Degree project in Computer Science and Engineering
Second cycle, 30 credits

A study of Oracle Cloud Infrastructure
Demonstration of the vulnerability or reliability of certain services through penetration attacks

SHANLY FELLER

Stockholm, Sweden 2023
A study of Oracle Cloud Infrastructure: demonstration of the vulnerability or reliability of certain services through penetration attacks

SHANLY FELLER
Abstract

This thesis aims to assess the security of Oracle Cloud Infrastructure (OCI) through penetration testing of some of its services. Targeted at cloud, cybersecurity, governance, and compliance professionals as well as administrators or cyber enthusiasts in general, this research uncovers specific best practices to OCI. We employ a methodology in three steps published by Astra aimed at cloud services auditing, combining penetration testing techniques and thorough documentation review to evaluate the security posture of OCI services. The scope encompasses IAM and MySQL Managed Databases. We found that unproperly supervised ABAC policies could lead to privilege escalation through the tagging of computing resources and that the MySQL service does not present the major issues that occurred in the managed services of OCI’s main competitors. This research contributes to the growing body of knowledge on cloud security and offers practical recommendations to strengthen OCI deployments, ultimately fostering greater confidence in adopting OCI services.

Keywords: Oracle Cloud Infrastructure, penetration testing, security practices, cloud security, vulnerability assessment.
Acknowledgements

As I conclude my master’s thesis, I cannot help but reflect on the journey that brought me here to an office in Stockholm, 2000 km from my birthplace.

First and foremost, I would like to thank my supervisors from WithSecure, Olle Segerdahl and Ingrid Waldheim for unwavering support, invaluable guidance, and continuous encouragement throughout this research.

My thanks also go to my supervisor from KTH, Nikolaos Kakouros for his guidance and reviews.

I would like to express my deepest gratitude to the technical help from the cloud security team at WithSecure, namely Nick Jones and Christian Phillippov. I would also like to extend my thanks to my lovely colleagues of WithSecure, Albert, Daniel, Johan, Kevin and all the others for their support and their availability to bounce my ideas.

This work would not have been possible without the resources and facilities provided by WithSecure.

I am profoundly thankful to Clara Lynn and Calle Svensson for the recommendation of this thesis without whom I would probably not be in Stockholm anymore. I would also like to thank Daniel Kruczek for the translation of the abstract in Swedish.

Finally, I would like to give heartfelt thanks to all my friends who have given me unconditional love, support, and reminded me that I am capable. Daniel Kruczek, Alexandra Wide, Maxwell Folkesson, I cannot express how grateful I am for your help in keeping me grounded, motivated, and focused on what truly mattered throughout the entire thesis journey, which also happened to be my first one. I want to express how grateful I am to PromenadorQuestern and everyone within to have given me a safe space to be myself and empowering myself as a young woman on the first steps of her career. It is with great emotion that I can say that you are still there as I am starting a new chapter and I look forward to celebrate my achievements with you in the future.

To my parents and my little sister, thank you for your unconditional support behind everything I undertake.

I am finally studying what I love and stepping onto a career that brings me joy, passion, challenge and playfulness, while serving my ideals of justice and moral. Thank you all.
# Contents

1 Introduction .......................... 1
   1.1 Research question(s) .......... 2
   1.2 Aim and targeted audience ... 2
   1.3 Contributions ............... 2
   1.4 Ethical considerations ...... 3
   1.5 Scope and Limitations ...... 3
   1.6 How to read this thesis ....... 3

2 Background .......................... 4
   2.1 Cybersecurity fundamentals ... 5
   2.2 Cloud computing fundamentals 8
   2.3 OCI architecture ........... 19
   2.4 Services and Infrastructure management 36

3 Related research ................... 42
   3.1 Access control misconfiguration 42
   3.2 Too trusted services ........ 45
   3.3 Flawed implementation of managed services 46
   3.4 Other known vulnerabilities . 48
   3.5 Cybersecurity insight summary 49

4 Methodology ....................... 50
   4.1 Chosen methodology .......... 50
   4.2 Additional considerations ... 54
   4.3 Choosing the scope .......... 55
   4.4 Types of Cloud penetration testing 56

5 Scope ................................ 58
   5.1 Preselection of the services ... 58
   5.2 Angle of attack ............. 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Legal considerations</td>
<td>59</td>
</tr>
<tr>
<td>5.4 Additional filtering</td>
<td>59</td>
</tr>
<tr>
<td>5.5 Conclusion</td>
<td>60</td>
</tr>
<tr>
<td>6 Attack plan</td>
<td>61</td>
</tr>
<tr>
<td>6.1 IAM</td>
<td>61</td>
</tr>
<tr>
<td>6.2 Oracle MySQL Managed Database</td>
<td>74</td>
</tr>
<tr>
<td>6.3 Attack plan</td>
<td>91</td>
</tr>
<tr>
<td>7 Results</td>
<td>97</td>
</tr>
<tr>
<td>7.1 IAM</td>
<td>98</td>
</tr>
<tr>
<td>7.2 Oracle MySQL Managed Database</td>
<td>124</td>
</tr>
<tr>
<td>8 Conclusions</td>
<td>135</td>
</tr>
<tr>
<td>Bibliography</td>
<td>138</td>
</tr>
<tr>
<td>A IAM</td>
<td>145</td>
</tr>
<tr>
<td>B SQL</td>
<td>149</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Customer versus CSP responsibility given the type of Cloud Service used. Source: [10] . . . . . . . . . . . . . . . . . . . 15
2.2 Compartments and tenancy. Source: [20] . . . . . . . . . . . 21
2.3 Ressources in the same compartment across multiple regions [20]. A realm is a collection of regions. . . . . . . . . . . . 22
2.4 Resource groups in AWS exist within one region although connections can exist between regions [21] . . . . . . . . . . . 23
2.5 Example of an OCID [4] . . . . . . . . . . . . . . . . . . . 25
2.6 View of the policy builder from the console . . . . . . . . . 29

6.1 Illustration of policies A and B applied to different sets of users 64
6.2 Sensitive bucket as seen from the administrator account . . . 68
6.3 Sensitive bucket as seen from the developer account . . . . . 69
6.4 Content of the bucket . . . . . . . . . . . . . . . . . . . . . . . 69
6.5 List of roles in GCP managed PostgreSQL service. Source: [43] 77
6.6 Permissions of "cloudsqlsuperuser" role. Source: [50] . . . . 77
6.7 Administrator parameters during database creation in OCI console . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
6.8 Illustration of the attack path for compute instances . . . . . 93
6.9 Illustration of the attack path for serverless functions . . . . . 94

7.1 View of the console from the user THESIS_USER after a change in policy within about 5 minutes. . . . . . . . . . . . . . . . 99
7.2 Illustration of the report issuer steps . . . . . . . . . . . . . . 104
7.3 View of the console from the administrator account. The file "summary.txt" has been uploaded in the folder "generated". . . 106
7.4 Illustration of the serverless function steps . . . . . . . . . . 107
7.5 View of the console from the administrator account. The file "secret.processed.txt" has been uploaded. . . . . . . . . . . . 108
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>Report of the tenancy from ScoutSuite, view of the Identity Dashboard.</td>
</tr>
<tr>
<td>7.7</td>
<td>Report of the tenancy from ScoutSuite, view of the policies.</td>
</tr>
<tr>
<td>7.8</td>
<td>Report of the tenancy from ScoutSuite, view of the warning &quot;Policy Affects User&quot;.</td>
</tr>
<tr>
<td>7.9</td>
<td>Report of the compartment &quot;Pwn&quot; from ScoutSuite, view of the Object Storage Buckets.</td>
</tr>
<tr>
<td>7.10</td>
<td>View of the bucket from the console. The metadata is readable but the content is not accessible.</td>
</tr>
<tr>
<td>7.11</td>
<td>Failed attempt at deletion of the bucket.</td>
</tr>
<tr>
<td>7.12</td>
<td>Metrics of the bucket. It gives information about uploads through time.</td>
</tr>
<tr>
<td>7.13</td>
<td>View from the console at database creation. The specified username for the user administrator is a banned username.</td>
</tr>
<tr>
<td>7.14</td>
<td>View from the console at database creation. The specified username for the user administrator contains a banned username.</td>
</tr>
<tr>
<td>7.15</td>
<td>View from the console at database creation. The specified username for the user administrator contains a banned username.</td>
</tr>
<tr>
<td>7.16</td>
<td>View of the console from the administrator account during Route Rule creation.</td>
</tr>
</tbody>
</table>
# List of Tables

2.1 List of possible verbs for a policy statement [27] . . . . . . . . 28
2.2 Cloud Service Name Equivalence . . . . . . . . . . . . . . 38

6.1 PostgreSQL sensitive predefined roles . . . . . . . . . . . . . . 80
6.2 Reserved Usernames [53] . . . . . . . . . . . . . . . . . . . . 82
6.3 Administrator Capabilities in OCI’s MySQL Managed Database
   Service [54] . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
6.4 List of ungranted privileges to the administrator account [53] . 96
6.5 IP addresses on our virtual cloud network . . . . . . . . . . . 96
Acronyms

OCI: Oracle Cloud Infrastructure
AWS: Amazon Web Services
GCP: Google Cloud Platform
OS: Operating System
GB: Gigabyte
USB: Universal Serial Bus
NIST: National Institute of Standards and Technology
API: Application Programming Interface
SaaS: Software as a Service
IaaS: Infrastructure as a Service
PaaS: Platform as a Service
IP: Internet Protocol (IP address)
CSP: Cloud Service Provider
IaC: Infrastructure as Code
S3: Amazon Simple Storage Service
EC2: Amazon Elastic Compute Cloud
IAM: Identity and Access Management
OCID: Oracle Cloud Identifier
SMTP: Simple Mail Transfer Protocol
**LIST OF TABLES**

**IP:** Identity Provider

**SSO:** Single Sign-On

**RBAC:** Role-Based Access Control

**ABAC:** Attribute-Based Access Control

**SQL:** Structured Query Language

**CI/CD:** Continuous Integration/Continuous Deployment

**SDK:** Software Development Kit

**CIA:** Confidentiality, Integrity, Availability (CIA triad)

**CVE:** Common Vulnerabilities and Exposures

**MITRE:** MITRE Corporation

**URL:** Uniform Resource Locator

**OWASP:** Open Web Application Security Project

**FAQ:** Frequently Asked Question

**OKE:** Oracle Kubernetes Engine

**UDF:** User-Defined Function

**SSH:** Secure Shell
Chapter 1

Introduction

Cloud computing has emerged as a transformative force in information technology, redefining how resources and services are provisioned and consumed. The shift has been driven by the migration of businesses from traditional local information systems to the cloud, thanks to scalability and cost-efficiency. Business-ready cloud solutions have now existed for two decades, starting with Amazon in 2006. In this context, Oracle Cloud Infrastructure (OCI) stands as a new provider in the landscape of cloud services, having existed since 2016, seeking to establish itself as a provider with a security-first design, unlike established competitors that did not take security considerations into account early on.

Despite its emphasis on security, OCI remains relatively unexplored in terms of rigorous penetration testing. The lack of academic or published research on OCI services is also intriguing. One plausible explanation for it is the scarce market share of OCI compared to its main competitors, AWS, Azure and GCP, who conjointly powers 66% of the internet. This may in turn deter research as investment seems better placed on the major providers, which in turn become more tested and more trustworthy in terms of security. On the other hand, already existing strong expertise on major cloud providers may not encourage organizations to consider OCI as an interesting option in the first place, which feeds the cycle and results in a lack of visibility for OCI on the market.

However, the multi-cloud approach is important to consider for resilience, limiting dependence on the business models and services of a single provider, risk managing and efficiency for businesses. Diversifying the offer for cloud providers helps maintaining flexibility.

This thesis resonates with a personal curiosity for cloud computing and a strong appeal for challenges. As cloud is at the center of every organisation’s
CHAPTER 1. INTRODUCTION

Problematic, I recognized its significance as a critical skill to master in order to be an efficient cybersecurity professional. This work provided an opportunity to learn about a subject that was so obscure to me before undertaking this work, and falls within the first stepping stones of my career plan.

This work was made possible through the support of WithSecure, a company specialized in cybersecurity.

1.1 Research question(s)

The goal of the thesis is to answer the following question:
"How secure are OCI services against misconfiguration vulnerabilities, past vulnerabilities discovered on other cloud providers, and accessible penetration testing?"

1.2 Aim and targeted audience

By answering this question, through penetration testing and study of the documentation, we aim to bring light on OCI’s stance on security first and bring more understanding of the platform’s maturity.

This work is targeted to a broad public from professionals to uninitiated enthusiasts. It provides valuable insights for cloud professionals, cybersecurity professionals, auditors, defenders working with cloud tenancies, but also tenancy administrators, devops and the Oracle Corporation itself as it aims to uncover good practices and considerations for OCI that could be incorporated into professionals’ security checklist. Moreover, this thesis also aims to apply pentesting on the cloud and make a cloud comparison with academic standards on which security enthusiasts and academia can draw inspiration from and build on.

1.3 Contributions

The following contributions are brought forward by this work:

- Analysis and testing of the cybersecurity model employed by OCI. Two services were studied and reviewed: Identity and Access Management (IAM) and Oracle MySQL Managed Database. IAM was found to be vulnerable to privilege escalation in particular environments and would
benefit in having more extensive documentation. MySQL presented no particular important vulnerability.

- The open source tools "ScoutSuite" and "oci-enum" were used, evaluated and in cases patched\(^1\), to perform cloud auditing of OCI. Both seemed insufficient for a professional and efficient use on OCI.

1.4 Ethical considerations

Responsible disclosure is a process followed in cybersecurity in order to safely report vulnerabilities. If vulnerabilities are found as results of our attacks, they will be disclosed responsibly according to Oracle’s policies.

