Continuous Auditing & Threat Detection in Multi-Cloud Infrastructure

Kennedy Torkura [1], Muhammad I.H. Sukmana [2], Feng Cheng [2], and Christoph Meinel [2]

[1]Hasso Plattner Institute
[2]Affiliation not available

October 30, 2023

Abstract

Efficient change control and configuration management is imperative for addressing the emerging security threats in cloud infrastructure. These threats majorly exploit misconfiguration vulnerabilities e.g. excessive permissions, disabled logging features and publicly accessible cloud storage buckets. Traditional security tools and mechanisms are unable to effectively and continuously track changes in cloud infrastructure owing to transience and unpredictability of cloud events. Therefore, novel tools that are proactive, agile and continuous are imperative. This paper proposes CSBAuditor, a novel cloud security system that continuously monitors cloud infrastructure, to detect malicious activities and unauthorized changes. CSBAuditor leverages two concepts: state transition analysis and reconciler pattern to overcome the aforementioned security issues. Furthermore, security metrics are used to compute severity scores for detected vulnerabilities using a novel scoring system: Cloud Security Scoring System. CSBAuditor has been evaluated using various strategies including security chaos engineering fault injection strategies on Amazon Web Services (AWS) and Google Cloud Platform (GCP). CSBAuditor effectively detects misconfigurations in real-time with a detection rate of over 98%. Also, the performance overhead is within acceptable limits.

Continuous Auditing & Threat Detection in Multi-Cloud Infrastructure

KA Torkura, Muhammad I.H. Sukmana, Feng Cheng and Christoph Meinel

Hasso-Plattner-Institute for Digital Engineering, University of Potsdam, Potsdam, Germany

ARTICLE INFO

Keywords:
Cloud-Security
Cloud Audit
Security Metrics
Security Risk Assessment
Secure Configuration
Cloud Security Posture Management
Threat Detection

ABSTRACT

Efficient change control and configuration management is imperative for addressing the emerging security threats in cloud infrastructure. These threats majorly exploit misconfiguration vulnerabilities e.g. excessive permissions, disabled logging features and publicly accessible cloud storage buckets. Traditional security tools and mechanisms are unable to effectively and continuously track changes in cloud infrastructure owing to transience and unpredictability of cloud events. Therefore, novel tools that are proactive, agile and continuous are imperative. This paper proposes *CSBAuditor*, a novel cloud security system that continuously monitors cloud infrastructure, to detect malicious activities and unauthorized changes. CSBAuditor leverages two concepts: *state transition analysis* and *reconciler pattern* to overcome the aforementioned security issues. Furthermore, security metrics are used to compute severity scores for detected vulnerabilities using a novel scoring system: *Cloud Security Scoring System*. CSBAuditor has been evaluated using various strategies including *security chaos engineering* fault injection strategies on Amazon Web Services (AWS) and Google Cloud Platform (GCP). CSBAuditor effectively detects misconfigurations in real-time with a detection rate of over 98%. Also, the performance overhead is within acceptable limits.

1. Introduction

In recent years, cyber-attacks against cloud infrastructure have increased resulting in massive data breaches that cost billions of dollars and exposed millions of sensitive documents [61, 58, 12]. Most of these attacks are caused by *customer* misconfigured cloud resources e.g. over-privileged users, publicly exposed databases, and lack of audit logging [34]. Consequently, the Cloud Security Alliance (CSA)'s *Top Cloud Computing Threats 2019* report [12] identified *data breaches* due to *misconfiguration and inadequate change control* as the most severe cloud security threats. Similarly, the Ponemon Institute's Data Breach Report 2019, asserts that 49% of breaches are caused by system glitches and human errors [23]. Over the years, the CSPs have improved the security of the underlying cloud infrastructure, hence attacks at this layer of abstraction are no longer common. However, the upper layer of cloud infrastructure, which customers are responsible for securing has increasingly become vulnerable to attacks due to misconfigurations and human errors. According to Gartner, until 2025, 99% of cloud failures will be directly caused by customer errors. This further highlights the importance of cloud security mechanisms that focus on addressing human errors and are customer-centric. Such mechanisms should be designed to aid cloud customers in identifying and fulfilling designated security responsibilities efficiently. Important features for such mechanisms include secure configuration of cloud resources during orchestration and subsequently enforce Security-Focused Configuration Management (SecCM) processes via continuous auditing and monitoring [28, 9]. Other important features include automated incident response and cloud resource tracking.

Existing SecCM processes [28] are not sufficient for tackling the contemporary cloud infrastructure challenges. Traditional mechanisms functioned in environments with well established protocols for creating and modifying resources, which normally took days for careful reviews and approvals. However, cloud infrastructure is abstracted by software and due to increasing market demands, contemporary practices e.g. *DevOps* [33, 16] facilitates agile processes where infrastructure is launched into production environments within seconds. In these scenarios, the traditional orchestrating processes are bypassed and the lifespan of infrastructure is highly unpredictable. Due to these factors, traditional inventory and auditing mechanisms struggle to keep track of infrastructure changes [62].

Several cloud-specific mechanisms have been proposed to overcome the aforementioned challenges, however, most of these focus on cloud compute and networks [13]. Other cloud services e.g Cloud Storage Services (CSS) and Identity and Access Management (IAM) are lack efficient SecCM techniques and the specif requirements for ensuring this differs. For example, Cloud compute and networks leverage security control e.g. firewalls and Intrusion Detection Systems (IDS) [52] to enforce detective and preventive security controls, however, these security controls are ineffective for protecting some cloud services e.g. cloud storage and IAM. Therefore, novel security mechanisms are imperative to tackle these open security challenges. A promising cloud security system recently proposed by Gartner to address these issues is the Cloud Security Posture Management (CSPM) [34].

Therefore we propose *CSBAuditor*, a novel cloud security system that continuously monitors and audits cloud infrastructure, to detect misconfigurations, malicious activities and unauthorized changes. CSBAuditor leverages two concepts to achieve these objectives: the *reconciler pattern* and *state transition analysis*. The *reconciler pattern* [32] enables tracking of cloud resources using *expected-state* and *cloud-*

kennedy.torkura@hpi.de (K. Torkura)
ORCID(s): 0000-0001-8967-1035 (K. Torkura)

state. The *expected-state* reflects the state at *orchestration time*, while the *cloud-state* refer to the state at anytime after resource orchestration. The *reconciler pattern* employs a *Reconciliation Loop* to ensure that the cloud-state remains the same with the expected-state. However, in order to introduce investigative capabilities i.e. how to determine what changed between the two or more states, we employ the *State Transition Analysis* [22]. The ability to detect and analyze each specific change enables threat detection capabilities, given unauthorized changes could be Indicator of Compromise (IoC). Combining these two strategies efficiently tackles security misconfiguration challenges, *configuration drift* [7] and detects threats thereby resulting to a robust cloud security model.

Furthermore, security metrics are used to compute severity scores for detected vulnerabilities using a novel scoring system: Cloud Security Scoring System (CSSS). This severity scoring system, therefore, helps in risk management e.g. risk prioritization. Ultimately, CSBAuditor is a CSPM, since it fulfills all the requirements for CSPM as proposed by Gartner [34] and CSA [39].

Contributions Initial results on CSBAuditor were presented in our earlier paper [57]. The focus was proactive security risk analysis techniques to Cloud Storage Broker (CSB) using *CloudRAID* [8, 49, 51] as a reference architecture. In this article, we have extended our initial concepts in several ways including:

- Implement and evaluate novel techniques for continuously auditing and monitoring security threats in multi-cloud cloud infrastructure.
- Propose innovative techniques for implementing SecCM for cloud resources using *state transition analysis* and *reconciler pattern*. The affords several advantages including efficient change control management, configuration drift control, threat detection and *reconciler pattern*.
- We propose CSSS, a risk model that leverages employs the Common Vulnerability Scoring System (CVSS), to score the severity of configuration vulnerabilities. CSSS is useful for unifying the risk assessment methodologies between several cloud services e.g. cloud compute and cloud storage.

The rest of this paper is structured as follows, a concise illustration of the problems tackled in this paper is outlined using a running example, in Section 2, followed by discussion on the design choices for CSBAuditor in Section 3. In Section 4, we provide implementation details of CSBAuditor, and evaluate its performance in Section 5. Related works are presented in Section 6 and the article's conclusion in Section 7, while a brief outline of future works is presented in Section 8.

2. Background and Problem Statement

In this section, we provide background information on the complexities of employing SecCM in cloud infrastructure. A representative *running example* is presented to clearly depict the challenges introduced by mis-configuration vulnerabilities in the cloud. Thereafter, the challenges of applying risk analysis to the cloud are discussed.

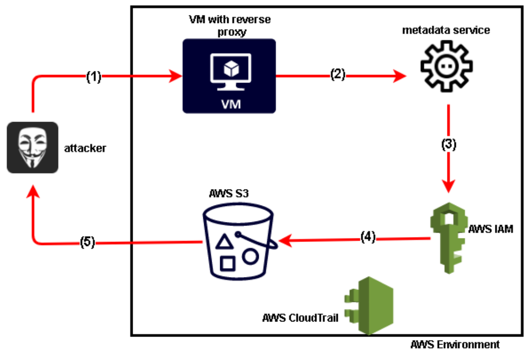


Figure 1: Running Example- an illustration of the Capital One Data Breach [36]

2.1. Running Example - The Capital One Data Breach

To provide a concrete illustration of the contemporary cloud security issues, we present a realistic running example based on the Capital One data breach [36, 40]. We used this running example for a related article [55] and we have decided to reuse it here since the topics are interrelated. The running example depicts the cyber-attacks against Capital One's AWS infrastructure between April - July 2019. Capital One became aware of the attack in July 2019 when an independent security researcher alerted them as part of an ongoing bug bounty program. Figure 1 is an illustration of the attack scenario. The initial *entry point* (**EP01**) was a misconfigured reverse proxy, that the attacker identified and leveraged to gain access to an Elastic Computing Cloud (EC2) VM (*Step 1*), where the reverse proxy server was hosted. Having gained an initial *foothold*, the attacker executed a Server-Side Request Forgery (SSRF) attack against the metadata server (*Step 2*), to obtain valid and extensive permissions. The metadata server, in turn, requested permissions from the IAM service, as defined in the profile access control policy (*Step 3*). These permissions were overly permissive (**EP02**), granting access to the entire Simple Storage Service (S3). Essentially, the VM (including any user inheriting the permissions scoped within the VM) can make root-level requests against all S3 assets including storage buckets and objects, bucket policies, bucket ACLs, S3 static hosted websites and S3 metadata. The attacker successfully inherits these privileges (*Step 4*), by taking control of the VM. Thereafter, the attacker gains access to several critical information from the S3 buckets (*Step 5*) e.g. customers' email addresses, social security numbers and credit card information (**EP03**). The attacker successfully exfiltrated data out of the AWS environment, completely unnoticed, leading to a massive data breach that cost Capital One over In the above scenario, we notice several security issues due to misconfigured cloud assets and ought to be prevented by implementing

