \( S_{\text{media}} \) are \( \bot \) before Line 01. The method at Line 01 transforms their values into \( T(\bot) = \varphi \). The method at Line 02 influences only URI field, \( T(S_{\text{URI}}) = (S - \text{KILL}) \cup \text{GEN} \) yields to \( T(S_{\text{URI}}) = \{ \text{http://base} \} \) as \( S = \text{IN} = \varphi, \text{KILL} = \varphi, \text{and GEN} = "\text{http://base}" \). Since we rely on the SSA IR, we leverage variable value analysis provided through the symbol table, which associates information (like the value, if exist) with each variable in an SSA IR. After Line 03, given that \( S = \text{IN} = "\text{http://base}" , \text{KILL} = \varphi, \text{and GEN} = "/rest" \), the value is transformed into \( \{ \text{http://base/rest} \} \). The value of the media field remains unaffected till Line 10 which transforms it into \( T(S_{\text{media}}) = \{ \text{text/plain} \} \). The analysis keeps solving the data flow equations till we reach Line 12. There is no need for further iterations as we reached concrete values. If we assume a more complex example that has multiple branches, then the analysis joins the values at the point where the branches meet.

To this end, our strategy resolves the targeted entry point method of the called service by matching the obtained string values of the URI, accepted media type, and
5.3. THE FRAMEWORK

HTTP verb with our entry point database. We follow Algorithm 2 (Lines 22-25) to bind the caller service to the called service and propagate the involved points-to sets. The binding happens as the points-to sets of call site receivers are computed. We create the called service’s synthetic model that includes invocation to the matched service’s entry point method and passes the identified query parameter and response object as the method’s parameters. Then, the call site representing the inter-service invocation is explicitly resolved to the model. This results in adding a synthetic edge in the call graph from the call site to the service model. This helps in propagating data flow facts from the caller to called service to precisely solve the data flow analysis. The purpose of the synthetic edge is to govern the computation of control dependence for building the SDG of the analyzed application. If the targeted service is not found in our database, we assume a conservative model that treats the effect of the inter-service invocation on the passed parameters and returning object as unknown.

B. Modeling external-libraries invocations
Modern applications usually rely on external or third-party libraries. We only consider libraries that are frequently used. We choose to create summaries (stubs) based on their official description to simplify the analysis of benign libraries. The summaries represent how data flow may be changed in the form of GEN/KILL sets. This eliminates the need to analyze the implementation code of the libraries that perform the invoked methods. Thus, our strategy is to model library invocation methods by dispatching them to their corresponding summaries at the time of
building the call graph as illustrated in Algorithm 2 (Lines 26-28). In turn, the analysis scalability is improved without any cost on the precision.

5.3.3 The Vulnerability Detector

The vulnerability detector component applies information flow control for inspecting the candidate application’s security. It aims to detect violations of data integrity and confidentiality. It reports detected insecure and sanitized paths inside inspection slices. In this section, we present the core idea and steps of our detection approach.

In this work, we choose to leverage IFC based on SDG and program slicing techniques for security inspection [141]. Different techniques can be employed in this component. The algorithmic foundation of IFC based on SDG and program slicing is explained in depth in Hammer and Snelting work [141]. SDaaS inherits the precision of such technique in terms of flow, context, and object sensitivity. The vulnerability detector component first prepares inspection points and slice nodes based on an application’s SDG. Then, it verifies the non-interference security policy (i.e., point x in the program does not affect information at point y during the execution of the program). Finally, it records insecure and sanitized paths in each slice.

To define the interesting points of inspection, we specified a list of possible sources and sinks of information based on domain knowledge. The context-sensitive backward slicing technique is applied to compute slice nodes in SDG that influence every node respective each inspection point (sink). Slice nodes that affect sink y can be identified formally by backward slicing $BS(y) = \{x | x \rightarrow y\}$, where a path $x \rightarrow y$
5.3. THE FRAMEWORK

in the SDG is said to exist, if an information flow exists between \( x \) and \( y \). Such a flow can be explicit due to data dependency or implicit due to control dependency.

The core idea is that IFC leverages a lattice \( \mathcal{L} = (L, \land, \lor) \) to express the security level of the application’s statements and expressions. The simplest lattice form can express only two levels: high and low [89]. Towards applying IFC on SDG, we allocate a security level denoted by \( S(x) \) for each node \( x \) in the slice nodes. We assign the security levels to the inspection points according to the type of the sources and sinks in the defined list.

For example, a user supplied input statements are considered untrusted sources of information. Hence, they are provided security level: low. In contrary, statements performing operations on the backend database are considered sensitive sink of information. Thus, they are provided security level: high. Other nodes in the slice are allocated security levels based on the security level of the information reaching each node.

The next step is verifying the non-interference policy. For example, in the case of integrity inspection, we need to verify that information should only flow from a source node that has a same or higher security level than that of a sink node. Accordingly, the policy is \( x \sim y \) if and only if \( \forall x \in BS(y): S(x) \geq S(y) \). Likewise, to verify confidentiality, information should only flow to a sink node that has a same or higher security level than that of a source node. This implies, \( x \sim y \) if and only if \( \forall x \in BS(y): S(x) \leq S(y) \).
In our work, to detect insecure information flow that can lead to injection attacks. Intuitively, we need to verify the integrity policy. The verification is performed by checking that no node in the slice has a lower security level than the sink. Recall that we treat the inter-service interactions as method invocations. Our approach, in turn, easily supports the propagation of security levels between an application’s services to accurately conduct security inspection of the candidate application. The last step in this component is to record any violations of the non-interference policies which indicate insecure flow paths representing vulnerabilities. Some of these violations are redundant. Recall that the verification of the non-interference is done by computing all information flow paths between the source and sink nodes from the SDG, if exist. Some of these paths represent the same logical flow. In this respect, we consider flow paths as redundant, if and only if, they share the same source and sink nodes. In this case, we report all these paths as one vulnerability in the final results.

The detector also considers the declassification and endorsement inspection. This is achieved by checking if the detected flow path contains a node that represents a method for performing encryption, anonymization, sanitization, etc. In these cases, we consider the security level of information that flows through these methods is changed in a way that eliminates the violation state. In another word, a path from $x$ to $y$ is not a violation, if there is only one declassification or endorsement node on that path [142].
5.3. THE FRAMEWORK

We refer to such information flow paths that involve declassification or endorsement nodes by sanitized paths. Discovered sanitized paths are recorded for the purpose of security diagnosis.

5.3.4 The Security Diagnosis Practitioner

The security diagnosis practitioner component is responsible to characterize an application’s behavior as benign, malicious, or vulnerable. It provides a comprehensive insight about the security status of an analyzed application in terms of potential risk, behavior diagnosis, and security risk rating.

Through this section, we introduce the workflow of our diagnosis approach. We also define the set of quantitative metrics and rules proposed for diagnosing an application’s security status.

Figure 5.5 demonstrates the workflow of our diagnosis approach. This component utilizes the detected sanitized and insecure paths for inspected slices to derive the relevant quantitative metrics. The core idea is to work on sinks belong to the same security issue (i.e., integrity, confidentiality).
First, we compute the quantitative metrics, defined at Definitions 5.2, 5.3, and 5.4, to gain a solid insight of the analyzed application’s behavior and its security risk rating against the detected security issues. Second, we follow a rule-based approach to diagnose the application’s behavior and determine the risk level against each security issue. The rules are defined based on the computed aggregated ratio values according to Rule 1, 2, and 3.

**Definition 5.2 (Diagnosis Tuple).** A diagnosis tuple \(<I, R, N>\) per each security issue, where \(I\) is the number of insecure flow paths to these sinks, \(R\) is the number of flow paths that involve declassification or endorsement operations to the sinks, and \(N\) is the total number of paths residing in the inspected slices to these particular sinks.

**Definition 5.3 (Aggregated Ratio Values).** Aggregated ratio values \(V_I\) and \(V_R \in [0\%, 100\%]\), where \(V_I = I/N\) and \(V_R = R/N\) to reflect the portion of flow paths that are insecure and sanitized, respectively per each security issue across the application.
5.3. THE FRAMEWORK

**Definition 5.4 (Security Risk Level Spectrum).** Security risk level spectrum $S$ is a vector that represents the normal distribution of the severity of detected insecure flow path per issue. The spectrum scale (e.g., critical, high, medium, low) is determined based on the security level lattices used in the vulnerability detection component. For each detected insecure flow, the risk level is the maximum of the security level of its source and sink. Risk level of each insecure flow per issue is distributed in the spectrum by increasing the value of its position in the vector by one. Then, the resultant vector is divided by $I$ to normalize the vector’s values.

**Rule 1.** Given the aggregated ratio values $V_I$ and $V_R$ per security issue, intuitively, an application is classified as benign if it is free from insecure information flow paths that can trigger security risks. An application is benign if there is a threshold $Th \in [0\%, 100\%], V_I \leq Th$, and $0\% \leq V_R \leq 100\%$. $Th$ can be specified for flexible classification. We applied rigid classification rule where the application is classified as pure benign and not prone to any security risk such that $Th$ is $0\%$.