1.5 Scope and Limitations

The thesis work has been limited by law considerations, prioritizing current cloud challenges and other technical concerns such as the inability to access certain services or the time-bounded nature of the exercise (see Chapter 5). For these reasons, the following services, first selected for their interest among the OCI offering, were not studied:

- Oracle Container Engine for Kubernetes (OKE)
- Oracle’s Mobile Cloud Services
- API Gateway
- Oracle Cloud Infrastructure Logging and Monitoring
- Serverless function eventing

1.6 How to read this thesis

Someone knowledgeable in cloud and security can go through sections OCI Architecture (2.3), and Related research (3) only, before focusing on the summary of the attack plan in Section 6.3 and the Results (7.1).

Chapter 2

Background

This chapter aims to give the reader context as well as technical background to understand the thesis. In the first part, we will cover cybersecurity fundamentals. The second part will study Cloud Computing basics. Then, technicalities and specificities of OCI recommended to understand the thesis will be studied. Finally, services and Infrastructure management will then be mentioned.
2.1 Cybersecurity fundamentals

In this section, we will study how information security models such as the CIA triad are used to secure a large amount of data in cloud tenancies. Then, as OCI is a cloud multi-service and multi-application provider, the concept of Common Vulnerabilities and Exposures will be presented.

2.1.1 CIA Triad

Data is very valuable to companies that host their infrastructure in the cloud. As Cloud services are used to store large quantities of data or information, the primary objective of cybersecurity within the context of cloud computing is to guarantee the security of this information. What specific attributes contribute to ensuring the security of such information?

To answer this question, the Center for Internet Security (CIS) has introduced the CIA triad as a framework. The CIS is a non-profit organization based in Washington whose aim is to protect entities against cyber threats. They are widely known and recognized in the world for their benchmarks and guides for security assessments. In particular, the CIA triad serves as a model to guide auditors in assessing and assuring the security of information.

The CIA Triad rests on three essential pillars, confidentiality, integrity, and availability. These three attributes are necessary to guarantee the security of information. The description of the CIA Triad can be found on the CIS official website [1]:

"The CIA Triad is a benchmark model in information security designed to govern and evaluate how an organization handles data when it is stored, transmitted, or processed. Each attribute of the triad represents a critical component of information security:

- **Confidentiality** – Data should not be accessed or read without authorization. It ensures that only authorized parties have access. Attacks against Confidentiality are disclosure attacks.

- **Integrity** – Data should not be modified or compromised in any way. It assumes that data remains in its intended state and can only be edited by authorized parties. Attacks against Integrity are alteration attacks.

- **Availability** – Data should be accessible upon legitimate request. It ensures that authorized parties have unimpeded
access to data when required. Attacks against Availability are destruction attacks.

Every cyber attack attempts to violate at least one of the CIA triad attributes."

The main concept behind the development of the CIA triad is the recognition of information as a crucial asset to protect. The CIA Triad aims to provide an exhaustive overview to delimit the measures on safeguarding information and the various attacks it needs to be defended against.

From this framework, we can deduce important challenges that cloud security faces such as maintaining tenant isolation, or maintaining high availability on an infrastructure that can be under consequent pressure due to the number of simultaneous customers.

The thesis will pay particular attention to confidentiality for the following reasons:

1. Breaches in confidentiality are the most straightforward type of attack for the general public to apprehend. The idea that files are accessed without authorization is a clear violation of security. This type of attack is frequently depicted as cyberattacks in the media (such as credential leakage), making it a familiar concept to most people and an effective way to communicate security risks.

2. Attacks on confidentiality tend to attract the most attention due to the considerable damage they can inflict, in terms of reputation damage, exposure of sensitive information, and privacy infringements...

3. Other types of attacks are typically more difficult to achieve in our particular context. Due to their large-scale infrastructure, over-flooding a cloud provider in the context of denial-of-service attacks would require a multitude of machines and files. In contrast, breaches in confidentiality are relatively easier to execute. Moreover, integrity concerns are often intertwined with confidentiality matters (although not systematically, tampering with a file can signify we have access to its contents).

2.1.2 Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures (CVE) [2] is a database of publicly disclosed vulnerabilities. The database is maintained by the MITRE corporation, a non-profit organization which aims to serve the public interest. In
that context, MITRE has a strong involvement in cybersecurity. The purpose of CVEs is to spread knowledge of vulnerabilities in a standardized manner. It helps actors prioritize their efforts thanks to the severity score and helps people defend themselves efficiently through documentation and common remediation. Each vulnerability documented as a CVE is identified by a CVE number. When delivering audits or threat reports, issues assigned with CVEs make it easier for cybersecurity professionals to communicate and mitigate issues.

The database is searchable per service and version. For example, one can find all known attacks on MySQL servers with versions anterior to 5.5. A service vulnerable to a CVE is very likely to be exploited, as the public knows about these issues and automatic scans exist for CVEs. For an auditor, the CVE database is particularly useful for identifying common existing attack paths and vulnerabilities in a service.

However, not all discovered vulnerabilities are CVEs. Some may be found through independent internet searches.

In the context of our work, as we are studying multiple versioned services, CVEs are a foundational base to study, to identify if Oracle has made necessary adjustments to protect its services against the most common threats.
2.2 Cloud computing fundamentals

Before we dive deep into the core subject of this thesis, general knowledge about cloud computing will be presented. This part will cover what Cloud Computing is and the current ecosystem of cloud providers where OCI fits in. The goal of this part is for the reader to understand the challenges that cloud computing is facing, and why security is a concern. A reader knowledgeable in cloud computing can skip this part.

2.2.1 Backend

Most companies are digitalized to some extent. The term “backend architecture” [3] refers to the core components that a company’s information system requires for its business to be operable. Components of the backend include hardware, storage, computing resources, or network infrastructure for employees to access the company network, for instance. The traditional model of backend architecture for companies was, for small companies, an on-site server room containing all the backend architecture. Larger companies could afford to displace their servers in remote data centers and connect to them through their on-site servers.[4] The emergence of cloud computing has completely shaken the way companies design the architecture of their backend. Nowadays, any small business can start operating with just a few personal user computers. Some larger companies do not have server rooms anymore. This resulted in cutting the costs of setting up a business. This is a revolution that has been permitted by the emergence of the cloud.

2.2.2 Virtualization & Shared Pool Concept

Cloud computing relies heavily on a technology called “virtualization.” Virtualization is a technology where computing resources are abstracted from the actual hardware physical component they rely on. In other words, virtualization is a logical computed representation (i.e. virtual) of hardware.

Here is a non-exhaustive list of physical resources that can be virtualized:

- CPUs
- RAM
- Disk space
- Network Infrastructure
The virtualization of processors works as follows: a computer may have multiple physical cores, for instance, four, but these cores can be logically divided, resulting in the operating system recognizing eight logical cores instead of four physical ones. The OS can utilize these logical cores just like physical cores because their functioning is identical. However, physical core resources are not always fully utilized. Logical cores offer the advantage of resource optimization since they rely on a shared physical core. The process of translating one physical core into two logical cores in an optimal way is made possible through virtualization.

In the same manner, disk memory can be virtualized. Thus, for example, a computer could see one big storage of 16 GB which is the virtual memory of two actual 8 GB physical memory modules. Virtual memory can be enlarged, reduced, and remapped while the same physical memory hardware is in use. This notion of grouping hardware resources and using parts of them is very connected to the notion of a shared pool in cloud computing as we will see later.

Virtualization plays a key role in cloud computing’s business model. When tenants rent hardware resources in the cloud, they are not renting the complete physical hardware but rather specific "portions" of it. For instance, if a tenant wants to rent 8GB of memory, they will be allocated a virtual memory space of 8GB from a much larger physical memory space owned by the cloud provider. If the tenant does not need the memory anymore, the memory is freed and can be allocated to a new tenant by the cloud provider. Similarly, when tenants request computing instances, they rent parts of a server that form an isolated environment exclusive to them. Other people can utilize available computing power when a tenant’s computer is not in use. Multiple tenants sharing the same server can achieve efficient resource allocation and eliminate idle time through isolation or distributed computing power reallocation, as well as cutting costs overall.

One of the challenges that cloud computing faces is achieving isolation between tenants while using the same shared hardware to reduce costs and optimize resource allocation. This is essential to provide tenants with a secure, isolated environment for their business operations to ensure safety and privacy.

2.2.3 Cloud Computing Definition

With the core concepts of cloud computing explained above, this section will present a definition of cloud computing, as defined by the National Institute of Standards and Technology (NIST).
In September 2011, the NIST defined cloud computing as: “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”[5]

The four deployment models range from public cloud to private cloud and will not be explained further.

The five essential characteristics are also defined by NIST [6] and are the following:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity or expansion
- Measured service

We will explain these concepts further now.

2.2.3.1 On-demand Self-Service

On-demand self-service echoes the “minimal management effort or service provider interaction” mentioned in the definition. On-demand self-service refers to the ability of users to provision the resources themselves without relying on any human interaction from the cloud provider. It means that customers should rely as little as possible on support and administrative hassles such as opening tickets online. This principle goes hand-in-hand with rapid provisioning as relying on support means usually high waiting time.

Concretely, to achieve this, cloud providers offer users self-service portals or APIs that users can leverage to provision, configure, and manage their resources instantaneously and with little involvement from the cloud provider. This allows businesses to rapidly answer changing needs.

2.2.3.2 Broad Network Access

Achieving broad network access is what makes cloud services ubiquitous. This means that users can access cloud services regardless of their location,
time, or the device they are using as long as they have an internet connection. Ubiquity has also led to the emergence of a new business model, such as software-as-a-service (SaaS) (see section 2.2.4), which allows users to access software applications over the internet rather than installing them on their device (such as online document editing and storage). Users can work remotely, but also collaborate easily and fast, which has greatly improved workflows, flexibility, and efficiency, but also cut software’s costs in general, improving accessibility of businesses of all sizes and means.

2.2.3.3 Resource Pooling

As we mentioned before, resource pooling has been made possible by virtualization (see 2.2.2). Multiple users share computing resources that are owned by a single cloud provider. This principle relies on tapping into a common pool of resources, which helps reduce hardware costs. When these resources are no longer in use, they are freed and returned to the common pool. This approach ensures that everyone receives exactly the resources they require, precisely when they need them. Users only pay for the resources they use, resulting in cost savings. Moreover, resource pooling decreases costs for all users and enhances flexibility, among other benefits. This contrasts with traditional computing models where users must purchase and maintain their physical hardware, which can be expensive and can result in underutilized resources.

2.2.3.4 Rapid Elasticity or Expansion

Users can scale their resources up or down very easily. A concrete example of scalability is the ability of users to copy entire compute instances without worrying about infrastructure or hardware concerns such as getting another computer, reinstalling a fresh OS manually, and restoring all needed software and data to have the instance fulfill its intended function. Being able to copy code, computing power, and configurations in a fast and easy manner makes businesses able to reply very fast to sudden changes, such as spikes in traffic toward their servers. Obsolete resources can also be shut down in a matter of minutes or seconds to save costs.

2.2.3.5 Measured Service

Measured service refers to the principle that resource usage is monitored to bill the customer accordingly. Different resources have different measurement parameters. For instance, storage will be billed based on storing size, while
serverless functions\(^1\) will be billed on the number of times of their execution. Different providers may have different billing strategies. Aside from billing, measured services can also help the customer understand the consumption of their resources detect anomalies, and alert users on signs of compromise. This is a core principle of cloud computing as having extensive information on their resources can help a customer understand when they are about to reach resource limits and thus scale up or down efficiently.

### 2.2.4 Types of Services

Cloud providers follow 3 service models. Different service models provide different types of control over the cloud provider’s infrastructure. The type of service model that a user will choose depends on the user’s needs.

#### 2.2.4.1 Software as a Service (SaaS)

According to NIST [7], SaaS is defined as “The capability provided to the consumer to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.”

To rephrase, SaaS refers to software applications delivered to users through the internet, compared with the traditional software application model consisting of downloading software on a local device to use them. Usually, all a user needs to access SaaS is an account and an internet connection on any kind of device. SaaS is probably the type of cloud service that most users have been exposed to while knowing very little about cloud computing overall. Document storage, editing, file sharing, and email services are all examples of SaaS. Popular SaaS are Google Docs, Microsoft Office 365, or project management software such as Trello or Asana.

#### 2.2.4.2 Infrastructure as a Service (IaaS)

NIST defines IaaS as: “The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing re-

\(^1\)One-purpose program delivered on-demand at execution, see 2.4.1.5
sources where the consumer can deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).” [8]

IaaS is the type of model that offers the most control and configuration possibilities to the user. The cloud infrastructure is not controlled by the user as this encompasses concerns such as hardware owned by the company, virtualization, optimization, etc. However, a user can leverage the cloud infrastructure to suit their own needs, through an API that is exposed to the user by the cloud provider. For instance, IaaS is very popular for building web applications: cloud providers provide certificates, public IPs, databases, computing power, and networks while developers can build and make their webserver accessible publicly very easily. IaaS can also be used to host companies’ entire information systems, which is an essential problem in digital transformation nowadays.

2.2.4.3 Platform as a Service (PaaS)

According to NIST, PaaS is “The capability provided to the consumer to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.” [9]

PaaS can be seen as a compromise between SaaS and IaaS, where a user needs to develop an application, but does not need control over the kind of OS that they are using to program, how their data is stored, or if the network configuration is suited to their needs. Heroku is an example of PaaS as it enables developers to develop, build, and run web applications entirely in the cloud. The user does not have to worry about if their web application is reachable and how, or how the network load is distributed, which would be concerns a developer would have using an IaaS.

To sum up, so far, companies moved from having dedicated all-paid-for hardware on-site or in data centers, to remote resources rented right to fit their needs. As the backend architecture model change, the way those resources are administrated and used have also changed, which entails potential misconfigurations from end-users and security issues.
2.2.5 Shared Responsibility

As we saw before, cloud providers can provide a combination of infrastructure, software, and managed services. Tenants, on the other hand, can configure their resources, upload data, or leverage the provided services in general.

Configuration possibilities need to be flexible enough to fit the needs of common use cases. However, this entails responsibility for the user to know how to configure resources adequately and securely. Let’s imagine that object storage by default is created with public access. If a tenant does not change the default configuration and happens to store sensitive data in the object storage that is inadvertently exposed to the public, who is responsible for the misconfiguration? The customer for not configuring the object storage properly or the cloud providers for not making a default as secure as possible? Many reasons could exist for a tenant to make this mistake in the first place: lack of information about the defaults, or knowledgable about the risks but rushed by quick deployment which leads to remaining with working but unsatisfying defaults, or even because the bucket was repurposed, and administrators forgot to change accesses from public to private.

