# Forensic Analysis in Access Control: a Case-Study of a Cloud Application

by

Xiaowei Huang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

## Statement of Contributions

In chapter 2, my colleague Nahid Juma and my supervisor Mahesh Tripunitara aided in proving computational complexity of forensic problem in ARBAC.

## Abstract

We discuss a case-study we have conducted on forensic analysis in access control. The case-study is an application in the Amazon Web Services (AWS) cloud provider. Forensic analysis is the investigation and analysis of evidence of possible wrongdoing. Access control is used to regulate accesses to computing resources. Both forensic analysis and access control are recognized as important aspects of the security of a system. We first argue that posing the forensic analysis problem in the context of access control is meaningful and useful towards the security of a system. We then summarize results on the computational hardness of the forensic analysis problem for two access control schemes from the research literature. We point out that these results suggest that meaningful logging information can render forensic analysis tractable, even efficient. We then instantiate the forensic analysis in access control problem in the context of a cloud application. A cloud application is a software service that can be accessed over the Internet and uses computing resources provided by a cloud provider. A cloud provider provides computing tools and services that can be administered over the Internet. The cloud provider we have adopted is AWS, and the application is "Hello Retail", an image-sourcing application for online retailers. In addressing forensic analysis in this context, our particular focus is the manner in which logging information can be leveraged. We ask two kinds of questions: (i) is particular logging information from AWS necessary to answer forensics analysis questions of interest, and, (ii) is particular logging information sufficient? We observe that from the standpoint of (i), default AWS logs have considerable redundancy. We propose an algorithm to prune logs for efficient forensic analysis. From the standpoint of (ii), we observe that it is not possible to definitively answer 'yes' or 'no' to forensic analysis questions of interest given only the information AWS permits us to log. We identify additional logging information that, if available, would be sufficient. Together, (i) and (ii) provide us with "goal-directed logging." We conclude by reiterating the benefits of forensic analysis in access control, and with suggestions for goal-directed logging in cloud systems.

## Acknowledgements

I would like to thank my parents for supporting my study here at University of Waterloo.

I would like to thank my colleague, Nahid Juma, for her previous work on Forensics Analysis in Access Control. Her research provides me with insight on forensic analysis in access control.

I would like to thank my supervisor, Professor Mahesh Tripunitara, for guiding me through this research.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **AWS** | Amazon Web Services 7, 27, 34 |
| **S3** | Simple Storage Service 27, 29, 31, 33 |

# Chapter 1

# Introduction

We address *forensic analysis*, which deals with the collection and analysis of data related to security incidents. It is a well-established discipline in settings outside of computing [9]. It is recognized as an important aspect of computer security as well [10]. An example of a security incident in computing is a leak of contents from a file that is to remain confidential to only personnel in a company. An example of forensic analysis of this incident would be tracing a suspect who possibly leaked the confidential information. An example of a forensic analysis question that we may ask is, "Did Alice leak the confidential information?", where Alice is a suspect.

Separately, *access control* comprises principles and mechanisms to regulate access to resources by active entities in a system. For example, computer processes of a user Alice may be allowed to both read and write a file, whereas those of another user Bob may be allowed to read it only. Such an expression of who may do what with a particular resource is called a policy. A mechanism for enforcement of the policy when an access is attempted is called a reference monitor. An issue with which research and practice in access control deals is changes to the policy, which may themselves be governed by policies and mechanisms which are said to fall under administration.

While each of forensic analysis and access control is recognized as an important aspect of information security, to our knowledge, they have largely been addressed in isolation to one another. Access control is perceived as a preventive security mechanism, that is, intended to prevent undesirable occurrences from happening at all. Forensic analysis, on the other hand, is part of a response mechanism — the investigation of an undesirable incident after one happens, presumably with the intent of holding a wrongdoer to account, and understanding what needs to be changed with the preventive security policies and mechanisms. To our knowledge, the work of Juma [22] is the first to unify forensic analysis and access control in a meaningful way. Specifically, that work poses the forensic analysis problem in the context of access control systems.

Figure 1.1 gives a high level perspective of the above discussions. The figure shows that an administrator, who may or may not be fully trusted, creates and edits access control policies. Any changes they make to the policy may be logged. The policy is used as input by the reference monitor to make allow/deny decisions when a user requests access to a resource. Both the request and the allow/deny decision may be logged. Forensic analysis can then use these logs to answer a query such as, "could Alice have edited the access control policy to allow Bob to read the file?"

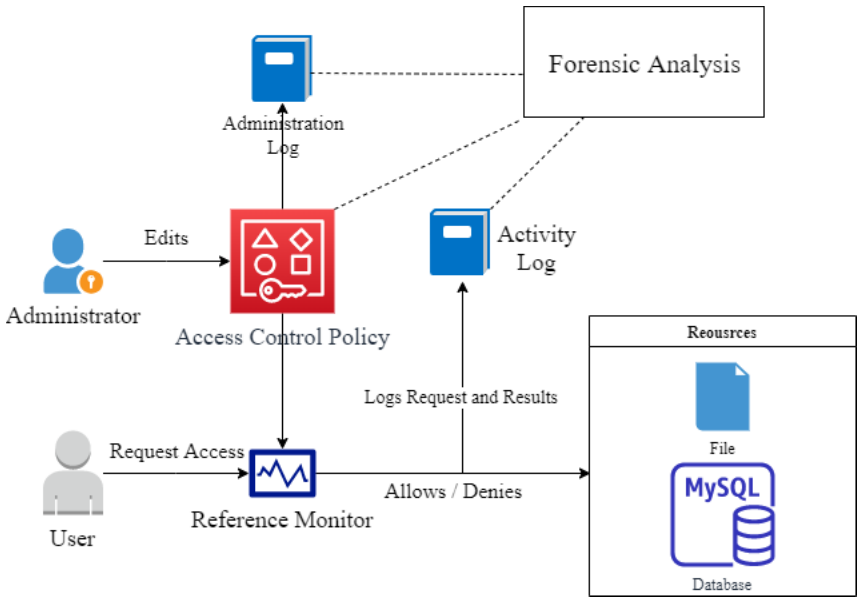


Figure 1.1: A high level view of forensic analysis and access control. As the figure shows, an administrator has the ability to grant/revoke a user's privilege in the access control policy. The changes made to access control system will be record in the Administration log. When user requests access to the resources in the system, the reference monitor compares user's request with user's privilege assignment in the access control policy and either allows or denies the request. The user's request and reference monitor's result will be logged in the activity log.

We give an example of forensic analysis in access control. We collect the evidence from Access Control Policy, the Activity Log and the Administration Log. We review the evidence and pose forensic questions base on the incident. If the contents of a confidential file have been leaked, we check the activity log to see who has accessed the file in the past. If it turns out that Bob requested access to a file he does not own, Bob is considered as a suspect. Furthermore, we can ask "Who would have given Bob the access to the file". We check the Administration log and it turns out Alice edited the access control policy to give Bob access to read the file. If this looks abnormal, Alice may be considered as an accomplice. Incident mitigation procedure includes changing access control policy so administrators cannot give other users read access to the file in the future.

This thesis presents a case-study of forensic analysis in access control in the context of a cloud application. A cloud application is a software service that can be accessed over the Internet that uses the computing resources provisioned by a cloud provider. For example, an online retailer can deploy a cloud application to source product images from photographers. The front-end of the application is a website hosted by the cloud provider. The back-end of the application is a server-side application running on Linux provisioned by the cloud provider. In our work, the cloud application is deployed in AWS [1], which is a cloud provider. Our choice of AWS is motivated by the fact that it is, to our knowledge, the most widely-used cloud provider [23].

Figure 1.2 gives a high level perspective on cloud applications and the manner in which they are administered and accessed. The figure shows that a cloud application is built with computing, data and storage services provided by the cloud provider. Users access the cloud application via Internet. The Cloud provider provides administrator with an access control system that can be used to monitor and control users access to the cloud application.

We choose to research forensic analysis in cloud application because ordinary digital forensic tools do not apply in the scope of the cloud environment [10]. By migrating enterprise application from a physical machine in a local data center to a public cloud, the sensitive data in the application suffers from an increased attack surface. Typical digital forensic tools require access to the physical machine to collect evidence[6]. In a cloud computing platform with the virtualization technology enabled, access to hardware is limited. When a security incident happens in cloud environment, the investigator may not be able to collect enough evidence to track down the suspect. The traditional forensic tools provided in Carvey's DVD focus on collecting evidence from access control system's activity log [6]. We look at administrative log and argue that state changes in the access control system may also provide us with useful information.

In the case-study, we examine AWS' access control scheme and logging mechanism. We simulate a security incident in the cloud Application we deployed to AWS. We study access

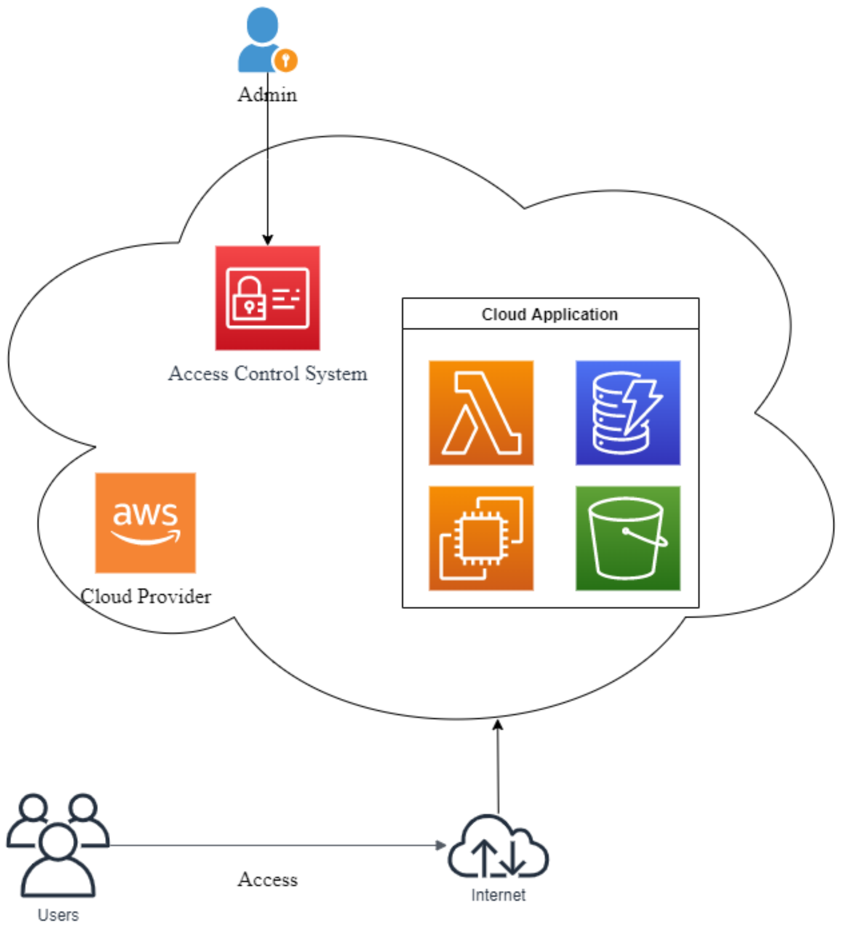


Figure 1.2: A high level view of a cloud application residing in a cloud provider. Cloud provider is built with computing and data storage services provider by cloud provider. The cloud application is to be accessed by users through the Internet.

logs generated by AWS and analyze whether they are sufficient to definitely answer 'yes' or 'no' to forensic analysis questions of interest, as motivated by the work of Juma[22].

## 1.1 Problem Statements

We propose the following problems that this thesis attempts to answer regarding forensics analysis:

1. Is it possible to perform the forensic analysis with the given access control model? Can we answer forensic problems by having knowledge to an access control model and its past states?
2. Can we perform forensics analysis in cloud computing environment, specifically Amazon Web Services, by leveraging its access control model and logging mechanisms?

## 1.2 Related Work

Digital security refers to various way of protecting computational resources from intrusion by outside users. The book by Gasser et al. discuss the approaches to build a secured computer system [11]. The book discusses digital security from different views, one of them being constructing security models. A security model is a scheme for specifying and enforcing security policies. Security model, also known as access control model, is an essential part of security in a system.

Graham-Denning model, introduced by Graham et al. is considered as a fundamental work in the context of security model [14]. Graham-Denning model shows how resources in a system can be securely created and destroyed. Graham-Denning also addresses how to assign specific rights in a system.

The HRU model designed by Harrison et al. extends Graham-Denning model [15]. Harrison's work also discuss the possibilities and limitation of proving the safety of systems using an algorithm, known as safety analysis. Safety analysis asks whether there exists a state in the system that a user may hold a specific privilege in the future. They claim that verifying safety of an HRU model is an undecidable problem.

Other security model has been proposed by researchers. Sandhu et al. proposed Role-based access control (RBAC) model in their paper [26]. RBAC assigns users into roles and restricts their access based on the role assignment. RBAC is adapted by many applications as underlying security model. The safety analysis problem of RBAC has also been researched. Li et al. showed that safety of RBAC model is PSPACE-complete [18]. The author of RBAC also introduced ARBAC(Administrative RBAC), an extension to RBAC that allows RBAC scheme to change over the time base on the sets of explicit rules [25]. Safety of ARBAC has also been proved to be PSPACE-complete by Sasturkar et al [29].

In this thesis we propose a case study in cloud computing environment. Wang showed that risks and security problems in traditional computer system also apply to the cloud computing environment [31]. Khan showed that security model can be used to manage resources in cloud computing environment [16]. Zhou et al. showed that RBAC model can be used to control access to encrypted data in a large public cloud infrastructure. Wang et al. introduced the trust model RBAC scheme for reliable access management in cloud computing environment [30].

Forensic science is an application of science to criminal and civil laws during the criminal investigation. Among forensic science, digital forensics is a branch encompassing the recovery and investigation of material found in digital devices[5]. Casey introduced methodology and technology in performing digital forensics and investigation in computers [7]. Carrier et al. gave an in-depth explanation on forensic analysis in the operating system's file system [4]. The book by Carvey introduced a digital forensic toolkit, **The Sleuth Kit**,that can be used to investigate volume and file system data[6].