**Rule 2.** Given the aggregated ratio values $V_I$ and $V_R$ per security issue, an application is considered to have a malicious merit if $V_I > Th$ and $V_I \geq V_R$ in case the security issue represents confidentiality violation. Otherwise, the application is considered to have a vulnerable merit if $V_I > Th$ and $V_I < V_R$ in case of confidentiality violation or if $V_I > Th$ in case of integrity violation. An application is classified as vulnerable in a general view if it has only vulnerable merits with respect to all the detected issues. Otherwise, an application is classified as malicious if it shows a malicious merit to confidentiality-related security issue. We defined $Th$ as $0\%$ to satisfy our rigid classification approach.

**Rule 3.** Given spectrum $S$ per each security issue, the risk level of an application against each security issue is determined by the range having the maximum distribution in the corresponding $S$. If for any security issue the normalized distribution is equal through some or all ranges, the highest range indicates the risk level of this security issue. The risk rating equals the percentage inside the indicated range. The overall risk level of the application is expressed in terms of risk levels for all the detected issues.
5.3. THE FRAMEWORK

Figure 5.6 illustrates an example of four paths, representing from left to right, XSS vulnerability, XSS sanitization, SQLI vulnerability, and data leakage violation. For each security issue, we compute three metrics referring to a tuple, an aggregated ration value, and risk level spectrum, as demonstrated at the right-hand side of Figure 5.6. XSS and SQLI paths are grouped to reflect integrity-related security issue, while data leakage path is considered for confidentiality violation. Then, we apply our rule-based diagnosis approach based on these quantitative metrics. In our example, Rule 1 is not satisfied as the application has insecure flow paths.

The idea simply revolves around diagnosing an application’s state with respect to confidentiality and integrity threats. First, Rule 2 attests an application as malicious if it sends sensitive data to unauthorized parties profusely compared to putting effort to declassify these data before transmission. However, if the declassification of sensitive data receives higher attention compared to insecure paths, the diagnosis eases the application's state to be vulnerable rather than malicious with respect to confidentiality threats. Next, Rule 2 checks if an application hits the bar by having insecure paths prone to integrity violations. At this point, Rule 2 indicates an overall insight of the application either as malicious or vulnerable. In case it is malicious against confidentiality, no matter about its state towards integrity threats, it is characterized as malicious. However, in case the application was vulnerable to any or both security issues, it is considered vulnerable. In our example, the overall diagnose is determined to be malicious.
Following Rule 3, as the normal distribution is equal through the critical and high ranges inside the integrity spectrum, we conclude that our example has 50% critical risk towards integrity threats. By inspecting the confidentiality spectrum, Rule 3 identifies that the example has 100% high risk towards confidentiality violations. Thus, the example is potentially prone to high confidentiality and critical integrity threats with 100% and 50% risk ratings, respectively.

Based on the diagnosis results, this component provides a comprehensive report about the security status of an analyzed application in terms of potential risk, behavior diagnosis, and security risk rating. It publishes detailed information about the detected vulnerabilities categorized by each security issue and broken down by security levels. It also provides an overall characterization about the application’s behavior, and a general security risk rating of the application.

The resultant report provides a solid insight into the application security. The report is beneficial to prioritize remediation efforts by focusing on the most significant vulnerabilities and critical issues. At this end, our solution provides an extra measure for promoting proactive protection and addressing the potential security issues before being exploited after production.

5.4 Experimental Validation

In this section, we present our implementation and experimentation details. The main objective of our experiments is assessing that our framework adheres to accurate detection as well as high performance and scalability. Our evaluation targets
5.4. EXPERIMENTAL VALIDATION

answering the following questions: 1) what is the effectiveness of our system for detecting vulnerabilities in cloud SaaS applications; and 2) what is the performance and scalability efficiency of our system. Sections 5.4.1 and 5.4.2 further discuss the set of experiments conducted to assess each question, respectively.

All components of our framework are implemented in JAVA. The main functionalities of the framework are exposed as RESTful web service. WALA [143] libraries are extended to perform points-to analysis and call graph construction. We use JOANA [144] as an implementation of IFC based on SDG and slicing techniques. JOANA runs on top of WALA. WALA and JOANA do not support the modeling of cloud applications which is considered the main challenge for this work. Different frameworks are offering points-to analysis. We select WALA as it is integrated with JOANA. We choose JOANA as it takes into consideration the flow, context, and object-sensitive features when constructing the SDG. These features are essential in analyzing applications written in a dynamic and object-oriented language like Java.

In our experiments, we employ our solution to perform diagnose task. The diagnose task carries out our solution from end-to-end until publishing the diagnosis report about the analyzed application. We deploy our system in public cloud on Amazon EC2. We generate the workload by Apache JMeter v3.1 to simulate several users. We also augment JMeter with several extensions to set up realistic intensive workloads and acquire results in a graphical way.
5.4. EXPERIMENTAL VALIDATION

5.4.1 Vulnerability Detection

We apply our system on two recent benchmark suites: OWASP-bench\textsuperscript{7} and WSVB-bench\textsuperscript{8}. These benchmark suites are for evaluating automated vulnerability detection tools and services. They contain web service-based applications that can resemble cloud-based SaaS applications. The benchmark applications represent real scenarios like simple and complex data and control flow.

The OWASP-bench comprises of 1,261 applications that cover a variety of vulnerabilities including command injection (CMDI), lightweight directory access protocol injection (LDAPI), path traversal, insecure cookie, trust boundary violation, XML path injection (XPathI), and XSS. We opt to use v2.1 as it is the latest stable version of the benchmark. This version contains 175,618 lines of code, 6,643 methods, 3,678 classes, 8 packages, 179 external libraries, and 22,472 files. It is implemented as servlets, one of the natural convenient ways to implement RESTful web services.

The WSVB-bench adapts web service implementation of the TPC-APP, TPC-C, and TPC-W benchmarks. It consists of 21 web services with 80 different versions and 14,826 lines of code that cover SQLI vulnerabilities.

We contrast our system with different vulnerability detection tools as outlined in Table 5.1. To demonstrate the effectiveness of our tool over the two benchmark suites, we use the results of open-source and commercial tools including seven static

\textsuperscript{7} OWASP-bench (2016): https://www.owasp.org/index.php/Benchmark
\textsuperscript{8} WSVB-bench (2015): https://github.com/nmsa/wsvd-bench
code analyzers, three penetration testers, and one anomaly detector. The sources of the results come from publicly published data [145, 146]. We choose the most recent versions of the tools included in these studies.

### Table 5.1: Tool Information.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Tool</th>
<th>Version(s)</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP-bench</td>
<td>FBwFindSecBug</td>
<td>Open Source - v1.4.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FindBugs</td>
<td>Open Source - v3.0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OWASP-ZAP</td>
<td>Open Source - vD-2016-09-05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PMD</td>
<td>Open Source - v5.2.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SonarQube</td>
<td>Open Source - v3.14</td>
<td></td>
</tr>
<tr>
<td>WSVB-bench</td>
<td>FindBugs</td>
<td>Open Source</td>
<td>Penetration Testing</td>
</tr>
<tr>
<td></td>
<td>Yasca</td>
<td>Open Source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IntelliJ IDEA</td>
<td>Open Source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HP WebInspect</td>
<td>Commercial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM Rational AppScan</td>
<td>Commercial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acunetix Web Vul. Scanner</td>
<td>Commercial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IPT-WS</td>
<td>Commercial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CIVS-WS</td>
<td>Open Source</td>
<td></td>
</tr>
</tbody>
</table>

We compare our system with the mentioned tools in terms of recall, precision, F-measure, and false alarm rate. These metrics are defined and measured as follows:

- **Recall**: The fraction of vulnerabilities that are correctly detected out of the total number of existing vulnerabilities. This metric is also known as True Positive Rate (TPR).

  \[
  \text{Recall} = \frac{TP}{TP + FN}
  \]

- **Precision**: The fraction of correctly detected vulnerabilities out of the number of the detected vulnerabilities.

  \[
  \text{Precision} = \frac{TP}{TP + FP}
  \]
5.4. EXPERIMENTAL VALIDATION

- **F-measure**: The weighted harmonic mean of precision and recall.

\[
F - measure = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}
\]

- **False alarm rate**: The fraction of incorrectly detected vulnerabilities out of the total number of existing false vulnerabilities. This metric is also known as False Positive Rate (FPR).

\[
FPR = \frac{FP}{(FP + TN)}
\]

In this work, we identify a set of possible sources and sinks that are prone to the various types of vulnerabilities covered in the benchmarks. The set is identified based on a domain knowledge of the specification of relevant APIs of Java RESTful web services, SOAP web services, and web Servlets. The knowledge is specified as method signatures, as briefly outlined in Table 5.2. We used a total of 14 sources, 45 sinks, and 5 declassification and endorsement methods in our experiments.