Providers should make sure that the services they are offering do not contain security vulnerabilities inherent to the service they are providing. However, customers have a responsibility to use services correctly. A user also has to acknowledge that there are parts of the provider’s infrastructure that they do not know about in detail and cannot act upon to strengthen security. For instance, if a data breach happens in a SaaS document editing service that the user has been using, the user has very little room to prevent such an event as they do not have any leverage on how the data is stored and protected. This raises the notion of trust and responsibility between the two parties, called "shared responsibility," that can legally be specified in terms of agreements.

On whom relies the responsibility of building a safe tenancy globally? Different kind of *aaS entails different responsibilities from the concerned parties (see Figure 2.1). The more flexibility the cloud provider provides, the more the users have management and security responsibility. The more infrastructure the cloud provider is responsible for, the less the users need to be concerned about security.

IaaS provides the most flexibility. Large room for configuration entails that configuration concerns are up to the customer. However, cloud providers still have a responsibility to make it as secure as it can be for users.

This can be done by providing:

- Clear and extensive documentation with appropriate warnings
Regular sensitization, teaching, and seminars from internal professionals

Warnings when performing sensitive operations (such as changing a bucket from private to public)

Secure defaults

It is important to note that documentation and warnings are not sufficient to prevent customers’ mistakes as documentation will, in practice, rarely be read [11]. This is due to the cognitive complexity of the studied system and verbose documentation that can be an obstacle to productivity. Furthermore, warnings before validating an operation can only be shown in the cloud provider’s console (see Section 2.4.2). This means that users managing their cloud resources through the command line or third-party tools such as Infrastructure as Code, which is how it is usually done and recommended [12], will not be exposed to warnings. This emphasizes further the importance of providing the users with good default policies.

A concern that users can also have, is that relying on cloud providers to store data or host an entire company’s information system means depending on a third party. Users have the responsibility of making sure that the cloud
services they are using are trustworthy before using them. This can also be challenging to do as the user generally does not have all the information needed to make this decision. For instance, cloud providers’s architecture cannot be disclosed for intellectual property reasons, or because disclosing code can expose potential vulnerabilities. How can one trust a service they do not know about? Usually, cloud providers will publish official documents that account for their security. Oracle’s security document is available online [13]. But reputation, track record, seniority, and the amount of published research done on the provider are all factors that can improve trustworthiness.

To sum up, cloud computing has revolutionized the way business operates. I also had the opportunity to mention a few of the security challenges that cloud computing has to face: maintaining robust tenant isolation & providing resilience against end-user misconfigurations.

2.2.6 Cloud provider ecosystem

I will now examine the current landscape of cloud providers and explore Oracle Cloud Infrastructure’s position within this context. In this work, the IaaS part of OCI will be studied. Many different providers share the IaaS cloud computing scene. To name only OCI’s most famous competitors:

- AWS by Amazon
- GCP by Google
- Azure by Microsoft

Numerous cloud experts on this infrastructure in the world exist, and regular research is done on these three cloud providers. Studying these providers and what are the recent flaws found in them will give us direction on what could be important points of failure of OCI. This section aims to give background to the related research (3) and underline the importance of their impact.

2.2.6.1 Major cloud providers

2.2.6.1.1 AWS (Amazon Web Services) by Amazon  AWS was born in 2003 [14], although AWS’s most popular services, S3 (Object storage service) and EC2 (cloud computing service) (see Services in 2.4) came to be in 2006\(^2\). Giant companies such as Netflix, Apple, Facebook, and Slack rely on AWS.

At the time of writing (June 2023), Amazon owns 33% of the market. Its closest competitors, Microsoft and Google respectively follow behind Amazon with 23% and 10% \(^3\). This implies that about a third of the internet is running on AWS and that two-thirds of the internet overall is owned by the big three.

AWS is a very popular cloud offering. Its strong points are mainly due to its early start as a major cloud service and that the community has been learning about it since it appeared. As AWS grew, it continued providing services covering most cloud scenarios, good support, and user-friendly services. Today, many customers use AWS because of the already preexisting widespread knowledge of AWS in the computer science community which makes learning it easier and deterring, time-wise or money-wise, to consider other providers. AWS’s popularity and long-standing presence have also made it the target of extensive research and pentest, which makes its security reputation widely trusted through continuous improvement.

2.2.6.1.2 Azure by Microsoft

Azure was first announced in 2008 [15] as “A cloud computing operating system which was targeted at Business and Developers without additional coding” and entered the market in 2010. Microsoft aimed to create a platform that was easier to use compared to its competitors. Nowadays, AWS has still better support and more advanced features, although Azure secured a part of the cloud market.

2.2.6.1.3 GCP (Google Cloud Platform) by Google

GCP was launched in 2008 [16] and secured its place in the top 3 by its different designs such as introducing per-second billing or better integration with third-party services. It offers users the possibility of building apps from a range of easy to complex while leveraging the same services that Google itself runs on.

2.2.6.2 OCI (Oracle Cloud Infrastructure by Oracle)

OCI was first announced in 2016 [17]. Compared to the 3 cloud providers we just mentioned, OCI is very much a newcomer in the field of cloud computing. Oracle describes its cloud service in the presentation of its security architecture [13]: “Oracle Cloud Infrastructure (OCI) is a next-generation infrastructure-as-a-service (IaaS) offering architected on security-first design principles”. Its late arrival in the history of Cloud computing makes a strong

\(^3\)Available at https://aag-it.com/the-latest-cloud-computing-statistics/
marketing argument as, unlike first-generation cloud that did not integrate security as a concern from the start, OCI had the time to consider security very early in the design and avoid the design pitfalls that cloud architects made at the time: “First-Generation Public Clouds focused on the efficient use of hardware resources enabled by virtualization and use of a hypervisor. [...] Security sometimes wasn’t a foundational principle of this design because private data centers relied on perimeter defenses” [13].

Despite this strong point, OCI is still very little used. Often OCI will be counted as "others" in the presentation of cloud providers or statistics, as its market share is not significant enough to be presented separately. According to a report on cloud providers⁴, OCI ranks in the 8th position worldwide with 2% of the market share in 2023. However, Oracle registered a fast growth compared to its competitors that can mainly be explained by TikTok’s choice to use OCI as host⁵.

As major businesses choose to rely on OCI more and more, testing the security of the provider has higher stakes. However, research on OCI is for the moment quite scarce as investment seems better placed on the major providers, which turns the latter more tested and more trustworthy in terms of security. On the other hand, already existing strong expertise on major cloud providers may not encourage organizations to consider OCI as an interesting option in the first place, which feeds the cycle and results in a lack of visibility for OCI on the market.

---

⁴12 Biggest Cloud Providers by Market Share in the World - Yahoo Finance
⁵Press Release - Oracle Chosen as TikTok’s Secure Cloud Provider
CHAPTER 2. BACKGROUND

2.3 OCI architecture

The goal of this section is to give the reader sufficient background on Cloud and OCI Architecture to understand further work. Essential security concerns made public by Oracle will also be discussed. Although this part covers theory which is common with other cloud providers, it is recommended reading to grasp the specificity of OCI.

First of all, general terminology about account governance and organization will be studied. Secondly, the main identifiers that will be used to perform attacks will be explained. Then, important IAM concepts as well as their implementation in OCI will be detailed. The latter is the most important part of the background, as IAM will be studied in detail during the pentesting chapter, and contains specificities to OCI. Then, the main studied services of OCI will be presented. Lastly, infrastructure management will be mentioned. The goal of this final part is to understand the developing habits of cloud computing users and how they can impact the overall security of OCI.

2.3.1 Account governance/organization

Cloud providers have different terminologies for the organization of their resources. We can usually distinguish the common parts:

- A global account that is supposed to represent an organization, a company, or business as its whole,
- User accounts within that global account that are each associated with a person,
- Regions, i.e. the physical geographical region where resources are stored,
- Resource grouping, that can be seen as folders or shelves for resources so that organizationally related resources are in the same place.

In the following, the OCI terminology will be used. Other cloud providers’ terminology as well as a comparison will be mentioned when relevant.

2.3.1.1 Tenancy

According to Oracle Help Center, documentation [18],

“a tenancy is a secure and isolated partition of Oracle Cloud Infrastructure to create, organize, and administer your cloud resources.”
A tenancy is an account of the scale of an organization or a company. Resources in the tenancy are the company assets and they are organized in such a manner that mirrors the structure of the company in the physical world.

A tenancy is an isolated collection of resources. Resources from other tenancies, in principle, cannot access or be accessed from other tenancies. Cloud isolation is a core principle of Cloud Computing and ensures the good functioning of organizations and companies. One of the most critical vulnerabilities that can exist in cloud computing is the one that breaks cloud isolation.

The tenancy name is known by users who own an account in the tenancy of the company, as it is needed for users to authenticate. The tenancy name cannot be changed.

In AWS, a tenancy is more generally called an organization\(^6\). (GCP: organization, Azure: root)\(^7\)

### 2.3.1.2 Regions

Although the word "cloud" can sometimes be synonymous with “dematerialized” in the minds of many, cloud resources are very real and physical. Resources are physically stored somewhere in the world in data centers. Regions, in OCI, conceptualize this geographical localization. Regions are relevant when considering local laws and regulations, proximity, geopolitics, etc. Given where in the world the client data is stored, the data might be subject to different local laws and regulations. Clients might also have concerns about the predictability of server availability, influenced by geopolitical or economic conditions. Geographical proximity between end users and servers can also be a factor for cloud clients as high-speed connection is also a factor to consider when developing a product.

In OCI, a region \(^1\) is an isolated collection of one or more data centers that are connected and geographically close at an international scale.

The structure of these data centers is such that a hardware component failure of one part of the data center, such as the electricity supply, does not imply the failure of the whole data center. More information can be found at Oracle Compute Best Practices \(^8\).

Other providers use the "region" terminology as well.

\(^6\)https://aws.amazon.com/fr/organizations/faqs/


\(^8\)Available at https://docs.oracle.com/en-us/iaas/Content/Compute/References/bestpracticescompute.htm
### 2.3.1.3 Compartments

A compartment [20] is a grouping of organizationally related resources. For computer users, they could be similar to folders. Compartments are meant to facilitate organization, cost-tracking, or other managerial concerns but also limit privileged access efficiently. In OCI, the root compartment is the default compartment, also referred to as "tenancy" in policies. All newly created compartments are sub-compartments of the root compartment and follow a tree structure.

Figure 2.2 illustrates how compartments can be efficiently used for organization. In the tenancy, there are three compartments, Development, Test, and Production. Different compartments are managed by the development team, the test team, and the prod team. Within each compartment, one could very well imagine further sub-compartments grouping network resources separately from computing resources belonging to another compartment. Good organization of these resources will simplify the process of creating fine-grained access policies later on. This is because policies are designed to cover entire compartments of resources, rather than individual resources. For example, an administrator would not want testers in the test group to access resources in the production compartment and vice-versa. They will compartmentalize the resources and policies as so.

AWS equivalent of compartments is AWS Resource Groups. AWS resource groups can only exist within one region, while OCI compartments can group resources belonging to different regions. This implies that resources from any region can be accessed from another, as shown in Figure 2.3. In
Figure 2.3: Resources in the same compartment across multiple regions [20]. A realm is a collection of regions.

Azure, the equivalent is Resource Groups and in GCP the equivalent is projects. All resources in yellow belong to the same compartment. This difference is fundamental with AWS (see Figure 2.4). This entails that IAM resources and strategies affect all regions: a policy (see section 2.3.3.2) defined in the home region will affect other regions as well by default, which is an unusual administration posture [22].

2.3.1.4 Users

OCI defines a user as [23]

“An individual employee or system that needs to manage or use your company’s Oracle Cloud Infrastructure resources.”

Essentially, a user is an account belonging to a tenancy that enables a real-life person to access cloud resources. One user can have multiple sets of credentials, such as API tokens, SSH keys, or a password, used to authenticate to the same account.

2.3.1.5 Groups

Groups are a collection of users who share the same access privileges. Users can be associated with multiple groups. In OCI, permissions are usually group-defined, and in that case, groups are meant to represent the “role” or the function that grouped users share. Permissions will be developed more in section 2.3.3.2.
Figure 2.4: Resource groups in AWS exist within one region although connections can exist between regions [21]
2.3.2 Identifiers

Every cloud provider has a feature for uniquely identifying resources.

2.3.2.1 Oracle OCID (Oracle Cloud Identifiers)

Since humans are the end users of resources, they often prefer assigning names that hold significance to them, to facilitate organization and memorization of the resources. Reading the name of the resource can give the end user an idea of the resource’s purpose with ease. However, human-attributed naming is insufficient and inadvisable for executing operations as they can be guessable. Knowing a resource’s name or gaining knowledge about company naming conventions should not grant the capability to manipulate company resources.

In Oracle Cloud, this is the purpose of OCID. According to the documentation [24], OCIDs use this syntax:

\[
\text{ocid1.<RESOURCE TYPE>.<REALM>.[REGION].[FUTURE USE].<UNIQUE ID>}
\]

Listing 2.1: OCID structure

The majority of the OCID’s components are related to the background previously explained

- The first part is the version of the OCID which is currently always fixed to “ocid1”
- Resource type is the type of resource (i.e. compute instance, bucket, virtual network, etc.)
- Realm is always set to “oc1” and denotes commercial use.
- Region is the region
- Future use is, for now, blank and not used
- Unique ID is a randomly generated string unique for every OCI resource. The purpose of randomness is to protect against brute-force attacks.

The human-readable name is called the “display name” and does not have to be unique within an OCI for most of the resources \(^9\) as long as they reside in different compartments. An exception exists for resources that are

\(^9\)https://support.oracle.com/knowledge/Oracle%20Cloud/2447063_1.html
tenancy-scoped such as storage buckets. Even though they exist within different compartments, their definition is at the tenancy level and thus, their names need to be unique in the tenancy. Human readable names can be changed but OCIDs are immutable. When performing operations on resources, OCIDs are required, while display names are not. An example of OCID of a compute instance is presented in Figure 2.5. “Vistula-1” is the display name.