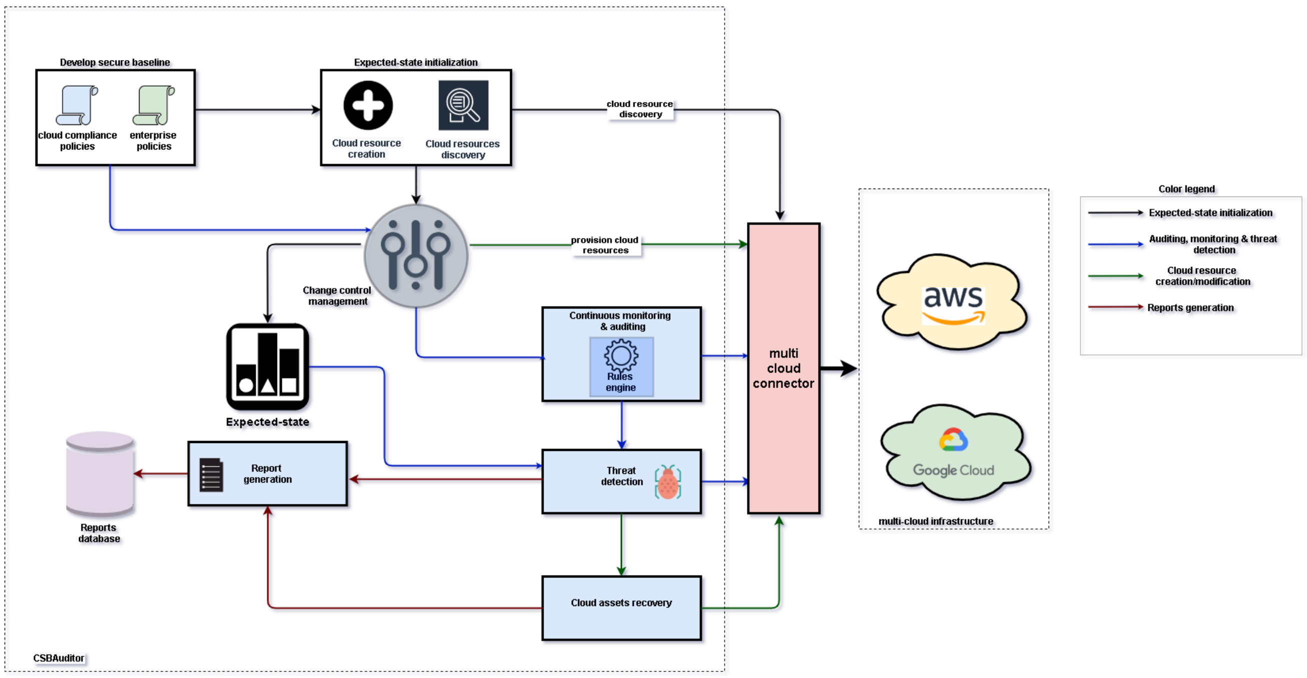

Develop secure baseline
cloud compliance policies
enterprise policies
Expected-state initialization
Cloud resource creation
Cloud resources discovery
cloud resource discovery
provision cloud resources
Change control management
Expected-state
Continuous monitoring & auditing
Rules engine
Threat detection
Report generation
Reports database
Cloud assets recovery
multi cloud connector
aws
Google Cloud
multi-cloud infrastructure
CSBAuditor
Color legend
Expected-state initialization
Auditing, monitoring & threat detection
Cloud resource creation/modification
Reports generation

Figure 2: CSBAuditor Process Workflow

security controls. This security lapses thereby exposed three entry points **EP**:

- **EP01:** Misconfigured reverse proxy
- **EP02:** Over-privileged profile policy, that does not satisfy *the principle of least privilege*
- **EP03:** Massive ex-filtration of sensitive data from the S3 bucket without triggering alarms.

2.2. Risk Analysis Challenges - Cloud Perspective

The Shared Security Responsibility Model (SSRM) is a popular security model employed by public CSPs [24, 6]. It clearly defines security responsibilities for cloud stakeholders. The ability to detect and mitigate the attacks highlighted in the previous subsection is the responsibility of cloud users, however, most cloud users do not clearly understand these responsibilities, or lack the tools to implement commensurate technical and organizational measures. Consequently, CSPs provide security services to aid cloud users, however, these services are not completely efficient for several reasons, hence the recurring security attacks against cloud infrastructure. A typical example is the implementation of industry security standards and compliance regulations for the cloud. Currently, the Centre for Internet Security (CIS) benchmarks, which define best practices for ensuring security for cloud infrastructure is well-adopted security and compliance standard. However, there is a gap between these high-level recommendations and commensurate low-level implementation [35]. A typical example of this misalignment is the lack of mapping of security metrics to the CIS benchmarks. Most security metrics are measured based on qualitative or quantitative metrics such as CVSS and are useful for guiding security professional in making risk-based decisions. Thus an implication of this lapse is challenged in implementing risk assessment techniques for cloud assets. Techniques like probabilistic attack graphs that rely on CVSS base scores to derive probabilities are difficult to implement. Risk assessment requires well-scored vulnerability reports for decision making, however, results provided by CSPs are either qualitative or are computed based on proprietary methods.

3. CSBAuditor: Design and Threat Model

In this Section, we briefly discuss our design choices for CSBAuditor including the development of secure baselines and how the two core concepts are leveraged: *reconciler pattern and state transition analysis*. Thereafter, a discussion of the proposed CSSS is presented and then, a threat model unique to our proposal is discussed. Figure 2 is a visualization of CSBAuditor's process flow.

3.1. Developing Secure Baseline Configuration

Secure baseline configuration are important especially for cloud security as they provide a common denominator for configuring cloud resources including access control policies and ACLs. Furthermore, employing secure baselines enables efficient planning, and prevents errors and misconfigurations by aligning infrastructure with recommended best practices e.g the CIS cloud security benchmarks [26] [25]. We aim at providing tooling support for the process of ensuring adherence to secure baselines by, this maps to phase one and two of the National Institute of Standards and Technology (NIST) SecCM phases [28]. There are two categories of policies to be employed:

3.1.1. Compliance Policies

These types of policies are designed to aid in complying with one or more security best practices e.g. CSA cloud security guidelines [1], CIS cloud benchmarks e.g for GCP[26].

3.1.2. Enterprise Security Policies

These category of policies are those that are based on enterprise security goals and internal policies. Examples of these includes the access control regulations the various teams e.g. Finance and Human Resources. Algorithm 1 implements a check to detect buckets that are publicly exposed. Rather than applying the best practices which recommend for all buckets to be private, a custom check is initiated since there serving public content e.g. pictures need to be public to enable access.

3.2. Expected-State Initialization

ISO 27001 [27] recommends the implementation of robust asset management and change control mechanisms for enterprise security. Such mechanisms are even more important tackling security issues in cloud platforms due to the rapid and often unpredictable changes the occur in the cloud. Therefore, CSBAuditor's employs an asset control technique that is initiated by establishing an *expected-state*, which ideally should be an integral part of the *cloud resource orchestration* system. This approach is based on Infrastructure-as-Software (IaS), a cloud operating model that allows for efficient manipulation of cloud resources [18, 15].

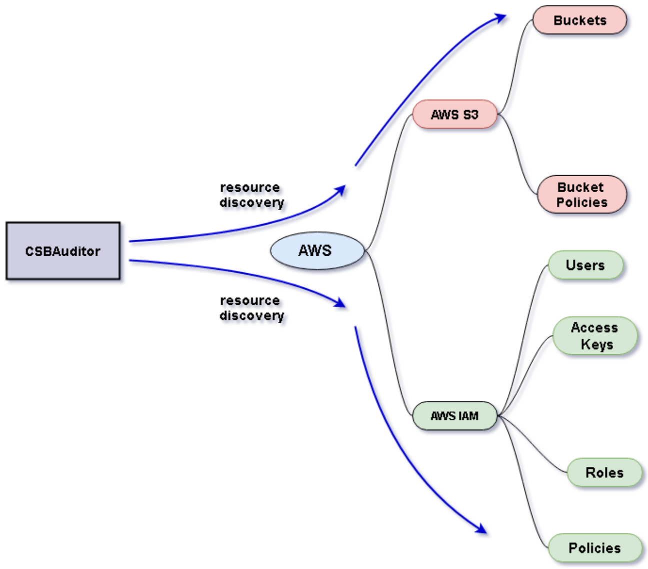


Figure 3: CSBAuditor Discovering Resources in an AWS Account

3.2.1. Initialization via Infrastructure-as-Software

According to Fitzgerald et.al [15], IaS is based on the principle that *all the data that defines a piece of infrastructure can be programmed in a language that allows it to be versionable, repeatable and testable in a similar manner to software code.* IaS differs from Infrastructure-as-Code (IaC) in three major ways: (1) programmers are able to manipulate cloud resources using programming languages, rather than Domain Specific Language (DSL). Therefore, already attained programming skills are leveraged to increase productivity. (2) IaS does not require special handling approaches for *state management*, established programming language persistence layers can be used, on the other hand IaC requires varying efforts to address state management. In order to implement IaS, cloud APIs or programming language SDKs are leveraged represent, deploy and manipulate cloud resources. For example, we programmatically represent a cloud bucket (for object storage) with a *Bucket* object, consisting of the following attributes: *bucketName, bucketId, bucketPolicy and bucketProvider*. The bucket object can therefore be used to create, modify and delete the bucket at runtime, this same principle is applicable to every cloud resource. A key advantage of IaS is that cloud resources are persisted in datastores as *states* [18], thus allowing for easy retrieval, versioning etc.

3.2.2. Initialization via Cloud Resource Discovery

In a scenario where cloud resources are already deployed on one or more cloud platforms, a discovery process is employed to enumerate all cloud resources, this are subsequently adopted as the expected-state. The discovery process leverages cloud APIs to acquire the full list of the resources already deployed in the cloud. During the discovery process (as illustrated in Figure 3) CSBAuditor queries the cloud infrastructure for important information of the deployed resources based on the already defined object using IaS paradigms. This aids in subsequent analysis of the ensures that all resources are securely, i.e. based on the policies established in Section 3.1.2 above. This is achieved by preventing manual creation of resources, rather an automated system is employed. For example, the *principles of least privilege* are ensured so that access control policies assign privileges to users based only as sufficient to perform their jobs. Details or our strategy for unifying access control across CSPs is in our previous paper [50].

3.2.3. Secure Baseline Enforcement

The last step of the *expected-state initialization* is the enforcement of secure baselines. This ensures that the cloud resources are *secure-by-default*, by testing for misconfigurations and other defined security policies before deployment to the cloud. Security issues found are remediated for both newly created resources and those already existing in the cloud, and detected via the *resource discovery*.

3.3. Change Control Management

Secure Configuration Controller (SCC) is the control point for cloud infrastructure orchestration and subsequent modification, it validates cloud resources change requests to enforce the secure baselines earlier established in Section 3.1. Due to the rapid changes that occur in the cloud, it is imper-

Algorithm 1 CheckBucketPublic Algorithm

```
1:  procedure ChkBucketPub($bucketName, bucketPublicRule$)
2:      Receive request to audit bucket
3:      getBktPolicyCloud (bucketName)        ▷ get bucket policy from the cloud-state
4:      getBktPolicyExp (bucketName)          ▷ get bucket policy from expected-state
5:      if $getBktPolicyCloud.equals\ getBktPolicyExpect$ then    ▷ reconciler pattern
6:          getRiskScore (bucketPublicRule)   ▷ no changes detected
7:      else
8:          getRiskScore (bucketPublicRule)   ▷ compute risk score
9:          recoverBktPolicy (bucketName)     ▷ recover the bucket policy to its original state
10:         sendAlert                         ▷ send alert to admin
11:     end if
12: end procedure
```

ative to deploy a control mechanism that leverages automation, hence the need for the SCC. The SCC leverages the following mechanisms to perform its functions :

- Access control systems are employed to regulate who is authorized to propose configuration changes. This helps in ensuring only those with the right privileges are allowed to make changes.
- Security tests are employed for validating infrastructure by leveraging IaS to ensure adherence to security and compliance policies as suggested by Fitzgerald et.al [15]. IaS use programmatic constructs to manipulate cloud resources at run-time. .
- Security Chaos Engineering (SCE) [53, 55] techniques are employed to validate that the changes do not introduce vulnerabilities to the cloud infrastructure. This could be done either in the development or production environment.