Table 5.2: Specifications of source, sink, declassification methods.

<table>
<thead>
<tr>
<th>Vul. Type</th>
<th>Method Signature</th>
<th>Method Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>javax.servlet.http.HttpServletRequest.getParameter(L java/lang/String;)L java/lang/String;</td>
<td>Source</td>
</tr>
<tr>
<td></td>
<td>javax.ws.rs.client.WebTarget.resolveTemplate(L java/lang/String;L java/lang/Object;)L javax/ws/rs/client/WebTarget;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>javax.ws.rs.core.UriBuilder.queryParam(L java/lang/String;L java/lang/Object;)L javax/ws/rs/core/UriBuilder;</td>
<td></td>
</tr>
<tr>
<td>LDAPI</td>
<td>javax.naming.directory.DirContext.search(L java/lang/String;L java/lang/String;[L java/lang/Object;L javax/naming/directory/SearchControls;]L javax/naming/NamingEnumeration;</td>
<td>Sink</td>
</tr>
<tr>
<td>SQLI</td>
<td>java.sql.Statement.executeQuery(L java/lang/String;)L java/sql/ResultSet;</td>
<td></td>
</tr>
<tr>
<td>CMDI</td>
<td>java.lang.Runtime.exec(L java/lang/String;L java/lang/String;)L java/lang/Process;</td>
<td></td>
</tr>
</tbody>
</table>
5.4. EXPERIMENTAL VALIDATION

We applied SDaaS over eight vulnerability types covered by the two benchmarks. Figure 5.7 demonstrates the vulnerability detection results over the two benchmarks broken down by some of the covered vulnerability types. Figure 5.7, part (a), (b), (c), (d), (e), and (f) depicts the evaluation over path traversal, CMDI, LDAP, XSS, insecure cookie, and SQLI applications, respectively.

Our system, SDaaS, demonstrates capability to detect different vulnerability types in contrast to FindBugs, OWASP ZAP, PMD, and SonarQube. A fine analysis to the results over the OWASP-bench indicates that only SDaaS and FBwFindSecBugs could cover the detection of all types of vulnerabilities.

FindBugs hits low false alarm rate and similar precision as SDaaS, while significantly losing recall and F-measure, when applied on path traversal and XSS applications. While, OWASP ZAP tool hits higher precision and lower false alarm rate than SDaaS at the cost of very poor recall and F-measure, specifically when tested over CMDI and XSS applications. Thus, FindBugs and OWASP ZAP tools are risky to miss vulnerabilities. In contrast, the results validate that SDaaS shows a significantly higher recall, a slightly improved precision, and as a consequence better F-measure. It also achieves relatively low false alarm rates. In comparison, SonarQube and FBwFindSecBug have a significantly higher false alarm rate and a relatively lower F-measure than SDaaS, when tested on LDAP in specific. The same observation holds as SDaaS achieves higher recall, precision, and F-measure with lower false positive rate than FBwFindSecBug for XSS and XPathI.
Figure 5.7. The SDaaS vulnerability detection evaluation over various vulnerability types: (a) Path Traversal; (b) CMDI; (c) LDAPI; (d) XSS; (e) Insecure Cookie; and (f) SQLI. Blanks denote that the corresponding tool could not detect any vulnerability.
Interestingly, SDaaS and FBwFindSecBug show similar effectiveness when applied on trust boundary violation applications. The four tools including SDaaS, FBwFindSecBug, OWASP ZAP, and SonarQube show ideal results achieving 100% recall, precision, and F-measure with no false alarm rate when experimented on insecure cookie applications.
On the other hand, the evaluation on WSVB-bench reveals that CIVS-WS aims for high precision while sacrificing recall, thus it is more prone to omit SQLI violations. In comparison, SDaaS shows a significantly higher recall and even a significantly improved F-measure.

### 5.4.2 Performance and Scalability

Several experiments are conducted to evaluate the performance of our framework. These experiments investigate various aspects including, overall response time of the framework for different analysis requests, the impact of various framework components to the response time, system scalability, and resources usage.
5.4. EXPERIMENTAL VALIDATION

A. Response Time

The environment setting for experimenting the response time is depicted in Figure 5.8, part (a). On Amazon EC2, we create an instance of the type “c3.2xlarge” to serve as the application server to host SDaaS. The instance is launched from a pre-configured image (AMI) of “Windows Server 2016, 64 bits” with 8 virtual CPUs and 15 GB memory. We configure the instance with Apache Tomcat v7.0.57 to deploy our system.

The JMeter is hosted on Amazon EC2 of type “t2.micro” with a pre-configured image (AMI) having “Windows Server 2016, 64 bits”, 1 virtual CPU and 1 GB memory. All VMs are located within the same region zone in Oregon.

The first experiment measures end-to-end response time of an analysis request sent from JMeter node (simulated users) to the application server. The overall response time includes the processing and communication time. The workload is generated according to a step-wise manner. The test plan is repeated 5 times and conducted at various times over 250 different applications and 10 concurrent users per run. We consider randomizing the applications and the time between requests in order to stimulate real scenarios. The sent requests perform the analysis over each application to indicate if it is benign, malicious, or vulnerable and retrieve a detailed report of the analysis results.

The chart appears at the top of Figure 5.9 portrays the average response time in seconds vertically against the five runs, and the underneath table draws more details.
about the average, min, max, and 95% percentile achieved per each run. The overall results are averages across all runs. From observing the overall results, the average response time is 34.6, the minimum period is 10.3 and the maximum is 59.5 seconds, while 95 percent of the simulated users experienced a response time of 48.7 seconds or less during this experiment.

The second experiment evaluates the impact of different tasks in our system to the response time. As outlined in Table 5.3, the first and second columns show the applications grouped by vulnerability types and their count, respectively. The sixth column shows the average response time in milliseconds including processing and communication time corresponding to application categories, while columns 3-5 breaks the time down into segments representing the execution time of main tasks like lifecycle modeling, SDG construction, and static IFC analysis.

We infer from the results in Table 5.3 that the SDG construction task dominates the response time by approximately 98%. Thus, the complexity and size of an analyzed application is the main factor affecting the performance of our solution. We employed an intensive program analysis to achieve the highest possible level of detection’s accuracy in terms of precision and recall. Lighter program analysis can be used when building the SDG in order to reduce the impact on the response time. Though, this can come at the cost of less accuracy.
B. System Scalability

The target of this experiment is to exercise our system performance and scalability to serve increasing number of requests. The experiment is conducted over the same set of benchmark applications. Figure 5.8, part (b) depicts the framework environment set up in Amazon for the scalability experiment. JMeter is deployed on three VMs, one dedicated to the master node and the other two are for the slave nodes. Hence, the workload can be conducted remotely in a distributed manner to avoid bottleneck at users’ side. The master node specifies the requests and slave nodes are responsible to send these requests, receive the results, and send them back to the master node.

We utilize an elastic web balancer to serve as the internet-facing point of our system. We conduct our experiments with two settings: a) fixed number of instances and b) auto-scale. In the former setting, two VM instances in two different availability zones are created to serve as the application servers.
5.4. EXPERIMENTAL VALIDATION

In the later setting, we set up the auto-scale group to increase the instances if SDaaS’s response time exceeds a threshold of 30 seconds and decreases the instances if the response time diminishes to 15 seconds or less. We configure our cloud environment to launch instances from an AMI of the type “c3.2xlarge” when scaling up. The AMI is created from the configured application server. We set 60 seconds to warm up new launched instances.

Figure 5.10. SDaaS scalability evaluation for different request loads: (a) with fixed number of instances; and (b) with auto-scale.
The load balancer automatically distributes incoming application traffic across the system’s healthy instances within the auto-scaling group. For both settings, we design the health check grace period to be 300 seconds, the number of consecutive successful health checks to be 2 to consider an instance as healthy, the number of consecutive health check failures to be 3 before considering an instance unhealthy, and the time between health checks to be 180 seconds.

As illustrated in Figure 5.10, part (a), the response time has a non-linear increase as the number of requests grows. We noticed that our system could not accommodate 32.6% of the requests because of the limited capacity of the fixed instances.

However, leveraging the auto-scale feature of the load balancer allows our system to scale-up to satisfy the requisite performance conditions and meet other quality of service (QoS) features like availability and fault-tolerance. Figure 5.10, part (b) depicts how our system scales up to satisfy the increasing number of requests while holding up the response time to the specified threshold, thanks to the automatically provisioned instances.

C. Resource Usage
This section presents our experiments to evaluate the CPU utilization and memory consumption of our system. We conduct the evaluation of CPU utilization following the same settings used for the scalability experiment. We leverage CloudWatch to monitor the CPU utilization of our system.
Figure 5.11. SDaaS CPU utilization evaluation for different request loads: (a) with fixed number of instances; and (b) with auto-scale.