Work by Khanuja et al. showed that it is possible to construct a framework to perform digital forensics in a database system [17]. They proposed a method to extract useful information from MySQL operational logs and export it for analysis. Olivier et al. also worked in the context of database forensics [24]. The work proposes an idea of transfer database forensics into file system forensics and apply mature techniques in file system forensics to the database system.

It is possible to bring digital investigation into cloud computing environment. Work by Garfinkel explained discussed challenges digital investigation may face in the upcoming ten years[10]. They suggested that growing size of storage devices may make current data retrieval techniques less effective. Use of cloud storage service may make existing tool-kits fail due to lack of access to physical storage devices.

Marturana et al. showed evidence can be extracted from a cloud storage system by leveraging existing digital forensics procedures [20]. Their work showed that evidentiary material of the user-to-Cloud interaction can be found by examining local artifacts without accessing cloud service directly.

Marty showed that logs are important analytical data in cloud environment [21]. They showed that user guidelines on what information to log is important to guarantee the availability of data needed for forensic investigation. He also showed that it is important for applications running on the cloud environment to generate and preserve logs to achieve complete forensic investigation data. This work leads to an important view in our thesis. We argue that it is not always possible to guide software in the cloud environment on what information to log. Thus, we propose forensic analysis from a complete different approach by analyzing access control scheme.

There are some key difference between our work and the work discussed above. First, we focus on forensic analysis regarding access control system. Second, we observe the similarity between forensic analysis and safety analysis for access control scheme. And we use this similarity to prove computational complexity of forensic analysis problems. Third, we take a different approach in building a forensic framework in the cloud environment. Unlike logging framework for data plane actions in the cloud, we build a logging framework for access control model and rely on that to perform forensic analysis.

## 1.3 Outline

The chapters of this thesis are organized as follows.

In chapter 2 we give a background introduction on access control model used in this thesis. We show that computational complexity of security analysis of the models discussed has been proved in previous research work. We summarize work of Juma [22] that is complementary to our work. That work addresses the foundational aspects of forensic analysis in access control scheme. Juma's work proves the computational complexity of forensic analysis problem on the access control model base on a reduction from safety analysis. Our work in the following chapter addresses a case-study of forensic analysis in a real system.

In chapter 3 we perform an case study on Amazon Web Services (AWS) to show that access control model with its logs can be used to answer forensic questions on cloud computing platform. We also propose a new logging mechanism, goal-directed logging, that makes it efficient to answer forensic questions.

In chapter 4 main conclusion of this thesis is discussed. Possible future works are also discussed.

# Chapter 2

# Forensic Analysis in Access Control: the Foundations

In this chapter, we summarize the work of Juma [22], who, to our knowledge, is the first to pose the forensic analysis problem in the context of access control. In section 2.1, we introduce terminology as it pertains to access control systems. In section 2.2, we formulate forensic problems in access control and forensic queries into 3 different types. In section 2.3, we summarize Juma's work that reduce safety analysis problem to forensic analysis problem.

## 2.1 Access Control

**Access control** is a way of regulating access to resources by active entities in a system. Access control deals with actions, such as read or write, that a principal may exercise on a resource. When a principal, such as a process in an operating system, seeks to exercise the action, access control ensures that attempt is mediated. An *authorization policy* is used by the mediator to determine whether the attempt should be allowed. In realistic systems, the authorization policy may change over time. For instance, Alice may be disallowed from writing to a file she used to have access to.

Here is some terminology used in the scope of access control:

- Subject : An active entity that exercise privileges on resources,typically a user or a process

- Object : A passive resource, typically a file or a program
- Operation: The actions subject performed on objects, typically read, write or deletion

Access control rights are defined in the form of **access control matrices** $M$, where:

$M = (M_{so})_{s \in S, o \in O}$ with $M_{so} \subset A$, $S$ is a set of *subjects*, $O$ is a set of *objects*, $A$ is a set of *operations*

The entry $M_{so}$ defines set of operations subject $s$ may perform on the object $o$. Figure 2.1 gives a simple example of access matrix for two users(subjects) and three objects(files).

| | File A | File B | Application C |
|---|---|---|---|
| Alice | Read<br>Write | | Read<br>Execute |
| Bob | | Read<br>Write | Read<br>Write<br>Execute |

Table 2.1: An example of access matrix $M$

In the access matrix above, File A can be read and written to by Alice while Bob has no access to it at all. File B can be read and written to by Bob while Alice has no access to it at all. Application C can be executed by both Alice and Bob but only Bob can write to it. In a system, a reference monitor would verify a subject's request against an access matrix and mediate the subject's request. The Access matrix can be used as a model of static access permissions of the system. It does not model the rules by which permissions can change. Thus, it should be considered as a static model of permissions at a given time. *Access control scheme* models how permissions can change in a system from time to time. Access control schemes consist of multiple access control matrices where each access control matrix represent a change of state in the system.

We define access control scheme used in this thesis by adopting meta-formalism introduced by Li et al [18] [19].

**Definition 1.** An access control scheme is a four-tuple, $\langle \Gamma, \Psi, Q, \vdash \rangle$, where $\Gamma$ is a set of states, $\Psi$ is a set of state-change rules, $Q$ is a set of queries and $\vdash : \Gamma \times Q \rightarrow \{true, false\}$ is the entailment function, which specifies whether given a state, a propositional logic formula of queries is true or not in that state.

A state, $\gamma \in \Gamma$, contains all the information necessary to make access control decisions at a given time. When a query, $q \in Q$, arises from an access request, $\gamma \vdash q$ means that the access corresponding to the request $q$ is granted in the state $\gamma$ and $\gamma \nvdash q$ means that the access corresponding to $q$ is denied. A state-change rule, $\psi \in \Psi$, determines how the access control system changes state. Given two states $\gamma$ and $\gamma_1$ and a state-change rule $\psi$, we write $\gamma \rightarrow_{\psi} \gamma_1$ if the change from $\gamma$ to $\gamma_1$ is allowed by $\psi$, and $\gamma \rightarrow^{*}_{\psi} \gamma_1$ if a sequence of zero or more allowed state-changes leads from $\gamma$ to $\gamma_1$. An access control system based on a scheme is a state-transition system specified by the four-tuple $\langle \gamma, \psi, Q, \vdash \rangle$, where, $\gamma \in \Gamma$ is the current state, and $\psi \in \Psi$ is the state-change rule that specifies how states may change. We assume that $\psi$ is static, that is, the state-change rule is defined when the access control system is first put in place, and is not changed thereafter.

We explain two access control schemes, the Harrison-Ruzzo-Ullman (HRU) scheme [15] and the Administrative Role-based access control (ARBAC) scheme [25]. HRU scheme is considered as a foundational work in the access control field. It provides a model that is sufficiently powerful to encode several access control approaches, and is precise enough so that security properties can be analyzed. RBAC scheme is one of the most used access control schemes nowadays. The target of case study in this thesis, Amazon Web Service, has implemented an RBAC scheme as its access control scheme. ARBAC provides administration to RBAC and is more practical in the real world due to the administration capability it provides.

### 2.1.1 HRU

The Harrison-Ruzzo-Ullman (HRU) access control scheme addresses the policies for changing access rights and for the creation and deletion of subjects and objects in a system[15]. In HRU model, there is:

- a set of subjects $S$
- a set of objects $O$
- a set of access rights $A$

- An access matrix $M = (M_{so})_{s \in S, o \in O}$ with $M_{so} \subset A$

For HRU, there exists six *primitive operations* for manipulating the $S$,$O$ and access matrix $M$:

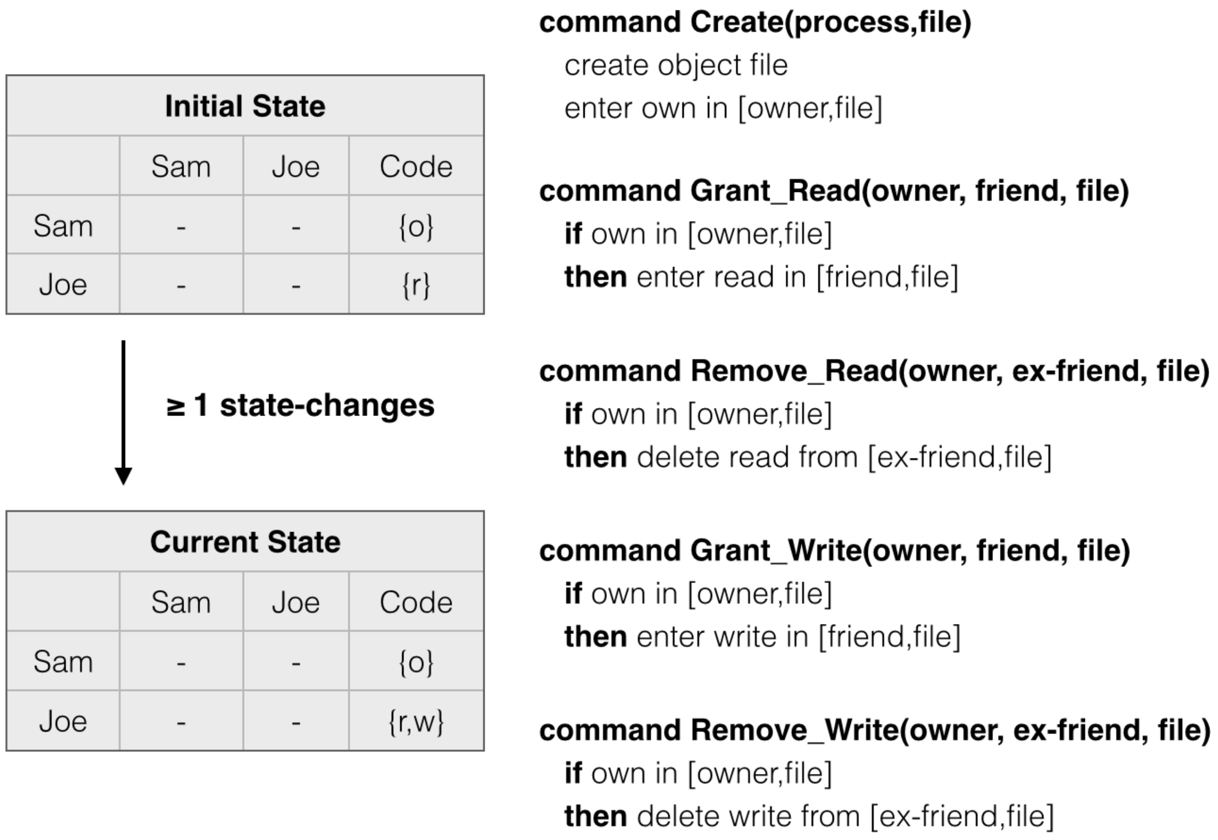
- **enter** r into $M_{so}$
- **delete** r from $M_{so}$
- **create subject** $s$
- **create subject** $s$
- **delete object** $o$
- **delete object** $o$

The command defines the possible state change rules in the HRU access scheme.

An authorization system is defined by a set of commands and by its state as captured by access matrix. Figure 2.1 gives an example of an actual authorization system using HRU access control scheme. In the system, Sam and Joe $\in S$ are two subjects and Code $\in O$ is an object. There are three rights $own(o), read(r), write(w) \in A$. The state change rules consists of five commands: `Create`, `Grant_read`, `Remove_Read`, `Grant_Write`, `Remove_write`. State are represented as access matrices. One or more state change command were fired to bring system from its initial state to its current state. In its current state, Joe has read and write access to the code.

We give a formulation of HRU scheme used in this thesis:

**Definition 2.** We assume two sets of subjects ($S$) and objects ($O$). Every subject is an object, i.e., $S \in O$. The set of states, $\Gamma$, is the set of all possible access matrices. Each state $\gamma \in \Gamma$ is defined by a finite set of subjects ($S_\gamma \in S$) and objects ($O_\gamma \in O$), and an access matrix, $M_\gamma[]$. The access matrix has a row for each subject and a column for each object, and the cell $M_\gamma[s, o]$ contains the set of e rights subject $s$ has over object $o$. The state-change rule is denoted by $\psi = \langle C_\psi, R_\psi \rangle$ where $C_\psi$ is a set of commands and $R_\psi$ is a set of rights. A state-change is brought about when a command is fired. Each command is composed of one or more primitive operations and can additionally have a list of pre-conditions that check for certain rights in certain cells of the access matrix. The primitive operations can modify the access matrix by adding or deleting rights from a cell and by creating or destroying subjects or objects. $\Psi$ is the set of all such tuples



| Initial State | | | |
|---|---|---|---|
| | Sam | Joe | Code |
| Sam | - | - | {o} |
| Joe | - | - | {r} |

| Current State | | | |
|---|---|---|---|
| | Sam | Joe | Code |
| Sam | - | - | {o} |
| Joe | - | - | {r,w} |

```
command Create(process,file)
  create object file
  enter own in [owner,file]

command Grant_Read(owner, friend, file)
  if own in [owner,file]
  then enter read in [friend,file]

command Remove_Read(owner, ex-friend, file)
  if own in [owner,file]
  then delete read from [ex-friend,file]

command Grant_Write(owner, friend, file)
  if own in [owner,file]
  then enter write in [friend,file]

command Remove_Write(owner, ex-friend, file)
  if own in [owner,file]
  then delete write from [ex-friend,file]
```