3.4. Continuous Security Monitoring and Auditing

Cloud infrastructure is constantly evolving due to changes required for the delivery of goods and services. Consequently, the rapid rate of change increases the complexity of managing and securing cloud infrastructure. Furthermore, some changes could have malicious intent, such changes could be initiated by attackers and therefore if detected early become crucial IoC. Therefore, continuous monitoring and auditing strategies are is imperative to detect IoC [10]. We achieve this by continuously inspecting cloud resource, including their respective configuration to detect deviations from the secure baselines. This process also aids in providing security visibility to the state of the cloud infrastructure. We leveraged the *reconciler pattern* to implement our continuous security monitoring and auditing strategy:

Reconciler Pattern: The Reconciler Pattern is a software pattern that provides programming models for efficiently managing cloud-native systems. The Reconciler pattern is employed in several cloud-native system e.g. Kubernetes [17] and Terraform [21]. At the core of the Reconciler Pattern is the notion that every resource consists of *two states* at time - t_o,

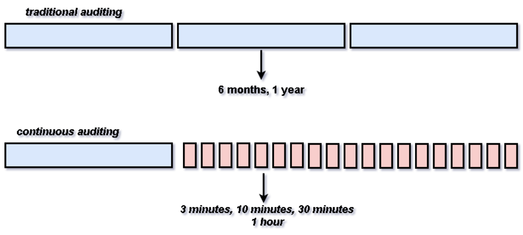


Figure 4: A Comparison of traditional Auditing Processes versus Continuous Auditing

- Desired State: Which is also known as the *expected-state*, refers to the ideal state in which the resource is expected to be at t_o .
- Actual State: This is real world situation of the resource, how it is currently seen. In this paper, we refer to this as the *cloud-state*, since we are dealing with cloud infrastructure.

The reconciler pattern strives to detect whenever the cloud-state drifts from the expected-state by employing the *Reconciliation Loop*, to continuously compare the two states to detect and reconcile detected differences. The recruiter pattern achieves the reconciliation tasks by using four main functions: *getExpected(), getActual(), create() and destroy()* .

3.5. Threat Detection

A limitation of the reconciler pattern is the lack of methods for tracking the reasons for detected changes or failures. This is a critical requirement for security assurance, the knowledge of *what went wrong?* is critical for employing detective and preventive counter-measures. Essentially, configuration changes e.g. disablement of logging, might indicate an on-going attack. Hence, investigative capabilities are imperative to augment the reconciler pattern. In traditional systems, the logging mechanisms might be analyzed in real-time to detect malicious activities. However, logs are not available in real-time in public cloud systems. Log delivery takes about 20 minutes on AWS, and over 90 minutes on GCP. Therefore it is imperative to evolve techniques that breach this window of of opportunity from malicious actions. We address the afore-mentioned lapses with a novel adaptation of the reconciler pattern to automatically track the root causes of configuration drift and secure baseline violations. Figure 6 illustrated our adaptation, which leverages *state transition analysis*.

State Transition Analysis: State transition analysis is an analytical model for detecting and representing malicious events in computer systems [22]. Malicious events are modeled as the transition of states from a secure state (good) S_o, the already established *expected-state* to an unknown state, a

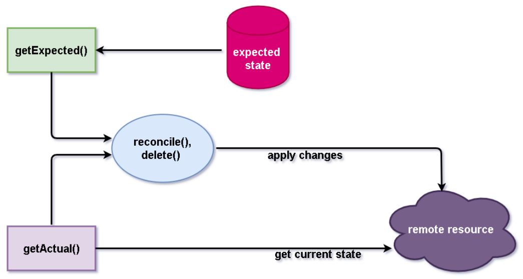


Figure 5: Reconciler Pattern showing its four methods: *getExpected(), getActual(), reconcile() and delete()*

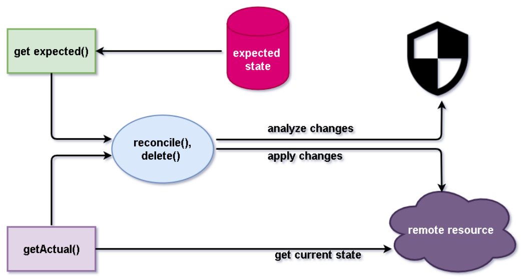


Figure 6: The Reconciler Pattern enhanced with the State Transition Analysis to real-time threat detection.

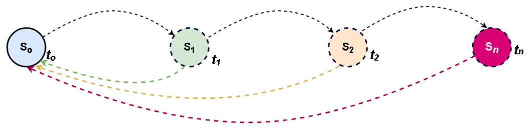


Figure 7: State Transition Analysis

state that (1) deviates from the expected-state (2) is not validated at the SCC. As illustrated in Figure 7, an unknown state can vary from S_1 to S_n. Within this range, each subsequent state represents a compromised state which might be due to misconfigurations or malicious action (attackers) e.g. change of an AWS access policies to escalate privileges, creation of new resources without the authorization of the SCC. To achieve this, CSBAuditor leverages the API of CSPs e.g. GCP to continually acquire the exact state of the cloud resources (*cloud-state*). Thereafter, the expected-state is reconciled with the cloud-state by comparing and detecting differences. CSBAuditor uses its Scheduler to make calls for these operations every 3 minutes. Figure 8 is alert generated due to a misconfigured AWS S3 bucket. The alert contains enough information to provide a security operator insights about the detected misconfiguration e,g, the name of the affected cloud resource (*resourceName* on Line 5 of Figure 8).

3.6. Cloud Assets Recovery

When combined with the reconciler pattern above (Section 3.4), CSBAuditor is able to investigate the exact changes between S_o and any state between S_1 and S_n. Furthermore, *expected-states* can be recovered by re-deploying the affected assets, from original definitions contained in the *expected-state*. The asset recovery can either occur automatically or an alert can be sent to a human operator to manually inspect and make a decision, these form the two operational modes of CSBAuditor (*automatic* and *admin*). In the case where the human operator decides approves the prefers the version of the cloud resources in the cloud (cloud-state), a synchronization operation is triggered, and the cloud-state is adopted as the expected-state. In this case, the cloud-state is retrieved and persisted in the database and the SCC is notified.

3.7. Report and Alert Generation

Alerts are generated at the end of each rule execution. The alerts contain important information for understanding the events leading to the security event including description of the executed rule, security event finding, affected cloud resource and severity score of the security finding, which is based on the (CSSS). Hence the alerts provide high value for incident response and forensic investigation. The generation of the alert is based on a constructed *Alert object*, the necessary information directly extracted from the rule and the algorithm. For example, in Figure 8 an alert about a misconfigured AWS S3 object has been constructed. The alerts are also presented on a dashboard as visualized on Figure 11.

3.8. Cloud Security Scoring System

Risk assessment is a critical aspect of organizational security since it provides the basis for quantifying security and also making critical decisions e.g. *security impact analysis* [28]. Consequently, the outcome of the continuous auditing and monitoring operations discussed in Section 3.4 are not expressed in binary categories e.g. *secure/insecure* or *true/false* as done in similar systems. Instead, fine grained security risk metrics are computed for these detected security events. The security metrics are computed for every violation alert using the CVSS, one of the most popular security metrics standard. Table 1 contains some of the rules CSBAuditor's Rules Engine (Section 4.4). The details of the methodology for calculating the CVSS scores is described below:

3.8.1. CVSS

We extended our previous works on threat modeling and proactive risk analysis for cloud infrastructure, where we used the CVSS version 2 to score vulnerabilities in cloud infrastructure [57, 56]. The CVSS metrics are expressed using with *base scores*, which are numeric representations of risks, assessed in terms of severity [37, 48]. The *base scores* are computed using the *Impact* (Eqn 2) and *Exploitability* (Eqn 3) metrics, as expressed in Eqn 1. We have used our expert knowledge to compute these metrics, comparing them with similar vulnerabilities and following the guidelines in the CVSS manuals [37, 48].

3.8.2. Deriving Security Metrics with CVSS

Let us consider how to compute security severity using the CVSS for three representative cloud attacks: *Cloud Storage Enumeration Attack*, *Cloud Storage Exploitation Attack*[57, 56, 9] and *Credential Report Abuse*.

$$BaseScore = round\_to\_1\_decimal(((0.6 * Impact) + (0.4 * Exploitability)\lambda 1.5) * f(Impact)) \quad (1)$$

$$Impact = 10.41 * (1 - (1 - ConfImpact) * (1 - IntegImpact) * (1 - AvailImpact)) \quad (2)$$

$$Exploitability = 20 * AccessVector * AccessComplexity * Authentication \quad (3)$$

Table 1
Some of the rules available in the rules engine with corresponding RuleType, **CSSM!** (**CSSM!**) scores and a brief description

RuleId	RuleName	RuleType	CSSM Scores	Description
SAR01	check_public_bucket	*storage_auditor*	7.2	checks if the bucket's ACLs are set to PRIVATE
SAR02	check_public_objects	*storage_auditor*	6.5	check if the ACLs of objects in a bucket are set to PRIVATE
SAR03	check_missing_buckets	*storage_auditor*	6.4	checks if there are missing buckets
SAR04	check_unknown_bucket	*storage_auditor*	4.0	checks if there are buckets existing without approval
SAR05	check_bucket_logging_enabled	*storage_auditor*	5.0	checks if the feature for logging all activities against bucket is enabled
SAR06	check_bucket_permissions	*storage_auditor*	4.0	check that the policies attached to bucket conform with to the secure baseline
IAR01	check_missing_user	*iam_auditor*	3.0	check if the users accounts approved are deployed in the cloud
IARO2	check_unknown_user	*iam_auditor*	5.0	check is a user account exists without authorization
IAR03	check_missing_role	*iam_auditor*	7.0	checks if the roles approved are all available
IAR04	check_unknown_role	*iam_auditor*	6.0	checks if roles available differ from those approved

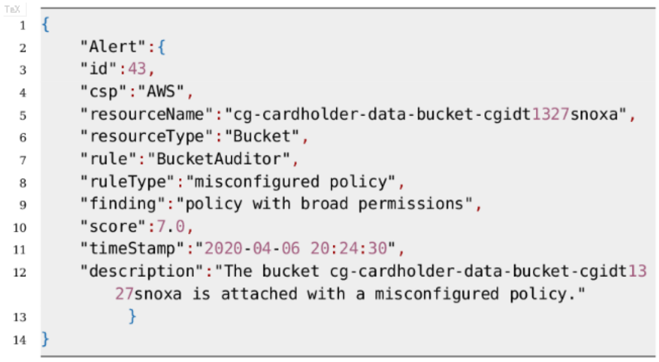

```
{
    "Alert":{
    "id":43,
    "csp":"AWS",
    "resourceName":"cg-cardholder-data-bucket-cgidt1327snoxa",
    "resourceType":"Bucket",
    "rule":"BucketAuditor",
    "ruleType":"misconfigured policy",
    "finding":"policy with broad permissions",
    "score":7.0,
    "timeStamp":"2020-04-06 20:24:30",
    "description":"The bucket cg-cardholder-data-bucket-cgidt13
       27snoxa is attached with a misconfigured policy."
        }
}
```

Figure 8: An Alert showing the result an executed rule involving a misconfigured AWS S3 Bucket

- *Cloud Storage Enumeration Attack:* This attack aims at detecting misconfigured buckets for a selected target e.g. a company's AWS S3 buckets that are publicly accessible. The attacker leverages previous knowledge about the target acquired via enumeration techniques [14], to construct possible keywords that are relevant to the target e.g. company name. These keywords are then fed into the word-list generation tool such as *Mentalist* [1], to generate a list of possible word combinations which can be used to construct AWS S3 bucket names for the company. Thereafter, the generated word-list is fed to Bucketfinder [2], a cloud exploitation tool, which conducts the attack. Bucketfinder uses the generated word-list to construct and probe AWS S3 URLs using HTTP GET requests, responses with code 200 are publicly accessible. Equations 1 - 3 are the equations used for computing the the base scores for the CVSS. Due to space limitations, some details of the equations are omitted e.g. static values for the AccessVector, AccessComplexity, Authentication, ConfImpact, IntegImpact and AvailImpact. These values are available at various resources such as the CVSS Implementation Guide [37]. However, for the *Cloud Storage Enumeration Attack* described here, we assign *Network* for the AttackVector metric since the attack can be executed over the internet. Similarly, the AccessComplexity is assigned *Low* given that attackers can execute this attack with tools available *in the wild* e.g Metasploit and on several GitHub repositories. The Authentication metric is set to *None*,