Figure 5.11 shows the average processor utilization of all active instances running behind the load balancer during the execution of this experiment. As depicted in Figure 5.11, part (a), when our system is deployed over the fixed number of instances, the processor utilization is proportional to the number of concurrent requests. We previously inferred that SDG construction dominates the execution of our system. The increase in the processor usage stems from the overhead incurred to
serve increasing number of requests as well as perform intensive computing task to construct the SDG.

By noticing Figure 5.11, part (b), the CPU utilization did not exceed 25% when leveraging the elasticity of the Cloud to provision more resources for executing our system on demand. Having enough compute optimized instances to cover the incurred overhead reduces the impact on the CPU usage.

We evaluate the memory consumption of our system during carrying out the second experiment of the response time. We use platform-independent runtime monitor API\(^9\) embedded in JAVA to capture the memory consumption.

Figure 5.12 shows the average physical memory consumed by our system in Megabytes during the experiment’s execution. The results demonstrate that the increase in the memory usage is non-linear with the increase of the size of analyzed applications, more specifically when more IFC analysis is conducted.

By analyzing the monitoring data, we observe that the peak points for memory usage incurred due to the SDG construction required mainly for the IFC analysis. The overall average consumption is 2861 MB, the maximum reached is 6664 MB, and the 95th percentile is 5204 MB. As observed from the results, our system achieves an efficient CPU utilization and memory consumption footprint.

---

\(^9\) Java Runtime API: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html
5.4. EXPERIMENTAL VALIDATION

D. Result Analysis Summary

As observed from the experimental results, our solution achieves high accuracy, performance, and scalability. We summarize our major findings as follows:

- SDaaS outperforms other tools covering static analyzers, penetration testers, and anomaly detectors. It attains wider coverage of vulnerability types and higher detection accuracy with enhanced balance between F-measure and false alarm rates. Our solution, in contrast to these tools, has the highest recall and F-measure rates over the analyzed benchmark applications. By excluding tools having significantly poor recall, SDaaS reports the lowest false alarm rates over most of the applications. In contrast, other tools are endangered to miss vulnerabilities and produce false alarms. SDaaS achieves F-measure of 81% and false alarm rate of 53% on average over 1,341 applications in the two benchmarks covering eight vulnerability types.

Figure 5.12. SDaaS memory consumption evaluation for different benchmark applications.
5.4. EXPERIMENTAL VALIDATION

- We observe that SDaaS is prone to false alarm cases that are caused either by complicated indirect control dependency or data dependency through complex arrays. The analysis treats composite data structures as a single location in memory to ignore the index which can be only known at runtime. These reasons are common, unavoidable, and quite acceptable due to the over conservative nature of static analysis techniques. Albeit, SDaaS still achieves better false alarm rates in comparison with other tools evaluated over these cases.

- Our solution achieves an average response time of 34.6 seconds when evaluated for 5 repeated runs at various times, over 250 different applications and 10 concurrent users for each run. The response time here represents end-to-end execution of our solution performing diagnose task including request processing and communication time. The experiments deduce that the complexity and size of an analyzed application is the main factor affecting the performance of our solution; where the SDG construction task dominates the response time by approximately 98%.

- The scalability experiments validate that leveraging the Cloud’s auto-scale feature in a way that gradually supports scalability and elasticity of our solution is a must. Our solution is deployed as a service that can dynamically meet efficient performance and quality of service features in order to promote its adaptability. As demonstrated by the experiments, our solution scales up to satisfy the increasing number of requests while holding up the response time to 30 seconds.
as well as scales down to preserve resources when the response time recedes to 15 seconds.

- As experimentally validated, the processor utilization and memory consumption are proportional to the increasing number of concurrent requests, in case we disable the scalability of our system. The proportional increase stems from the overhead incurred to perform the intensive computing task required to construct the SDG as part of our analysis. Our solution benefits from its scalable design to achieve an efficient CPU utilization and memory consumption footprint. Activating the scalability limits the CPU utilization to 25% and the memory consumption to 2861 MB, on average.

5.5 Summary

With the maturity of service-oriented architecture (SOA), microservices architecture, and Web technologies, web services have become critical components of cloud operational applications. Although these technologies promise impressive benefits, they put SaaS applications at risk against novel as well as prevalent attack vectors. This security risk is further magnified by the loss of control and lack of security enforcement over sensitive data manipulated by SaaS applications.

In this chapter, we introduce a novel Security Diagnosis as a Service (SDaaS) framework for cloud applications. The benefit of our solution is twofold. First, it offers protection of SaaS applications by vetting information flow vulnerabilities and mitigating the risk early during an application’s development phase. Second, it attests
the end-to-end security of consumers’ data manipulated by SaaS application. These benefits can promote the adoption of cloud SaaS applications in security-critical domains.

Based on our experiments, the proposed framework has the following implications: 1) ease of deployment that facilitates its adoption in cloud environments; 2) comprehensive analysis that holistically capture information flow paths, including explicit and implicit flows; 3) wide coverage of vulnerability types; 4) accurate detection by achieving high F-measure and low false alarm rates in contrast to other tools; 5) high performance with reasonable response time and no runtime burden over analyzed applications; 6) scalability that supports its adaptability on real applications without sacrificing the accuracy; and 7) solid insight about an analyzed application’s security status.
CHAPTER 6

Security Monitoring as a Service for Cloud Analytic Applications

6.1 Introduction

The rapid adoption and investment in cloud analytic applications is magnified by the 4Vs: velocity, volume, variety, and veracity of big data. Such increasing momentum has given rise to the hype around analytic technologies to perform data-parallel computing on commodity hardware, empowered by leveraging on-demand cloud services. As the glory of cloud analytic applications grows in popularity, security concerns grow in importance even more.

Analytic applications are prone to data breaches due to insecure computations, misconfiguration, and unauthorized access as a result of vulnerable, malicious, or misconfigured nodes or tasks [7]. The risk is further amplified due to the loss of control for running analytic applications in the Cloud. Furthermore, the unique features of analytic clusters, which provide distributed large-scale heterogeneous computing environments, render traditional security technologies and regulations
ineffective. Even though, there is a lack of effective security measures provided by cloud analytic providers to detect such malicious and anomalous activities.

Recall from Chapter 2, several research efforts, proposed to fortify analytic world against security threats, have span different research directions, ranging from differential privacy [123], integrity verification [124-127], policy enforcement [128-131], data provenance [132-135], honeypot-based [136], to encryption-based [137] mechanisms, among others. The effectiveness of differential privacy as a widespread solution is still not proven. The integrity verification approaches require intercepting the application execution for verifying result integrity, which comes at the cost of performance penalty. Access control policies cannot prevent misuse activities breaching data security, after an access has been granted. Provenance mechanism incurs overhead due to collecting, storing, and analyzing provenance data, which can lead to impracticability. Honeypot-based and encryption-based approaches entail modifications to the analytic applications to add the security attestations.

In this chapter, we propose Security Monitoring as a Service (SMaaS) framework. SMaaS is a novel information flow-based log analysis solution for detecting security anomalies in cloud analytic applications. It exemplifies the runtime monitoring service provided in the deployment phase, as highlighted in Chapter 4.

Hadoop, the most-shiny analytic technology, was not originally designed with security, compliance, and risk management support in mind. Recently, it is evolved to support authentication and encryption mechanisms for protecting data at rest and in
transit. Despite the evolving efforts in securing Hadoop, it is still exposed to weak authentication and infrastructure attacks. Such attacks increase the security risk of analytic applications against data confidentiality and integrity.

The distinct features of computations and data in the distributed large-scale analytic systems arise several challenges to develop an effective log analysis for anomaly detection solution. These challenges are summarized as follows: a) handling log data that is characterized by the 4Vs and collected across the cluster nodes; b) involving the complex data and control flows enclosed among the cluster nodes to execute analytic applications; c) considering the different roles of core daemons responsible for running such analytic applications; and d) mining for tangible evidence of security anomalies from log data.

In this work, we propose a novel approach to boost our solution in solving the aforesaid challenges. We leverage streaming data pipeline for security inspection. The data pipeline mixes advanced software technologies (Apache NiFi, Hive, Zeppelin) to automate the collection, management, processing, analysis, and visualization of log data from multiple sources, making it valuable, comprehensive, and cohesive for security inspection. Cluster log data is but one part of the whole picture. Thus, SMaaS relies also on system logs to complete the picture for security inspection. SMaaS works towards extracting information flow profile from log data to model the execution of a candidate application. Upon the information flow profile,
SMaaS employs several techniques for the detection of security anomalies. These anomalies indicate data integrity and confidentiality violations.

Our overall contributions are as follows:

- We propose a novel framework called Security Monitoring as a Service (SMaaS) for analytic applications.
- We introduce an advanced approach that leverages streaming data pipeline to automate log data ingestion, processing, analysis, and visualization for real-time security inspection.
- We propose several techniques for detecting different types of security anomalies based on information flow analysis.
- We demonstrate the detection effectiveness and performance efficiency of our framework through a set of experiments over benchmark applications.