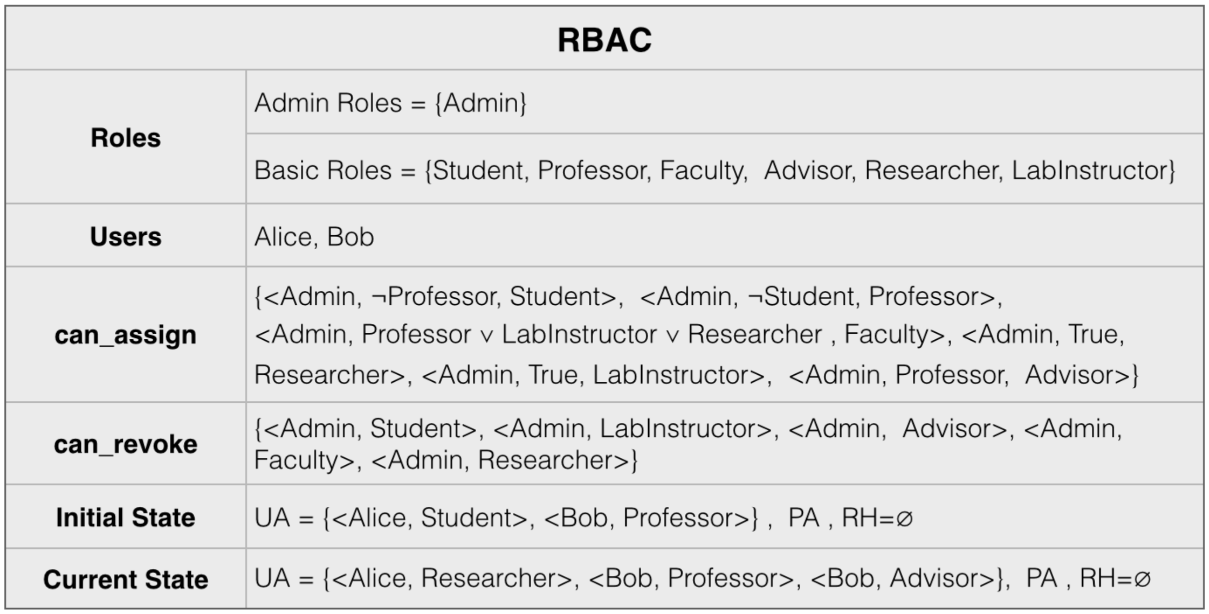
Figure 2.1: Example of an HRU access control scheme

of commands and rights. Examples of queries, $q_1, q_2 \in Q$, are $q_1 = r \in M[s,o]$ and $q_2 = r' \in M[s,o]$. The queries, $q_1$ and $q_2$, ask whether the subject $s$ has the rights $r$ and $r'$ over the object $o$, respectively. Given a state, $\gamma$, we say $\gamma \vdash q_1 \wedge \neg q_2$ if and only if $s \in S_\gamma \wedge o \in O_\gamma \wedge r \in M_\gamma[s,o] \wedge r' \notin M_\gamma[s,o]$.

### 2.1.2 ARBAC

RBAC is role-based access control scheme introduced by Sandhu et al[26]. RBAC is used to define *Authorization policy*, which mediate access attempt. In RBAC, permissions are associated with roles, users are assigned to roles, and a user has a certain permission if and only if at least one of the roles they have been assigned to has that permission associated with it. Roles can be related to one another in a partial order called a role hierarchy. Consequently, given the set $U$ of users, $P$ of permissions and $R$ of roles, an RBAC state $\gamma$ is a triple $\langle UA, PA, RH\rangle$ where $UA \in U \times R$ is the user-role assignment relation, $PA \in P \times R$ is the permission-role assignment relation and $RH \in R \times R$ is the role hierarchy. A state-change occurs when a modification is made to either of the three components, $UA$, $PA$ or $RH$. For the scope of this thesis, we only focus on state changes made to $UA$ and assume $PA$ and $RH$ do not change. This is because in a system user-role assignment changes most frequently while premission-role assignment and role hierarchy rarely change.

ARBAC(Administrative RBAC) is an extension to RBAC. It is a scheme that uses RBAC itself to administer state-changes in an RBAC system. ARBAC is the first and most comprehensive administrative scheme to have been proposed for role-based access control (RBAC) [25]. ARBAC specifies the way in RBAC may change over the time. We note that the administrative part of ARBAC makes the scheme comparable to HRU, since the state change rules are defined for the access control scheme. In short, an ARBAC policy defines administrative roles, and specifies how members of each administrative role can modify the RBAC policy [28].



| **RBAC** | |
|---|---|
| **Roles** | Admin Roles = {Admin} |
| | Basic Roles = {Student, Professor, Faculty, Advisor, Researcher, LabInstructor} |
| **Users** | Alice, Bob |
| **can_assign** | {<Admin, ¬Professor, Student>, <Admin, ¬Student, Professor>, <Admin, Professor v LabInstructor v Researcher , Faculty>, <Admin, True, Researcher>, <Admin, True, LabInstructor>, <Admin, Professor, Advisor>} |
| **can_revoke** | {<Admin, Student>, <Admin, LabInstructor>, <Admin, Advisor>, <Admin, Faculty>, <Admin, Researcher>} |
| **Initial State** | UA = {<Alice, Student>, <Bob, Professor>} , PA , RH=∅ |
| **Current State** | UA = {<Alice, Researcher>, <Bob, Professor>, <Bob, Advisor>}, PA , RH=∅ |

Figure 2.2: Example of an ARBAC access control scheme

Since this thesis only focus $UA$ part of RBAC scheme, in ARBAC we focus the $URA$

portion. $URA$ defines the way which users are authorized to or revoked from the roles. Under our assumption that $PA$ and $RH$ do not change, the state change of ARBAC is either addition of entry $< u, r >$ to set $UA$ or removal of entry $< u, r >$ from the set $UA$. The state change rule $\psi$ are defined by two relations: *can_assign* and *can_revoke*. These two relations address two issues with respect ARBAC scheme: who may perform these operations, and under what conditions may he or she performs these conditions.

*can_assign* specifies under what conditions a user may be assigned to a role. A *can_assign* rule is of the form *can_assign*$\langle a, C, t\rangle$ where $a$,$t$ are roles, $C$ is a condition. $C$ is a set in which each entry is either a role $r$ or a role negation $\neg r$. The semantics of *can_assign* is that administrative role $a$ is able to assign target role $t$ to an assignee provided that the assignee is already a member of every non-negated role in $C$ and is not a member of any negated role in $C$.

*can_revoke* specifies the roles which users' membership can be revoked. A *can_revoke* rule is the form of *can_revoke*$\langle a, t\rangle$. where $a$ is an administrative role and $t$ is a target role. The semantics of *can_revoke* is that a member of administrative role $a$ is able to revoke a target user's membership from target role $t$.

We take figure 2.2 as an example. In figure 2.2's ARBAC scheme. *can_assign*$\langle Admin, \neg Professor, Student\rangle$ means a user who is member of *Admin* role is able to assign user to *Student* role, given the user is not a member of $Professor$ role. *can_assign*$\langle Admin, Professor, Advisor\rangle$ means a user who is member of *Admin* role can assign user to *Advisor* role given that user is already a member of $Professor$ role. *can_revoke*$\langle Admin, Student\rangle$ means a user who is member of *Admin* role is able to revoke *Studet*-role membership from any user that is assigned to *Student* role. Note that for *can_revoke* rule, there is no condition since revocation is considered as a safe operations[25].

### 2.1.3 Safety analysis in Access Control

A state of an authorization system is said to **leak** the right $r$ if there exists a command $c$ that adds the right $r$ into a position of access matrix $M$ that previously did not contain r. Formally, there exist $s$ and $o$ so that $r \notin M_{s,o}$ but $r \in M'_{s,o}$.

A state of an authorization system, i.e access matrix $M$, is said to be **safe** with respect to right $r$ if no sequence of commands can transform $M$ into a state that leaks $r$. [13]

We give two examples of Safety analysis in HRU and ARBAC.

**HRU Safety analysis** HRU-safety takes three inputs:

1. A query, a triple $(s, o, r)$, where s is a subject, o is an object an r is a right.
2. $\gamma$, current state of HRU scheme
3. $\psi$, state change rules

The output of the HRU-safety instance is 'false', if there is a state $\beta$ that is reachable from current state $\gamma$ that subject $s$ hold right $r$ over object $o$ i.e the system is not safe. Otherwise, the output is 'true'.

**ARBAC Safety analysis** ARBAC-safety takes three inputs:

1. A query, which is a pair $\langle u, r \rangle$, where $u$ is a user and $r$ is a role
2. $\gamma$, current state of ARBAC scheme
3. $\psi$, state change rules, which is instance of *can_assign* and *can_revoke* rules

The output of ARBAC-safety insance is 'false' if there exist a state $\beta$ from current state $\gamma$ that user $u$ can be assigned to role $r$, i.e. the system is not safe. Otherwise, the output is 'true'.

Previous work has shown the computational complexity of HRU and ARBAC safety analysis. HRU-safety is shown to be undecidable [15]. ARBAC-safety is analysis is shown to be PSPACE-Complete [28].

**Theorem 1.** *Safety analysis on HRU is undecidible.*

**Theorem 2.** *Safety analysis on ARBAC is PSPACE-Complete.*

## 2.2 Forensic Analysis in Access Control

In this section, we introduce forensics analysis in access control. Forensic analysis in access control deals with collection and analysis of evidence in access control scheme. We pose a forensic question to the access control scheme. We analyze state changes performed to an access control scheme and use that to resolve forensic question.

Safety analysis is complementary to forensic analysis. Safety analysis asks whether a query can be true in the future while forensic analysis asks whether a query is true in the past. Because of this correlation, properties of safety analysis problem such as computational complexity may be applied to forensic analysis. The computational complexity of safety analysis in access control scheme has been studied in previous papers. Safety in HRU control scheme has been showed to be undecidable (HRU)[15]. Safety in other schemes has been showed to be decidable, but intractable (PSPACE-Complete for ARBAC) [28]. We wonder the same computational complexity analysis can be applied to forensic analysis questions. The computational complexity of forensic analysis gives us insights on feasibility of applying forensics analysis in the context of access control system.

### 2.2.1 Classification of Forensic Question

During forensic analysis of a security incident, investigators may pose a set of questions that they are interested in. Those questions are considered as forensic questions. This thesis focuses on forensic analysis in the scope of access control. We classify forensics questions into 3 different types.

1. Possible possession of an access
2. Actual possession of an access
3. Exercise of an access

**Possible possession of an access** In this type of question, we ask whether it is possible that a user possesses an access. This type of question is useful in the forensic analysis. For example, if a confidential file is compromised in a system and investigator want to generate a list of suspects to investigate on them, knowing who would possibly had access to the confidential file in the past would be useful. Another example would be if Alice wanted to be exonerated from the suspects list, she can show that it is not possible for her to have had access to the file in the past state. In this way, an investigator can use negative possible query to exempt users from the investigation. Possible possession questions may be used to exonerate a suspect.

- **Possible possession question in HRU** Take HRU control scheme in 2.1 as an example. Suppose we ask if it is possible that Sam had read access to the Code in the past. And we have system's initial state $\iota$, its current state $\gamma$. However we do not have information on actual state changes from $\iota$ to $\gamma$. The query $q$ is true because there exist a possible sequence of state changes: `Grant_Read(Sam, Sam, Code)`, `Revoke_Read(Sam, Sam, Code)`, `Grant_Write(Same, Joe, Code)`. On the other hand, if we know the state change from $\iota$ to $\gamma$ is due to the command `Grant_Write(Same, Joe, Code)` and nothing else, then the query $q$ is false. The existence of log $L$ will affect the answer to the forensic instance. The detailed explanation of log in forensic instance will be described shortly.

- **Possible possession question in ARBAC** Take ARBAC scheme in 2.2 as an example. Suppose we ask whether it is possible that Alice was assigned to Advisor in the past, and we have no information on state changes. By examining the state change rules $\Psi$, we see that query $q$ evaluates to false because in order to be assigned to Advisor role, Alice has to be assigned to Professor first. And Professor role is not revocable.

**Actual possession of an access** In this type of question, we ask if a user actually holds an access. This type of forensic question is useful in investigation as well. For instance, if a file is compromised in a system, if answer to 'Did Alice actually have access to the file in the past' is 'No'. Then Alice can be exonerated from suspicion. Possible possession questions may be used to exonerate a suspect as well as confirm someone's suspicion.

- **Actual possession question in HRU** Take HRU scheme in 2.1 as an example again. Suppose we ask whether Sam actually had access to the code in the past. The answer to the query is unanswerable because there are multiple paths that lead from initial state $\iota$ to current $\gamma$. If we do not have information about the path taken, we are not able to answer the forensic query. However, if we know the state change is due to the following command: `Grant_Write(Same, Joe, Code)` and nothing else. We are able to the answer that the query is false since Sam did not have access to the code in the past. Again the existence of Log $L$ will change the answer to the forensic instance.

- **Actual possession question in ARBAC** Take ARBAC scheme in 2.2 as an example again. Suppose we ask if Alice was actually assigned to Advisor in the past. The answer to that query is false because similar to the possible access question, Professor role is not revocable. In this case, even without the knowledge of the log, we are able to answer the forensic query.

**Exercise of an access** The third type of questions an investigator may ask is the use of the access. This type of question may provide stronger evidence than the other two types of questions. However, the drawback is the queries related to this type of questions are outside of the scope of access control scheme. Information about user activities is necessary to answer this type of questions. As a result, this type of questions are excluded from the forensic analysis in access control.

We classify forensic questions into three types because the amount of evidence required to answer them are different. The answer to “Exercise of an access” implies “Actual possession of an access”. And the answer to “Actual possession of an access” implies “Possible possession of an access”. Figure 2.3 shows a access control system with multiple states. To answer “Did Alice actually have had read access to file 1” we need to collect logs on all state changes within a system. To answer “Did Alice actually accesse file 1”, we need to collect activity logs. The sheer volume of activity logs may prevent investigator from asking this type of questions. In chapter 3, we show that we can pose this type of the question in forensic analysis. However, we are not able to generate an answer as the activity log in AWS is not sufficient to answer the question.