[1]https://github.com/sc0tfree/mentalist
[2]https://digi.ninja/projects/bucket finder.php

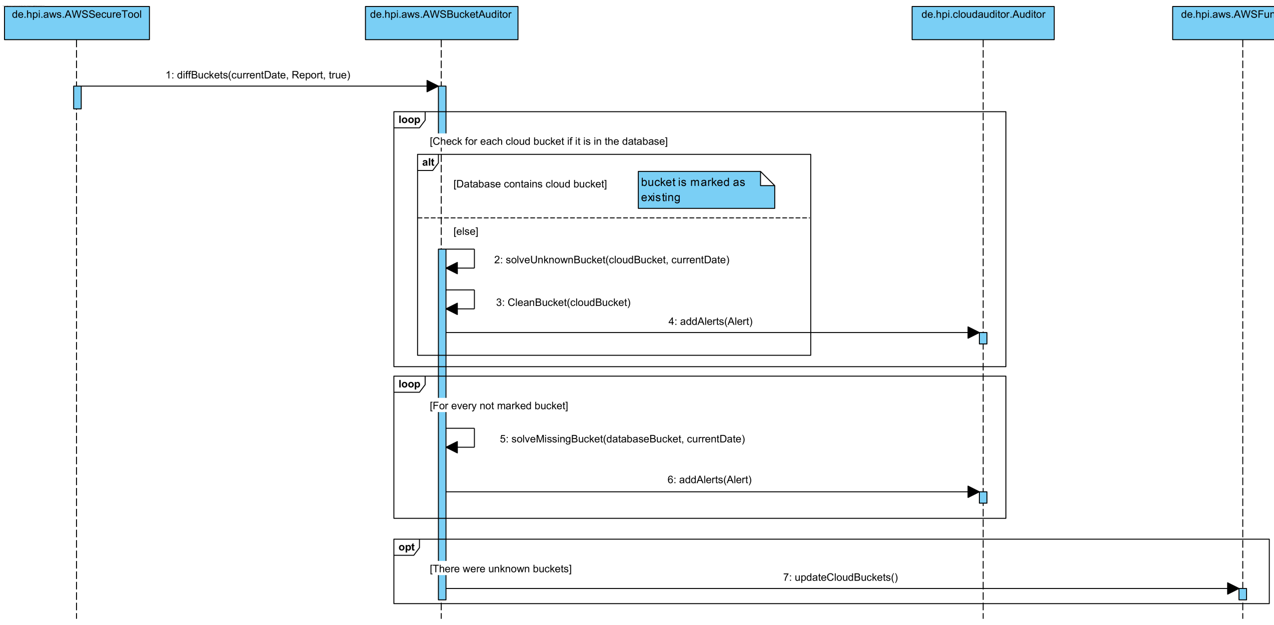

de.hpi.aws.AWSSecureTool
de.hpi.aws.AWSBucketAuditor
de.hpi.cloudauditor.Auditor
de.hpi.aws.AWSFunctions
1: diffBuckets(currentDate, Report, true)
loop
[Check for each cloud bucket if it is in the database]
alt
[Database contains cloud bucket]
bucket is marked as existing
[else]
2: solveUnknownBucket(cloudBucket, currentDate)
3: CleanBucket(cloudBucket)
4: addAlerts(Alert)
loop
[For every not marked bucket]
5: solveMissingBucket(databaseBucket, currentDate)
6: addAlerts(Alert)
opt
[There were unknown buckets]
7: updateCloudBuckets()

Figure 9: Sequence Diagram Showing An Automated WorkFlow Where Several CSBAuditor Rules are Executed e.g. SAR03 & SAR04 (Table 1)

because no authentication is required for the attack. For the Impact metrics, IntegImpact, ConfImpact and AvailImpact is set to *Partial* since there is a possibility of either acquiring materials encrypted in buckets/objects with properly configured Access Control List (ACL). Based on these metrics ($AV : N/AC : L/Au : N/C : P/I : P/A : P$)[3] we derive 7.5, as the base score. The Cloud Storage Enumeration Attack is comparable to *brute force password guessing attacks* e.g. CVE-2012-3137[4].

- *Cloud Storage Exploitation Attack:* The *Cloud Storage Enumeration Attack* leverages the previous attack as a staging step. The actual attacks against misconfigured buckets are conducted during this attack by employing cloud exploitation tools e.g. Bucketfinder. To compute the severity scores, we assign *Network* to the AttackVector metric, since the buckets are reachable via the internet. The AccessComplexity is assigned *Low*, while the Authentication metric is set to *None*, given there is no authentication mechanism protecting the bucket. The Impact metrics is more severe given the previous attack informs the attacker of buckets that are *publicly accessible*. Thus, the IntegImpact, ConfImpact and AvailImpact are set to *Complete*. We thus have the base metrics as *(AV:N/AC:L/Au:N/C:C/I:C/A:C)*, and arrive at a score of 10.0. The score is reasonable considering it affords an attacker full access to AWS S3 bucket.

- *Credential Report Abuse:* AWS IAM provides credential reports [5] to support auditing of account users. The credential report can be generated by users with at least the *GenerateCredentialReport* permission. Due to the subtlety of this permission, it is common to have it assigned to users who do not require it e.g. together with other credentials or erroneously. Hence, an attacker who gains access to valid AWS credentials could easily request AWS IAM for the credential report of an account, thereby gaining insights into the details of the users provisioned in that account including user_creation_time, password_enabled and mfa_active. The credential report can be queried directly using the AWS CLI, SDK or attack tools like the Rhino Security Pacu Post Exploitation tool [43]. We assign scores to this attack using the CVSS similar to the previous attacks, since the attacks is conducted over the internet, *Network* is assigned to the AttackVector metric. The AccessComplexity is *Low*, while the Authentication metric is set to *Partial*, due to the requirement for valid permissions. For the Impact metrics, the ConfImpact is assigned partial, since the attacker has some amount of visibility to the other users in the account. However the IntegImpact and AvailImpact are set to *None*, in this case as the attacker has only read access. Therefore, the computed base metrics representation is *(AV:N/AC:L/Au:S/C:P/I:N/A:N)*, with a score of 4.0.

3.9. Example Scenario - Bucket Audit

We present a simplified example of a bucket audit operation (illustrated in Figure 9). The audit operation executes two rules (*SAR03 and SAR04*) against an AWS environment. The *auto* flag set to *true* in the triggering func-

[3]this is a vector string representation of all computed metrics for a vulnerability

[4]https://nvd.nist.gov/vuln/detail/CVE-2012-3137

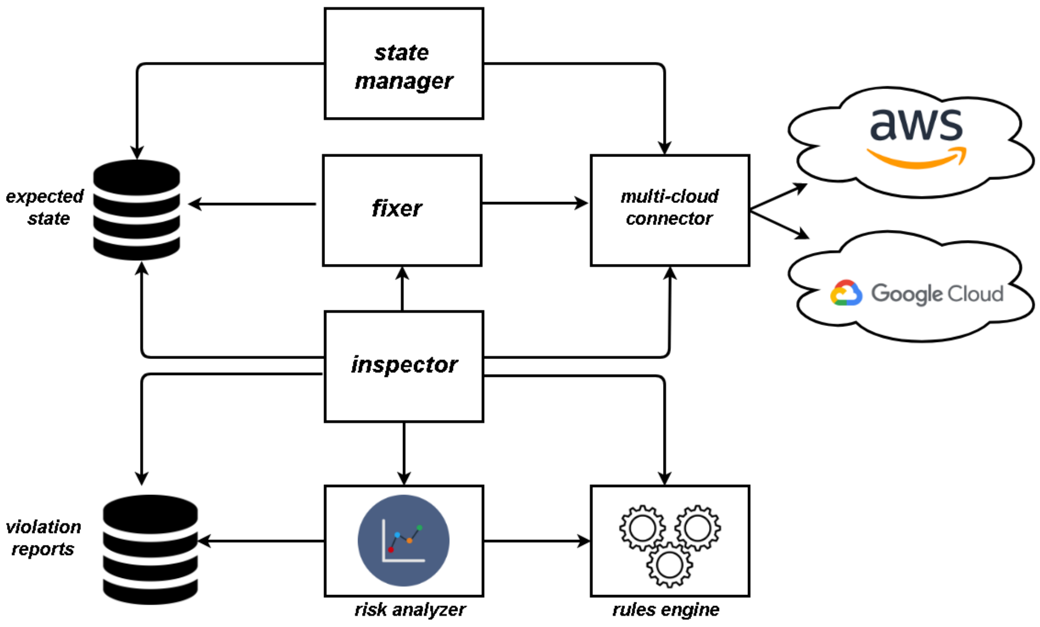


Figure 10: High Level Architecture of CSBAuditor

tion - $diffBuckets(currentDate, Report, true)$. The *true* flag implies necessary action can be automatically taken to remediate detected security misconfigurations based on *conditions* and corresponding *actions* defined in the rules. An important aspect of the audit is the report generation, hence the triggering function takes a *Report* object as an argument. Furthermore, as earlier introduced in Section 3.4, the *reconciler pattern* (Figure 5) is employed as a core aspect of CSBAuditor. Therefore, the expected-state and cloud-state are collected, and the *reconciliation* process is conducted. This process involves comparing the buckets existing in the cloud with the representation of the buckets retained in the expected-state (see comment in sequence diagram - check for each cloud bucket if it is in the database). Three possible outcomes emerge from the reconciliation process:

- No change: If the expected-state is the same as the cloud-state then no action is taken, a report is generated and the audit ends.
- Unknown Buckets: The *unknown bucket* scenario a condition where there are buckets in the cloud-state that do not exist in the expected-state. Here, the function $solveUnknownBucket$ reconciles the situation using the reconciler pattern's ($reconcile()$) method. Next, the $CleanBucket()$ function is activated to delete the unknown buckets including the objects in the bucket, this is in accordance with the $delete$ method of the reconciler pattern.
- Missing Bucket: The last possibility is a situation where one or more buckets originally deployed to cloud are no longer found which could be due to intentional or mistaken deletion. This is detected and reconciled by rule *SAR03*, implemented by $solveMissingBucket()$ function. The rule works by acquiring the appropriate bucket information from the expected-state, creating the bucket using and redeploying it in the cloud. These actions are in accordance with the reconciler pattern's $reconcile()$ method.

One point to note is that the execution procedure CSBAuditor rules (e.g. those employed for the above bucket audits) are the same for both AWS and GCP. However some low level implementation details differ per provider due to the unique Cloud Service Provider (CSP) APIs and architecture. More details about CSBAuditor implementation is provided in Section 4.

3.10. Threat Model

We assume CSPs may have implementation flaws, misconfigurations and vulnerabilities which are potentially exploitable by malicious entities to violate security properties specified by enterprise cloud administrators. Furthermore, we also assume that cloud users, and cloud operators, are potentially malicious. However, we assume that the CSP may be trusted for the integrity of the audit data (e.g., access policies and IAM policies) collected through API calls. Nevertheless, CSBAuditor will detect violations of specified security properties e.g. unauthorized bucket policy alterations, user, policy, role or group creation/detection/modification.

4. Implementation

This Section outlines the implementation details of *CSBAuditor*. All components of CSBAuditor are implemented in Java programming language. The Java Software Development Kit (SDK) for AWS & GCP were leveraged to support cloud infrastructure integration. The architecture of CSBAuditor is shown in Figure 10.