The remainder of this chapter is organized as follows: the operational overview of our proposed framework, the threat model it assumes, and Hadoop in a nutshell are introduced in Section 6.2. Section 6.3 presents the details of the framework. The framework implementation and experimental evaluation are presented in Section 6.4. Section 6.5 draws the concluding remarks of the chapter.

### 6.2 Overview

This section outlines the operational overview of our proposed framework, the threat model it assumes, and a brief outline about Hadoop.
6.2.1 The SMaaS Operational overview

We consider three main entities that comprise the Cloud service models for this framework: Cloud analytics provider, trusted party, and consumers running data analytic applications over the provided cluster or service.

There are different architectural deployment offerings for analytic technologies (e.g., Hadoop) in the Cloud. These offerings range from basic services (e.g., IaaS, PaaS) to specific-tailored services (e.g., Data Analytic as a Service). Such offerings facilitate running analytic application in the Cloud. SMaaS design supports these different offerings.

We offer SMaaS as an advanced security monitoring feature from the Cloud analytics provider. As depicted in Figure 6.1, a provider for analytic technology (e.g., Hadoop) offers its consumer the option to subscribe for security service (1). For
subscribed consumers, the provider enables collecting log data from the clusters respective these consumers. The provider delegates the trusted party for further analyzing the collected data (2). The trusted party, in turn, employs the proposed framework to detect anomalous activities indicating data breaches. Monitoring reports are published in the dashboard to the consumers and email alerts are sent with detailed analysis reports upon detecting security violations (3).

6.2.2 Threat Model
Analytic applications (e.g., MapReduce jobs) can be misconfigured, malicious, or vulnerable and may breach the security of processed data. This can be done throughout their execution via multiple activities (e.g., modifying, copying, or deleting data) at different levels (i.e., input, intermediate results, output) in a way that violates data integrity and confidentiality. Our solution aims to detect five anomaly types: 1) data leakage; 2) data tampering; 3) access violation; 4) misconfigurations; and 5) insecure computation. We assume the correctness and integrity of log files upon which we build our security analysis solution. We also assume the security of the cluster, the underlying platform, and infrastructure where SMaaS is deployed on.

6.2.3 Hadoop in a Nutshell
In this work, we are mostly interested in Hadoop's latest versions (2.x and 3.x series). Hadoop stack consists of core modules. These modules contain: 1) YARN, responsible for job scheduling and cluster resource management; 2) MapReduce,
based on YARN for parallel processing of large data sets; and 3) HDFS, a distributed
file system for high-throughput access to application data. Each module consists of
several daemons. To be specific, YARN comprises of resource manager, node
manager, and job history server daemons. HDFS has many daemons such as name
node, and data node, among others.

An analytic application performing MapReduce job breaks the input data into
multiple splits, equivalent in size to HDFS block. Then, the application breaks the
processing into two main phases: map and reduce. The map phase is responsible to
map the input data into key/value pairs forming intermediate results. Multiple map
tasks are initiated on cluster's nodes to simultaneously process each input split. Then,
the reduce phase takes the intermediate key/value pairs and produces the final output.
Multiple reduce tasks are commenced to process all relevant intermediate pairs
together in parallel to perform the required processing task.

YARN executes MapReduce application as follows: 1) the resource manager
assigns a unique ID to the application and copies the resources required to run it; 2)
then, the resource manager starts an application master to coordinate the execution of
the application's tasks; and 3) the node managers control containers on each
individual node to concurrently run the tasks.

The infrastructure knowledge about Hadoop adds another obstacle to
implementing security monitoring initiatives. Hadoop can produce various types of
log data from different sources (e.g., applications, daemons, audit actions). Such log
data is considered a rich source of information for troubleshooting and performance debugging issues. However, it has a complex confounding structure that precludes mining a useful knowledge for security inspections. It is a very critical issue to derive a comprehensive profile of the behavior of an analytic application from the emitted log data, with the goal of fostering the detection of security anomalies.

There is no direct mean to relate information about an executed application (in YARN) and processed data (in HDFS) from log data. In a typical Hadoop cluster, each individual node (i.e., daemon) generates its own log data. Logging and auditing are further configured through complicated settings to expose data which can have various granularity levels (e.g., DEBUG, WARN, INFO), reside in different storage locations, and have different retention and deletion policies. In this sense, logging and auditing in Hadoop can convey rapidly-growing quantities of data with low quality in terms of redundancy, heterogeneity, and diversity. In addition, it falls short to provide cohesive insights about user activities running analytic operations. Thus, Hadoop log data can be burdensome to mine for meaningful information or tangible evidence of security anomalies.

6.3 The Framework

As illustrated in Figure 6.2, we build the architecture of the SMaaS framework on a distributed model. Such architecture supports the scalability required for monitoring distributed analytic applications, which may execute on a cluster spanning thousands of nodes.
The proposed framework comprises of four main components: Data Operator, Data Aggregator, Security Analyzer, and Visualization Manager. These components, according to their essential tasks, logically form two core engines: observation and inspection. To support our distributed architecture, the observation operates transparently on the monitored cluster without introducing any changes or overhead, while inspecting the security of the monitored cluster takes place separately by performing the log data analysis on a different cluster. The observation engine involves the data operator component acting as a transparent agent to collect log data from each individual node in the monitored cluster. The inspection engine entails the data aggregator, security analyzer, and visualization manager components.
The data aggregator component consumes the collected data from the data operator. It consolidates the data to model the execution events in the context of the whole application. This component leverages novel techniques to profile the execution behavior in terms of information flow including data and control dependencies.

Upon the consolidated profile, the security analyzer component detects security anomalies as well as conduct alerting and reporting actions. The profile is inspected against expected features that characterize benign applications. Such inspection approach boosts this component to detect deviations indicating anomalous and suspicious events.

Monitoring reports are displayed in graphical web-based frontend dashboards, managed by the visualization manager component. The components are further detailed in the light of the aforesaid engines in the following subsections.

### 6.3.1 Observation Engine

**Data Operator**

The data operator component represents the observation engine in SMaaS. It is responsible for observing the monitored cluster through auditing and logging. We devise an approach for log data collection that promotes transparency and overhead-efficiency. Instead of setting up a custom log collection process, the data operator
component leverages log4j API and Syslog protocol as its workhorse to collect log data from the cluster nodes.

We specifically leverage log4j API to enable the extensive native logging capabilities in Hadoop. Log4j API is the heart of the data operator component to employ logging process for Hadoop applications during the course of their execution. We focus on collecting YARN application logs and HDFS data node daemon logs in order to capture both control and data flow activities relevant to an application execution. We automate storing the application logs in an aggregated fashion in HDFS. In this respect, logs from all containers, allocated to run the monitored application in the cluster distributed nodes, are unified in one location ready for ingestion by the inspection engine. The benefit of our approach is twofold:

- It facilitates serving and managing the collected logs directly by YARN daemons (i.e., ResourceManager or JobHistoryServer) in HDFS.
- It enables, in turn, retrieving the collected data in an easily-managed way.

Users are allowed to run or execute analytic applications and operations through command line from any node within the cluster. The command line facilitates an option which provides an environment to run an application under another user (specified in the command). Hadoop log data falls short in recording this information. It records information about the specified user whose name appears in the command but not the actual user who submits the command to be run under the specified user's

10 Log4j API: https://logging.apache.org/log4j/1.2/apidocs/index.html
environment. In this case, a user executing a malicious or vulnerable application can go undetected and incorrectly another user can be accounted for breaching the security.

To cover this gap and achieve deeper monitoring, we augment our approach to log user activities from the host operating system (OS) in Hadoop nodes. The data operator component leverages Syslog protocol to configure the host OS to collect system logs of user command activities. Our system supports a transparent distributed hierarchical architecture to collect system logs from the cluster nodes. The cluster is forked into groups of nodes forming sub-clusters. A localized relay is configured to poll the system logs from each individual group of nodes (sub-cluster) and instantly forward the received logs to be segregated into integral remote collector server, ready for ingestion by the inspection engine.

In this sense, our approach exposes both Hadoop and system log data transparently without requiring any intrusive changes, installing custom agents, or introducing any overhead over the monitored cluster. It also supports scalability by efficiently managing log data collection.

6.3.2 Inspection Engine

The data aggregator, security analyzer, and visualization manager components embody the inspection engine in SMaaS. These components work together towards automating the processing, analysis and visualization of log data, making it valuable, comprehensive, and cohesive for security inspection. They employ data pipeline as an
advanced log data processing and analysis approach to reason about the security of analytic applications.

This section starts with presenting the main idea of the data pipeline. Then, it dives into the detailed steps conducted by each component in the following subsections.