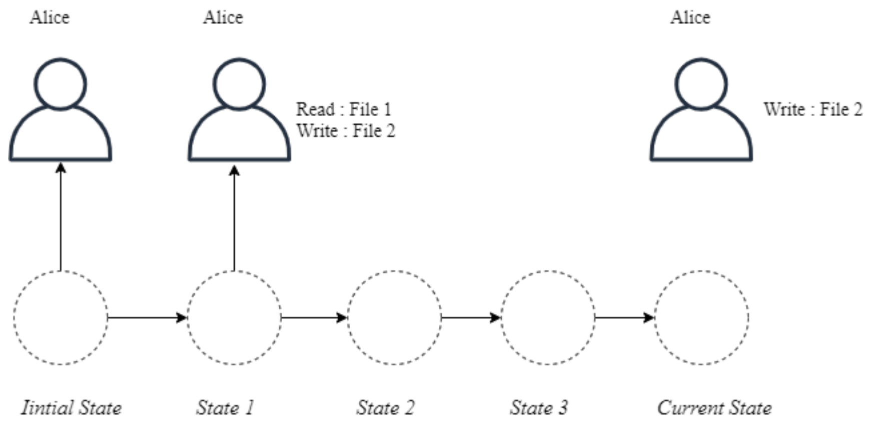


Figure 2.3: Example of state changes in an access control scheme. In the initial state, Alice does not have access to the files. In state 1, Alice has read access to file 1 and write access to file 2. In system’s current state, Alice lost read access to file 1.

### 2.2.2 Formulation

We keep adopting definition introduced by Li et al. to define a forensic instance[18] [19].

**Definition 3.** Given an access control scheme $\langle \Gamma, \Psi, Q, \vdash \rangle$, a forensic instance has the form $\langle \iota, \gamma, \psi, q, \pi, L, T \rangle$, where $\iota \in \Gamma \cup \{?\}$ represents an initial state, $\gamma \in \Gamma$ is the current state, $\psi \in \Psi$ is the system’s state-change rule, $q$ is the forensic query, $\pi \in \{\texttt{possible}, \texttt{actual}\}$ denotes the type of analysis, $L$ is a log, and $T$ is a set of trusted users.

If $\pi =$ `possible`, then the output of the instance is either true or false. It is true if there exists a path from the initial state to the current state such that the state-changes are initiated by untrusted users, and the query is true in some state on that path. It is false otherwise.

If $\pi =$ `actual`, the output can be true, false or unknown. It is true if all state-changes on the actual path taken were initiated by untrusted users, and the query was true in a state along this path and false if either of these two conditions are not satisfied. The output is uncertain if the input information, in particular, the log, $L$, is insufficient to determine whether the instance is true or false.

We give a detailed explanation of each elements in forensic instance tuple:

**Initial State** $\iota$ One may wonder why there is a need to capture initial state of a system in a forensic instance. Firstly, initial state allows investigators to limit how far back in the time the forensic analysis should go. Secondly, the initial state of the system may not be empty. It is typical for administrator to set up access control system initially then hand it to users. If the initial state is empty for a system, $\iota$ can be empty so there is no loss of generality here.

**Current State** $\gamma$ As important as initial state, current state together with initial state provide a range of time for investigators to look into for forensic analysis. Furthermore, some forensic query can be answered by looking at current state directly.

**State-change rules** $\psi$ State-change rules limit how access may change over time. It is the most important component in forensic instance. When collecting forensic evidence, state-change rule must be collect completely to ensure correct forensic query result. Empty logs $\psi$ may result in undecidable forensic result. Incomplete state-change rule will result in wrong forensic result however.

**Forensic Query** $q$ The query in forensic is permitted to be general, similar to the query in the previous study with respect to security analysis and safety analysis.[19] The query may also be negative. For instance, the query may ask if a subject did not hold access to an object in the past. The query may also be a complex Boolean logic statement with $\wedge$ and $\vee$ operators. For example, the query $(s_1, o_1, r_1) \wedge \neg(s_2, o_2, r_2)$ asks about whether subject $s_1$ had right $r_1$ over object $o_1$ and subject $s_2$ had right $r_2$ over object $o_2$ at the same time in the past. The complexity of the query allowed depends on the design and implementation of the forensic analysis system. In this thesis, we only consider basic positive queries in forensic instance.

**Access type** $\pi$ In this thesis we focus on two different types of forensic analysis i.e $\pi \in \{\texttt{possible}, \texttt{actual}\}$. $\pi = \textit{Possible}$ indicates that forensics instance asks whether it is possible for a query to be true in the past state. $\pi = \mathit{Actual}$ indicates that forensics instance asks whether a query is actually true in the past state. We separate actual possession and possible possession questions into two different types because the amount of evidence required to answer them are different. However we argue these two type of forensic questions have its own usage in forensic analysis and are equally important.

**Logs** $L$ A log records state changes that happened in a system. Logs may be partial or complete, they may contain a subset of state changes or all past changes. If no log is available to a forensic instance, then $L = \perp$. Similar to example illustrated in possible access and actual access in HRU, the answer to forensic query may differ based on available logs.

## 2.3 Reduction from Safety Analysis to Forensic Analysis

In this section, we summarize the reduction performed in Juma's work[22]. Juma's work unifies forensic analysis with access control. That work reduces forensic analysis problem from safety analysis problem and gives a lower bound complexity of forensic analysis problem. Juma's work is complementary to our work in the following chapter where our work address a case-study of forensic analysis problem in a real system. We show computational complexity of forensic analysis problem in two access control schemes, HRU and ARBAC respectively.

By adopting the notation in the definition, an access control scheme has the form of $\langle \Gamma, \Psi, Q, \vdash \rangle$. An safety analysis instance to this access control scheme has the form of $\langle \alpha, \psi, T, w \rangle$ where $\alpha$ is the current state of the system, $\psi$ is state change rules, $T$ is trusted users and $w$ is the safety query. For the same access control scheme, a forensic analysis instance has the form of $\langle \iota, \gamma, \psi, q, \pi, L, T \rangle$. Note that current state $\gamma$, state change rules $\psi$, trusted users $T$ and query $q$ exist in the safe analysis tuple. We ask whether it is possible to find a mapping that maps every safety instance to the forensics instance. Forensic instance asks whether a query is true in the past state while safety instance ask if a query is true from current state to the future state.

### 2.3.1 HRU

We show that there exist a mapping from a safety instance to a forensic instance. And forensic instance is true if an only if the corresponding safety instance is true. Since safety analysis for HRU is undecidable[13], forensic analysis is undecidable as well. The theorem is illustrated as following:

**Theorem 3.** *Forensic analysis for HRU, in which $\pi = possible$ and $L = \emptyset$, is undecidable.*

The proof for the above theorem is a reduction from the problem of safety instance $\langle \alpha, \psi, T, w \rangle$ where $\alpha$ for HRU, in which the primitive operations are as following:

- **enter** r into $M_{so}$
- **delete** r from $M_{so}$
- **create subject** $s$
- **create subject** $s$

In short, the primitive operations that destroy subjects and objects are absent in the access control scheme. This restricted version of safety analysis is also undecidable for following reasons:

1. Previous work [15] does not use any primitive operations that destroy subjects and objects.
2. Previous work does not use set of trusted users $T$ to prove the undecidability of safety analysis.
3. If the input query $w$ is true for current state $\alpha$, safety instance is evaluated to true. Thus eliminating this query does not affect undecidability.

The forensic instance we reduced to is a restricted version of forensic instance where $\pi = possible$, $q = w$ i.e forensic query is equivalent to safety analysis query and query $q$ is not true for initial state $\iota$ and current state $\gamma$.

To maintain generality, forensic query $q$ should be false for initial state $\iota$ and current state $\psi$, otherwise the forensic will constantly evaluate to true. Then, we observer that forensic instance $\langle \iota, \gamma, \psi, q = (s^*, o^*, r^*), \pi = Possible, L = \emptyset, T = \emptyset \rangle$ is true if and only true if following two conditions hold:

**Condition 1** $\exists$ a state $\gamma_1$ such that $r^* \in M_{\gamma_1}[s^*, o^*]$ and $\iota \rightarrow^*_{\psi} \gamma_1$

**Condition 2** $\exists$ a state $\gamma_2$ such that $r^* \notin M_{\gamma_2}[s^*, o^*]$, and $\gamma_1 \rightarrow_{\psi}^{*} \gamma_2$, and $\gamma_2 \rightarrow_{\psi}^{*} \gamma$

**Condition 1** says that there must exist a state $\gamma_1$ for initial state $\iota$ to transfer into so that $s^*$ hold right $r^*$ over $o^*$ in that state $\gamma_1$. This is because in initial state $q = (s^*, o^*, r^*)$ is false as defined. **Condition 2** says that there must exist another state $\gamma_2$ for $\gamma_1$ to transfer into so that $s^*$ does not hold right $r^*$ over $o^*$ in that state. This is because $q = (s^*, o^*, r^*)$ is false for current state $\gamma$. Only if both **Condition 1** and **Condition 2** hold, the forensic instance will be evaluated to true.

We map the safety analysis to **Condition 1** and choose current state $\gamma$ so that if **Condition 1** is satisfied, **Condition 2** is also satisfied. This is done by defining appropriate HRU commands. To summarize, we have following safety analysis instance and forensic analysis instance:

- Safety instance $\langle \alpha_s, \psi_s, T_s = \emptyset, w_s = (s^*, o^*, r^*) \rangle$
- Forensic instance $\langle \iota_f, \gamma_f, \psi_f, q_f = (s^*, o^*, r^*), \pi = \mathit{Possible}, L_f = \emptyset, T_f = \emptyset \rangle$

In the forensic instance, $\iota_f$, $\gamma_f$ and $\psi_f$ are defined as follows:

- $\iota_f = \alpha_s \cup s_{new} \cup o_{new}$ $\iota_f$ is $\alpha_s$ with an extra subject $s_{new}$ and an extra object $o_{new}$
- $\gamma_f = \{s_{new}, o_{new}\}$ i.e current state in forensic analysis only contains subject $s_{new}$ and object $o_{new}$ where $r_{new} \in [s_{new}, o_{new}]$
- $R_{\psi_f} = R_{\psi_s} \cup \{r_{new}\}$ and $C_{\psi_f} = C_{\psi_s} \cup \{\mathit{Remove}_{r^*}, \mathit{Destroy}_s, \mathit{Destroy}_o\}$.

The extra commands we defined for $C_{\psi_f}$ requires some explanation. $\mathit{Remove}_{r^*}$ checks if $r^* \in [s_1, o_1]$ and if yes, it removes it and adds $r_{new}$ to $[s_2, o_2]$. This command is used in **Condition 1** to move the right $r^*$ from access matrix $M_{\gamma_1}[s^*, o^*]$ to $M_{\gamma_1}[s^*, o^*]$. $\mathit{Destroy}_s$ destroys subject and $\mathit{Destroy}_o$ destroys object. These three commands introduced work together to ensure that $\mathit{Condition}1 \rightarrow \mathit{Condition}2$. That is, if **Condition 1** is satisfied, then **Condition 2** is satisfied. An algorithm is designed to achieve this:

**Algorithm 1:** Algorithm to reach final state $\gamma$ from **Condition 1**

```
   Data: Current state $\gamma_1$, where $r_* \in [s_*, o_*]$
   Result: Finial state $\gamma_f$,where $\gamma_f = \{s_{new}, o_{new}\}$
1  initialization;
2  if $r_* \in [s_*, o_*]$ then $Remove_{r^*}([s_*, o_*], [s_{new}, o_{new}])$ ;
3  foreach $s \in S \setminus s_{new}$ do
4      $Destroy_s$
5  end
6  foreach $o \in O \setminus o_{new}$ do
7      $Destroy_o$
8  end
```

With this mapping, input safety analysis maps to **Condition 1**. We have

- $Condition1 \rightarrow Condition2$
- $Condition1 \land Condition2 \iff Forensic\ instance$

At this point, if we can prove that our safety instance is true if and only if forensic instance satisfies **Condition 1**. We have a reduction from safety analysis to forensic analysis. The bidirectional proof is as follows:

1. From safety instance to **Condition 1**: Assume safety instance is evaluated to true, this means $\exists$ a state $\beta$, such that in state $\beta$, $r_* \in [s_*, o_*]$ And this state is reachable from current state $\alpha_s$. Since the initial state in forensic instance $\iota_f = \alpha_s \cup s_{new} \cup o_{new}$ and commands in forensic instance $C_{\psi_f} = C_{\psi_s} \cup \{Remove_{r^*}, Destroy_s, Destroy_o\}$ i.e $\alpha_s \subseteq \iota_f$ and $C_{\psi_s} \subseteq C_{\psi_f}$. This implies state $\beta$ can always be reached in forensics instance. Thus, if safety instance is true, condition 1 is satisfied.

2. From **Condition 1** to safety instance: In forensic instance, we introduced three new commands and an extra subject $s_{new}$, an extra object $o_{new}$ and right $r_{new}$. We argue that sequence required to get from initial state $\iota_f$ to condition 1 state $\gamma_1$ does not need to contain these three commands introduced i.e $C_{\iota \rightarrow \gamma} \subseteq C_{\psi_s}$. This is due to HRU commands check presence of a right for primitive operations. Thus removing subjects, objects is not useful for fulfilling conditions of commands in $C_{\psi_s}$. Also, we argue that moving $r_{new}$ is not beneficial for fulfilling conditions of commands as well because none of the commands in safety instance checks $r_{new}$ as a condition.

Therefore, if **Condition 1** can be satisfied in forensic instance, the corresponding safety analysis can be evaluated to true as well.

To summarize the proof above, we prove that Safety analysis is true and only true if **Condition 1** is true. **Condition 1** satisfies **Condition 2**, which altogether satisfies forensic instance. Hence, safety instance in HRU is true if and only if the forensic instance is true. We observe the reduction from safety analysis to forensic analysis. According to previous research that safety analysis is undecidable in HRU, forensic analysis with $\pi = possible$ and Ł $= \emptyset$ is undecidable.

### 2.3.2 ARBAC

To analyze computational complexity of forensic instance in ARBAC, we use the same proof strategy that reduces safety analysis to forensic analysis in HRU. We show there exist a mapping between safety analysis and forensic analysis for ARBAC. We prove that the mapping is bidirectional, that is safety analysis is true if and only if forensic analysis is true.

In the ARBAC model, the set of administrative roles is disjoint from the set of basic roles. Assume the former is denoted by $R_{admin} \in R$, and the latter by $R_{basic} \in R$.