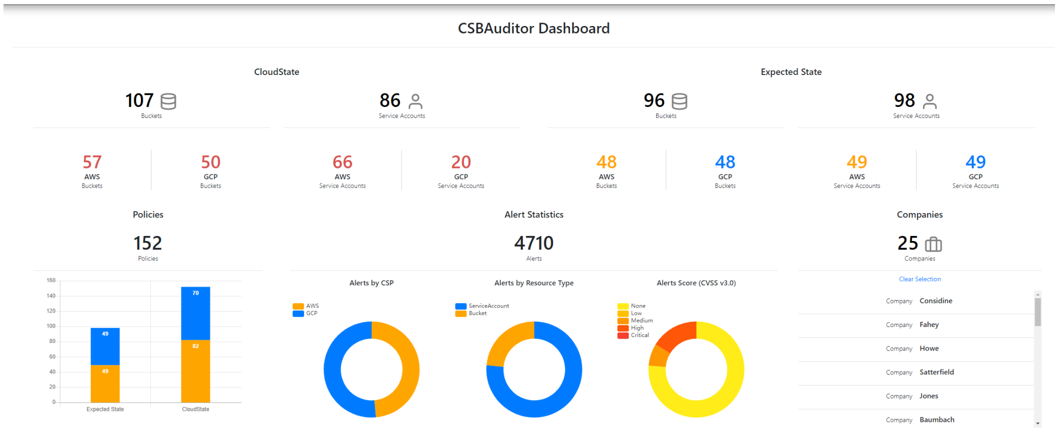


Figure 11: CSBAuditor Dashboard Showing the *expected-state*, *cloud-state and summary of alerts according to severity and CSP*

4.1. Overview of CSBAuditor Workflow

CSBAuditor enforces established secure baseline, earlier described in Section 3.1 by leveraging a rule-based system. The rules are divided into several *RuleTypes* and are pre-assigned with CSSS scores. CSBAuditor can detect misconfigured cloud resources and automatically fix the issues or notify a human administrator. Figure 9 is a sequence diagram that shows the execution of several rules to check the security posture of AWS S3 buckets. In step (1), the *diffBuckets* function triggers an auditing session, the expected-state is pulled from database and to be used by the *solveUnknownBucket* function (Rule SAR 04 on Table 1) in step (2). Next, the *CleanBucket* function deletes unknown buckets found in the cloud-state, while the details of the findings are added as violation *alerts* in step 4 (*addAlerts(Alert)*). Similarly, step (5) is an audit of the expected-state, that uses (Rule SAR 03 on Table 1) to detect buckets, not in the cloud. An update of the expected state is enforced in step (7). This rules implements the *reconciler pattern* earlier explained in Section 3.4. The above procedure is also applied for Google Cloud Storage (GCS) buckets, with slight implementation details due to the differences in Java SDK.

4.2. State Manager

The State Manager initializes and maintains the expected-state by directly interfacing with the orchestration system of the cloud infrastructure and the SCC (Section 3.3). The state manager, therefore, works with the SCC to ensure effective change control management. Resource creation, modification after the *initialization* of the expected-state, is coordinated with the State Manager. During expected-state initialization, the assets already deployed in the cloud are discovered and added to the expected-state. This process was explained in Section 3.2

4.3. Inspector

The Inspector implements the drift detection component of based on the Reconciler pattern (described in Section 3.4). This is ensured by continuously monitoring and auditing the cloud infrastructure by retrieving the *cloud-state* and comparing it with the *expected-state*. Furthermore, the Inspector implements the *state transition analysis* earlier described in Section 3.4, to perform more fine-grained audit focused on detecting the exact reason for cloud resource and configuration drifts. Line 5 of Algorithm 1) illustrates this step. Details of the rules are in Section 4.4. In auto-mode, anomalies are automatically corrected (Line 9 of Algorithm 1), while alerts are persisted in the database. In manual mode, alerts are sent to the cloud infrastructure administrator for further action. The Inspector also generates and persists alerts of every scan, as described in Section 3.7.

4.4. Rules Engine

The Rules Engine contains rules that specify details of audit checks. There are two categories of rules: *compliance rules*, and *enterprise security rules*. The compliance rules are derived from several cloud security best practice definitions such as the CIS benchmarks for AWS [25] and GCP [26] respectively. For example, in Algorithm 1, the *bucketPublicRule* ensures compliance of buckets by inspecting all buckets in the cloud infrastructure to detect wrong configurations (public instead of private ACLs). The enterprise security rules are custom rules that are specific to an enterprise, including security policies for enterprise governance. These rules specify how to validate enterprise-specific security requirements e.g. how specific teams have access to specific cloud resources to implement the *principle-of least-privilege*. The rules are implemented specific to each CSP. A typical example is the rule to check for public buckets. the rule functions by inspecting the ACL attached to each respective bucket. Listing 1 shows the ACL of a GCP bucket that is public. In this case CSBAuditor reads the *members* array, where the *allUsers* membership means the bucket is public. However the implementation differs between AWS and GCP. Similarly, rules different within the services that constitute each CSP. The rules for IAM differ from those for S3. Ultimately, each rule is unique. Furthermore, there are various RuleTypes: *storage_auditor* and *iam_auditor* for cloud storage and IAM respectively.

```
{
  "bindings":[
    {
      "members":[
      "allAuthenticatedUsers",
      "allUsers",
      "alice.test@gmail.com"
    ],
 "role":"roles/storage.admin"
    }
  ],
 "etag":"CAM="
}
```

Listing 1: A Mis-configured Bucket ACL on Google Cloud Platfprm

4.5. Fixer

The Fixer has the responsibility of resolving misconfigured assets to their expected-states. It receives a *recover-state* request from the Inspector, together with the information of the asset to be recovered e.g. *Asset-Name, Asset-Id and Asset-Tag*. Based on this information, the Fixer retrieves the expected-state of the asset and redeploys it. Essentially, the *fixer* implements the *reconcile() and delete()* methods of the reconciler pattern. The delete() method deletes the cloud resources that differ from the expected-state. The Inspector watches the cloud, detects the drift, get the name of the resource and hands it over to the fixer. The fixer then deletes the resource and employs the reconcile methods to redeploy the resource based on the expected-state.

4.6. Risk Analyzer

The algorithms for deriving the CSSS are implemented by the Risk Analyzer. In order to support risk analysis, the Risk Analyzer implements algorithms for deriving security metrics from the alerts generated by the Inspector. These security metrics are based on CVSS [48] and details of our methodology was provided in Section 3.8. The appropriate scores are already pre-computed and assigned for every rule, an example of these is in Table 1. The CVSS equations (equations 1 - 3) are employed for computing the base scores in order to make the scoring compatible with CVSS as proposed in Common Configuration Scoring System (CCSS) scoring guidelines [47]. One advantage of employing the CVSS is for reasons of generating probabilistic attack graphs for threat modeling. Such models are useful when assessing the risk of cloud infrastructure, across several services e.g. VMs, IAM and S3 [56]. The Risk Analyzer retrieves additional vulnerability information from public vulnerability databases such as the HPI-VDB [5].

4.7. Multi-Cloud Connector

The multi-cloud connector leverages the APIs of the supported CSPs for transmitting and receiving requests and responses, therefore CSBAuditor employs an *agent-less* strategy. However, appropriate access permissions are required for the respective providers, *READ* permissions are sufficient for retrieving the cloud-state, while *WRITE* permissions are required for automatically applying fixes to the cloud infrastructure (reconciliation process). Alternatively, alerts can be sent to a human agent to manually trigger fixes. An important aspect of the multi-cloud connector is the support for extensibility. Since the objects representing cloud resources are cloud agnostic, the major effort required for adding more CSP is to add the respective cloud connector interface.

[5]https://www.hpi-vdb.de/vulndb/

5. Evaluation

Experimental evaluations for CSBAuditor were conducted using a multi-cloud test-bed comprising cloud resources deployed on AWS and GCP. The multi-cloud environment was structured like an enterprise cloud environment based on guidelines provided in the AWS *well architected framework* [42]. Furthermore, the security tests are drawn from recommended guidelines available in the CSA cloud penetration testing playbook [11]. These guidelines advocate for cloud resources to be are organized into three categories: IAM, cloud services and cloud applications. However, we narrow our focus to IAM and cloud storage both are core cloud services.

5.1. Experiment Setup

The aim of these experiments are to evaluate the performance of CSBAuditor based on response times and latency in both *steady* and *transient* states. These measurements are useful for determining performance overhead. The test environment comprises 3 components:

5.1.1. Multi-Cloud Enterprise Environment

The multi-cloud enterprise environment consists of 50 user accounts, provisioned on AWS and GCP infrastructure i.e. 25 per cloud. Each user account is properly configured based on privilege separation and principle of least privilege.

5.1.2. CSBAuditor

CSBAuditor is deployed locally i.e. on a Windows 10 PC with the following configuration: Intel (R) Core (TM) i5-5200U CPU, 2.20GHz processor, 8GB RAM and 1 TB HDD. CSBAuditor executes all the tests from this environment, however, the application can be easily deployed on cloud platforms.

5.1.3. Fault Injection Testing

In order to produce the workloads, for the transient state, faults are generated to produce realistic, malicious events. Hence, we leverage *chaos engineering*-style approach as proposed in [45]. We have employed a tool developed in previous work: CloudStrike [53**?**], which injects various levels of *faults* to the test cloud environments (AWS and GCP).

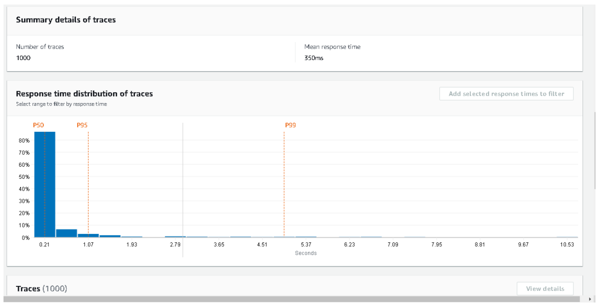


Figure 12: AWS S3 Response Time Distribution of Traces -*Steady State*

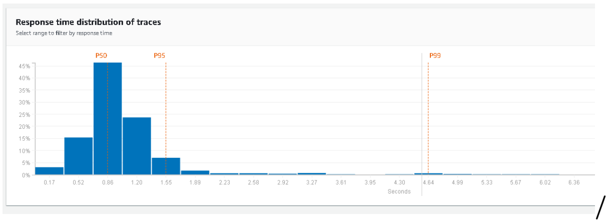
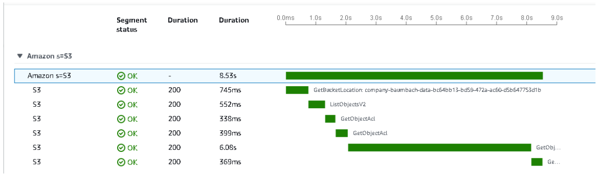


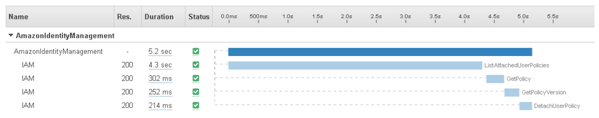
Figure 13: AWS S3 Response Time Distribution of Traces - Transient



	Segment status	Duration	Duration
▼ Amazon s=S3			
Amazon s=S3	✓ OK	-	8.53s
S3	✓ OK	200	745ms
S3	✓ OK	200	552ms
S3	✓ OK	200	338ms
S3	✓ OK	200	399ms
S3	✓ OK	200	6.08s
S3	✓ OK	200	369ms

0.0ms 1.0s 2.0s 3.0s 4.0s 5.0s 6.0s 7.0s 8.0s 9.0s
GetBucketLocation: company-baumbach-data-bc64bb13-bd59-472a-ac60-d5b647753d1b
ListObjectsV2
GetObjectAcl
GetObjectAcl
GetObj...
Ge...