Other cloud providers have a way to identify resources uniquely. The AWS equivalent of OCID is Amazon Resource Names (ARNs).

2.3.2.2 Object storage namespace & buckets

For more information about buckets see 2.4.1.1.

The object storage namespace [25] is an identifier for object storage. The object storage namespace is immutable. The namespace is a random and unique string. The namespace for storage objects created since the early days of OCI is the tenancy name. Unlike other resources, buckets and objects are uniquely identified by names and not OCIDs. Details about naming generations are not important. The main point is that, depending on how old the tenancy is, the namespace can be more or less guessable or easy to brute force, which is relevant for our penetration testing in chapter 6.

Because buckets rely solely on the data of the object namespace, while buckets have to be placed in compartments, their names are required to be unique tenancy-wise, even though they belong to different compartments. However, unicity is necessary only within a region. Moreover, other tenants can have bucket names similar to any other tenants.

In that regard, OCI is different than other cloud providers, as AWS requires unicity in bucket naming across all tenants in a region.
2.3.3 Identity and Access Management

Individuals within a company possess various levels of access and authority based on their role in the organization. For instance, high positions in the hierarchy may be entitled to have access to company secrets. Developers should be able to develop code that is considered company intellectual property, but financial teams should not have access to it. The latter group though should instead be able to see expenses that can be considered sensitive information. From these examples, we can see an emerging need to define access control to the different organization’s assets. This separation must be done in the cloud tenancy of the organization as well.

IAM stands for Identity and Access Management (IAM) and constitutes a fundamental organizational concept. Because of the distinctive nature of the cloud reflecting real-life organizations, IAM is an essential feature of cloud providers. It is essentially centered around defining and enforcing policies that determine who can access what resources and under what conditions.

2.3.3.1 Least privilege principle

The least privilege principle is a principle in information systems where entities are granted minimal privileges for them to perform their function. The goal is to reduce the risks of attackers gaining access to sensitive information if they compromise low-level accounts. Applying the least privilege principle can also prevent attackers from moving upwards i.e. by performing privilege escalation to higher-level accounts. Having fine-grained policies and a good organization of groups and users can help achieve the least privilege principle.

2.3.3.2 Policies

By default, groups or users do not have any permissions. Newly created users still have some capabilities that can be disabled by an administrator [26]:

- Use Console password (native users only)
- Use API keys
- Use auth tokens
- Use SMTP credentials
- Use customer secret keys
These capabilities are meant to let users generate their credentials, as there are multiple ways to authenticate to the cloud provider.

For users to access resources, administrators need to explicitly define the appropriate policies. A policy is a group of one or more policy statements. In OCI, the structure of a policy statement is as follows:

```
Allow group <group_name> to <verb> <resource-type> in compartment <compartment_name>
```

Listing 2.2: Structure of a policy statement

From this syntax, we can remark the following:

- The statements always begin with the word allow. Policies only allow access, they cannot deny it. This implies that, by default, users cannot do anything and have to be granted access through policies.

- Policies apply to groups of users or all users. This implies that users need to belong to a group to be granted capabilities.

- Policies apply to compartments, which implies that resources identical in type, grouped in the same compartment, will be subject to the same level of access. This further underlines the importance of compartmentalized resource organization.

- Policies targeting a compartment can only be defined at the compartment level or parent compartments. This feature is meant to protect against privilege escalation in unrelated compartments.

Verbs are words representing a level of capability (see Table 2.1)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Types of Access Covered</th>
<th>Target User</th>
</tr>
</thead>
<tbody>
<tr>
<td>inspect</td>
<td>Ability to list resources, without access to any confidential information or user-specified metadata that may be part of that resource. Important: The operation to list policies includes the contents of the policies themselves, and the list operations for the Networking resource types return all the information (e.g., the contents of security lists and route tables).</td>
<td>Third-party auditors</td>
</tr>
</tbody>
</table>
read  Includes inspect plus the ability to get user-specified metadata and the actual resource itself.  
Internal auditors

use  Includes reading plus the ability to work with existing resources (the actions vary by resource type). Includes the ability to update the resource, except for resource types where the "update" operation has the same effective impact as the "create" operation (e.g., UpdatePolicy, UpdateSecurityList, etc.), in which case the "update" ability is available only with the manage verb. In general, this verb does not include the ability to create or delete that type of resource.  
Day-to-day end users of resources

manage  Includes all permissions for the resource.  
Administrators

| Table 2.1: List of possible verbs for a policy statement [27] |

Policies can have conditions applied with the WHERE keyword and cloud variables. It is important to note that condition matching is case insensitive, which is a default in common query languages like SQL. However, it can be problematic as resources defined in the tenancy are not case sensitive: different resources can exist with the same name if the casing is different. Oracle warns about the consequences of it in the documentation: "Condition matching is case insensitive. This is important to remember when writing conditions for resource types that allow case-sensitive naming. For example, the Object Storage service allows you to create both a bucket named "BucketA" and a bucket named "bucketA" in the same compartment. If you write a condition that specifies "BucketA", it will apply also to "bucketA", because the condition matching is case insensitive."

In OCI, defining policies in OCI resembles writing a sentence in English that is meaningful for a human. This differs from other cloud providers where specifying the different parameters of the policy statement is closer to code (see for example the AWS IAM User Guide\textsuperscript{10}).

\textsuperscript{10} https://docs.aws.amazon.com/IAM/latest/UserGuide/access_po
Here is an example of a valid policy statement.

**Listing 2.3: Example of a complete policy statement**

Allow group A-Admins to manage all-resources in compartment Project-A where target.compartment.id != 'ocid1.compartment.oc1..aaaaaaaaexampleocid'

Because it can be difficult to find the appropriate names to build policies, OCI has default pre-built policies for quick implementation (see Figure 2.6). Default policies are meant to suit the most common needs. They must be as safe as possible by default, as certain users might adopt these default policies and not necessarily modify them as their requirements become more specific over time, or from the start. This could result in excessively broad permissions.

Some services, such as Kubernetes, also need permissions to fulfill their functions. These can be activated by implementing a default policy very quickly. To cover most users’ needs, these permissions tend to be permissive by default, which could lead to a service with high privilege. For this reason, services with default permissive policies will also be studied in this work.

In summary, users compose groups, and users are given permissions through policies applied to the groups they belong to. Implementing this is called Identity & Access Management. An important factor that determines if a tenancy...
is secure is the accuracy of the implemented IAM. When users have too wide permissions compared to their attributed capabilities, or if their permissions convey more trust than their role in the organization, permission escalation could arise. Account compromise (for instance, an employee falling victim to phishing and leaking their credentials), should also have the least damage possible on the company by applying the principle of least privilege.

2.3.3.3 Authentication

Authentication is the process of validating an entity’s identity against a trusted database. The entity must successfully prove that they are the person they claim to be to perform the operations they are allowed to do or access services. Usually, this is done by entering credentials to log in to an authentication authority. The authentication authority verifies the credentials against a database and authenticates or not the requested user.

Different authentication authorities can be used in OCI:

- Features of OCI IAM itself

- Third-party identity providers (through OpenID connect/OAuth protocol)

Third-party Identity Providers (IP) – Third-party identity Providers are trusted entities that manage and store user identity information and perform authentication. The user has a unique set of credentials that the user authenticates with the identity provider. The IP validates or not their identity. If they are successfully authenticated, users are returned with temporary tokens that can be used to access services in OCI. Other cloud providers usually provide this possibility as well.

Third-party identity providers are connected to the concept of Single Sign Ons (SSO). SSO is a method in which a user accesses multiple platforms with just one pair of credentials, as the same Identity Provider will be integrated into different services. The concept of SSO is particularly relevant as people usually own a large amount of user accounts with different usernames and passwords.

SSO is still a security concern as it is a unique point of failure: compromising the SSO of a user means compromising all the accounts connected to this SSO. However, it is still recommended to use SSO rather than IAM native users in most cloud providers, for several reasons:
• SSO and third-party IP in general give short-lived credentials to users. Compromise of this short-lived credential limits the attack scope as the credential is revoked automatically after a certain time. On the other hand, native IAM uses persistent credentials.

• In a company, a cloud tenancy is usually one of the many services that a company will use (emails, building access, etc.). It can happen that when an employee is terminated, administrators forget to revoke credentials for cloud access. The use of SSO simplifies account management for administrators in general. See for instance Termination Best Practices11.

In this work, SSO won’t be studied. Authentication will be performed using the default OCI IAM. This choice is motivated by considering that it is more unsafe to use native IAM capabilities than third-party identity providers, as explained above. The intention is to evaluate the security of the tenancy in scenarios where the administrator makes unfavorable decisions.

2.3.3.4 Authorization & access control

When a user authenticates, the authentication authority determines your level of authorization, i.e. the user capabilities which include what services, resources, and accounts they have access to, and what operations they are permitted to perform on them. The process of gaining authorization is called access control. There are two methods of controlling the authentication & authorization process: RBAC & ABAC [28]. The difference lies in the way in which those permissions are attributed.

RBAC stands for Role-Based Access Control. Role-based means that a user is granted access based on the role that they have in the company. For example, HR workers might be attributed access to sensitive information about employees but not granted access to source code about the ongoing project being developed by developer workers. The exact opposite could be imagined for developers.

ABAC stands for Attribute-Based Access Control. Permission can be granted on multiple kinds of attributes: user attributes (i.e. what attributes the requesting user has, such as user ID), resource attributes (file name matching a certain rule, time of creation, compartment, etc.), or environment attributes such as

the current time. ABAC can be seen as a much more complex way of performing access control but permits in fine to write more grained rules [29].

Using RBAC or ABAC is a choice made by the administrators which is reflected in how the policy statements are written. RBAC and ABAC are not mutually exclusive and can coexist in a tenancy: some policies can be role-based, and others can be attribute-based.

Because of its complexity, ABAC is more challenging to do accurately as access rights can be more dynamic, and harder for an administrator to keep track of who has access to what (see related research at 3.1.3).

Moreover, in OCI, there is no current straightforward way of seeing what permissions a group/dynamic group has except looking at the policies one by one (see [30], "How can I tell which policies apply to a particular group or user?") or performing permission enumeration as we will do in our attacks.

This behavior is unlike other cloud providers, as they usually provide an API to list specific users' permissions.

Considering this, we have found it relevant in this work to study how ABAC in OCI is performed. In OCI, ABAC can be done notably with dynamic groups.

2.3.3.5 Dynamic groups

Dynamic groups [31] are groups defined by the data of a (or more) matching rule. When the matching rule is created, membership is automatic and retroactive: all resources making the matching rule true are added to the group, and every resource created follows the matching rule. When the resources are modified, such as the matching rule is not true anymore, resources are automatically removed from the group. Azure and GCP also possess this capability.

According to the documentation, matching rules can be constructed with any of these parameters:

"-

- **compartment ID** - include (or exclude) the instances that reside in that compartment based on compartment OCID
- **instance ID** - include (or exclude) an instance based on its instance OCID
- **tag namespace and tag key** - include (or exclude) instances tagged with a specific tag namespace and tag key. All tag values are included. For example, include all instances tagged
the with tag namespace department and the tag key operations.

- tag namespace, tag key, and tag value - include (or exclude) instances tagged with a specific value for the tag namespace and tag key. For example, include all instances tagged with the tag namespace department and the tag key operations and with the value '45'.
- resource.compartment.id - the OCID of the compartment where the resource resides
- resource.id - the OCID of the resource
- resource.type - the type of the resource"

Such a way of defining membership to a group is very time-efficient and highly scalable: as the system scales up or down, resources are automatically added or removed from the group of interest without manual intervention.

On the other hand, it requires careful supervision and organization from the administrators, as created resources are automatically added to a group if they fit the matching rule, not manually added by an administrator. Administrators are usually not the only users with permission to create resources, as lower privileged users also are (developers for instance), which means that lower privileged users can automatically add created resources to groups. Improperly supervised, this can lead to privilege escalation, as we will see in Chapter 6.

2.3.3.6 Defined-tags

Tagging associate metadata (i.e. strings) to resources to help describe and manage them. Good tagging strategies help fine-grain and categorize cost and consumption for Billing and Cost management.

Defined tags are specific to OCI. The primary purpose of using defined tags is to let individuals use tags predefined by administrators to align with their organization, such as billing split per purpose or department. This ensures that adequate departments are using appropriate tags. If users were to create their tags, there could be a risk of inadequate naming convention, leading to resource loss, as management might not be aware of the large number of different tags that exist. By employing defined tags, resources are systematically categorized according to organizational rules, promoting uniformity and effective resource management.
For example, if the organization has multiple cost centers and would like to generate bills per cost center, a tag such as "Finance.CostCenter=w1234" could be applied to the resources, where w1234 is an example cost center.

The structure of defined tags is as follows:

- tag namespace: a container, group, or collection of tags
- tag key: a tag label belonging to a tag namespace
- tag value: the value entered or selected by the user for the tag key.

As we saw earlier, defined tags can be matching rules for dynamic groups. Possessing use right for tags can imply giving permissions to arbitrary resources. This implies that using defined tags requires to be granted permission to do so by an accredited user such as an administrator as it is a privileged feature. This seems counterintuitive as tagging is usually an organizational management tool only. Moreover, such capability is not mentioned in the tagging overview possibilities [32]. An idle administrator might be tempted to authorize all use for all users and forget about it as policy-making changes.

On the other hand, free-form tags can be used unrestricted although they cannot be uniformed and controlled.

2.3.3.7 Instance principals/resource principals

As every resource needs explicit "allow" permission to use other resources in the cloud, in particular, compute instances need them too. In general, compute instances need to have access to other resources to perform operations on them such as reading and exploiting databases and object storage.