**Theorem 4.** *Forensic analysis for ARBAC, in which $\pi =$ **possible**, $L = \emptyset$, $q = (u^*, r^*)$ is **PSPACE**-Complete.*

The proof for the above theorem follows:

The forensic instance for ARBAC we have is in the form of $\langle \iota, \gamma, \psi, q, \pi, L, T \rangle$ where $\iota$ is initial state, $\gamma$ is current state, $\gamma$ is state-change rules, $q = (u^*, r^*)$ is forensic query, $\pi = Possible$ is access type, $L = \emptyset$ is a empty log, $T = \emptyset$ is trusted users. Query $q$ is asking whether is is possible for user $u^*$ to be assigned to role $r^*$ in the past. We say that $q$ is not true in initial state $\iota$ and current state $\gamma$, otherwise the forensic instance is naturally evaluated to true. Similar to proof in HRU section 2.3.1, we observe that forensic instance is true and only true if following two conditions hold:

**Condition 1** $\exists$ a state $\gamma_1$ such that $\gamma_1 \vdash (u^*, r^*)$ and $\iota \rightarrow_{\psi}^{*} \gamma_1$

**Condition 2** $\exists$ a state $\gamma_2$ such that $\gamma_2 \not\vdash (u^*, r^*)$, and $\gamma_1 \rightarrow_{\psi}^{*} \gamma_2$, and $\gamma_2 \rightarrow_{\psi}^{*} \gamma$

**Condition 1** says that there must exist a state for initial state $\iota$ to transit into such that user $u^*$ is assigned to role $r^*$. **Condition 2** says that state $\gamma$ must transit into a state where $u^*$ is no longer assigned to role $r^*$. Because query to the system's initial state and current state is not true, **Condition 1** and **Condition 2** must hold. In order to evaluate to a true forensic instance, the system must transit into a state where the query is evaluated to true.

We reduce the problem from safety instance $\langle \alpha, \psi, T, w \rangle$ where query $w$ is not true for current state. We say that there is no loss of generality because if $w$ is true for current state, the safety instance naturally holds. The query $w$ is the same as query $q$ in forensic instance, that is, we are asking if it possible for user $u^*$ to be assigned to role $r^*$ in the future. We map this safety instance to **Condition 1** described above. Then we choose the current state $\gamma$ for forensic instance such that if **Condition 1** is satisfied, then **Condition 2** is satisfied. This is done by choosing appropriate *can_assign* and *can_revoke* rules. We give a formal definition of the forensic instance and safety instance:

- Safety instance $\langle \alpha_s = \langle UA_s \rangle, \psi_s = \langle \mathit{can\_assign}_s, \mathit{can\_revoke}_s \rangle, T_s = \emptyset, w_s = (u^*, r^*) \rangle$
- Forensic instance $\langle \iota_f, \gamma_f, \psi_f, q_f = w_s = (u^*, r^*), \pi = \mathit{Possible}, L_f = \emptyset, T_f = \emptyset \rangle$

In the forensic instance $\iota_f$,$\gamma_f$ and $\psi_f$ is mapped from safety instance as follows:

- We introduce a new user $u_{new}$ and a new role $r_{new}$
- $\iota_f = UA_s \cup \{u_{new}, r_{new}\}$
- $\gamma_f = UA_s \cup \{u_{new}, r_{new}\} \cup \{(u^*, r) : r \in R_{basic}, r \neq r^*\}$ i.e. $u^*$ is a member of all basic roles except $r^*$
- $\mathit{can\_assign}_f = \mathit{can\_assign}_s \cup \{(r_{new}, r^*, r) : r \neq r^*\}$
- $\mathit{can\_revoke}_f = \mathit{can\_revoke}_s \cup \{r_{new}, r^*\}$

With these mapping, if **Condition 1** holds, then **Condition 2** is satisfied. This is because once $u^*$ is assigned to $r^*$ n state $\gamma$, $u_{new}$ can revoke $u^*$'s membership from $r^*$. Thus if **Condition 1** holds, **Condition 2** also holds. Now we show that safety instance can be reduced to **Condition 1**, that is, safety instance is true if and only if **Condition 1** is true. The bidirectional proof is as follows:

1. From safety instance to **Condition 1**: Assume safety instance is true, then it is possible that user $u^*$ to be assigned to role $r^*$ give current state $\alpha_s$ and state change rule $\psi_s$. Since $\iota_f$ is a superset of $\alpha_s$ and $\psi_f$ is a superset of $\psi_s$, **Condition 1** must hold. That is it is possible for user $u^*$ to be assigned to role $r^*$ in the past.

2. From **Condition 1** to safety instance. Assume **Condition 1** holds, we need to argue that the changes we made to state change rule do not affect whether **Condition 1** holds. This will in turn show that if **Condition 1** holds then safety instance is true. In addition to the *can_assign* rule in the $can\_assign_s$, we allow for $u_{new}$ to assign any basic role to any user, provided the user is a member of $r^*$. If this rule is exercised for any user other than $u^*$, it will not affect $u^*$'s role membership in any way. If this rule is exercised for $u^*$, the precondition ensures that $u^*$ must already be a member of $r^*$. Hence, the addition of *can_assign* rule does not affect the result of safety analysis. In addition to the *can_revoke* rule in the $can\_revoke_s$ set, we allow for $r_{new}$ to revoke any user who is member of $r^*$. Again, to exercise this rule on user $u^*$, $u^*$ must already be a member of $r^*$. We prove that if textbfCondition 1 holds, the safety instance in the mapping also holds.

Since we prove that Safety instance is true if and only if **Condition 1** is true. From the obeservation that **Condition 1** satisfies **Condtion 2** and $Condition1 \wedge Condition2$ satisfies forensic instance. We furtuher prove that the safety instance is true if and only if forensic instance is true. Since the computational complexity of safety instance is PSPACE-Complete. We infer that the forensic instance is also PSPACE-Complete.

# Chapter 3

# Case Study : Hello-retail on AWS

In this chapter, we analyze access control system on AWS. AWS adapts an RBAC-like access control scheme to control resources used within AWS. We deploy an cloud application that uses several AWS resources like Simple Storage Service (S3) (File Storage) and DynamoDB (Database) in AWS. We simulate an security incident and perform forensics analysis based on logs generated by AWS and the cloud application. We show that we are able to generate useful forensic result based on the evidence we captured. Finally, we propose a object-directed logging scheme that eliminate redundancy in default AWS logs.

## 3.1 AWS IAM

IAM(Identity and Access management) is an AWS component that is used to authenticate and authorize AWS resources. The IAM is AWS' implementation of a Role-based access control system. Administrator can use AWS IAM to manage access to AWS services and resources. By creating IAM roles for different users. Administrator can grant different privilege to users depending on their service requirements. In the AWS IAM, IAM users are assigned to IAM roles, and IAM roles are associated with IAM policies. IAM policies define an identity's access to the AWS resources. An IAM role may be associated with multiple policies, and an IAM user may be assigned to multiple IAM roles. IAM user, IAM role and IAM policy correspond to the user, role and permission in the RBAC model described in the paper [26]. Figure 3.1 illustrated how IAM secures user's access to the resources.

AWS IAM's access does not strictly follow the definition of $RBAC_0$ scheme in [26]. In the access control model Ravi et al. defined, roles are associated with permissions and no user

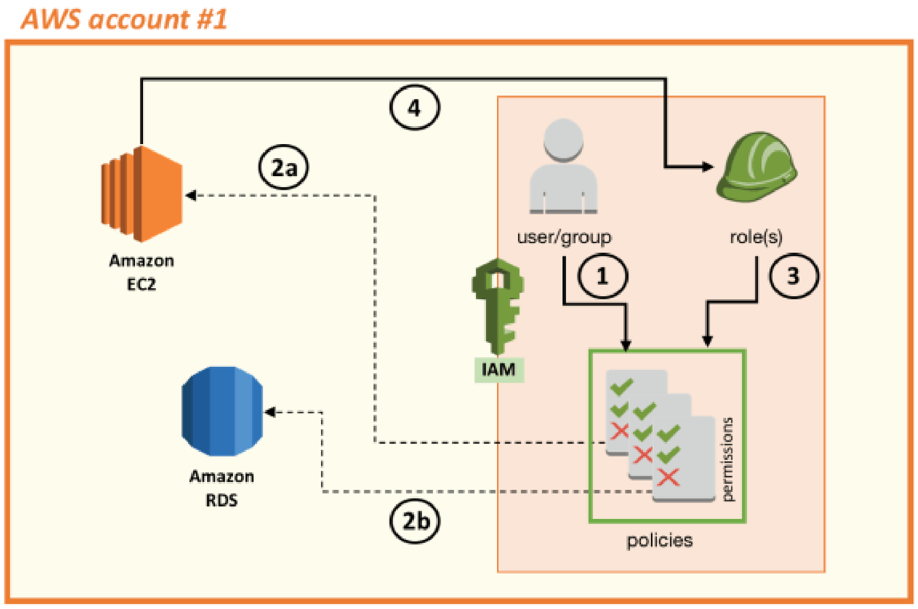


Figure 3.1: Example of AWS IAM. The Amazon EC2 computing instance is assumed as a role in IAM. IAM use the access policy defined in that role to decide whether the EC2 computing instance has access to Amazon RDS database.

can be assigned to a permission directly. However, in AWS IAM's access control model, IAM user can be associated to a policy directly. In the figure 3.1, users and roles are attached to the same policy. The policy is examined to authenticate users' requests to the AWS resources. Note that another type of identity, IAM group, is described in the figure. IAM group is just a collection of IAM users with same policy attached. It has nothing to do with RBAC scheme. Thus, without loss of generality, we do not discuss IAM groups in this thesis.

If an AWS service has an intention to access AWS resource, the resource request has to be authenticated by IAM as well. AWS service can be assumed with an IAM service role. Service role is associated with AWS services rather than real users. In that regard, AWS service behaves like a IAM user and is associated with an IAM role. In the figure 3.1, if a virtual machine in Amazon EC2(Amazon Elastic Compute Cloud, an cloud virtual machine service) wants to access a table in Amazon RDS(Amazon Relational Database Service), it can be assumed with an IAM role, `AllowDBAccess`, so the IAM will approve virtual machine's request to the database. An example of a policy is listed below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": ["arn:aws:s3:::bucket1"]
        }
    ]
}
```

The **action** field specifies the actions to be allowed. Each AWS resource has its own set of actions to be performed on it. `ListBucket` is an action to be performed on AWS S3, a file storage service, to list names of all files in the storage bucket. The **Resource** field specifies targeted resource. The **Effect** field can be 'Allow' or 'Deny' to either allow or deny the action performed on the resource. The policy can be associated to a user or a role. For example, if the policy above is associated with user Alice, Alice will have access to list all objects in **bucket1** storage buckets in AWS S3.

### 3.1.1 IAM Logs

AWS CloudTrail logs activities across AWS infrastructure. The logs are in JSON format and are delivered into a S3 storage bucket. CloudTrail characterizes events into data events and management events[3]. Data events log read/write access to objects in S3 storage and databases in DynamoDB. Management events log all control plane operations in a AWS account. For instance, accessing a table in AWS DynamoDB would be logged as a data event, while creating a new IAM role would be logged as a management event. An example of a management event is shown below, where Alice adds a new user Bob in the IAM system. All changes made to IAM are logged by CloudTrail. In the case-study, we use logs generated by AWS CloudTrail as the source of evidence to perform forensic analysis.