Figure 14: AWS S3 Response Time Distribution of Traces -*Steady State*



Name	Res.	Duration	Status
▼ AmazonIdentityManagement			
AmazonIdentityManagement	-	5.2 sec	✓
IAM	200	4.3 sec	✓
IAM	200	302 ms	✓
IAM	200	252 ms	✓
IAM	200	214 ms	✓

Figure 15: AWS Response Time Distribution of Traces - *Transient State*

5.2. Performance Evaluation - AWS Environment

Steady-State The first experiments was to evaluate the performance time of CSBAuditor against the AWS environment. This analysis is critical since CSBAuditor employs an agent-less approach, by leveraging the cloud provider APIs. Distributed tracing [41] provides a detailed overview of requests and responses, which is useful for understanding application performance issues e.g. bottlenecks. Therefore, we deployed AWS XRAY [4], an AWS service that offers this feature. AWS XRay affords deep observability of interactions between third party tools and AWS services, as well as the interaction between AWS services. Hence AWS service is very beneficial for evaluating the performance of downstream calls that trickle down into the AWS services we are interested in analyzing: AWS S3 and AWS IAM. CSBAuditor, especially as the AWS Java SDK is used for implementation. Two experiments were conducted in the steady-state of the AWS environment. In this state, no workload was deployed against the test environment. CSBAuditor was executed to detect if there were changes to the state of the cloud assets. CSBAuditor ran continuously for 2 hours, using a scheduled *cron job* that sleeps for every 2 minutes, in between runs. AWS XRAY was configured to record all interactions between CSBAuditor and the AWS services. Thereafter, the results of the measurements are extracted and analyzed based on response times and latency measurements.

Transient State Essentially, it is imperative to evaluate the performance of CSBAuditor using a target in *transient state*. In order to translate the earlier introduced test environment into a transient state, we employ security fault injection techniques. By injecting security faults into the test environment, several cloud assets are changed from secure states to compromised states, similar to what a malicious attacker would do. These is achieved by causing random actions including creation, deletion and modification of resources in the cloud test-bed. The fault injection is implemented with CloudStrike, a tool we designed based on the principles of chaos engineering [53, 55]. CloudStrike employs three *chaos modes*: *LOW*,*MEDIUM* and *HIGH*, corresponding to random alterations in magnitudes of 30 %, 60 % and 90 % respectively, for these experiments, we used the *MEDIUM* chaos mode. We assume that CloudStrike has the correct privileges to perform all the required actions by providing appropriate access to the cloud test-bed. Based on this, CloudStrike makes API calls to retrieve the *deployed resources* and executes the *security fault injection algorithms*. In the following subsections, the results of the experiments for both the steady and transient states are described.

5.2.1. Response Time

The Response times is the time taken by CSBAuditor to conduct audit checks, detect and remediate misconfigurations. In the steady-state, the only the checks are conducted, with neither misconfiguration detection nor fixing. The response time, therefore, is the total time from the execution of the first check to the last check against the cloud test-bed. This procedure is executed continuously for 2 hours, then the measurements from AWS XRAY collected. Figure 12, which is based on 1000 traces captured, illustrates the response time for the steady-state. The overall response time is 350ms, about 90% of the requests were completed in 0.21 seconds, while the worst performance of about 5secs was observed for about 2% of the requests. In comparison, to the transient-state at the 50th percentile (Figure 13), requests are completed in 0.88 secs, though it takes a longer time, it is still performant. The mean response time was 14.63s, and more detailed illustrations of performance for selected rules is shown in Figures 11 and 12. The traces of the several API calls for the *iam_auditor ruletypes*. The rule executes 3 API calls: *ListGroupsForUser, ListAttachedUserPolicies*

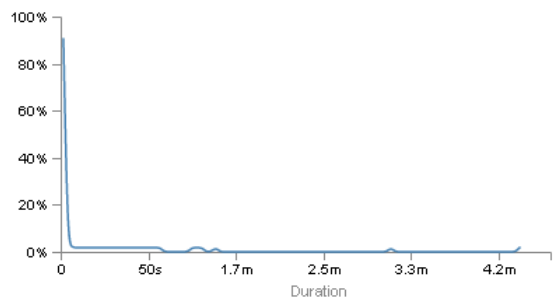


Figure 16: AWS IAM Latency -*Steady State*

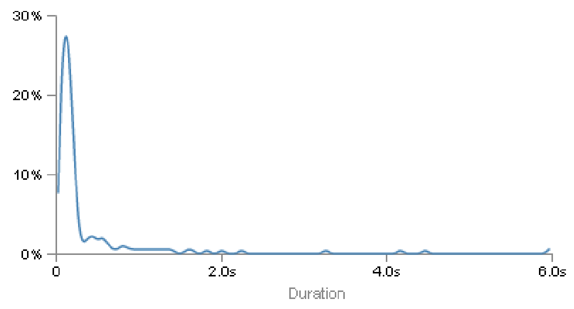


Figure 17: AWS S3 Latency Steady State - *Transient State*

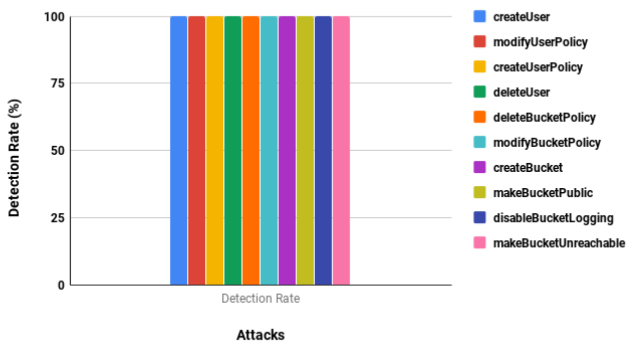


Figure 18: The Attack Detection Rate of 100% based on 10 Different Attacks Instrumented against CSBAuditor

and GetPolicy, the total time taken is 29.93s. 13 shows the response time distribution for the transient state

5.2.2. *Latency*

We used the *Latency Distribution* histogram to illustrate the performance of the API calls against the AWS IAM service. Based on the same number of traces above (1000). The histogram in Figure 13 shows that 90 % of the requests are completed in about 12 seconds.

5.2.3. *Intrusion Detection Rate*

The Intrusion Detection Rate (IDR) is used as a security metric for measuring the performance of CSBAuditor. The motivation for selecting the IDR is because the security category of CSBAuditor (CSPM) serves several purposes including detecting intrusions. In this case, intrusions are slightly different network layer intrusions, which are normally the detected by IDS and firewalls. Here *intrusions are evidence of unauthorized changes to cloud resources and their configurations* e.g. cloud buckets and policies respectively. This is especially relevant since traditional IDS are not suitable for deployment for cloud storage and IAM. Hence we IDR is a suitable metric evaluating the security performance of CSPM systems like CSBAuditor. IDR is the *ratio of total number of true positives to the total number of intrusions* [**?**], expressed in Eqn (4). To compute the IDR, the number of faults injected in the *attack phase* were noted. Afterwards, this was compared with the Alerts generated per

$$IDR = \frac{Total Number of True Positives}{Total Number of Intrusions} \tag{4}$$

experiment to determine if there were either false positives or false negatives. Initially, some false positives where detected due to an error in the retrieval of the cloud-state. The number of access control policies in the cloud were not comprehensively computed by the retrieval algorithm (*getActualState*). The reason was that the AWS API enforces pagination for IAM access control policies such that only the first 100 policies are retrieved for a *getPolicies()* request. To overcome this limitations, a pagination strategy provided by the AWS SDK was implemented to loop through all the policies in batches of 100, where 100 is the maximum number of retrievable policies.

5.3. Evaluation with CloudGoat - *Running Example Scenario*

To further evaluate CSBAuditor, we employed CloudGoat [44], a *vulnerable-by-design* framework provided by RhinoSecurityLabs. CloudGoat is recommended for cloud security testing by the CSA since it provides realistic scenarios based on contemporary cloud attacks. We used the *cloud_breach_s3* scenario, which replicates the cyber-attacks against Capital One's AWS infrastructure that lead to a huge data breach [36].

5.3.1. *CloudGoat Deployment*

The *cloud_breach_s3* scenarion involves the following cloud resources deployed using Terraform IaC templates: 1x EC2 VM, 1x Nginx reverse proxy, 2x security groups, 1x S3 bucket, and 1x Role.

5.3.2. *Attack Conduct*

After the deployment, we manually conduct the attack using the provided *cheat-sheet* and instructions. The attack sequence, illustrated in Figure 1 is as follows:

- *Step 1*: The initial *entry point* is a misconfigured reverse proxy, which we and leveraged to gain access the EC2 VM, where the reverse proxy server is hosted.
- *Step 2*: Next, a SSRF attack is launched against the

metadata server (ip address is 169.254.169.254) using the command in Listing 2. The command requests for the *Role* assigned to the EC2 instance. Following the response, the command in Listing 3 is issued with the name of the Role appended (*cg-banking-WAF-Role-cgidtvskgb6fy9*).

```
curl -s http://54.163.196.206/latest/
meta-data/iam/security-credentials/
-H 'Host:169.254.169.254'
```

Listing 2: Initial SSRF Request to get Role Assigned to the EC2

```
curl -s http://54.163.196.206/latest/
meta-data/iam/security-credentials/
cg-banking-WAF-Role-cgidtvskgb6fy9
-H 'Host:169.254.169.254'
```

Listing 3: Second SSRF Request with the Name of the Role Appended

- *Step 3*: The metadata server forwards requests in Listing 3 to the AWS IAM, and receives the a *Access Key ID*, *Secret Access Secret* and *Session Token* of the IAM Role attached do the EC2 instance (Figure 19).

  Based on these privileged credentials, we successfully executed arbitrary commands from our local system against the AWS environment using the locally installed AWS CLI. The Role is scoped with full access to AWS S3 resources. In this case, the policy grants read, write, delete, and list access to the AWS S3 service. Consequently, the access to S3 buckets is acquired, enabling ex-filtration of sensitive documents e.g. customers' email addresses, social security numbers and credit card information. The documents can be easily downloaded to the local machine using the AWS CLI using the command in Listing 4.

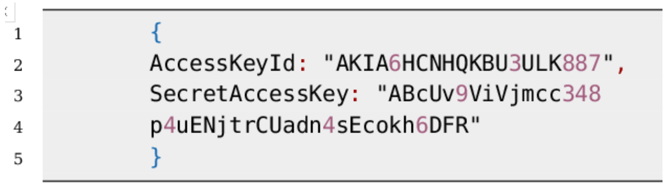
```
aws s3 sync s3://<bucket-name>
    ./<local-folder-name>
```

Listing 4: S3sync command for Exfiltrating Documents out of the AWS Environment

- *Step 5*: This information is ex-filtrated out of the cloud account, unhindered. Though the AWS Cloudtrail is enabled, there is no active monitoring and assessment of the events captured by Cloudtrail. Hence the attacker's activities through recorded are not detected.