With standard IAM, to get those permissions, every instance would require to be associated with a user. This would entail generating user credentials for each instance, generating API key pairs, storing them in the instance, and using them to authenticate when running code. Storing hardcoded credentials is in general not very recommended, as code is usually meant to be shared either internally or publicly such as on platforms like GitHub. It is very common for developers to forget to remove the hardcoded credentials in the code or previous commits, leaking them on the public internet. This is why digging for hardcoded credentials on GitHub is so lucrative, to the extent that GitHub itself has a built-in tool for secret scanning, to prevent breaches.

Moreover, each instance would need its user credentials to differentiate instances for monitoring, which requires the administrator to create and manage
a large number of users, which is a lengthy process. This process gets in the way of fast scalability.

In that context, Instance Principals were introduced in 2018 [33], similar to Service Accounts in GCP. As defined by Oracle[34],

“Instance Principal is the capability in the Oracle Cloud Infrastructure Identity and Access Management (IAM) service that allows you to make service calls from an instance. With instance principals, you no longer need to configure user credentials on the services running on your compute instances or rotate the credentials. Instances themselves are a new principal type in IAM.”

With instance principals, every instance has an identity that can be dynamically retrieved. For example, in Python, authenticating an instance would look like adding this line in a code running on the instance of interest [35]:

```python
rps = oci.auth.signers.get_resource_principals_signer()
```

Listing 2.4: Recuperation of an instance principal on Python

“rps” is equal to the identity of the instance. This variable can be used further in the code to identify the instance. Instance principals have their permissions granted when instances are members of dynamic groups [36], and can only be members of dynamic groups, unlike regular users. Instance principals also work for serverless functions A.

The goal of instance principals is for the instances to abstract themselves of static credentials: in this way, an admin does not need to generate specific users and key pairs for instances to have the right to access services. Furthermore, this makes any code very portable and scalable. Provided that another instance belongs to an appropriate dynamic group, the same code will make the instance perform the same operation, even though the identity of the instances is different, as their identity is dynamically retrieved at the execution of the code. An administrator could very well duplicate the instance a thousand times without concern about code adaptation.

The disadvantage is that any user who has access to the instance can also impersonate it and access the associated rights. Unsupervised access to a privileged instance can therefore lead to privilege escalation as we will see in Chapter 6.
2.4 Services and Infrastructure management

Services are applications served through the cloud provider. Their purpose is to enable users to leverage computing resources to perform operations on their resources. They provide interconnection and relation between the existing resources while abstracting them from the need to write custom software.

Cloud providers usually provide base services that are very similar to what administrators would work with on self-hosted business infrastructure such as databases, containerization, storage spaces, access control, logging, etc. Most Cloud Providers implement the same type of services due to practical reasons for users as well as competitive considerations.

2.4.1 Providers key services

The following services are services that developers most commonly know how to use with proficiency and expect to find in Cloud Providers:

- Compute services:
  - Virtual Machines
  - Containers and Container Orchestration (e.g., Kubernetes)
  - Serverless Computing (Function as a Service)

- Storage Services: Object Storage, File Storage, Block Storage

- Database Services: Relational Databases (SQL), NoSQL, Database as a Service (DBaaS)

- Networking Services

- Security Services: IAM, Firewalls

- Analytics/Big Data

- Backup as a Service (BaaS)

- Monitoring and Logging Services

and many more.

While AWS has the most extensive catalog of services with more than 200+ services\footnote{https://aws.amazon.com/fr/what-is-aws/}, different Cloud platforms rely on different technologies or systems,
based on their design choices. As companies will usually rely on multiple cloud providers at the same time, similarly implementing services can help developers not make mistakes when migrating from one cloud provider to another. This is one of the reasons new ways of using OCI services will be particularly studied in this work as unexpected features can lead to misconfigurations and thus security exploits.

However, in a scenario where multiple cloud platforms are built upon the same foundational technology, any vulnerabilities within that technology could have a widespread impact across all these providers. Diversity can be an advantage.

To have more clarity through the plethora of names that can exist across providers, we present in Table 2.2 equivalent services between them.

Services are important components to pentest as the creation of said services usually predates the widespread use of Cloud computing. Service developers did not foresee the rise of cloud computing and thus are used differently than intended. For example, PostgreSQL databases were meant for a local installation on a server, where the default user was a fully privileged admin. PostgreSQL administrators could have the right for instance to read local files and execute them by default.

Nowadays, when cloud providers want to implement basic services in their cloud platform, services become managed to some extent. The goal is to abstract the user for maintenance, migration, optimization, or any kind of task that can get in the way of efficiency and prevent the user from using the core provided service straight away. For example, this means that for managed databases, the user does not need to know where and how the database is installed, if they have access to it. That implies that the user should not have access to the underlying OS that hosts the local database. Default administrator accounts provided by PostgreSQL therefore give too wide default permissions. An extra security layer needs to be added by the Cloud provider so that the users do not have access to the file system of the hosting OS by default. Every cloud provider made different implementation choices when adapting services into managed services.

The core idea of managed services is that they are very similar in functionalities to the services that users have used for forever, with an added layer of isolation or permission control in terms of security or access. This layer is not always perfectly implemented as we will see later in Chapter 3.
<table>
<thead>
<tr>
<th>Area</th>
<th>OCI</th>
<th>AWS</th>
<th>Azure</th>
<th>GCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>Compute</td>
<td>EC2</td>
<td>Virtual Machines</td>
<td>Compute Engine</td>
</tr>
<tr>
<td>Container</td>
<td>Container Registry</td>
<td>ECS</td>
<td>Container Instances</td>
<td>Kubernetes Engine</td>
</tr>
<tr>
<td>Container</td>
<td>Container Engine</td>
<td>EKS</td>
<td>AKS</td>
<td>Kubernetes Engine</td>
</tr>
<tr>
<td>Serverless</td>
<td>Oracle Functions</td>
<td>Lambda</td>
<td>Functions</td>
<td>Cloud Functions</td>
</tr>
<tr>
<td>Database</td>
<td>Oracle MySQL</td>
<td>DB, RDS</td>
<td>SQL DB, MySQL, PostgreSQL, DB for MySQL, DB for PostgreSQL</td>
<td>Cloud SQL, Cloud Spanner</td>
</tr>
<tr>
<td>Monitoring</td>
<td>APM</td>
<td>CloudTrail</td>
<td>Monitor</td>
<td>StackDriver (Monitoring, Logging, Error Reporting, Trace, Debugger)</td>
</tr>
<tr>
<td>Mobile</td>
<td>Mobile Cloud</td>
<td>Cognito</td>
<td>App Center</td>
<td>Cloud Tools for Android Studio</td>
</tr>
<tr>
<td>Identity</td>
<td>Identity</td>
<td>IAM</td>
<td>Azure AD, RBAC</td>
<td>Cloud IAM</td>
</tr>
<tr>
<td>Storage</td>
<td>Object Storage</td>
<td>S3</td>
<td>Blob Storage</td>
<td>Cloud Storage</td>
</tr>
<tr>
<td>API Management</td>
<td>API Platform</td>
<td>API Gateway</td>
<td>API Management</td>
<td>Cloud Endpoints</td>
</tr>
</tbody>
</table>

Table 2.2: Cloud Service Name Equivalence
2.4.1.1 Object storage

Object storage, also called buckets, is a data management service. An object is an individual file (such as a text file, an image, or a video), containing data (the content) as well as metadata containing the object’s attribute, such as time creation, author, etc.

Buckets contain collections of objects, in a similar fashion that folders contain multiple files in a file system. Object storage is commonly a service where sensitive data is stored.

The most popular example is S3 on AWS.

2.4.1.2 Databases

Databases are collections of data stored in the form of arrays meant for simple and quick querying and retrieval of information. Sensitive data can be found in database services.

2.4.1.3 Compute

Compute is a service leveraging hardware and processing power to perform operations and manipulate data. Computing can be done multiply, most commonly through virtual machines (compute instances), where the user has full control of the operating system or serverless functions.

2.4.1.4 Containers & Container Registry

Containerization is a development software practice where applications are developed on isolated packaged environments called containers. The purpose of containerization is to ship applications to different environments and ensure they run. Containers are deployed based on images containing the blueprint or template of the application called "image". Multiple independent containers can be deployed using the same base image. The container registry is a service whose purpose is to manage and store container images.

2.4.1.5 Serverless computing

Serverless functions are containers that are activated when triggered by specific events. Once the serverless functions fulfill their purpose, these containers are automatically terminated. Because serverless functions are temporary containers, they also depend on the container registry of the cloud provider. The term "serverless" originates from the fact that a developer does not need
to develop an application on a particular operating system, resulting in an apparent abstraction of the application from the underlying operating system.

Serverless are commonly exposed to external users through a reachable API. Manipulation of the data sent through this channel can cause exploits.

2.4.1.6 Logging

Logging records information about use, access, and performances on applications or services in the cloud tenancy. The purpose of logging is to detect anomalies and provide information for troubleshooting or monitoring.

2.4.2 Infrastructure management

Infrastructure management is the orchestration and management of the resources in the cloud tenancy. To achieve this, the cloud provider exposes API that can be leveraged by users for self-provisioning, and autonomous management of their resources. Infrastructure management can be done in numerous ways. The default one is through the graphical web application provided by the cloud provider called "console". However, the most common approach is to use Infrastructure as Code tools such as Terraform.

2.4.2.1 Console

The Oracle Cloud Infrastructure (OCI) console serves as the default web-based interface for users to interact with cloud services. Although valuable for its filtering and visualization features, the console is not the most common means of using cloud resources. Providers typically grant third-party tools access to their services for tasks such as Continuous Integration/Continuous Deployment (CI/CD), such as Infrastructure as Code or Software Development Kit (SDK).

However, as it is provided by the cloud provider and has a graphical interface, the console is usually beginner-friendly. The console is most suited for access to cloud resources by an individual.

2.4.2.2 Software Development Kit SDK (python)

The SDK provides libraries that permit developers to access the resources programmatically. It is useful for automated processes such as compute instances that need to access cloud resources.
2.4.2.3 Terraform/Infrastructure as Code

IaC tools are meant to ease the management of a large part of the cloud infrastructure and are usually targeted at administrators. The most common IaC tool is Terraform. IaC tools describe the environment as code which can ease tracking and understanding of resources and their connections.

For third-party tools such as Terraform to operate, users can generate API signing keys that let third-party tools authenticate on their behalf. These API signing keys need to be properly stored: a major issue is people storing their AWS credentials on public open-source repositories for example.

Although different ways API eventually lead to the same operations, each method has its advantages and disadvantages. The console has useful filtering and visualization tools that can be used for monitoring billing and expenses, SDKs are used to develop code that is going to be deployed on compute instances for example, and IaCs are used by developers with an administrative or architectural design role in the company to keep an overview on the structure of the infrastructure of the cloud of the company.

Different tools perform their checks when users perform operations on resources. For example, when destroying a bucket, the console prompts the user to enter the exact name of the bucket as a safeguard against mistakes or oversights. Moving a compartment into another also raises a warning that policies and privileges affecting the resources inside the compartment might change.

In Terraform, resource-specific checks are not performed. However any changes are reviewed and necessitate approval of the developer to proceed, unless the option “auto-approve” is used. No confirmation is needed for any operation with the Python SDK. As it is used in programs, no validation is performed to not block the execution of the applications.

It’s important to note that the console may not always signal sensitive actions with warnings. Therefore, the user should not rely on the console to avoid mistakes, but rather on proper governance posture and infrastructure management.

In this work, the console as well as Terraform will be used to perform the attacks. The Terraform code is accessible on the associated GitHub repository [37].
Chapter 3

Related research

State-of-the-art cloud research among multiple providers will be presented to understand the stakes and our possibilities for OCI penetration testing. Finally, I will recapitulate the challenges that cloud has to face.

3.1 Access control misconfiguration

Access control misconfiguration is a type of misconfiguration where policies fail to give the correct access rights to users, due to incorrect assumptions about how the environment works. These assumptions can be wrong because the provider implementing a flawed service or because administrators and developers not using the tool as intended. Generally, too wide permissions are given to users. This can lead to privilege escalation where attackers can, for example, have access to sensitive data that was not meant to be shared with them.

3.1.1 Incorrect assumptions for access control

Attack vectors exploiting this misconfiguration were found in AWS API Gateway Lambda Auth in 2022 [38]. The API Gateway lambda authorizer in AWS is a special lambda function that checks if calls to an API are authorized. The principle of this function is that a user makes an API call with an identification token that is caught by the lambda authorizer. Policies that determine access control are implemented in a third party of the user’s choice (IAM, Amazon Cognito, OAuth). The lambda authorizer checks against the third party what the implemented policies are, and returns a policy document that explicitly what API calls can the requesting user make. The issue with this policy doc-
ument is that these calls are given as a list of URLs where wild cards expand greedily, i.e., they don’t stop at slashes.

For example, with a policy where the API call

```
*/test/*
```

is authorized, the intention would be that any API call from any API at the “test” stage would be callable. For example,

```
myAPI1/test/get/foo/bar
```

and

```
myAPI2/test/post/hello/world
```

would be authorized calls. The issue is that statements such as

```
myAPI1/admin/test/
```

would also be authorized calls. This is possible because wildcards could be replaced by strings containing slashes.

AWS also failed to grasp how their service worked, as they wrote documentation that indicated they understood their policies were authorizing statements restricted only to the first two examples and effectively blocking the third one. There was also no possible manner of writing policies that would authorize statements such as the first and the second and blocking the third until the researchers disclosed this issue. Real-life examples on GitHub showed that developers were also making wrong use of these statements when creating policies, and therefore potentially giving too broad permissions to their users.

The takeaway from this research is that access control misconfiguration can be done for the following reasons:

- Service providers implemented a flawed and insufficient access control policy
- Documentation writers made incorrect assumptions about how the service was implemented
- Developers assumed the documentation was right.

It is worth noting that Oracle Cloud has an equivalent service for AWS API Gateway Lambda Auth called API Gateway, which also performs token identity validation.
3.1.2 Flawed service implementation by the developers

The use of AWS Amazon Cognito was also discovered to be wrongly implemented by developers, resulting in privilege escalation and confidentiality breaches [39]. Amazon Cognito is an identity platform and authentication service meant for mobile and web applications. The purpose of Amazon Cognito is to grant end users temporary credentials to consume other AWS services such as S3 and DynamoDB. Those temporary credentials are generated from identity pools.