```
{"Records": [{
    "eventVersion": "1.0",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-03-24T21:11:59Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "CreateUser",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-cli/1.3.2 Python/2.7.5 Windows/7",
    "requestParameters": {"userName": "Bob"},
    "responseElements": {"user": {
        "createDate": "Mar 24, 2014 9:11:59 PM",
        "userName": "Bob",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "path": "/",
        "userId": "EXAMPLEUSERID"
    }}
}]}
```

## 3.2 Hello-retail

Hello-retail is an open-source event-sourcing application developed by Nordstrom team. The typical use case for Hello-retail would be for an e-commerce company to request the images for their product from the photographers. When items are added into the product databases, hello-retail application sends a text message to photographers asking for product images. Photographers reply with images via MMS(Multimedia Messaging Service). Application then associates images with products and displays on the website. The source code of Hello-retail can be acquired at https://github.com/Nordstrom/hello-retail.

Hello-retail is a serverless, event-driven application, meaning there is no server hosting the application. A serverless application runs in stateless compute containers that are event-triggered, ephemeral, and fully managed by the cloud provider. A typical application requires provision of the computing resources. A developer compiles the application's source code into the binary executable and deploys the executable on a server. With the serverless execution model, the cloud service manages allocation and provisioning of the servers. For example, to deploy hello-retail on AWS, there is no need to allocate a Linux Virtual Machine in AWS EC2 and host the application on the virtual machine. Developer uploads source code to AWS Lambda and Lambda takes care of assigning computing resources to run the code. The front-end interface is a website hosted on Amazon CloudFront, a content delivery network service. The back-end consists of multiple AWS Lambda functions. The block diagram of hello-retail application is illustrated in 3.2.

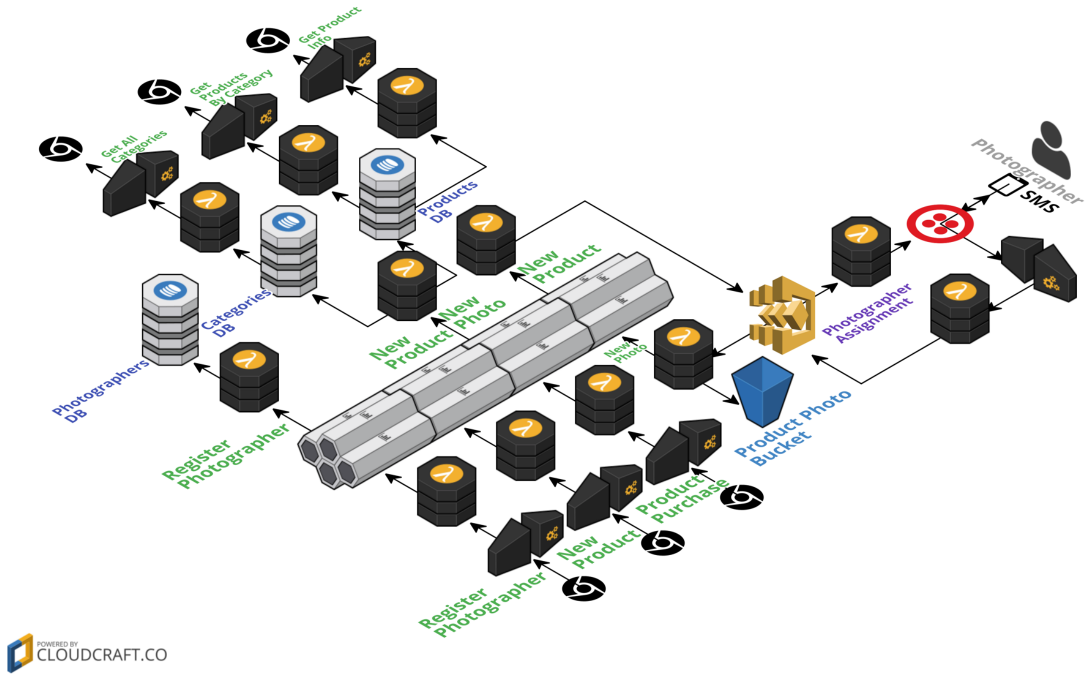


Figure 3.2: Hello-retail block diagram

The hello-retail application consists of 14 individual components. When the application is deployed onto AWS, each of the components run as a Lambda function. Depending on the purpose, lambda function may have access to other AWS resources. For instance, the Lambda application `hello-retail-product-photos-receive` which receives image from photographer requires access to AWS S3 to store the image file in the image file bucket. AWS S3 is an object storage service that stores files in the Internet. Another Lambda application `hello-retail-product-catalog-api` requires access to the AWS

DynamoDB database to write product information into the tables. DynamoDB is a non-relational database service hosted by AWS in the cloud. Lambda function may be assumed into a service role, and IAM may grant access to the resources based on the policy attached to the service role.

Table 3.1 gives a detailed explanation on purpose of all Lambda functions used in Hello-retail application. Each function is composed as a *Node.js* application. The source code is then uploaded to AWS Lambda to be deployed.

| **Lambda Function** | **Description** | **Service Role** | **Resources Used** |
|---|---|---|---|
| cart-builder | Build shopping cart | CartBuilder | Amazon DynamoDB<br>Amazon Kinesis |
| cart-api | An API for web-page to call | CartApi | None |
| event-writer | Write Event to Kinesis stream | EventWriter | Amazon Kinesis |
| product-catalog-builder | Add product into database | ProductCatalogBuilder | Amazon DynamoDB<br>Amazon Kinesis |
| product-catalog-api | An API for web-page to call | ProductCatalogApi | None |
| product-photo-processor | Publish new photo event to other function | ProductPhotoProcessor | Amazon DynamoDB<br>Amazon Kinesis |
| product-photo-assign | Assign task to available photographer | ProductPhotoAssign | Amazon DynamoDB |
| product-photo-message | Send text message to photographer | ProductPhotoMessage | None |
| product-photo-recrod | Record task assignment in database | ProductPhotoRecord | Amazon DynamoDB |
| product-photo-receive | Receive product image from photographer | ProductPhotoReceive | Amazon DynamoDB<br>Amazon S3 |

|  |  |  |  |
|---|---|---|---|
| product-photo-fail | Handle event when photographer fails to take photo | ProductPhotoFail | Amazon DynamoDB |
| product-photo-success | Handle event when image taken successfully | ProductPhotoSuccess | None |
| product-photo-unmessage | Remove task assignment from database | ProductPhotoUnmessage | Amazon DynamoDB |
| product-photo-report | Associate image with the product in database | ProductPhotoReport | Amazon DynamoDB<br>Amazon Kinesis |

Table 3.1: Hello-retail Lambda functions explained

In a serverless architecture, computing and data storage are isolated. When Lambda functions are executed, a ephemeral virtual machine is spawned so code can be run inside the virtual machine. When the code finishes executing, the ephemeral virtual machine is destroyed. This means persistent data generated by Lambda function needs to be stored. Files need to be stored in AWS S3. Other information may need to be stored in a database such as AWS DynamoDB. In hello-retail application, `product-photo-receive` function store the images it receives from photographers into the S3 image bucket, thus it needs to have write permission to AWS S3. Such access request needs to be mediated by AWS IAM. As an RBAC scheme, IAM determine whether to authorize the request based on policy. Since Lambda function is not a IAM identity, it has to be assumed into a service role. The service role defines what kind of access it has by associating itself with policies. As an example, **Product-photo-receive** will have following policies attached to it to access S3 and DynamoDB:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:ExampleDataBase",
            "Effect": "Allow"
        }
    ]
}
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:putObject"
            ],
            "Resource": "arn:aws:s3:::ImageBucket",
            "Effect": "Allow"
        }
    ]
}
```

Figure 3.3 illustrate a typical work flow of hello-retail application running on the AWS. An Inventory Manager logs into the website and register a new product. Underlying the system, `Product-catalog-api` function is triggered. It then triggers `Product-catalog-builder`, where product information is added into 'Product DB' database. Then `Product-photo-assign` function is called and it finds an available photographer from the 'Photographer DB' database and request picture of product from him. The event-sourcing process is done by sending a text message to photographer's phone and expecting an image through photographer's reply. `Product-photo-receive` store the image into a S3 data bucket. `Product-photo-success` then associates image with the product in the 'Product DB' database and displays the image in the website.

Figure 3.4 shows a screenshot of product registration component of hello-retail application. Inventory manager can use this component to register new product into the application. Figure 3.5 shows screenshot of phone registration component. Photographers register themselves on this page. When new product is available in the database, photographers will be assigned to take pictures of items. Figure 3.6 shows screenshot of an item with picture in hello-retail. The description of product is entered by Inventory manager and picture of the product is sourced from photographers.

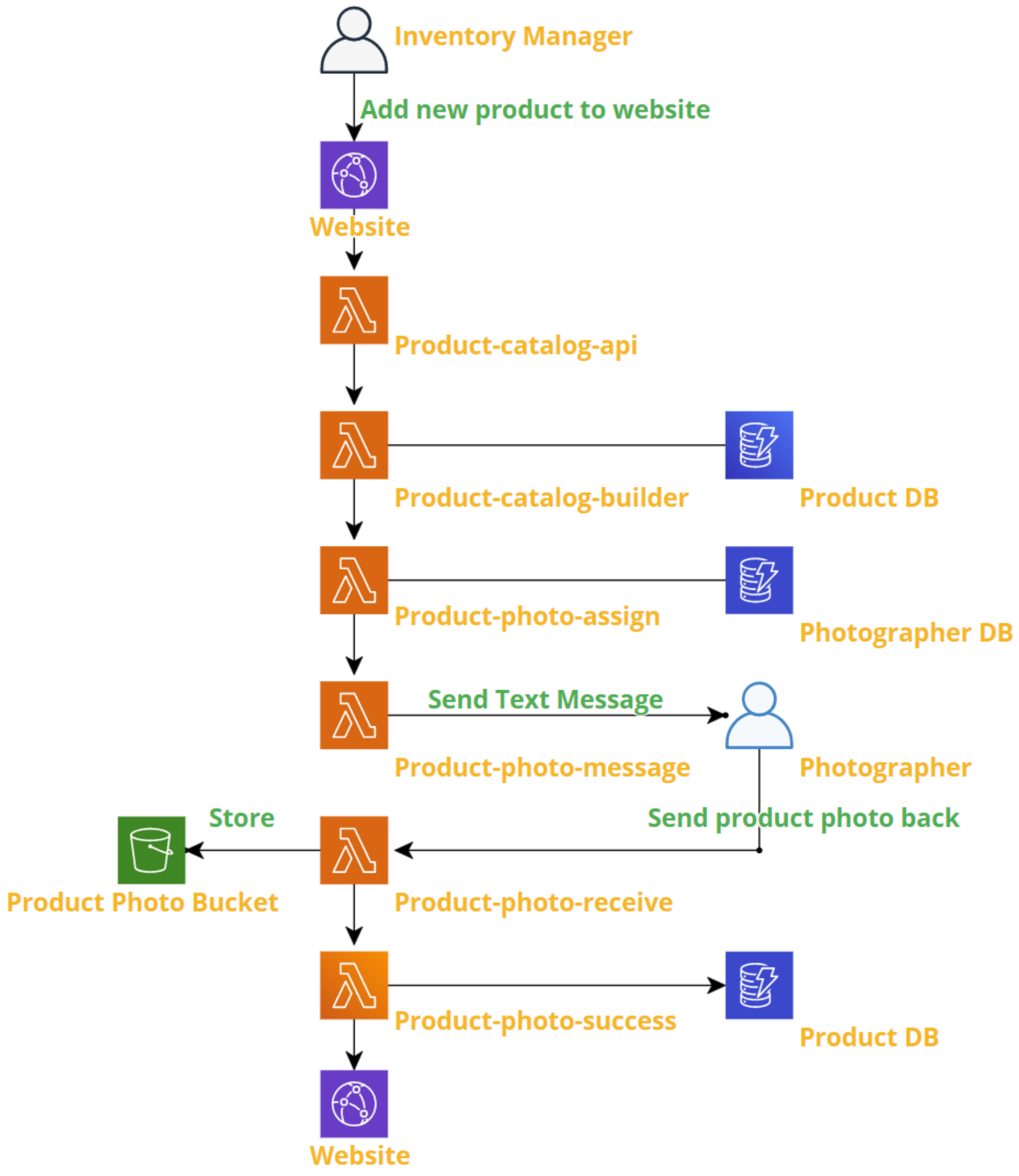


Figure 3.3: Workflow of Hello-retail application

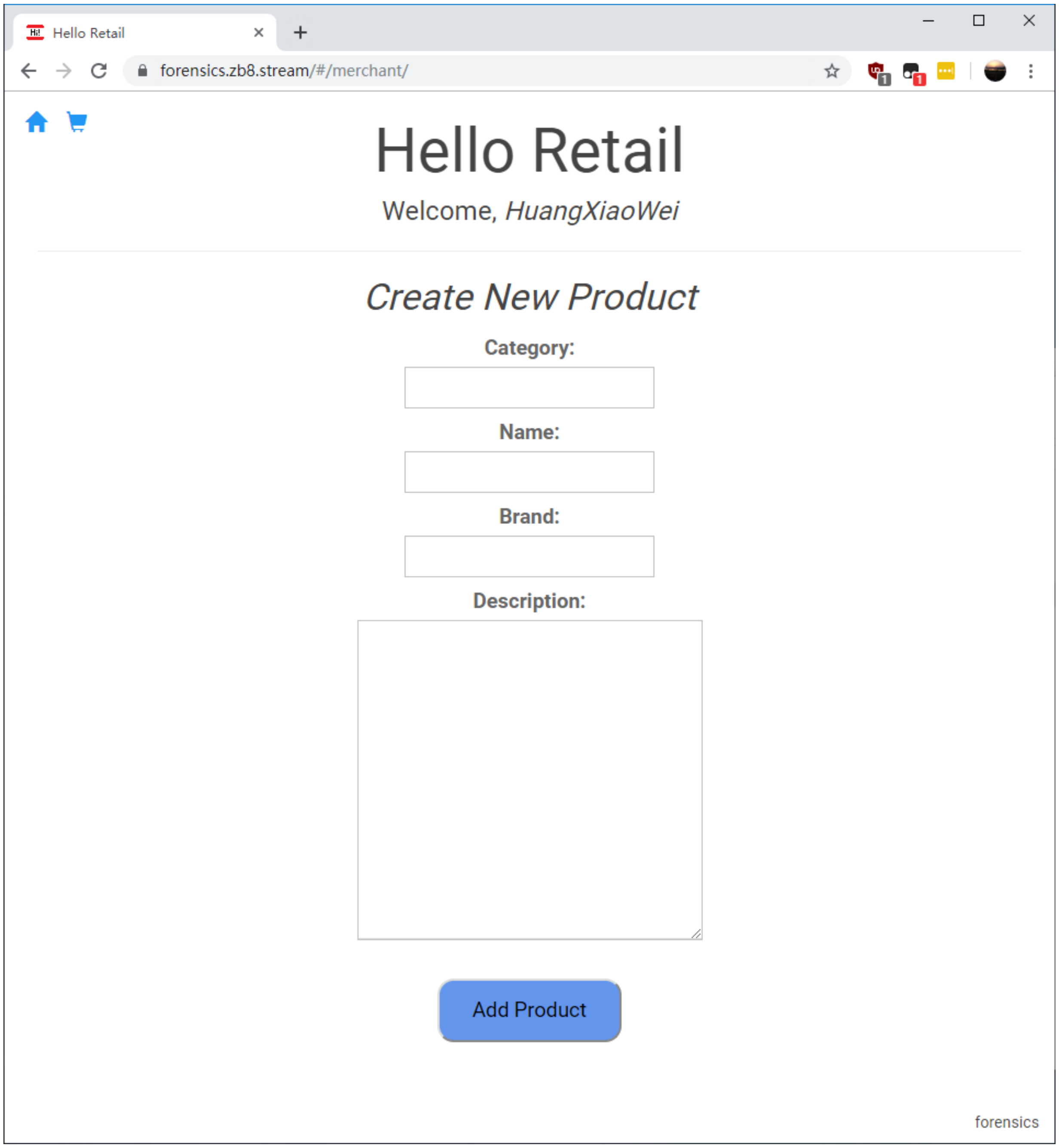
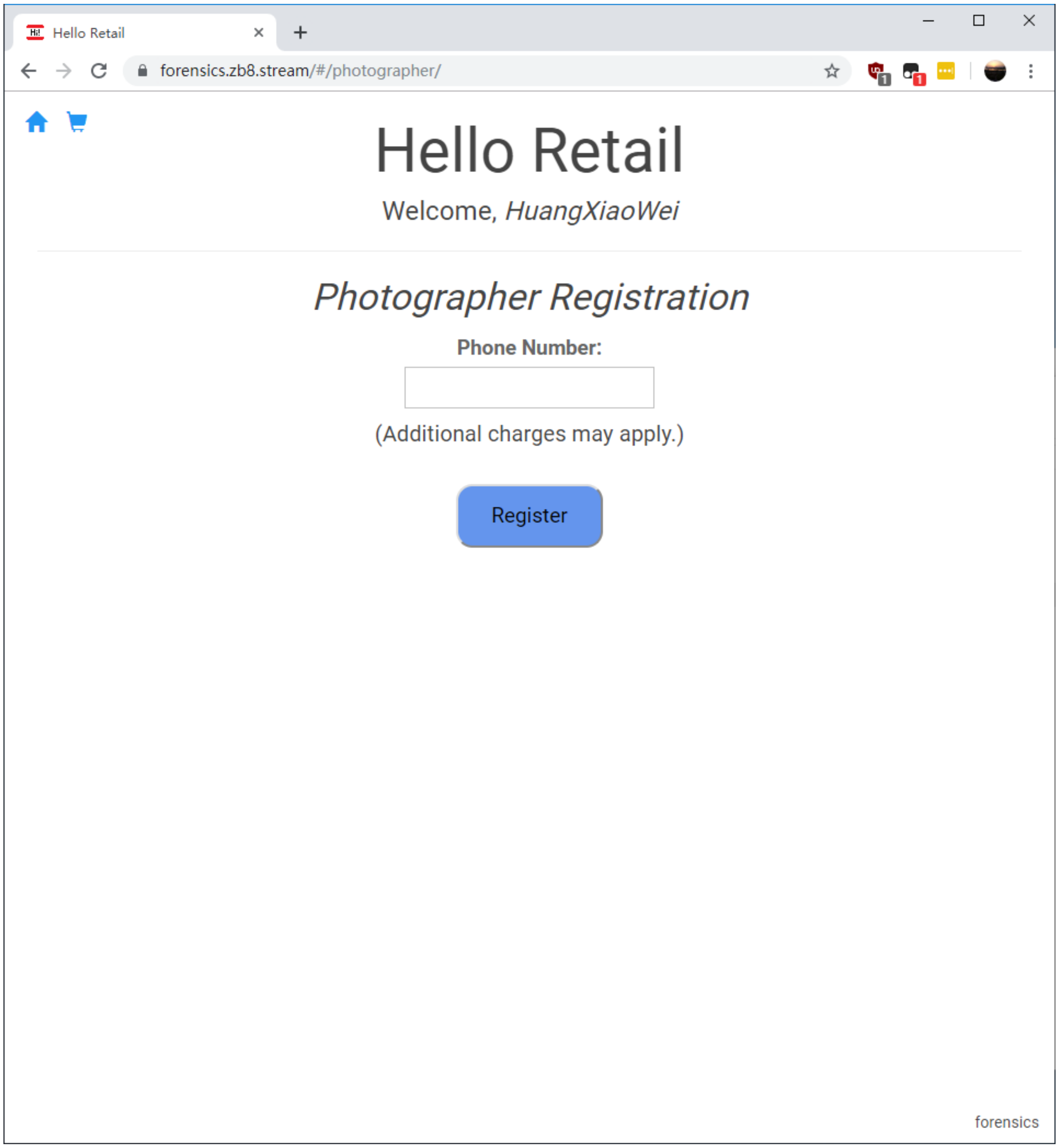