Our approach leverages Apache NiFi\textsuperscript{12}, Hive\textsuperscript{13}, and Zeppelin\textsuperscript{14} to build the data pipeline as outlined in Figure 6.3. We build the data aggregator and security analyzer components on top of Apache NiFi (Niagara Files) platform. These components are designed as groups of processors, running in an integral workflow, inside NiFi. This design boosts our system with the ability to stream log data from different sources (Hadoop and system), ingest high volumes of data in real-time, consume and transform many log data formats, and process log data in a distributed scalable manner. These features, in turn, empower our system with real-time security detection and decision-making capabilities. The aggregated data is streamed into Apache Hive to enable historical analysis. The visualization manager component provides on Zeppelin dashboard relational views extracted from aggregated data residing in Hive. These visualized views are valuable to determine effective responses and decisions to the detected security issues.

\textsuperscript{12} Apache NiFi: https://nifi.apache.org/
\textsuperscript{13} Apache Hive: https://hive.apache.org/
\textsuperscript{14} Apache Zeppelin: https://zeppelin.apache.org/
6.3. THE FRAMEWORK

Figure 6.3: The SMaaS data pipeline.

**Data Aggregator**

The data aggregator component is responsible to ingest the log data, collected by the data operator, at runtime. It then consolidates the data to profile the execution of MapReduce applications in terms of information flow view.

This component utilizes a combination of NiFi standard processors. It starts with remotely retrieving information about the running and finished applications in the monitored cluster. We optimize the component to fetch log data only if the retrieved information is changed indicating new submitted applications.

For each candidate application, its log data is fetched from YARN daemons. Recall that the fetched data represents logs from all containers allocated to run the candidate application over the cluster's distributed nodes. The data aggregator component consumes and parses the data in order to obtain every piece of information relevant to the application's execution. Then, it transforms and
6.3. THE FRAMEWORK

consolidates these pieces of information into a single profile in a condensed format as JSON.

This profile represents an information flow view that models the application's execution. The JSON schema of the profile is illustrated in Figure 6.4. The profile models the application's execution from two angles: global and partial.

The former captures global attributes about the application such as id, name, start time, finish time, average map time, etc. Furthermore, attributes detailing the data flow processed by the application broken down into input, output, and intermediate data are also tracked. Such attributes are driven directly from HDFS and nodes where the data is stored. The latter portrays the application's processing activities (Map and Reduce tasks) and dependencies (data and control) between them. For each Map task, the profile captures the data split that flows as input to the task. Similarly, for each Reduce task, the profile shows the list of dependent Map tasks that flow input data to the reduce task and the data split that flows as output from the reduce task.

The last step is streaming the constructed profile into Apache Hive. This step facilitates incrementally storing the profiles of applications, which ran on the monitored cluster, into a unified database. This database enables performing historical analysis by security administrators.
Figure 6.4: The JSON-Schema of Information Flow Profile.
6.3. THE FRAMEWORK

Security Analyzer

The security analyzer component utilizes the information flow profile received from the data aggregator component to detect anomalies.

This component is designed as a combination of NiFi standard processors. We implement the security analysis logic as scripts that automatically execute inside the processors. The analyzer reasons about five anomaly types: 1) data leakage; 2) data tampering; 3) access violation; 4) misconfigurations; and 5) insecure computation. The analyzer takes from the information flow profile as a baseline for the security analysis. It checks the profile against expected features that govern benign applications. Any deviation from the expected features indicates a security anomaly. The analyzer also involves correlating information from daemon logs and Syslogs upon needed to identify a compromise. These logs are collected by the data operator and ingested by the data aggregator to be ready for this component. The analyzer sends email alerts to the security administrator with detailed analysis reports upon detecting security violations. The analysis techniques to detect the anomalies are further explained in Section 6.4.1.

Visualization Manager

The visualization manager component provides a dashboard as an important feature of our monitoring solution. We leverage Apache Zeppelin for the visualization dashboard. The dashboard augments security administrators with the ability to visualize the aggregated information flow profile as well as relational views.
about the aggregated data from Hive database. This is important to determine effective responses to security issues. For example, an administrator can query about other applications executed by a malicious and a victim user or node involved in a detected security violation. As a remedy, the administrator may block or isolate activities from this user or node until proper mitigation actions are conducted.

6.4 Experimental Evaluation

In this section, we present our experimental setup and evaluation results. Our evaluation targets answering two questions: 1) what is the effectiveness of SMaaS for detecting security anomalies in MapReduce applications; 2) what is the performance efficiency of our system. The following sections describe the experiments conducted to assess each question, respectively.

We set up our experiments over private cloud consisting of six VMs. Each VM has "Ubuntu 14.04", 16 GB RAM, 4 CPUs and 100 GB storage. We choose Hortonworks\textsuperscript{15} data platform distribution to build the monitored Hadoop cluster. We build our system over a NiFi cluster using Hortonworks data flow distribution. Each cluster has one master and two slave nodes. We managed both clusters via Ambari\textsuperscript{16} server.

The experiments are conducted over three popular MapReduce benchmark applications [147]: TeraSort, TeraGen, and WordCount. They are shipped with

\textsuperscript{15} Hortonworks products: https://hortonworks.com/
\textsuperscript{16} Apache Ambari: https://ambari.apache.org/
Hadoop distribution. TeraSort aims to sort data stored in files. We used TeraGen to generate data as input to Terasort with size of 1GB, 5GB, and 10GB. WordCount aims to count word occurrences in files.

6.4.1 Anomaly Detection
We assess the detection effectiveness of our solution over the aforesaid five types of anomalies. We crafted the code of the WordCount application to implement data leakage, data tampering, access violation, and misconfigurations types. For the insecure computation type, we change the permission of the default locations with open access. At the end, we have six versions of WordCount applications, where five of them are malicious, vulnerable, and misconfigured. We applied our solution that detected all anomalies with 100% accuracy. In what follows, the detection techniques, employed by the security analyzer, are explained as regards each anomaly type.

A. Data Leakage
A malicious application may copy input or output data to unauthorized location. To detect data leakage activity, the analyzer reasons about the "BlkNo" attribute of all splits of the input and output dataflow in the application's profile. Then, it analyzes the datanode daemon logs looking for unauthorized write operations of any block of data which is not related to the expected MapReduce control flow. The information is correlated based on the "StartTime" and "BlkNo" attributes in the
profile. An occurrence in the datanode logs indicates the data leakage activity. Only MapReduce related-activities are expected from a benign application.

By knowing the compromised data split, the analyzer can further track the affected map and reduce tasks by traversing the profile. In case of input leakage, the "DataFlowSplit" attribute can lead to the affected map task. From there, we can infer the reduce task having dependency with it from linking its "TaskId" with the reduce task's "DependentTaskIds". In case of output leakage, the "DataFlowSplit" suffices to identify the affected map task.

B. Data Tampering

A malicious application may replace input or output data with wrong data. To detect data tampering, the analyzer reasons about the "DirPath" attributes of the input and output dataflow in the application's profile. Then, it analyzes the Syslogs to reason about the input and output paths that are configured when a user submitted the application. The information is precisely correlated based on the "SubmitTime" and "mapClassName" attributes from the profile. Mismatch between configured and processed paths indicate the data tampering activity. This is because a benign application is expected to process the input and output data as configured by the user.

C. Access Violation

A vulnerable application may be submitted to process input at or produce output to unsafe locations. In addition, a malicious application may change the access
permissions of the input or output making them prone to security compromise. Access violation is detected by reasoning about the "Permission" attributes of input and output dataflow in the profile. Permission, specifying that the input or output paths have open access, indicates a violation. This is because a benign application is expected to process and produce data that has restricted access to authorized users only.

D. Misconfigurations

Throughout the execution of an application, YARN stores the application's files inside local cache. The cache specifically contains the work directory of each individual container assigned to execute the application's tasks. It also stores the intermediate results throughout the application's execution. On the other side, input and output data processed by an application typically is stored in HDFS. The datanode daemons store data blocks in local directories. The location of the local cache and directories are normally configured at the time of the cluster set-up. A misconfigured application may change the configurations to different locations than the typical expected ones. Misconfiguration violation is detected by checking the "DirPath" attribute of cache and intermediate dataflow in the profile and linking them with the values of the corresponding properties in Hadoop's configuration file.
6.4. EXPERIMENTAL EVALUATION

E. Insecure Computation

A vulnerable application may execute while the intermediate data, application files, or data blocks are stored in the locations that have open access. In this sense, the application is prone to insecure computations due to the risk of tampering with its processing in terms of data and control flows. The analyzer detects insecure computations by examining the "Permission" attribute of the intermediate and cache dataflow in the profile. Such data is expected to have restrictive access.

6.4.2 Performance and Scalability

We conducted several experiments to evaluate the performance of SMaaS as streaming analytic solution. The experiments exercise different aspects including I/O performance, execution time, and resource usage. They are designed to appraise the SMaaS performance from two levels: component and system. Component-level experiments focus mainly on the data generator and security analyzer, the two main components that involve processing streaming data in SMaaS. System-level experiments reflect the overall performance of SMaaS, hosted on NiFi cluster.