The calling mobile or web application sends an identity pool to Amazon Cognito in the form of a uuid4 string. It gets AWS credentials in response, which the user can use to consume the services. Those credentials match the level of permission that the user has access to. An identity pool can have multiple access levels. For example, with the same uuid4, there can exist two access profiles: one that defines permissions for identified users, and one that defines permissions for unidentified users. Permissions can be enumerated when knowing the uuid4.

The finding of this research is that developers were hardcoding the uuid4 in their application. By decompiling apps found on, for example, Google Play store, or examining open-source code on GitHub, Riancho was able to find the uuid4s for these identity pools and gain access to confidential data. More than 13000 S3 buckets (which are not publicly exposed), 1,200 DynamoDB tables, and 1,500 Lambda functions were able to be accessed in this way. One of the reasons that developers were hardcoding the uuid4 is that there exists an (unofficial) tutorial for using Amazon Cognito that implements it in this way. Once again, existing documentation and global internet-wide knowledge about a service is important to use it securely.

OCI has an equivalent service of Amazon Cognito called Mobile Cloud Services.

3.1.3 Flawed implementation of access control by the cloud provider

AWS Bucket tags were found to be modifiable without the explicit permission to do so or with an explicit deny permission set [40], unlike as specified in the documentation. This is an attack vector, as tags can be used to grant permissions to tagged resources. For example, if there exists a policy that states that every resource with the tag “123” has access to sensitive buckets or database (which is useful when that said resource is an instance and needs data to com-
pute), and anyone can manipulate that “123” tag unrestrictedly, then a user that does not have access to buckets or database could attribute the tag a resource to access those restricted resources. Therefore, uncontrolled tagging such as the one described in the paper is an important security issue. The recommendation from the researcher is to avoid ABAC and use RBAC instead until ABAC becomes more mature.

ABAC can be implemented through tagging in OCI as well.

Regarding OCI, in 2022, Wiz found that OCI did not perform sufficient checks to authorize the attachment of storage volumes to a virtual machine [41]:

“Any unattached storage volume, or attached storage volumes allowing multi-attachment, could have been read from or written to as long as an attacker had its Oracle Cloud Identifier (OCID), allowing sensitive data to be exfiltrated or more destructive attacks to be initiated by executable file manipulation.”

In particular, this affected storage volumes from any tenancy which broke cloud tenant isolation. Although the documentation required specific permissions to attach storage volumes, they were not checked during the operation.

“Insufficient validation of user permissions is a common bug class among cloud service providers.”

### 3.2 Too trusted services

Trusted services are services for which policies are defined to have access to sensitive resources. For example, logging services require permission to monitor computing instances or buckets to log who uploaded or accessed a resource. Services related to containerization require permission to access the image repository to pull images. In general, cloud services require privileged permissions to perform useful functionalities. These permissions are generally set by the administrators, entailing that they are disabled by default. Cloud providers have default permission policies that administrators can easily activate. By default, those permissions can be wide. For instance, when activating OCI cloud logging, the default policy is that logging affects all resources in the tenancy, even though the administrator could only wish to monitor buckets. Thus, refining the policy to respect the least privilege principle requires additional work.

If an attacker can have access to those services, they can leverage them to use the permissions those services have access to.
3.2.1 Cloud logging exploit

Because companies usually use multiple cloud accounts, potentially from multiple platforms, cloud services can be used to aggregate data from different accounts for easier management such as gathering logs collected from different accounts. Those services require explicit permissions defined in all target accounts to perform their task. The company Wiz studied all the services that could be used to access other accounts in AWS [42]. They found that 3 services were vulnerable and allowed read-and-write operations into other tenants’ accounts.

For example, Cloud Trail, an AWS logging service was leveraged by an attacker to write in another tenant’s account. The issue is that Cloud Trail default policy has read and write access to S3 buckets when activated, which is not thought over by administrators. This is a feature for service logs to be able to be written in a storage space into other customer’s accounts, upon generation. This means that if two accounts set the appropriate policies to give each other access to Cloud Trail to make log aggregation effective, an attacker could have access to the logging service of another account and write in a bucket. Write operations are particularly dangerous as they can erase or replace data for instance.

Similarly, OCI has services that give the possibility to access resources across multiple tenancies and aggregate logs through Service Connector Hub.

3.3 Flawed implementation of managed services

As we mentioned earlier in section 2.4, Cloud providers ported popular services on their platform to improve compatibility with other services. The reason for using existing services in place of creating their own can be the following 1:

- The community has been using these services for a long period and shared knowledge of these services is particularly extensive
- Developers are used to this service/language and therefore already have the necessary skills to use the service and are less likely to make mistakes. It also augments the attractiveness of the platform as they do not have to spend time mastering a new service.

• These services are usually known to be quite performant, reliable, and secure in their intended purpose

• Developing costs and time for cloud providers are reduced

All these reasons are motivations for cloud providers to use existing services instead of developing their own. However, as original developers did not foresee the rise of cloud platforms, services developed before the cloud usually lack isolation between users or sufficient access control for users. The isolation layer has to be implemented by the cloud provider when porting those services.

In that context, the company Wiz studied managed PostgreSQL databases in multiple cloud vendors and found that it was incorrectly implemented [43], leading to being able to access databases of other tenants and thus breaking the cloud isolation principle.

“Its been shown that managed services are correct to popular open-source software that runs on top of vendor-managed compute instances.” “Some CSPs introduced these changes via extensions or custom configurations. Others went so far as modifying the source code of the PostgreSQL engine itself and maintaining their fork. All of these approaches can lead to unexpected security issues.”

Successful exploits were found both in GCP and Azure. In GCP, the exploit relies on the fact that ownership of PostgreSQL tables could be changed to the highest privileged user reserved to the Cloud Provider, meant to be inaccessible to any tenant. This change of ownership could then lead to executing functions stored in tables as the owner of the table, because of leveraging other features from PostgreSQL like the COPY command. This led to ability to read files and execute commands on the OS that hosted the database. This kind of privilege escalation led both providers to access to internal network as root and access cross-tenant resources. In Azure, the exploit relied on the fact that the user could create another user with higher privileges than the one they initially had.

Oracle Cloud Infrastructure chose to implement managed databases based on MySQL, which does not have a concept of table ownership. It is considered worthwhile to see if a similar issue could be found in OCI as well.
CHAPTER 3. RELATED RESEARCH

3.4 Other known vulnerabilities

3.4.1 Containers & Kubernetes

Kubernetes is an open-source platform for the automation of containers. Most cloud providers implement a Kubernetes service. Thus, Kubernetes vulnerabilities affect multiple cloud providers at once.

The most commonly exploited vulnerability in Kubernetes is container escape. Container escape refers to when an attacker gains unauthorized access to the underlying system from within a container running on a Kubernetes cluster, similar to the managed service escape explained above. A more detailed list of common vulnerabilities for Kubernetes can be found here\(^2\). Kubernetes incidents are usually due to misconfigurations although vulnerability exploitation is common too.

To limit the attack surface, Kubernetes recommends best practices adapted to most common cloud providers\(^3\). While not covered in the thesis, Kubernetes remains a critical target for penetration testing of cloud services.

3.4.2 Serverless computing & Eventing

The purpose of serverless is multiple. Commonly, serverless will be exposed to some extent to the internet. It can be done for multiple reasons such as exposing a public API. Some serverless functions require authentication. As serverless functions are exposed, vulnerable ones are a critical gateway to a cloud tenancy. An overview of the most common threats in serverless can be found here\(^4\).

\(^2\)https://www.armosec.io/blog/kubernetes-vulnerabilities-2022/
\(^3\)https://kubernetes.io/docs/concepts/security/overview/
3.5 Cybersecurity insight summary

Cloud computing faces many challenges, due to its nature, the rapid expansion of the technology and its complexity. Having mentioned briefly some of them throughout the background, We will recapitulate the key challenges that cloud computing faces:

- Achieving isolation between tenants while providing efficient resource sharing. Maintaining isolation guarantees the confidentiality of data.

- Providing resources while pressured by high simultaneous demand or sudden sharp changes in loads. Achieving this ensures the availability of data.

- Providing services (managed services, containers) without compromising the underlying structure on which the services run while relying on services that are not meant to function on cloud as they are.

- Diversifying the service offering, relying on different underlying technologies for services to not be subject mass cyberattacks, while managing to attract clients who can be intimidated with learning something new.

- Provide relevant warnings, comprehensive documentation and training as cloud providers are heavily reliant on the shared-responsibility model and cannot ensure the safety of the whole tenancy by themselves.
Chapter 4

Methodology

Traditional penetration testing usually occurs when a client has a particular item, device, application, or tenancy in a cloud provider for which they would like to audit its security. The aim of this work is not to audit multiple real specific tenancies belonging to a person and identify what is misconfigured to improve their security, but rather to evaluate the security of the service that OCI offers and test their robustness against possible misconfigurations. Because we do not study a finished product, application or environment belonging to an organisation with a defined use-case, most existing standards methodologies such as NIST or OWASP are not applicable in the context of our work as they are made to exploit set environments made by real-life idle users.

Cloud penetration testing depends on the Shared Responsibility Model (see 2.2.5), as all involved parties do not have the same responsibilities or leverage on improvement over different cloud components. This feature also makes it so that existing methodologies are unapplicable in our context as they usually consider one party only (user-end misconfigurations or issues inherent to the service or product), while we will study a combination of both. This will also impact our suggestions for remediation, as the responsibility for fixing issues can differ giving the type of found vulnerability.

4.1 Chosen methodology

We will follow a methodology in 3 steps published by Astra Security [44], an India-based cybersecurity company. They were awarded Most Innovative Security Company at the Global Conference on Cyber Security in 2017\(^1\). The methodology is as follows:

\(^1\)https://www.linkedin.com/company/getastra/
1. Understand the cloud service provider’s policies

2. Create a cloud penetration testing plan

3. Execute the plan

4.1.1 Understand the cloud service provider’s policies

This step aims to ensure that our penetration testing plan complies with Oracle Cloud Security Testing Policies [45] and the law. It will determine the type of tests and the scope of the tests that we can conduct.

4.1.1.1 Permitted Penetration and Vulnerability Testing

Some of the attacks, such as the ones on IAM, will concern voluntarily misconfigured tenancies. Such attacks fall into IaaS testing. Oracle’s policy regarding IaaS testing is the following:

"IaaS: Using your own monitoring and testing tools, you may conduct penetration and vulnerability tests of your acquired single-tenant Oracle Infrastructure as a Service (IaaS) offerings. You must notify Oracle prior to conducting any such penetration and vulnerability tests in accordance with the process set forth below. Pursuant to such penetration and vulnerability tests, you may assess the security of the Customer Components; however, you may not assess any other aspects or components of these Oracle Cloud services including the facilities, hardware, software, and networks owned or managed by Oracle or its agents and licensors."

Conformingly to this policy, we notified Oracle before this work. Oracle’s FAQ also premises further that permission is not needed as long as they are notified [46].

Regarding SaaS,

"SaaS: Penetration and vulnerability testing is not permitted for Oracle Software as a Service (SaaS) offerings."

Conformingly to this policy, attacks susceptible to abuse vulnerabilities for SaaS offerings will not be performed. However, they will be presented.
4.1.1.2 Oracle Cloud Security Testing Rules of Engagement

Penetration tests are built in compliance with Oracle’s Rules of Engagement [46]. In particular, in light of this rule:

"You must not attempt to access another customer’s environment or data, or to break out of any container (for example, virtual machine)."

In particular, this rule is affecting our work on services such as MySQL-managed databases and Kubernetes. For this reason, attacks that would be susceptible to breaking out of containers or accessing sensitive data will not be performed or performed with parameters that are not susceptible to disclosing personal information.

4.1.1.3 Responsible disclosure

If vulnerabilities are found as results of our attacks, they will be disclosed responsibly according to Oracle’s policies [46]:

"If you believe you have discovered a potential security issue related to Oracle Cloud, you must report it to Oracle within 24 hours by conveying the relevant information to My Oracle Support. You must create a service request within 24 hours and must not disclose this information publicly or to any third party. Note that some of the vulnerabilities and issues you may discover may be resolved by you by applying the most recent patches in your instances."

4.1.1.4 Oracle Cloud Functional Testing

"The purpose of functional testing is to validate features of Oracle Cloud services to ensure they meet particular functional requirements or specifications.

- You can perform functional testing using manual or automated tools.
- You can conduct functional tests to validate the main functions of the Oracle Cloud service to meet business requirements including usability, accessibility, and error handling." [46]
We assume that the specifications include specifications in the documentation. This entails that we have the right to check that detailed features in the Oracle documentation are correct. This includes ensuring that proper authorizations are validated during the execution of operations. Moreover, these operations will be written as such that, in the event of failure, they will not access sensitive information or break out of containers, all by previously defined detailed policies.

4.1.1.5 Conclusion

To recapitulate:

- Penetration testing of misconfigured (purposely) IaaS is allowed (for example IAM).
- Documentation compliance for SaaS will be tested, in such a way that, in case of improper checks, containers or data will not be broken or accessed.
- Lastly, attacks that are susceptible to going against the policy rules will be presented but not executed.
- Oracle will receive notification through the required administrative pipeline.

4.1.2 Create a cloud penetration testing plan

Because the services have different purposes and functionalities, different strategies will be followed. When testing for misconfiguration possibilities, the following strategy will be used:

1. Creating an asset. On a new tenancy, no assets are existing in the account. Thus, we will create an asset that represents sensitive data that a tenant wants to be secure.

2. Setting up a vulnerable infrastructure. The infrastructure is by design vulnerable and is meant to account for possible misconfigurations that administrators could be led to make. Reasons for misconfigurations can be multiple: bad defaults, missing documentation or a wrong intuition on the functionalities of some features that has been cultivated by a long-term exposure to similar tools.
3. Finding attack vectors and paths on that vulnerable infrastructure. Those paths are usually obvious to us as we voluntarily designed the infrastructure to be vulnerable to these specific attack paths.