Figure 3.4: Screenshot of Hello-retail application



Hello Retail

Welcome, *HuangXiaoWei*

*Photographer Registration*

**Phone Number:**

(Additional charges may apply.)

forensics

Figure 3.5: Screenshot of Hello-retail application

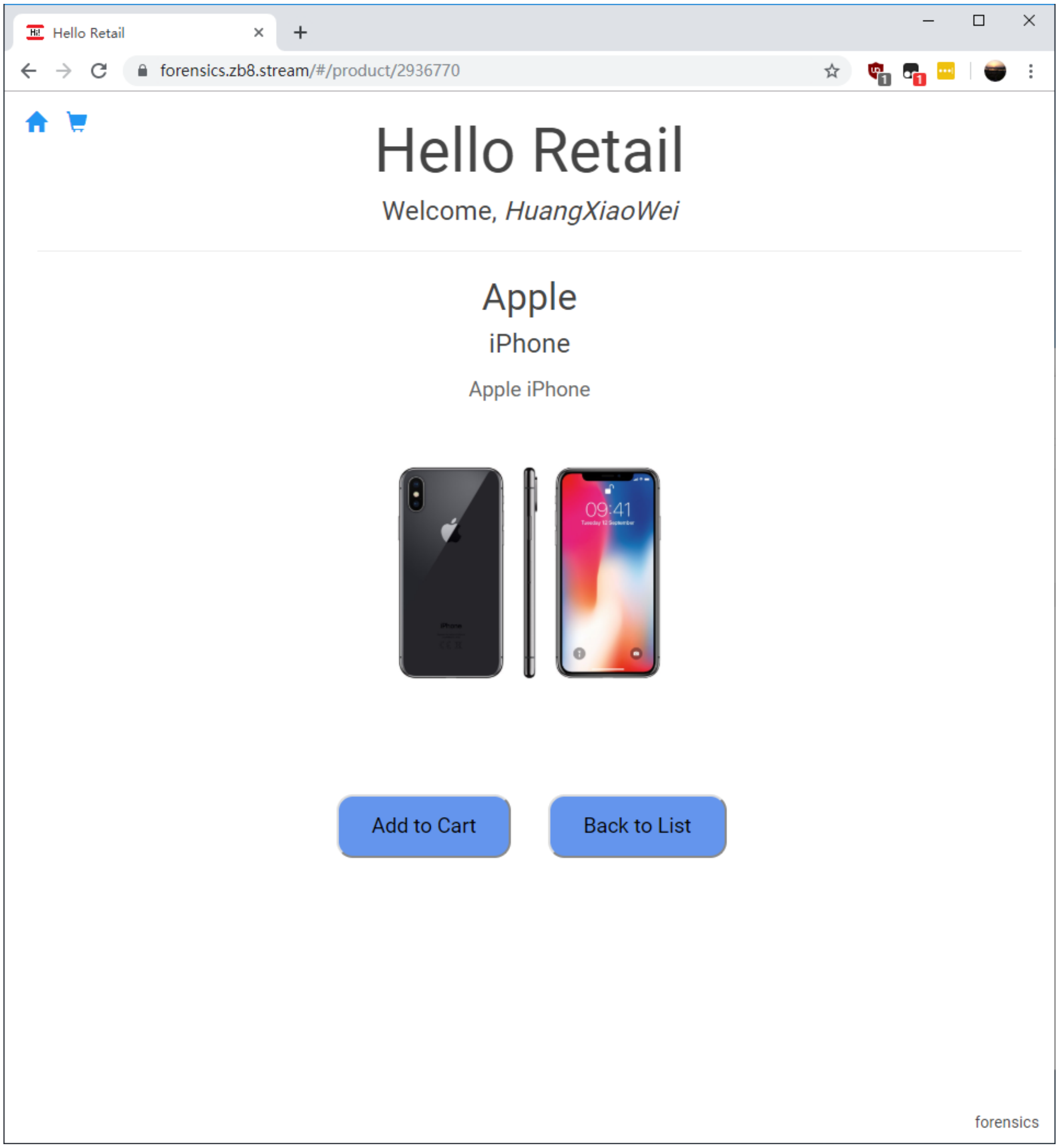


Figure 3.6: Screenshot of Hello-retail application

## 3.3 Forensic Analysis

One way to attack an ordinary application would be exploiting its underlying infrastructure through operating system vulnerabilities. An attacker will be able to stole application data if he gains access to the underlying server. Unlike ordinary application which requires a infrastructure to be deployed, serverless application runs on ephemeral virtual machines, meaning the attack surface on the infrastructure is relatively small. However, a malicious user may still be able to attack hello-retail application even it is serverless. In the hello-retail application, there are 14 different AWS Lambda functions serving as a back-end framework of the application. If a malicious user is granted privileges to modify the code of Lambda functions, the expected behaviour of the application might be altered. Multiple methods were discovered to perform a privilege escalation on Amazon Web Services [12]. For this case-study, we make an assumption that a malicious user is granted privilege to alter existing lambda functions. We perform forensic analysis based on the fact that lambda functions has been modified.

Here is the explanation to the security incident we simulated. We propose a scenario that Product MacBook and Product iPhone has been added to the hello-retail application already. Photographer Alice is assigned to take picture for a MacBook, while Bob is assigned to take picture for an iPhone. Alice and Bob took photo of an MacBook and an iPhone respectively and send them back. However, in the front-end website, an iPhone is shown on both products' page, while the expected behavior is a MacBook on product MacBook's page and an iPhone on product iPhone's page. This leads to an incident:

> Photo on the website for the MacBook is different from the product's actual photo.

While the truth of this incident is:

> A malicious user altered the code in $\Lambda\mathit{Receive}$ function, which caused photo of an iPhone to be displayed on Product MacBook's webpage

| Product | MacBook | iPhone |
|---|---|---|
| Assigned photographer | Alice | Bob |
| Photo taken by Photographer | A MacBook | An iPhone |
| Photo showing on Website | iPhone | iPhone |

The forensic analysis is performed solely based on the access control scheme and logs available. For the logs, we assume the CloudTrail logging has been enabled. The CloudTrail logs capture all management events in the Hello-retail application.

Product image is taken by assigned photographer. If a product's photo went wrong, naturally photographer may be considered as a suspect. So the investigator may be interested if it is photographer who performs the wrong task. So the first step of forensics is to either ensure photographer is the culprit or to exonerate the photographer. If the photographer is exonerated, we check whether the code of Lambda function has been modified by comparing their result with expected behaviour. The algorithm below illustrated the step-by-step analysis investigator may perform to track down the suspect.

**Algorithm 2:** Investigative process to narrow down suspect

```
Check logs of product-photo-receive;
if Photo sent by Photographer == iPhone Photo then
    Photographer is suspect;
    Check product-photo-message;
    if Proper text message sent to photographer then
        Photographer is suspect;
        Check product-photo-assign;
        if Correct assignment is made between product and photographer then
            Photographer is suspect;
            Lambda functions are not compromised;
            EXIT;
        else
            Exonerate Photographer;
            product-photo-assign may be altered by malicious user;
            EXIT;
        end
    else
        product-photo-message may be altered by malicious user;
        Exonerate Photographer;
        EXIT;
    end
else
    Exonerate Photographer;
    Check product-photo-receive Again;
    if Photo sent by Photographer != Photo stored in S3 then
        product-photo-receive may be altered by malicious user;
        EXIT;
    else
        Exonerate product-photo-receive;
        Check product-photo-success;
        if Photo in success event != Photo sent by Photographer then
            product-photo-success may be altered by malicious user;
            EXIT;
        else
            Exonerate product-photo-success;
            product-photo-success may be altered by malicious user;
            EXIT;
        end
    end
end
```

While the algorithm above seems sound to track down the suspect in the post incident analysis, realistically the result of it is unanswerable. Firstly, the above logic is constructed based on the fact that investigator has knowledge of the detailed implementation of all Lambda functions. If investigator does not know purpose of each Lambda function, he can not construct a logic tree to exonerate photographer first and narrow down to specific Lambda functions. Secondly, performing algorithm above requires the access to extensive amount of logs generated by Lambda functions. For instance, to validate *Proper text message sent to photographer* in Line 5 above, we would need to log all messages sent out to photographers by `product-photo-message` function. If such log is not available, the result of the investigative process above is unanswerable. Indeed, AWS does not log activity logs required to answer this question[3]. This means we are not able to answer type iii forensic questions(exercise of an access) posed in subsection 2.2.1.

Even in a system where logs to answer exercise of an access question are comprehensive, the sheer volume of the logs makes it difficult to perform forensic analysis[5], known as *quantity problem*.

A different approach to the forensic analysis focus on a system's access control scheme and its state change logs. We notice that in Table 3.1, only `product-photo-receive` assumes a role that have access to the photo S3 bucket. The modified Lambda function must have access to the S3 bucket in order to change the photo assignment. To investigate which lambda application has access to the photo bucket, we pose forensic queries to the access control model.

For the hello-retail application, assume we have no knowledge of the purpose of each Lambda functions in Table 3.1. An investigator may ask which component have access to the AWS S3 image storage bucket. This falls into the forensic queries where $\pi = Actual$, that is, we are asking whether a role actually holds privilege to access to the resources. We generate 14 forensic queries for hello-retail program, listed below:

- $\langle I, \gamma, \psi, q = (CartBuilder, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (CartApi, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (EventWriter, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductCatalogBuilder, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductCatalogApi, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoProcessor, AmazonS3Access), \pi = Actual, L\rangle$

- $\langle I, \gamma, \psi, q = (ProductPhotoAssign, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoMessage, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoRecord, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoReceive, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoFail, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoSuccess, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoUnmessage, AmazonS3Access), \pi = Actual, L\rangle$
- $\langle I, \gamma, \psi, q = (ProductPhotoReport, AmazonS3Access), \pi = Actual, L\rangle$

In each of the queries, $I$ is the initial state of IAM access model, $\gamma$ is the current state of the model, $\psi$ is the state change rule, $q$ is the forensic query, $\pi$ is the access type and $L$ is the CloudTrail logs. Algorithm 3 illustrated below returns result of forensic instance to the forensic query. These 14 forensic problems are fed into Algorithm 3, and we get a input of either 'Yes' or 'No' form the result of algorithm. If output to a forensic query is 'Yes', then investigator know that such a role must have access the S3 resources. If output to a forensic query is 'No', then such an role can be exonerated in the investigation. Since algorithm 3 terminates in polynomial time, the whole forensic analysis problem terminates in polynomial time.