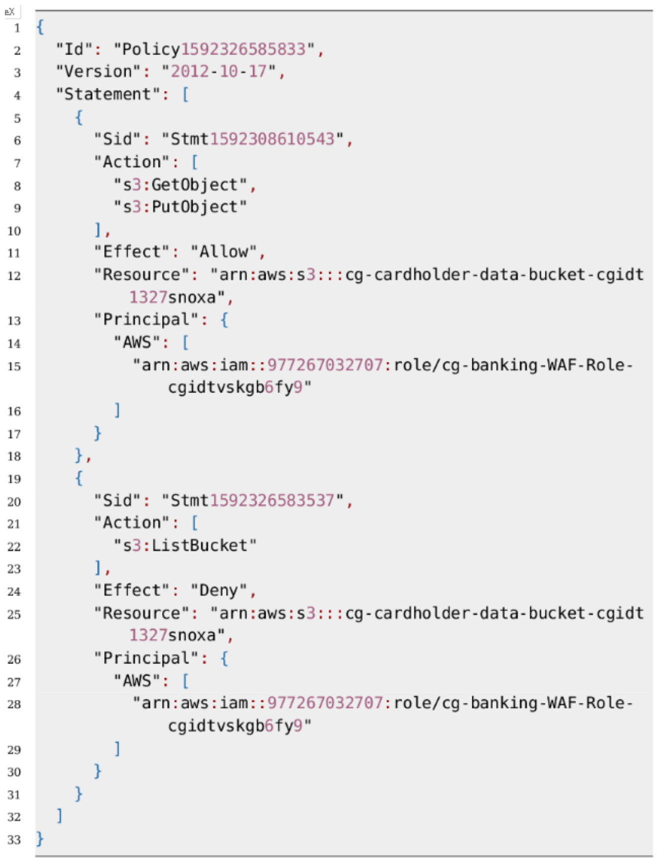
5.3.3. CSBAuditor Counter-measures

CSBAuditor is launched against the cloud account to detect and mitigate security issues. First, the initialization phase (Section 3.2) during which all the cloud assets are enumerated and an inventory of the cloud resources is established. The next step is to evaluate the discovered resources for security issues. During this stage, CSBAuditor uses its rules



```
        {
        AccessKeyId: "AKIA6HCNHQKBU3ULK887",
        SecretAccessKey: "ABcUv9ViVjmcc348
        p4uENjtrCUadn4sEcokh6DFR"
        }
```

Figure 19: Access Key ID and Access Key Secret for EC2 Role



```
{
  "Id": "Policy1592326585833",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1592308610543",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::cg-cardholder-data-bucket-cgidt
        1327snoxa",
      "Principal": {
        "AWS": [
          "arn:aws:iam::977267032707:role/cg-banking-WAF-Role-
            cgidtvskgb6fy9"
        ]
      }
    },
    {
      "Sid": "Stmt1592326583537",
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Deny",
      "Resource": "arn:aws:s3:::cg-cardholder-data-bucket-cgidt
        1327snoxa",
      "Principal": {
        "AWS": [
          "arn:aws:iam::977267032707:role/cg-banking-WAF-Role-
            cgidtvskgb6fy9"
        ]
      }
    }
  ]
}
```

Figure 20: AWS S3 Bucket Policy with Least Privilege Configuration

engine to check that the cloud resources comply with the cloud security best practices and the enterprise security policy.

- **Fix AWS S3 Level Permissions** - Here CSBAuditor runs several rules against the buckets using the *BucketAuditor* category of rules: (1)*public_bucket* to detect if the bucket is publicly available, the bucket passes since it is correct configured with the *PRIVATE ACLs*. (2) *bucket_logging* to detect if the all API calls and activities against the bucket are logged. The bucket passes this audit check since the logging feature is enabled. *BucketPolicy_check* - this is to check if *bucket level* permissions are configured. This ensures only fine-grained permissions that are assigned to a specific user or role are configured. The bucket fails this check since there is no policy attached, meaning the permissions are granted at the IAM level via Role. There-

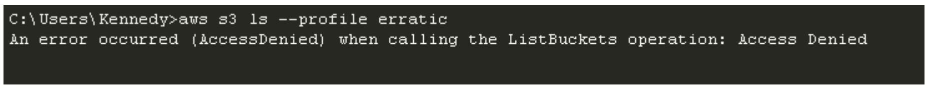
```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": "s3:*",
7              "Resource": "*"
8          }
9      ]
10 }
```

(a) Over-privileged EC2 Role Policy

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action": [
         "s3:GetObject",
         "s3:PutObject"
         ],
   "Resource": "arn:aws:s3:::cg-cardholder-data-bucket-cgidt132
      7snoxa"
      }
   ]
}
```

(b) Least Privileged EC2 Role Policy

Figure 21: Comparing Role Policies (a) Insecure and (b) Secure