4. Choose what kind of tools will be used for the attacks. Both automatic and manual tools will be considered.

5. Evaluating the success of the attack. The attack is considered successful if the attack can breach the confidentiality of the sensitive asset.

In the case where we are testing for the security of the service itself, the following attack plan will be used:

1. If there is a related research, understand the steps of the attacks that were in the related research of other cloud providers’ similar service.

2. Research on the mechanism and design of the managed service in the research using documentation

3. Research on the mechanism and design of the managed service in OCI using documentation

4. (Optional) Setting up a vulnerable infrastructure. This step is optional as the service in itself can be vulnerable, but the service could also be vulnerable due to bad implementation from the developers. When applicable, we will attempt to reproduce misconfigurations that developers could perform.

5. Finding attack vectors and paths on that vulnerable infrastructure. Those paths will be close to the ones that are presented in the related research. New attack paths might be introduced from CVEs or ideas.

4.1.3 Execute the plan

Execution and result of the attacks will be presented in part 6.

4.2 Additional considerations

Additional considerations to the methodology were taken into account based on a Cloud Penetration Testing guide published by Guidepoint Security [47].
• "Work with an experienced provider of cloud penetration testing". The role of WithSecure in that context, the partner company for our work, is to provide us with experts in Cloud Security available for our consultation.

• "Understand the Shared Responsibility Model" covered in part 2.2.5. This step helps us understand which areas are relevant to look at and what point of view we should adopt towards a certain service: should we play the role of an unaware developer or should we try to evaluate the posture of OCI in a particular service?

• "Define the scope"

• "Determine the type of testing", i.e. black-box, grey-box, or white-box.

### 4.3 Choosing the scope

To determine the scope of our study, the following methodology was used:

- Study of discovered vulnerabilities in other cloud providers. This part helped pinpoint which services in OCI are susceptible to present vulnerabilities.

- Study of OWASP Cloud-Native Application Security Top 10. This part helped find out the intended use cases of services and what are the most common threats to services, and thus what angle of attack we should adopt for our pentesting.

- Finding equivalent services in OCI. This step resulted in a preselection of the services in OCI that are going to be studied.

- Studying Oracle Cloud Security Testing Policies [46]. After the services preselection, this part determined which services we were allowed to work on.

Although we are not allowed to perform some of the attacks because of the Security Testing Policies, attack ideas will still be presented in this work.

#### 4.3.1 OWASP Cloud-Native Application Security Top 10

Open Worldwide Application Security Project (OWASP) is a project which aims at raising awareness of security. The scope of the Top 10 project was
originally on web applications. As applications nowadays are more and more
developed natively on Cloud, a top 10 project about Cloud-Native Applications
was created. The top 10 lists the most common threats to these applications
[48]. The project is still ongoing. Here is the list as updated of April 2022:

1. Insecure cloud, container, or orchestration configuration
2. Injection flaws (app layer, cloud events, cloud services)
3. Improper authentication & authorization
4. CI/CD pipeline & software supply chain flaws
5. Insecure secrets storage
6. Over-permissive or insecure network policies
7. Using components with known vulnerabilities
8. Improper assets management
9. Inadequate ‘compute’ resource quota limits
10. Ineffective logging & monitoring (e.g. runtime activity)

The review of the top 10 will help us determine the Scope in Chapter 5. The
relevant threats will be chosen based on the intended use case of the services
and the security posture we should evaluate for a given application.

4.4 Types of Cloud penetration testing

Types of penetration testing are classified given how much information and
initial capabilities the attacker has on the targeted system. There are three
types:

- Black-box Penetration Testing: The attacker has no access to the target
  or prior knowledge. Accessible information such as the one that can be
  found online about the system is very limited.

- White-box Penetration Testing: The attacker is granted the highest level
  of privilege to the target and full access. Information such as source
  code might also be available to the attacker.
• Grey-box Penetration Testing: The attacker has partial knowledge of the target and may have partial administrative capabilities.

As we are attacking OCI’s services, we are choosing to have full access to a tenancy we created, in compliance with the Oracle Cloud Security Testing Policies [46]. The penetration testing can be described as white-box from the point of view of the tenancy as we have access to everything within the tenancy (resources, user accounts, administration privileges of the tenancy). White-box attacks will be carried out for some services such as the IAM.

However, we do not have access to the source code of the managed services provided by OCI or full privilege access to the managed services, although we have some administrator capabilities. In some attacks, the testing will thus be classified as grey-box.
Chapter 5

Scope

The scope was chosen according to the methodology 4.3. This chapter aims to select which services to study, ensuring that the angle of approach aligns with the relevance of the known challenges within the cloud community, complies with the law, and considers our personal resources.

5.1 Preselection of the services

After the related research (see Chapter 3), the following services were considered relevant to the study:

- ABAC on Computer instances and serverless functions (IAM)
- Oracle Container Engine for Kubernetes (OKE)
- Oracle’s Mobile Cloud Services (Amazon Cognito equivalent)
- API Gateway (equivalent of AWS API Gateway Lambda)
- Oracle Cloud Infrastructure Logging and Monitoring (equivalent of CloudTrail)
- Serverless function eventing
- Oracle MySQL managed database
5.2 Angle of attack

5.3 Legal considerations

Considering Oracle Cloud Security Testing Policies covered in section 4.1.1.2, attacks aimed at escaping managed services or containers will not be attempted due to the following rule[46]:

"You must not attempt to access another customer’s environment or data, or to break out of any container (for example, virtual machine)."

Services that fall under such conditions are Kubernetes and Oracle MySQL. Kubernetes will not be studied at all. MySQL attacks are susceptible to break containerization and will be presented but not executed for substantiality reasons.

5.4 Additional filtering

Studies of some services were abandoned due to other factors.

5.4.1 Accessibility

Mobile Cloud Services is a service that is not available to an OCI account by default, even with an upgraded paid account. For the service to be accessible, Oracle’s sales services needed to be reached to get an invoice and access to usage. The Mobile Cloud Services study had to be abandoned, as this service is usually sold to an entire organization/company, and not an individual. Sales determined that thesis work was quite an unusual reason to purchase the service and did not fulfill our request.

5.4.2 Time management

The study of some other services, however interesting, were chosen to not be studied, as the current work is to be carried under a fixed period (6 months). To have a consequent amount of attacks to be presented while fitting the schedule, two final services were selected, by choice of preference.
5.5 Conclusion

The above considerations and the study of OWASP Cloud-Native Application Security Top 10 in Section 4.3.1 resulted in the study of the following services:

- IAM
- MySQL managed database

5.5.1 IAM

The Top 10, item 3 "Improper authentication & authorization", as well as the related research indicates that creating a set of permissions in such a way that IAM cloud roles are not over-permissive is a challenge. ABAC on computer instances and serverless functions will thus be studied with this goal in mind. The objective will be to design an infrastructure and study if and in what way IAM cloud roles can be over-permissive with ABAC posture when designing access policies.

5.5.2 Oracle MySQL managed databases

As explained in the related research when studying the PostgreSQL services in other providers (see 3.3) one of the challenges that managed services can face is an attacker breaking out of the managed part of the service and accessing the underlying infrastructure meant to be reserved to the cloud provider. This type of flaw is also related to item 1, "Insecure cloud, container or orchestration configuration", as the attack means to escape the scope of the managed service. The attacks will aim to determine if escaping this scope is possible or not in managed databases.
Chapter 6

Attack plan

In these section, the specific retained services, IAM and MySQL, will be studied in more detail. Attack plans will be presented for each services. The aim of this section is to understand our reasoning to form the attack plan for each service.

6.1 IAM

The objective of this section is to understand how the vulnerable infrastructures were built and that errors are likely to be repeated by users.

To understand our possibilities of pentest for IAM, we use the methodology described in 4.1.2.

Two aspects of IAM were reviewed. The first one is the documentation: as we were studying it, we encountered some parts of the documentation that were considered unclear or incomplete. It is important to have a sound documentation when implementing policies, as the documentation details best practices and ensures a safe tenancy. The second one is the implementation of ABAC policies through the use of tagging. Because we saw in the related research that ABAC could have flaws and that defined-tags are a novelty of OCI (see 2.3.3.6), we wanted to imagine scenarios in which they could be misused and abused.

6.1.1 Documentation

6.1.1.1 Giving permissions to a specific user or restricting a specific user

One can very well imagine a scenario where a single user belonging to a group needs either special access to a resources or needs to be restricted from it.
The documentation [30] details that:

“If you need to grant access to a particular user, you can add a condition to the policy that specifies the user’s OCID in a variable. This construction restricts the access granted in the policy to only the user specified in the condition. For example:

allow any-user to read object-family in compartment ObjectStorage where request.user.id='ocid1.user.oc1..<user_OCID>,'

Listing 6.1: Policy statement which allows one specific user only to read objects in the compartment ObjectStorage

[...]

If you need to restrict a particular user’s access, you can:

• Remove the user from the particular group of interest
• Delete the user entirely from IAM (you have to remove the user from all groups first) ”

What is interesting about this quote, is that, in the case where an administrator would need to restrict a particular user’s access, the documentation does not advise to use "!=", which seems like the most natural way to build such a policy, like so:

allow my-group to read object-family in compartment ObjectStorage where request.user.id!='ocid1.user.oc1..<user_OCID>,'

Listing 6.2: Policy statement that allows everyone but the specified user in the group my-group to read buckets in the compartment ObjectStorage. Behavior on the specified user is not precised by this policy statement.

Such a statement is useful when the user designated by the ocid belongs to the group "my-group". It would be expected to allow the whole group except this particular user to read objects in the compartment ObjectStorage.

The documentation does not particularly advises this method, because banning a certain user from having a right when they belong to a group, within which all users share the same permission, it is not necessarily considered good practice. This way, Oracle hopes to sensitize the administrator to create a better group distribution instead. Moreover, because policies can only allow, not deny, this policy does not effectively "denies" the user from accessing the
bucket. In other words, this policy just authorizes everyone else in the group but the user to accept the bucket. Behavior for the particular user is not determined, and if no other policy is specified, then the user would be effectively denied from accessing the bucket, unless other policies specify otherwise.

As it is not documented, one could still wonder if such a policy, intended to allow everyone but one user, would work. The reason this question is very relevant to answer, is because an idle administrator that would want to achieve such a result would be probably tempted to write such a statement naturally. For instance, we can imagine a scenario where the user is temporarily banned from one particular certain privilege because of a policy infraction. Rather than creating a special temporary group to place the banned user in, it might seem easier to modify existing policies. In the case where it does not work, this policy would have no effect to restrict the user which could cause security breaches.

6.1.1.2 Precedence on "contradictory" policy statements

When both "allow" and "deny" policies can be written, cloud providers have to choose which policy take priority over the other. The reason is to result in a deterministic behavior if contradictory policies are written (for instance a policy that allows a user to read and a policy that denies the same user to read the bucket). Usually, precedence rules are either based on creation date (older or newer takes precedence) or with a "deny always first" (deny policies override all the "allow" existing ones, as less access is safer than more access). Precedence rules are documented by the provider.

Writing contradictory policy statements could be practical in some cases. For example, if an administrator would like to ban temporarily an user from a certain right, and deny policies always take precedence, then it would be simple to just add a temporary policy that overrides existing ones, and delete it when it becomes obsolete.

Unlike in other cloud providers, writing contradictory policy statements in OCI is impossible, as policies can only allow, not deny. This entails that writing policies always has to grant more (or equal) rights to the already existing permissions that a user has. Thus policies cannot contradict each other.

We imagine that this is the reason why no information on policy precedence was found in the documentation.

For this reason, the following policies, which can look like contradictory statements:
Figure 6.1: Illustration of policies A and B applied to different sets of users

POLICY A: Allow group 'THESIS\_USERS' to read buckets in compartment Pwn where request.user.id='ocid1.user.ocl..aaaa...hurwta'

POLICY B: Allow group 'THESIS\_USERS' to read buckets in compartment Pwn where request.user.id!='ocid1.user.ocl..aaaa...hurwta'

Listing 6.3: Policy statements which "first" allow everyone else in the group but the specified user to access buckets, then allows the specified user in particular to access the bucket. "First" is in quotes as if we do not know if it is applied first in reality.

Are, in fact, not contradictory, rather complementary: - policy A authorizes the user specified by the OCID to read buckets. The behavior of the rest of the group THESIS\_USER is not specified - policy B authorizes the rest of the group THESIS\_USER to read the bucket. The behavior of the user specified by the OCID is not specified.

In other words, these policies are equivalent to taking a partition of the group THESIS\_USER (see figure 6.1), and determining a behavior for each set of users. Default or existing behavior apply for the unspecified behaviors.

We think that, in the case where they are validated, such statements could
lead to misconfigurations, as it is a logic mistake that could easily be made by someone. As it is not documented, we would like to test if the policies result in the scenario that we imagine.

Thinking that policy B would result in restricting the user with targeted OCID to not have access to the bucket could result in a vulnerability and break confidentiality as the user would then have too large privileges.

Policies statements are also going to be tested in reverse order, in order to confirm our theory. We expect to see all users to have access to the bucket regardless of the order in which the policies are written.

6.1.1.3 **Spellchecks on WHERE clauses**

As we were studying, we found that some parts of the policies were spellchecked but some others were not. It happened as we were designing such policies and realised that the query validated when the resource name specified was wrong.

In an example policy statement:

```plaintext
Allow group <group\_name> to <verb> <resource-type> in compartment <compartment\_name> WHERE <variable>=<value>
```

Listing 6.4: Structure of a policy statement containing a WHERE clause

The base components of the policy statement, `<verb>`, `<resource-type>` and `<compartment\_name>` are spell checked, meaning that the policy statement creation won’t succeed if specified resources don’t exist (by introducing a typo in the name for example). Anything after the WHERE statement is not spellchecked. Moreover, the name of the group is not checked either, meaning that policies can be written with groups that could not exist.