**Algorithm 3:** Algorithm to return result of forensic analysis problem.

```
  Input: An instance of the forensic analysis problem,
         $\langle I, \gamma, \psi, q = (r, p), \pi = Actual, L\rangle$
  Output: $result \in \{Yes, No, Unsure\}$
1 if $I \vdash q \lor \gamma \vdash q \lor L \vdash q$ then result = Yes;
2 else if Management events are enabled and complete after time-stamp of $q$ then
    result = No;
3 else results = Unsure;
4 return result
```

The python code that implements algorithm 3 is illustrated below.

```python
import os
import json

"""
Parse AWS cloudtrail log in json format
Usage: Put Uncompressed Cloud json logs in ./json folder
      Run python forensic_cloudtrail.py
"""

# Here is an example query
RoleName = 'ProductReceiveRole'
AccessType = 'S3ReadWriteAccess' # We are asking whether someone has access to
    S3 object storage
pi = 'Actual' # The access type we are asking is actual

def parse_cloudtrail():
    os.chdir("./json")
    record_count = 0
    createRoleEvent = []
    attachPolicyEvent = []
    detachPolicyEvent = []

    for json_file in os.listdir("."):
        if (os.path.splitext(json_file)[1] == ".json"):
            with open(json_file,"r") as f:
                log = json.loads(f.read())
                records = log.get("Records",[])
                record_count += len(records)
                for record in records:
                    eventName = record.get('eventName',"")
                    if (eventName == 'CreateRole'):
                        createRoleEvent.append(record)
                    if (eventName == 'AttachRolePolicy'):
                        attachPolicyEvent.append(record)
                    if (eventName == 'DetachRolePolicy'):
                        detachPolicyEvent.append(record)
    print(f"Total records: {record_count}")
    return (createRoleEvent,attachPolicyEvent,detachPolicyEvent)
```

```
def resolve_premission(arn,access):
    """
    An function that detect whether a given policy allows the access
    The detailed implementation is omitted here
    An example is given a policy arn:aws:iam::aws:policy/AdministratorAccess
    and access type S3ReadWriteAccess,
    This function should return true because Administrator has access to S3
       bucket
    For simplicity, in this code snippet, we always return true
    """
    true

if __name__ == "__main__":
    createRoleEvent,attachPolicyEvent,detachPolicyEvent = parse_cloudtrail()
    print(f"Forensics related records:
       {len(createRoleEvent)+len(attachPolicyEvent)+len(detachPolicyEvent)}")
    for event in attachPolicyEvent:
        role = event.get('requestParameters')['roleName']
        arn = event.get('requestParameters')['policyArn']
        if (role == RoleName):
            result = resolve_premission(arn,AccessType)
            if (result):
                print(event)
            sys.exit(0)
```

The python program takes CloudTrail logs as the input. The output of the python program is shown below. The program print the logs related to the forensic query. The logs show a new policy has been attached to *ProductPhotoReceive* role. And the policy has *PowerUserAccess*. By referring AWS document [2], *PowerUserAccess* policy indicates that user have read and write access to all S3 buckets.

```
{
   'eventVersion':'1.05',
   'userIdentity':{
      'type':'IAMUser',
      'principalId':'AIDAZ52W23FFWMVON5KIP',
      'arn':'arn:aws:iam::682545043787:user/xh_admin',
      'accountId':'682545043787',
      'accessKeyId':'ASIAJMNYFKPPMZCOZU7Q',
      'userName':'xh_admin',
      'sessionContext':{
         'attributes':{
            'mfaAuthenticated':'false',
            'creationDate':'2019-08-12T18:09:39Z'
         }
      },
      'invokedBy':'cloudformation.amazonaws.com'
   },
   'eventTime':'2019-08-12T18:09:51Z',
   'eventSource':'iam.amazonaws.com',
   'eventName':'AttachRolePolicy',
   'awsRegion':'us-east-1',
   'sourceIPAddress':'cloudformation.amazonaws.com',
   'userAgent':'cloudformation.amazonaws.com',
   'requestParameters':{
      'roleName':'forensicsReceiveRole1',
      'policyArn':'arn:aws:iam::aws:policy/PowerUserAccess'
   },
   'responseElements':None,
   'requestID':'60ff859b-bd2c-11e9-bd67-75e78d780a59',
   'eventID':'37076e62-ac3e-4af0-a873-86dc47271b93',
   'eventType':'AwsApiCall',
   'recipientAccountId':'682545043787'
}
```

As $\langle I, \gamma, \psi, q = (ProductPhotoReceive, AmazonS3Access), \pi = Actual, L\rangle$ evaluates to true, we confirm that `product-photo-receive` might be potential suspect of the incident. Since this Lambda function is the only once that have access to the S3 storage bucket. The code of the Lambda function may be altered by malicious user to generate a undesired behaviour.

## 3.4 Evaluation

From the section above, we show that the logging capability of AWS IAM are not sufficient to answer all the forensic questions. Specifically, the CloudTrail log cannot answer "exercise of privilege question" questions in subsection 2.2.1 due to the lack of activity logs. However, we are able to pose "Actual Possession" questions and answer them using CloudTrail logs.

| | Total Log Entry | Log Entry Related to Forensics |
|---|---|---|
| Light-usage scenario | 4491 | 47 |
| Heavy-usage scenario | 23331 | 57 |

Table 3.2: Logs generated for Hello-retail application

Furthermore, we are able to solve them in polynomial time given sufficient CloudTrail logs. With logs presented in the forensic instance, we are able to efficiently generate answers to the forensic questions.

In our test scenario, AWS CloudTrail logs all state changes happened in the access control system. We note that only a small portion of the raw logs collected are related to forensics queries of our interest. As the users and time grow, the sheer size of raw logs make it difficult to analyze those logs. We present an algorithm that runs in linear time to extract information related to forensic analysis forom the CloudTrail logs. However as the size of the logs grows, the execution time of that algorithm will grow as well.

Table 3.2 gives a benchmark on size of logs used to perform forensic analysis. We perform the forensic analysis under multiple scenarios. In each of the scenario, we deploy hello-retail application to AWS, add products and registered photographers to the application's database. We then simulate a security incident discussed above by changing `Product-Photo-Receive`'s code and altering a item's photo in the photo S3 object bucket. In Scenario 1, we simulate a light use situation where only **1** user and **1** photographer is registered in the system. In Scenario 2, we simulate a heavy use case where there are **5** users and **5** photographers registered in the system. We then compare the amount of total logs generated by AWS and amount of logs used to narrow down the suspect. The raw data of the logs is hosted on-line and can be achieved by accessing `https://github.com/clive2000/aws_logs`. In Scenario 1 system generates 4491 log entries and 47 of them are related to actual forensic queries. That is, with the help of those 47 entries, we are able to generate the same forensic result compared to the full CloudTrail logs. In Scenario 2 system generates 23331 entries, where 57 of them are related to forensics. This is as expected as only portion of CloudTrail logs are related to IAM system. And we only collect evidence on changes made to IAM because the forensic analysis is performed in the scope of the access control system.

The result shows that default AWS logs have considerable redundancy. In a real system, the sheer volume of the CloudTrail logs may make it difficult for investigator to analyze the incident. We propose the idea of "Goal-directed logging", where we only record sufficient and necessary logs for forensic analysis.

## 3.5 Goal-directed Logging

We introduce the idea of goal-directed logging. In the forensic analysis example illustrated in section 3.3, we proved that investigators are able to perform forensic analysis if sufficient logs are available. However, AWS's logging feature logs unnecessary information related to forensic analysis. This is understandable since AWS logging feature is designed for auditing purposes. In the event of security incident, the sheer volume of logs makes it difficult for investigator to examine the logs. Although our algorithm in analyzing CloudTrail logs terminates in linear time, excessive amount of logs still makes forensics analysis time-consuming.

In forensic analysis, investigators focus on the components related to the security incident. In the event of a data breach in AWS, malicious attacker usually steals data from file storage system or databases. This leads to the idea of objective-directed logging. That is, the system only logs the event that investigator cares for forensics questions. For example, we can log just the state changes on the access control model. This will ensure the log $L$ contains all access change in a certain time range. We can further reduce the logs by only logging state changes related to critical users. For example, in hello-retail, if we believe only databases contains sensitive data, we only log IAM state changes related to read/write permission in AWS DynamoDB. An example algorithm is illustrated below.

**Algorithm 4:** Algorithm to generate minimal logs sufficient for forensic analysis. EventName $E$ is the action that is interested to investigators and Target Resource $R$ is the resources focused on. The reduced logs can be used to improve efficiency of forensic analysis in access control

```
  Input: EventName of interest $E$, Target Resource $R$, Logs $L$
  Output: Reduced logs $\tau \in \{*, NULL\}$
1 foreach Log Entry $l \in L$ do
2     if Event in $l$ matches $E$ and Target in $l$ matches $R$ then Add $l$ to $\tau$;
3 end
4 return $\tau$;
```

We set $EventName$ to be events related to access scheme updates. Through the algorithm above, we are able to generate log $l$ that is a subset of the raw CloudTrail logs. The log $l$ contains all related information in order to perform forensic analysis and is relative small. Take "Scenario 1" from Hello-retail case study as an example, the raw logs we acquired from AWS contains 4491 entries. Assume we use algorithm to generate a log that only contains 47 entries. We are then able to generate same forensic result compare to using raw logs. This algorithm results in a 98.9% reduction from raw CloudTrail logs.

With goal-directed logging, we can deliver the reduced version of logs to a separate location. In the event of forensic analysis, if $P = (Event, Resource)$ matches $P$ in the forensic instance $\langle I, \gamma, \psi, q = (R, P), \pi = Actual, L\rangle$, we can use the reduced logs to perform analysis.

# Chapter 4

# Conclusions and Future Work

In this thesis we conduct a case-study on forensic analysis in access control system. We reviewed two access control scheme, HRU and ARBAC. We summarize computational complexity of the forensic problem in access control scheme. For each of the access control scheme, the forensic analysis problem is no easier than the safety analysis problem from the standpoint of access control. We deploy a serverless application, Hello-retail, to AWS. We simulate a data breach and perform forensic analysis on the application. We find that the default logging information generated by AWS is not sufficient to answer "Exercise of an privilege" question. We analyze control-plane logs, recorded by CloudTrail, and show that they can be used to resolve "Possession of an access" question. We show that CloudTrial logs are redundant in forensic analysis. We discuss the possibility of a goal directed logging mechanism that can be used to generate sufficient and necessary logs for forensic analysis.

Future work can be done based on the work discussed in this thesis. Other access control scheme has yet to be studied in the scope of forensic analysis. We wonder whether for any access control scheme, the computational complexity of forensic analysis and safety analysis is the same. We did not give a formal proof on complexity of forensic analysis when logs are indeed presented. Our case study shows that if sufficient logs are available, forensics instance can be solved in polynomial time. However, the proof is not formalized. In the case study, we focus on a specific cloud application in AWS. A forensic framework can be constructed on AWS to facilitate forensic analysis of all cloud applications.

# References

[1] Amazon web services, 2019.

[2] Aws managed policies for job functions, 2019.

[3] Logging data and management events for trails, 2019.

[4] Brian Carrier. *File system forensic analysis*. Addison-Wesley Professional, 2005.

[5] Brian Carrier et al. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of digital evidence*, 1(4):1–12, 2003.

[6] Harlan Carvey. *Windows forensic analysis DVD toolkit*. Syngress, 2018.

[7] Eoghan Casey. *Handbook of digital forensics and investigation*. Academic Press, 2009.

[8] Louis Columbus. Roundup of cloud computing forecasts and market estimates, 2018. *Forbes magazine*, 2018.

[9] Ayn Embar-Seddon and Allan D Pass. *Forensic science*. Salem Press, a division of EBSCO Information Services, 2015.

[10] Simson L Garfinkel. Digital forensics research: The next 10 years. *digital investigation*, 7:S64–S73, 2010.

[11] Morrie Gasser. *Building a secure computer system*. Van Nostrand Reinhold Company New York, 1988.

[12] Spencer Gietzen. Aws iam privilege escalation - methods and mitigation, 2018.

[13] Dieter Gollmann. *Computer Security-ESORICS 94: Third European Symposium on Research in Computer Security, Brighton, United Kingdom, November 7-9, 1994. Proceedings*, volume 875. Springer Science & Business Media, 1994.

[14] G Scott Graham and Peter J Denning. Protection: principles and practice. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 417–429. ACM, 1972.

[15] Michael A Harrison, Walter L Ruzzo, and Jeffrey D Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[16] Abdul Raouf Khan. Access control in cloud computing environment. *ARPN Journal of Engineering and Applied Sciences*, 7(5):613–615, 2012.

[17] Harmeet Kaur Khanuja and DS Adane. A framework for database forensic analysis. *Computer Science & Engineering: An International Journal (CSEIJ)*, 2(3):27–41, 2012.

[18] Ninghui Li and Mahesh V Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.

[19] Ninghui Li and William H Winsborough. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *2003 Symposium on Security and Privacy, 2003.*, pages 123–139. IEEE, 2003.

[20] Fabio Marturana, Gianluigi Me, and Simone Tacconi. A case study on digital forensics in the cloud. In *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 111–116. IEEE, 2012.

[21] Raffael Marty. Cloud application logging for forensics. In *proceedings of the 2011 ACM Symposium on Applied Computing*, pages 178–184. ACM, 2011.

[22] Juma Nahid. Forensic analysis in access control. 2020.

[23] Reno Nv. The leading cloud providers increase their market share again in the third quarter — synergy research group, 2019.

[24] Martin S Olivier. On metadata context in database forensics. *Digital Investigation*, 5(3-4):115–123, 2009.

[25] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):105–135, 1999.

[26] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[27] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[28] Amit Sasturkar, Ping Yang, Scott D Stoller, and CR Ramakrishnan. Policy analysis for administrative role based access control. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 13–pp. IEEE, 2006.

[29] Scott D Stoller, Ping Yang, C R Ramakrishnan, and Mikhail I Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 445–455. ACM, 2007.

[30] Wenhui Wang, Jing Han, Meina Song, and Xiaohui Wang. The design of a trust and role based access control model in cloud computing. In *2011 6th International conference on pervasive computing and applications*, pages 330–334. IEEE, 2011.

[31] Ziyuan Wang. Security and privacy issues within the cloud computing. In *2011 International Conference on Computational and Information Sciences*, pages 175–178. IEEE, 2011.

[32] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Achieving secure role-based access control on encrypted data in cloud storage. *IEEE transactions on information forensics and security*, 8(12):1947–1960, 2013.