```
C:\Users\Kennedy>aws s3 ls --profile erratic
An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied
```

Figure 22: Failure attempt of attacker to List S3 Buckets

fore, the *bucket level policy* in Figure 21a is configured and attached to the bucket. The policy has to important permission categories, the *Allow* (Line 11) and the *Deny* (Line 24). Due to the *Deny* permission, the *S3:sync* [6] command used for ex-filtrating documents is no longer possible as shown in Figure 22.

- **Fix IAM Level Permissions** - Next CSBAuditor launches the *UserAuditor* rules, which audits the users, roles and policies created in the AWS IAM service. Each of these resources is audited against secure baselines and *enterprise policies* e.g. ensuring the *principle of least privilege* is enforced. The statements expressed in the policies are checked, to identify privileges with wide permissions. For example in Figure 21a,we observe that the *Action* statement on line 6, states *"s3:*"*, where the wildcard means every action is permitted for the s3 service. Similarly, Line 7 has a wildcard value for the resource signifying all buckets and objects can be reached. Consequently, these misconfigurations are remediated in the policy on Figure 21b. Note only two actions are allowed *GetObject & PutObject*, against a specific resource: *Line 25 in Figure 20*. Due to this policy, the *list buckets command aws s*3 *ls* fails, as shown in Figure 22.

5.4. Discussion

5.4.1. *Continuous Auditing*

In the above evaluation, we used the Capital One data breach as a practical use-case to evaluate the efficiency of CSBAuditor. CloudGoat deployed the replicated environment of the attack scenarios, and CSBAuditor was deployed afterwards to detect the misconfigurations and mitigate the security misconfigurations that lead to the data breach. This was a reactive approach since the attack was orchestrated before the mitigation phase, however, in reality, CSBAuditor would be already deployed before the commencement of the attack, therefore most the attack would fail given that the *preconditions* for the attack would be detected and resolved beforehand. However, mitigating these preconditions is a short term effort since the attacker would attempt to find other entry points possibly based on the acquisition of other *preconditions* e.g. AWS and GCP access Keys. Hence, it would be desirable to follow the above measures with Incident Response (IR), where the event logs are retrieved and analyzed, Such an analysis will provide insights into the attacks paths employed by the user e.g. IP addresses used, API keys. The output of the information could be used proactively e.g. by adding the IP address to a blacklist. We have demonstrated some of these techniques in a Cloud IR system: *SlingShot* [54]. CSBAuditor alerts are forwarded to SlingShot for IR. These alerts are thereafter correlated with GCP Stackdriver logs [19] and AWS CloudTrail [3] to provide more contextual details, for accurate incident response.

5.4.2. *Limitation of Reversibility*

A limitation of our methodologies: *reconciler pattern and state transition analysis* is inability to reverse changes against some cloud resources. These resources include databases, object storage. Essentially, advanced efforts are required to achieve these since the actual content cannot represented in the *expected-state* using either IaC or IaS. Such advanced methods include redundancy methods such as backups and versioning of databases and object storage to facilitate restoration. At the core of our reversibility is the fact that an expected-state is created for every cloud resource. However, the expected-state is limited to configuration and meta-

[6] https://docs.aws.amazon.com/cli/latest/reference/s3/sync.html

data of cloud resources. Therefore, more stricter control for these resources is required e.g. stronger access control, encryption of contents in cloud buckets. These strategies can be enforced by the enterprise security polices introduced in Section 3.1.2.

5.4.3. Evaluation Challenges

CSPM is a relatively new category of security tools and there are very few implementation. Most CSPM are commercial and proprietary therefore it is quite difficult to compare these tools with CSBAuditor for evaluation reasons. Similarly, traditional security tools like firewalls and IDS are much easier to evaluate since their are public data sets available for conducting experiments. This is not the case also for CSPM. These two major factors pose challenges for deeper evaluation.

6. Related Work

This article is related to cloud security research in the context of security audit monitoring, SecCM and threat detection. In [60], *ConfEx* is proposed an framework for discovering and extracting text-based configurations in multi-tenant cloud platforms for configuration analysis and validation. ConfEx detects configuration defects in software applications e.g. *apache http*. Unlike ConfEx, we focus on the cloud infrastructure layer e.g. the interface between applications and Infrastructure-as-a-Service (IaaS) clouds. In [9], a framework for detecting security misconfigurations in AWS S3 such as misconfigured bucket ACLs is presented. The framework is tested against a wide range of AWS S3 infrastructure in several AWS regions. This is similar to our approach, with the difference that we fix detected issues and also integrate our framework into the lifecycle of cloud infrastructure to support automated and continuous cloud security management.

In [20], several cloud auditing strategies are studied, the authors recommend the use of automated risk analysis tools to improve cloud security. Similarly, Doelitzscher et. al [13] presented an architecture for automating public clouds audits and argued for the creation of a cloud audit policy language. However, the authors focus on auditing cloud VMs hence most of their techniques are not applicable to cloud storage and IAM. Mulazzani et'al [38] investigated the abuse of cloud storage by malicious users e.g storing malicious documents in cloud storage while preparing for attacks. Our focus is on the techniques for detecting, analyzing and managing risks against several cloud infrastructure. Rubsamen et.al [46] presented an agent-based auditing system that works by deploying agents in the cloud VMS. These agents collect and forward audit trails to designated endpoints for analysis. Ryan et.al [30] proposed *TrustCloud*, a framework for ensuring accountability and auditability in cloud computing via technical and policy-based approaches. Though some CSP already provide the audit trails and log information presented by the authors, efficient approaches for monitoring, retrieving the information as emphasized in the Cloud Accountability Life ycle (CALC) [31] are still lacking.

The concept of Risk Assessment-as-a-Service (RaaS) was explored in [29] as techniques that ensure continuous risk assessments such that *risk scores* of cloud tenants can be produced. This remains an open challenge, security metrics provided by CSPs are either qualitative or proprietary hence not easily integratable into open, well known and widely adopted systems. Therefore, in this article, we proposed CSSS, which is compatible with the CVSS. Almorsy et.al [2] argued for standardization of cloud risk assessment and compliance regulations, and mapping of security metrics to enable cloud security automation. The authors thereafter propose a collaborative risk model. The Cloud Security Automation Framework was proposed in [59] as a customer-centric approach for automating cloud security based on customer security requirements. This work is similar to our as it tackles user-centric security, however, it does not address the contemporary challenge of misconfigurations in cloud storage and IAM, but focuses on cloud compute.

The major difference between our work and related works is that we focus specifically on SecCM in order to tackle the current user-centric security challenges in the cloud as highlighted in [12] - [23] while providing risk assessments using appropriate security metrics. To the best of our knowledge, our work is the first addressing these contemporary cloud security challenges.

7. Conclusion

This article has presented *CSBAuditor*, a CSPM system designed to address the contemporary cloud security challenges including misconfiguration vulnerabilities, change control management and configuration drift issues that eventually lead to cyber-attacks and data breaches. *CSBAuditor* continuously monitors cloud infrastructure, to detect and remediate misconfigured resources, and malicious activities and unauthorized changes. This is achieved by leveraging two concepts: *state transition analysis* and *reconciler pattern*. The *reconciler pattern* enables tracking of cloud resources using *expected-state* and *cloud-state*, while the *state transition analysis* allows for discrete auditing of the difference between each version of the cloud-state to detect malicious events. Furthermore, a custom security reporting format is formulated to represent the detected vulnerabilities which are scored using a proposed cloud security metrics system - CSSS. Consequently, CSBAuditor overcomes the contemporary security issues caused by misconfigured cloud resources and other cloud-specific attacks. CSBAuditor has been evaluated using various strategies including *security chaos engineering* fault injection strategies on AWS and GCP, with realistic attacks including those based on the Capital One data breach.

8. Future Work

CSBAuditor's rules engine contains rules for detecting misconfigurations and malicious activities in cloud storage and cloud IAM for AWS and GCP. In the future, it will be useful to implement rules for other services e.g. cloud network, databases, Lambda serverless functions. Similarly, the performance might be improved by improving the caching and multi-threading, basic techniques of these were employed in this work. Furthermore, a closer analysis of the integrated CSBAuditor alerts and Security Information and Events Management (SIEM) alerts could increase the quality of incident response. As observed in the discussions section, there are limitations with our reversibility strategy that employs IaS. In order to provider better guidance for practitioners, such limitations must systematically handled, we intend to provider strategies to mitigate these in a future work.

References

[1] Alliance, C., 2017. Security guidance for critical areas of focus in cloud computing v4. 0. Cloud Security Alliance .

[2] Almorsy, M., Grundy, J., Ibrahim, A.S., . Collaboration-based cloud computing security management framework, in: Cloud Computing (CLOUD), 2011 IEEE International Conference on, IEEE.

[3] AWS, a. Aws cloudtrail-track user activity and api usage. Online. URL: https://aws.amazon.com/cloudtrail/. accessed 02 July 2020.

[4] AWS, b. Aws x-ray analyze and debug production, distributed applications. Online. URL: https://aws.amazon.com/xray/. accessed 02 July 2020.

[5] AWS, 2020a. Getting credential reports for your aws account. Online. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_getting-report.html1. [Accessed: 20 September 2020].

[6] AWS, 2020b. Shared responsibility model. Online. URL: https://aws.amazon.com/compliance/shared-responsibility-model/. [Accessed: 08 June 2020].

[7] Chang, W.Y., Abu-Amara, H., Sanford, J.F., 2011. Building and configuring enterprise cloud services 3, 2, in: Transforming Enterprise Cloud Services. Springer, pp. 273–308.

[8] CloudRAID, 2018. Cloudraid. URL: https://hpi.de/en/meinel/security-tech/cloud-security/cloudraid.html.

[9] Continella, A., Polino, M., Pogliani, M., Zanero, S., 2018. There's a hole in that bucket!: A large-scale analysis of misconfigured s3 buckets, in: Proceedings of the 34th Annual Computer Security Applications Conference, ACM. pp. 702–711.

[10] CSA, . Csa's perspective on cloud risk management. Online. URL: https://cloudsecurityalliance.org/artifacts/csa-s-perspective-on-cloud-risk-management/?utm_source=linkedin&utm_medium=post. accessed 05 September 2020.

[11] CSA, 2019a. Cloud penetration testing playbook .

[12] CSA, 2019b. Top threats to cloud computing the egregious 11. Cloud Security Alliance .

[13] Doelitzscher, F., Ruebsamen, T., Karbe, T., Reich, C., Clarke, N., 2013. Sun behind clouds-on automatic cloud security audits and a cloud audit policy language. International Journal on Advances in Networks and Services .

[14] Engebretson, P., 2013. The basics of hacking and penetration testing: ethical hacking and penetration testing made easy. Elsevier.

[15] Fitzgerald, B., Forsgren, N., Stol, K.J., Humble, J., Doody, B., 2015. Infrastructure is software too! Available at SSRN 2681904 .

[16] Forsgren, N., Humble, J., Kim, G., 2018. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution.

[17] Foundation, K., . Kubernetes. Online. URL: https://kubernetes.io/. accessed 02 July 2020.

[18] Garrison, J., Nova, K., 2017. Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment. " O'Reilly Media, Inc.".

[19] Google, . Gcp operations (formerly stackdriver). Online. URL: https://cloud.google.com/products/operations. accessed 02 July 2020.

[20] Gul, I., ur Rehman, A., Islam, M.H., 2011. Cloud computing security auditing, in: Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on, IEEE.

[21] Hashicorp, . Terraform. Online. URL: https://www.terraform.io/. accessed 02 July 2020.

[22] Ilgun, K., Kemmerer, R.A., Porras, P.A., 1995. State transition analysis: A rule-based intrusion detection approach. IEEE transactions on software engineering , 181–199.

[23] Institute, P., 2019a. 2019 cost of a data breach report. Online. [Accessed: 08 September 2019].

[24] Institute, P., 2019b. Shared responsibility in the cloud. Online. [Accessed: 08 June 2020].

[25] for Internet Security, C., 2018. Cis amazon web services benchmark. URL: https://d0.awsstatic.com/whitepapers/compliance/AWS_CIS_Foundations_Benchmark.pdf.

[26] for Internet Security, C., 2020. Cis google cloud platform foundation benchmarks.

[27] ISO, 2015. ISO:IEC 27001:2015. International Standard Organisation.

[28] Johnson, A., Dempsey, K., Ross, R., Gupta, S., Bailey, D., 2011. Guide for security-focused configuration management of information systems. NIST special publication 800, 16–16.

[29] Kaliski Jr, B.S., Pauley, W., 2010. Toward risk assessment as a service in cloud environments., in: HotCloud.

[30] Ko, R.K., Jagadpramana, P., Mowbray, M., Pearson, S., Kirchberg, M., Liang, Q., Lee, B.S., a. Trustcloud: A framework for accountability and trust in cloud computing, in: Services (SERVICES), 2011 IEEE World Congress on.

[31] Ko, R.K., Lee, B.S., Pearson, S., b. Towards achieving accountability, auditability and trust in cloud computing, in: International Conference on Advances in Computing and Communications, Springer.

[32] Luka, M., 2017. Kubernetes in action. Manning Publications.

[33] Lwakatare, L.E., Kuvaja, P., Oivo, M., 2015. Dimensions of devops, in: Lassenius, C., Dingsøyr, T., Paasivaara, M. (Eds.), Agile Processes in Software Engineering and Extreme Programming, Springer International Publishing, Cham. pp. 212–217.

[34] MacDonald, N., 2019. Innovation insight for cloud security posture management. Gartner Research .

[35] Majumdar, S., Madi, T., Wang, Y., Tabiban, A., Oqaily, M., Alimohammadifar, A., Jarraya, Y., Pourzandi, M., Wang, L., Debbabi, M., 2019. Cloud Security Auditing. Springer.

[36] McLean, R., 2019. A hacker gained access to 100 million capital one credit card applications and accounts. URL: https://edition.cnn.com/2019/07/29/business/capital-one-data-breach/index.html.

[37] Mell, P., Scarfone, K., Romanosky, S., 2007. A complete guide to the common vulnerability scoring system version 2.0, in: Published by FIRST-Forum of Incident Response and Security Teams.

[38] Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M., Weippl, E.R., 2011. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space., in: USENIX security symposium.

[39] Nguyen, S., 2019. Cloud security posture management: Why you need it now. Cloud Security Alliance .

[40] O'Donnell, L., . Is aws liable in capital one breach.

[41] Picoreti, R., do Carmo, A.P., de Queiroz, F.M., Garcia, A.S., Vassallo, R.F., Simeonidou, D., 2018. Multilevel observability in cloud orchestration, in: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/-CyberSciTech), IEEE. pp. 776–784.

[42] Pillar, R., 2018. Aws well-architected framework. Amazon Web Services , 45.

[43] RhinoSecurity, 2020. Pacu: The open source aws exploitation framework. Online. URL: https://rhinosecuritylabs.com/aws/pacu-open-source-aws-exploitation-framework/. [Accessed: 20 September 2020].

[44] RhinoSecurityLab, . Cloudgoat - cloudgoat is rhino security labs' "vulnerable by design" aws deployment tool. URL: https://github.com/RhinoSecurityLabs/cloudgoat. [Accessed: 06 August 2019].

[45] Rosenthal, C., Aschbacher, N., Jones, N., 2020. Chaos Engineering:

System Resiliency in Practice. O'Reilly Media, Incorporated. URL: https://books.google.de/books?id=UxuFxAEACAAJ.

[46] Rübsamen, T., Reich, C., Knahl, M., Clarke, N., 2014. An architecture for cloud accountability audits. BW-CAR| SINCOM .

[47] Scarfone, K., Mell, P., 2008. Vulnerability scoring for security configuration settings, in: Proceedings of the 4th ACM workshop on Quality of protection.

[48] Scarfone, K., Mell, P., 2010. The common configuration scoring system (ccss): Metrics for software security configuration vulnerabilities. NIST Interagency Report 7502.

[49] Schnjakin, M., Meinel, C., 2013. Implementation of cloud-raid: A secure and reliable storage above the clouds, in: International Conference on Grid and Pervasive Computing.

[50] Sukmana, M.I., Torkura, K.A., Graupner, H., Cheng, F., Meinel, C., 2019. Unified cloud access control model for cloud storage broker, in: 2019 International Conference on Information Networking (ICOIN), IEEE. pp. 60–65.

[51] Sukmana, M.I., Torkura, K.A., Meinel, C., Graupner, H., 2017. Redesign cloudraid for flexible and secure enterprise file sharing over public cloud storage, in: Proceedings of the 10th International Conference on Security of Information and Networks, ACM.

[52] Takabi, H., Joshi, J.B., Ahn, G.J., 2010. Security and privacy challenges in cloud computing environments. IEEE Security & Privacy .

[53] Torkura, K.A., Sukmana, M.I., Cheng, F., Meinel, C., 2019a. Security chaos engineering for cloud services: Work in progress, in: 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), IEEE. pp. 1–3.

[54] Torkura, K.A., Sukmana, M.I., Cheng, F., Meinel, C., 2019b. Slingshot-automated threat detection and incident response in multi cloud storage systems, in: 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), IEEE. pp. 1–5.

[55] Torkura, K.A., Sukmana, M.I., Cheng, F., Meinel, C., 2020. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. IEEE Access 8, 123044–123060.

[56] Torkura, K.A., Sukmana, M.I., Meinig, M., Kayem, A.V., Cheng, F., Meinel, C., Graupner, H., . Securing cloud storage brokerage systems through threat models(to appear), in: Advanced Information Networking and Applications (AINA), 2018 IEEE 32nd International Conference on, IEEE.

[57] Torkura, K.A., Sukmana, M.I., Strauss, T., Graupner, H., Cheng, F., Meinel, C., 2018. Csbauditor: Proactive security risk analysis for cloud storage broker systems, in: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), IEEE. pp. 1–10.

[58] Trendmicro, 2017. Data on 123 million us households exposed due to misconfigured aws s3 bucket. URL: https://www.trendmicro.com/vinfo/au/security/news/virtualization-and-cloud/data-on-123-million-us-households-exposed-due-to-misconfigured-aws-s3-bucket.

[59] Tunc, C., Hariri, S., Merzouki, M., Mahmoudi, C., De Vaulx, F.J., Chbili, J., Bohn, R., Battou, A., 2017. Cloud security automation framework, in: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W), IEEE. pp. 307–312.

[60] Tuncer, O., Bila, N., Duri, S., Isci, C., Coskun, A.K., 2018. Confex: Towards automating software configuration analytics in the cloud, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), IEEE. pp. 30–33.

[61] Upguard, 2017. Data leaks,cyber risk,and cstar URL: https://goo.gl/1WrUXZ.

[62] Vehent, J., 2018. Securing DevOps: Security in the Cloud. Manning Publications Co.

Kennedy A. Torkura received BSc in Computer Science from the Nigerian Defense Academy, Nigeria, and a MSc. degree in Cyber Security from the Lancaster University, UK. He is a cybersecurity doctoral candidate at the Internet Technologies and Systems Research Group of the Hasso Plattner Institute for Digital Engineering, Potsdam, Germany. His research focuses on security risk and threat analysis of cloud-native environments, security chaos engineering, and incident response.

Muhammad I.H. Sukmana received B.Sc. degree in Information Technology from Asia Pacific University of Technology and Innovation, Malaysia and M.Sc. degree in Communication Systems and Networks from Technical University of Applied Science Cologne, Germany. He is currently pursuing the Ph.D. degree at Hasso Plattner Institute, University of Potsdam, Germany. His current research interests include applied cryptography, cloud security management, and enterprise access control.

Feng Cheng received B.Eng. degree from Beijing University of Aeronautics and Astronautics, China, MEng. degree from Beijing University of Technology, China, and PhD degree from University of Potsdam, Germany. He is a senior researcher heading the IT Security Engineering (Sec-Eng) Team at Hasso Plattner Institute (HPI) at University of Potsdam, Germany. His research is mainly focused on Big Security Data analytics, network security, firewall, IDS/IPS, attack modeling and penetration testing, SOA and Cloud Security, SDN, etc

Christoph Meinel received PhD degree at Humboldt University, Germany. He is currently the President and CEO of the Hasso Plattner Institute, Germany and full professor at the University of Potsdam, Germany for Internet Technologies and Systems chair. His research focuses on Future Internet Technologies, in particular Internet and Information Security, Web 3.0, Semantic-, Social- and Service-Web, as well as on innovative Internet applications, such as e-Learning and Telemedicine.