A. Component-level performance

This section presents the experiments employed to measure a) I/O performance and b) execution time.
6.4. EXPERIMENTAL EVALUATION

We employ customized reporting tasks inside NiFi to send the collected metrics about each component to Grafana\(^\text{17}\). The metrics are measured during five minutes rolling window. The I/O performance is measured in terms of BytesRead and BytesWrite metrics. To estimate the execution time, we use the TotalTaskDurationSeconds metric. These metrics can be defined and measured as follows:

- **BytesRead**: The total number of bytes that the component read from the disk during the rolling window.
- **BytesWrite**: The total number of bytes written by the component to the disk during the rolling window.
- **TotalTaskDurationSeconds**: The total time that the component used to complete its task during the rolling window.

We perform two different experiments to evaluate each component. Firstly, we assess the aggregator component to analyze different volume of application logs. We execute TeraSort and TeraGen benchmark applications over various data volumes (1 GB, 5 GB, 10 GB). These various volumes consequently result in increasing the volume of each application's logs that need to be analyzed. On the flip side, we exercise the analyzer component in processing various workloads (1.2 GB, 6.9 GB, 15 GB) of Syslog and datanode log data. We created the workloads by combining the logs throughout a period of month.

\(^{17}\) Grafana: https://grafana.com/
6.4. EXPERIMENTAL EVALUATION

![Bar chart showing BytesRead and BytesWrite for varying data quantities for both the aggregator and the analyzer.](image)

Figure 6.5: I/O performance evaluation over various data size of the SMaaS components: (a) data aggregator; and (b) security analyzer.

a) I/O Performance

The BytesRead and BytesWrite for varying data quantities for both the aggregator and the analyzer are shown in Figure 6.5 part (a) and part (b), respectively.
As inferred from the figures, the performance involves proportional growth with the increasing size of data. The data aggregator involves more I/O operations in contrast to the security analyzer. The latter performs the analysis on the fly, whereas; the aggregator requires interacting with the disk while preparing the information flow.
profile. Thus, the performance of the analyzer is not affected by the size of the processed data. The aggregator's performance whereas is dependent on the data volume. Yet, it hits an efficient range between 127 and 224 MB.

\( b) \) Execution Time

Figure 6.6 part (a) and part (b) illustrate the execution time of the aggregator and the analyzer over different data loads, respectively. The execution time has a non-linear increase as the volume of data grows.

We notice that the execution time of the analyzer is higher than the aggregator. The main reason is that the analyzer executes diverse algorithms for the security inspection that entail processing time. Both components are highly efficient as their execution does not exceed 1 sec. Thanks to the proposed data pipeline that boosts our system to efficiently support data-parallel computing.

\( B. \) System-level performance

This section highlights our experiments to gauge two aspects: a) CPU utilization and b) memory consumption of our system. Monitoring the resource consumption is not supported through NiFi reporting tasks. Thus, we implement our own monitoring component as a group of NiFi processors.

The monitoring component leverages NiFi API to fetch system diagnostic report about the SMaaS cluster. The report captures the heapUtilization and
processorLoadAverage metrics. Then, the monitoring component publishes the metrics after refinement into Grafana through AMS API\textsuperscript{18}.

Recall that, SMaaS runs on top of NiFi. As NiFi executes within a Java Virtual Machine (JVM) on the host VMs, the SMaaS resources are limited to the CPU capacity and memory space afforded by NiFi. In this sense, the SMaaS components share the same resources dedicated to the JVM.

We perform the experiments in real-world settings. SMaaS is an online solution, thus it continuously runs while we execute 12 versions of the benchmark applications over various data size and anomaly types.

\textit{a) CPU Utilization}

Figure 6.7 shows the average processor utilization of our system. As noticed, the utilization varies throughout the experiment time. It does not exceed 15\% and reaches an average of 8\%. Having our system on NiFi to support data-parallel processing diminishes the impact on the CPU usage.

\textit{b) Memory Consumption}

Figure 6.8 shows the percentage of heap memory consumed by SMaaS during the experiments. The average consumption is 37\% and the maximum reached is 46\%. The average used heap memory is 3.7 GB, the maximum heap memory used is 4.6 GB, and the 95th percentile is 4.2 GB. As observed from the results, our system achieves an efficient CPU utilization and memory consumption footprint.

\textsuperscript{18} Ambari Metric Service (AMS) API: https://cwiki.apache.org/confluence/display/AMBARI/Metrics+Collector+API+Specification
6.5. SUMMARY

Cloud computing is empowering new innovations for big data. At the heart, cloud analytic applications become the most-hyped revolution. Cloud analytic applications have remarkable benefits for big data processing, making it easy, fast, scalable and cost-effective; albeit, they pose many security risks. Security breaches due to malicious, vulnerable, or misconfigured analytic applications are considered the top security risks to big data. The risk is further expanded from the coupling of data analytics with the Cloud. Effective security measures, delivered by cloud analytic providers, to detect such malicious and anomalous activities are still missing.

This chapter presents real-time security monitoring as a service (SMaaS) framework to detect security anomalies in cloud analytical applications. SMaaS provides an advanced online defense in depth for analytic applications. The benefit is
two-fold: 1) it hardens the security of analytic clusters (e.g., Hadoop) by inspecting applications running over them and 2) it protects big data processed by analytic applications by detecting anomalies that breach its security.

By intelligently leveraging streaming big data pipeline and cloud technologies, SMaaS gains benefits to elude the challenges originate in analytic applications. The SMaaS pipeline automates the collection, management, processing, analysis, and visualization of log data. The framework extracts information flow profile to model the execution of the analyzed application. The profile captures both control and data dependencies across the distributed tasks (i.e., nodes) running the analyzed application. Several techniques are employed for the detection of security anomalies based on analyzing the information flow profile. Our solution checks the profile against expected features that govern benign applications. Any deviation from the expected features indicates a security anomaly. The system helps security administrators to take mitigation actions when alerted about discovered anomalies.

In this sense, our system conceals the log processing for security inspection behind the analytic cluster's scene. It does not require any intrusive changes, installing custom agents, or introducing any overhead over the monitored cluster. It has the following implications: a) handling log data that is characterized by the 4Vs and collected across the cluster nodes; b) involving the complex data and control flows enclosed among the cluster nodes to execute analytic applications; c) considering the different roles of core daemons responsible for running such analytic
applications; and d) mining for tangible evidence of security anomalies from cluster and system logs.

Our experiments demonstrate that SMaaS attains high detection accuracy of five different anomaly types. Yet, it achieves high performance with a pretty lightweight footprint on resource utilization.
CHAPTER 7

Conclusions and Future Directions

Nowadays, more and more business and individuals tune to Software-as-a-Service (SaaS) applications to rapidly access various software capabilities through the Internet. The more SaaS adoption evolves, the more software service providers compete for fast development to cope with the market pace. This trend pushes security after functionality-needs in the priority list. This, in turn, results in delivering applications prone to various prevalent severe threats violating data confidentiality, integrity, and availability (CIA). The risk is further elevated from running these applications on the Cloud due to the lack of visibility, control, and regulatory enforcement over the applications and consumers' data they process. With the evolving expansion of threat landscape (i.e., internal and external) and the growing shortage of cybersecurity resources (i.e., tools and skills), Security as a Service (SecaaS) is gaining a momentum to fill this pressing gap. Whereas traditional techniques (access control and encryption) bound information security from access perspective, information flow control (IFC) tackled it from processing perspective to
guarantee end-to-end security. Since back to the early 70’s, IFC started to receive the research attention, however, it has been almost exclusively leveraged in type-based and taint-based systems, which may be impractical or ineffective for realistic security-sensitive cloud applications.

7.1 Summary and Concluding Remarks

This thesis takes the advantages of Security as a Service (SecaaS) model and information flow control (IFC) to thrive towards our ultimate goal for advancing the security of cloud SaaS applications. This is achieved through proposing and developing novel architectures and techniques addressing the unique challenges originating in cloud SaaS applications.

In Chapter 2, we unveiled the prevalent top security threats targeting cloud applications as part of a brief background on these applications and their hosting cloud environment. The chapter also demonstrated how these threats can lead to various information flow violations affecting data confidentiality, integrity, and availability (CIA). At the end, the chapter surveyed the literature spanning different directions staring from security-aware cloud service development, vulnerability vetting, to anomaly detection in order to bring forward various challenges and open research issues, addressed later in the subsequent chapters of this thesis.

In Chapter 3, we further explored existing SecaaS work based on our proposed tri-dimensional taxonomy: service operation, security solution, and threat. We discussed, compared, and contrasted the effectiveness of existing SecaaS work
7.1. SUMMARY AND CONCLUDING REMARKS

according to their security mechanisms in terms of visibility, control, and robustness. We also presented a comparative analysis that details the common strengths and weaknesses of existing SecaaS work. The analysis takes into consideration the implications of the service operation and the security solution as the central dimensions affecting the effectiveness of the surveyed work in mitigating the target threats. Our study helps not only in choosing an appropriate existing SecaaS but also in evaluating the new SecaaS offerings. The taxonomy aimed to present informed understanding of existing SecaaS work to help consumers and researchers in making educated decisions regarding the SecaaS adoption.

Chapter 4 presented a comprehensive platform to integrate security from inception through deployment and beyond for cloud applications. This platform encompasses IFC-based activities and services throughout the different phases of the development lifecycle. Such activities and services devote for defense-in-depth via three main tactics: administrative, preventive, and detective for improved response and resilience. Accordingly, the platform can achieve various principles to help in mitigating and defending against the top security threats to SaaS applications. Considering the different methodologies for developing cloud applications, the platform is established from a generic standpoint to provide different options for engaging the proposed activities and services in the engineering process. The platform also highlights the responsibility distribution among the involved entities (i.e., service providers, hosting providers, and trust security providers) promoting the
7.1. SUMMARY AND CONCLUDING REMARKS 184

incentive to consider security as a first-priority. The two frameworks (i.e., SDaaS and SMaaS) presented in Chapter 4 and 5 are proposed to address the different challenges originating in two different types of applications (i.e., operational and analytical) for adopting two of the services (i.e., static analysis and runtime monitoring) offered in the platform, respectively.

In Chapter 5, we considered proactively protecting cloud operational applications from top security threats involving stealth vulnerabilities as well as insecure APIs and interfaces. The risk from these threats can extend to cause data breaches and shared technology violations. Our objective was achieved by proposing Security Diagnosis as a Service (SDaaS) framework relying on static analysis. Several challenges originate in SaaS applications hindering static analysis such as runtime lifecycle, inter-service interactions, and external library invocations. With the help of our proposed lifecycle and dependency modeling strategies, our framework can provide comprehensive analysis. These strategies are crucial to holistically capture all data (explicit) and control (implicit) flows inside an application required for SDG’s construction. As part of handling inter-service interactions, we propose value computation approach as an IDE analysis. Grounded on SDG and program slicing techniques, SDaaS focuses the quantitative information flow analysis for security diagnosis. It releases a comprehensive report about the security status of an analyzed application in terms of potential risk, behavior diagnosis, and security risk rating. Based on the results, it decides if a security certificate should be granted to the
7.1. SUMMARY AND CONCLUDING REMARKS

A candidate application. This capitalizes the benefit of reducing the remediation efforts required to mitigate the detected vulnerabilities early in the development phase. The SDaaS framework can be deployed and adopted in current cloud environments without introducing any modifications in the application’s code or the underlying environment. We implemented, deployed, and evaluated SDaaS in public cloud. Our experiments validate the detection accuracy, performance, and scalability of our framework. We conducted our experiments over benchmark applications for evaluating vulnerability detection tools. We contrasted SDaaS with different tools including seven static code analyzers, three penetration testers, and one anomaly detector. Experimental results demonstrate that SDaaS outperforms other tools. It attains wider coverage of vulnerability types and higher detection accuracy with enhanced balance between F-measure and false alarm rates. It also achieves high performance and scalability with a lightweight footprint on resource utilization.

Chapter 6 tackled detecting the top security threats to analytical applications involving malicious and security anomalous activities such as data leakage, data tampering, unauthorized access, insecure computations, and security misconfiguration. This is done by proposing Security Monitoring as a Service (SMaaS) framework relying on log analysis. Developing an effective log analysis for anomaly detection solution is challenged by the distinct features of computations and data in the distributed large-scale analytic systems. Specifically, an effective solution needs to handle log data that is characterized by the 4Vs and collected across the
7.2. LIMITATIONS

cluster nodes, involves the complex data and control flows enclosed among the cluster nodes to execute analytic applications, considers the different roles of core daemons responsible for running such analytic applications, and mines for tangible evidence of security anomalies from log data. We leveraged streaming data pipeline for security inspecting to overcome these challenges. Hence, SMaaS is boosted to automate the collection, management, processing, analysis, and visualization of log data from multiple sources, making it valuable, comprehensive, and cohesive for security inspection. SMaaS particularly monitors a candidate application by collecting log data on real-time and extracts information flow profile from collected data to model the application’s execution. Then, SMaaS employs several techniques that inspect this profile against expected features which govern benign applications. Any deviation from these features indicates anomalous and suspicious activities. We evaluated the detection effectiveness and performance efficiency of our framework. The experiments are conducted over benchmark applications. The evaluation results demonstrate that our system is a viable solution, yet very efficient. Our system does not make modification in the monitored cluster, nor does it impose significant overhead to the monitored cluster's performance.

7.2 Limitations

In this section, we present the limitations of this thesis as follows:
1- Our platform mainly advocates for IFC-inspired security services to protect against application-level threats. It does not consider other security measures that
can complement the security puzzle for protecting the applications and their underlying environment.

2- SDaaS may produce false alarms by incorrectly detecting benign information flow paths as insecure especially when such paths involve complicated data dependency (i.e., complex data structures). The analysis treats composite data structures (e.g., arrays) as a single location in memory to ignore the index which is only known at runtime. Such a limitation is common and unavoidable due to the over-conservative nature of static analysis leveraged in SDaaS. In our experiments over XPathI benchmark applications, SDaaS reported the worst false alarm rate (18 false positives), albeit still lower than other tools evaluated over the same applications.

3- SDaaS has wide coverage of information flow vulnerabilities that it can detect. However, it may miss vulnerabilities for falsely considering the information flow paths as sanitized and secured even though they are prone to weak cryptography, hashing, or randomness.

4- SMaaS focuses on detecting some common types of anomalous activities involving 1) data leakage; 2) data tampering; 3) access violation; 4) misconfigurations; and 5) insecure computation. However, it does not cover the detection of unknown types or variant patterns of anomalies other than the addressed cases.
7.3 Future Directions

Some research problems and open issues can stem from our thesis work. In this section, we indicate some of them.

1- Our future work can proceed in exploring Cloud-assisted Internet of Things (CoT) applications, realized from the innovative paradigm integrating cloud computing, Internet-of-Things (IoT) and edge computing environments. Expanding on how CoT trend is differently impacting embedding security in the development cycle of these applications is a future goal. This expansion is challenged by the heterogeneity, scalability, and rapid-adaptability of the CoT applications that mainly undergo the development cycle as a complex multi-discipline engineering process. This work can consider exploring different security measures that can be employed hand-in-hand for complete security protection across the involved environments.

2- Complementary research directions can contribute towards the unimplemented security services proposed in our platform (i.e., secure coding and runtime analysis). These directions need to consider the unique requirements intrinsic to the Cloud environment towards architecting and building these services to provide effective and efficient solutions. They also entail addressing integrating the secure coding service into developer’s IDEs to vet vulnerabilities as code changes during development or automated builds and augmenting the underlying
cloud platform with runtime analysis service as bundled feature from the hosting providers.

3- In our SDaaS framework, we confined the value computation approach to handle the inter-service invocations. Our approach leverages Inter-procedural Distributed Environment (IDE) analysis to compute the value of the relevant fields required to statically reason about the targeted service. This approach can be extended to reason about other values that arise during an application’s runtime execution. For example, concrete values of the encryption, hash, or randomness algorithms; which appear as arguments to endorsement and declassification methods, can tell if an application applies a proper sanitization or not. This helps to extend the framework’s functionality to cover the detection of other vulnerability types like weak cryptography, hashing, and randomness.

4- With the speed driven by time to market, integrating SDaaS as an in-line part in continuous delivery and integration pipelines will be foremost interesting to investigate. This can provide advantage to speed up the security assessment and keep up with the acceleration of SaaS development. This requires extending SDaaS to support incremental analysis to cover recently and frequently changed code before being pushed to production. SDaaS provides security risk rating to prioritize patching the most significant vulnerabilities and critical issues. However, there will be still a need to overcome the trade-off between the analysis
7.3. FUTURE DIRECTIONS

intensity and velocity due to the tight time available to apply the analysis before delivering the changed code.

5- SMaaS address the detection of some common types of anomalous activities. Coping with the evolved patterns of anomalies, future extension of SMaaS can be driving predictive analytics geared to add intelligence for detecting these anomalies. This extension can benefit from the power of graph database and deep learning (DL). The graph database can model security data in a way that allows for identifying more features for the analysis. Whereas, the deep learning can allow detecting complex non-linear relationships and drawing significant insights to uncover advanced types of anomalies.
BIBLIOGRAPHY


BIBLIOGRAPHY


[136] H. Ulusoy, M. Kantarcioglu, B. Thuraisingham, and L. Khan, "Honeypot based unauthorized data access detection in MapReduce systems," *Proc. of*
2015 IEEE Int. Conf. on Intelligence and Security Informatics (ISI), pp. 126-131, 2015.