We can verify it by entering such a policy statement:

```plaintext
Allow group <group\_name> to <verb> <resource-type> in compartment <compartment\_name> WHERE 123="abc"
```

Listing 6.5: Example of policy to test if fields after the WHERE keyword are spellchecked (correspond to an existing ressource)

Because 123 is never equal to abc, this statement is never applied, although being successfully implemented in the console. One can then wonder if the policy could still be accepted if a spelling error is made on the value of a variable specified in the WHERE clause, for example a resource OCID with a wrong character. After research, this particularity was not found to be explained in the documentation. We think it is worthwhile to tackle as an ad-
ministrator that does not know it, might validate policies with spelling errors, leading to policies that does not reflect the intended levels of access.

6.1.1.4 Creating an infrastructure

To test that the policies that we create are enforced or not enforced, we created a bucket whose access is going to determine if the policy was enforced. The bucket is called thesis-reports. If the user can still read the content of bucket, it means that we failed to create a policy that reflected its intended goal.

The OCID of the user we are going to test access for is ocid1.user.oc1..[...].hurwta. The user belongs in the group THESIS_USERS.

6.1.2 ABAC

The related research showed (see 3.1.3) that ABAC policies can be wrongly enforced by the cloud provider. Because ABAC policies are in general considered harder to keep track of, or secure properly, we will study how can ABAC insecure policies can look like and what could their consequences be. In particular, we want to study how flawed implementations of defined-tags and dynamic groups permissions can lead to privilege escalation.

In this scenario, we consider an unprivileged user with the developer role. This user has access to some computing services. Because we usually want compute instances to compute on data, computing services need to that leverage other services, for instance, object storage, while the developer does not have access to the storage themselves. The developer is our attacker and is considered an inside threat. The attacker can be also have a different profile, for instance if they are an external threat who has gotten access to the credentials of the developer (through phishing or other method).

With this context, we would like to show that unsafe policies can be implemented, leading to the developer leveraging the compute services to access the storage. We will also show that fixing such policies is not straightforward, and given the design choice, mistakes could be made leading to remaining unsafe policies.

Different use case will be studied:

1. The developer has access to compute instances only
2. The developer has access to serverless functions only
6.1.2.1 Creating an asset

We have created an object storage (bucket). This object storage is considered to contain sensitive data such as a .txt file containing company secrets.

The bucket is called "thesis-report" and our namespace is censored as it is considered sensitive. We also assume that the attacker knows the name of the bucket. This can be done by gathering information on the company and their naming conventions, or bruteforcing.

Figure 6.2 shows the bucket in the console from the administrator account. We see that the bucket has some folders and files in. Figure 6.3 shows the bucket in the console from the developer account. The bucket is not seen in the listing. The developer managed to get to this view by guessing the url of the bucket which has a set structure knowing the namespace, the bucket name and the region.

The bucket contains one file called "secret.txt" as seen in figure 6.4.

Permission linked to this bucket are:

- The unprivileged user (developer) does not have access to the bucket (read/write/listing (cannot know if it exist or not))

- Instances tagged with the defined-tag “bucket-access” belong to a dynamic group which has access to the bucket for computing purposes.

The motivation behind the second policy is to let compute instances access stored data to compute outputs from it. The rule for adding instances to the dynamic group "bucket-instances" is done with this matching rule:

```
Instances that meet the following criteria will be included in the dynamic group: criteria defined by any of these rules:
tag.priv-tags.bucket-access.value
```

Listing 6.6: Matching rule for the dynamic group "bucket-instances"

The policy for authorizing entities belonging in the dynamic group bucket-instances to access the "thesis-reports" bucket can be written as follow.

```
"allow dynamic-group bucket-instances to manage objects in compartment Pwn where target.bucket.name='thesis-reports'"
```

Listing 6.7: Policy statements affecting the group bucket-instances
Figure 6.2: Sensitive bucket as seen from the administrator account
Figure 6.3: Sensitive bucket as seen from the developer account

Figure 6.4: Content of the bucket
Note that the restriction to the bucket "thesis-reports" is not necessary. However, since sensitive buckets on this tenancy exist, it’s good practice to specify resource names when authorizing access, to follow the least-privilege principle.

### 6.1.2.2 Setting up the different scenarios

The following subsections will present the additional policies that will constitute our different scenarios.

#### 6.1.2.2.1 Case 1: The developer has access to compute instances only

We added the default oracle policy "Let users launch compute instances" with the Policy builder (see 2.3.3.2). The policies are set as follow:

```plaintext
Allow group 'THESIS_USERS' to manage instance-family in compartment Pwn
Allow group 'THESIS_USERS' to read app-catalog-listing in compartment Pwn
Allow group 'THESIS_USERS' to use volume-family in compartment Pwn
Allow group 'THESIS_USERS' to use virtual-network-family in compartment Pwn
```

Listing 6.8: Policies affecting the group THESIS_USER

#### 6.1.2.2.2 Case 2: The developer has access to serverless functions only

Because the developer actually works on containers, the FaaS (service handling the functions) service needs the following policies in order to access the image repository everytime the serverless function is called:

```plaintext
"allow service FaaS to use virtual-network-family in compartment Pwn",
"allow service FaaS to read repos in tenancy"
```

Listing 6.9: Policies allowing the FaaS service (Functions as a service)

This policy is the default policy for Faas. The following policies authorize our developer to develop serverless functions:

```plaintext
allow group 'THESIS_USERS' to manage functions-family in compartment Pwn
allow group 'THESIS_USERS' to read metrics in compartment Pwn
```
allow group 'THESIS_USERS’ to use virtual-network-family in compartment Pwn
allow group ‘THESIS_USERS’ to manage repos in tenancy where target.repo.name = '/thesis/

Listing 6.10: Policies letting THESIS_USERS group develop serverless functions

This is the default policy when authorizing a user to develop serverless functions. These policies authorize the developer to use a certain folder of the image repository and deploy them.

6.1.2.3 Setting up a vulnerable infrastructure

The administrator wishes to grant the developer the right to use and attribute the defined tag. The reason is that the developer is going to create and use compute instances or serverless functions. Creation will not be limited to one but potentially hundreds or thousands of them. According to the company organisation policies, the administrator wants the developer to be able to organise their resources, and a very straightforward way to do that, is to let them tag the resources.

In order to do this, the administrator unknowingly writes an unsafe policy. The following policy allows users in the group THESIS_USERS to use the tag-namespace “priv-tags”:

"Allow group 'THESIS_USERS’ to use tag-namespaces in compartment Pwn

Listing 6.11: Policies letting the group THESIS_USERS to use all the available defined-tags

The mistake here, is not that the administrator allowed the developer to use tags, as it is an intended feature. It is that through this policy, the developer can now use every tag, including a tag that can grant privileges to compute instances, privileges that are superior to their own, which was unintended. We think it could be a likely source of error as authorizing users to use tags is not something that is commonly associated with privilege granting.

6.1.2.4 Permission enumerators

Usually, after credential compromission of a cloud account, a pentester wants to know the capabilities of the stolen account or what resources does it has access to. Cloud providers often have an API that will list all the permissions
associated with a user. However, using this API to uncover all permissions of
the user has shortcomings: sometimes specific resources will be still readable
by the user while not given explicit permission, because of their public status.

A common strategy to find all readable permissions or resources of an
account is bruteforce enumeration:

1. List all the possible actions or resources one can access with the highest
level of permission. Usually a look at the documentation is enough for
this step.

2. For each permission, find the associated API and call it authenticated as
the user

3. Study the response: if data is returned, the resource is readable, if there
is an error message the resource does not exist or is not readable

In other words, instead of asking the cloud provider what it thinks our
permissions are, we are finding it out ourselves by testing our read access to
all discovered resources or testing all known APIs. If the call returns an error
message, the permission is flagged as ungranted. Through this method, we can
find that we have access to much more resources than our permission list let
us think we do, because of misconfigurations (often it is because of resources
set to public by mistake).

On the hand, the number of API calls is significantly higher which makes
it easy to detect if the tenancy is monitored.

As we saw in OCI’s architecture (see 2.3.3.4), unlike other cloud providers
(AWS "aws iam list-user-policies", GCP), OCI admits in the documentation
that they have no endpoint to return the list of capabilities of a user. This means
that if we do not have access to the policies (because we are not administrator
for example), our only way to know what we can do with a newly acquired
credentials is bruteforce enumeration.

Open-source enumerators for cloud providers can usually be found online.
Two open-source repositories for OCI enumeration were found:

- oci_enum\(^1\) by orca-toolbox
- ScoutSuite\(^2\) by nccgroup

\(^1\)https://github.com/orcasecurity/orca-toolbox/tree/main/oci
_enum/oci_enum
\(^2\)https://github.com/nccgroup/ScoutSuite
At the time of writing, oci_enum enumerates resources in an OCI tenancy by leveraging the available "list" built-in APIs for given resources (buckets, instances...) with a set of credentials. This means that using the enumerator does not give more information that the credentials’ account posses in the console. Moreover, the tool does not enumerate permissions for an user. However, the readme shows that the creators considered to implement this feature with a bruteforce strategy: "Add "Brute Force" mode to enumerate all available (read) permissions"

The second tool, ScoutSuite is a multi-provider auditing tool. In the case of OCI, the tool retrieves all information available in the tenancy in a comprehensive report. By studying the source code, the tool does not seem to perform bruteforce enumeration either but rather just retrieves the policies that are all available to the administrator. The tool also gives insight to the auditor on what it thinks are unsafe and wrong. One can wonder if the tool might be able to detect if users have access to resources by other means.

6.1.2.5 Choice of tools

First, we will use Python to exploit the privileges granted to us through the compute instance or serverless functions and evaluate the results manually.

Both tools presented in the previous section, oci-enum and ScoutSuite, will be tested against our vulnerable tenancy. The aim of this step is to check if the enumerator can effectively detect that our user has "extra" unexplicit permissions. In general, we also want to check the amount of information that the attacker could get out of these tools.

6.1.2.6 How to use the terraform folders

You can replicate the attacks by visiting the associated git [37].

1. Setup the infrastructure common to both ABAC attacks by running the folder 1-setup-infra with an administrator account.

2. Setup the different scenarios by running one of the policies folders in 2-scenarios-policies. Choose scenario 1 or 2 and run it as administrator.

3. The attacks can be run by deploying the infrastructure in the attack folder 3-attacks with the credentials of the user generated in step 1.
6.2 Oracle MySQL Managed Database

In this scenario, the objective is to attack the Oracle MySQL Managed database service in itself. We will consider the attack successful if the tenant can execute any action that is not supposed to be in the scope of the managed service. For instance, running commands on the underlying OS, executing code as the root account belonging to OCI etc. For this service, we assume that the attacker has access to a database. This is possible for anyone opening an OCI free-tier account.

When we create a database with any cloud platform, the provider makes the user select an username for the administrator which they can use to administrate their database. This account is different from the managed administrator dedicated to the cloud provider that is the real superuser. We will distinguish them by calling them "user administrator" vs "managed administrator". The "user administrator" is not a system superuser but possesses some superuser capabilities.

Some commands are being run in this section in order to gain information about the studied system. These do not constitute the attack part and are in compliance with the law.

To understand our possibilities of pentest for MySQL, we use the methodology described in 4.1.2.

6.2.1 Context (reminded): related research

As we mentioned earlier in section 2.4, Cloud providers ported popular services on their platform to improve compatibility with other services. The reason for using existing services in place of creating their own can be the following 3:

- The community has been using these services for a long period and shared knowledge of these services is particularly extensive
- Developers are used to this service/language and therefore already have the necessary skills to use the service and are less likely to make mistakes. It also augments the attractiveness of the platform as they do not have to spend time mastering a new service.
- These services are usually known to be quite performant, reliable, and secure in their intended purpose

• Developing costs and time for cloud providers are reduced

All these reasons are motivations for cloud providers to use existing services instead of developing their own. However, as original developers did not foresee the rise of cloud platforms, services developed before the cloud usually lack isolation between users or sufficient access control for users. The isolation layer has to be implemented by the cloud provider when porting those services.

In that context, the company Wiz studied managed PostgreSQL databases in multiple cloud vendors and found that it was incorrectly implemented [43], leading to being able to access databases of other tenants and thus breaking the cloud isolation principle.

"Many managed services offered by CSPs are based on popular open-source software that runs on top of vendor-managed compute instances.[...] Some CSPs introduced these changes via extensions or custom configurations. Others went so far as modifying the source code of the PostgreSQL engine itself and maintaining their fork. All of these approaches can lead to unexpected security issues."

Successful exploits were found both in GCP and Azure. In GCP, the exploit relies on the fact that ownership of PostgreSQL tables could be changed to the highest privileged user reserved to the Cloud Provider, meant to be inaccessible to any tenant. This change of ownership could then lead to executing functions stored in tables as the owner of the table, because of leveraging other features from PostgreSQL like the COPY command. This led to ability to read files and execute commands on the OS that hosted the database. This kind of privilege escalation led both providers to access to internal network as root and access cross-tenant resources. In Azure, the exploit relied on the fact that the user could create another user with higher privileges than the one they initially had.

Oracle Cloud Infrastructure chose to implement managed databases based on MySQL, which does not have a concept of table ownership.
6.2.2 Study of the architecture of unmanaged and managed PostgreSQL

6.2.2.1 COPY

In order to prove that the research team successfully escaped the scope of the database, they leveraged the COPY PostgreSQL statement that enables to run shell commands from SQL queries: "PostgreSQL has a built-in feature for command execution, using the COPY statement. The COPY statement can be used to execute commands and read from or write to files." [43]

6.2.2.2 PostgreSQL permissions model

In PostgreSQL, the permission model consists of roles that are similar to users or group in IAM, associated to role attributes. These role attributes define what a role can do, similarly to policies in IAM.

Every created object in PostgreSQL is defined with an owner as explained in the documentation [49]:

"When an object is created, it is assigned an owner. The owner is normally the role that executed the creation statement. For most kinds of objects, the initial state is that only the owner (or a superuser) can do anything with the object. To allow other roles to use it, privileges must be granted."

By "can do anything", the documentation implies that the owner can read, write, execute statements in the table or use other privileges described in the documentation [49].

6.2.2.3 Managed PostgreSQL permissions model in GCP

In the Google Cloud Platform PostgreSQL managed service, the following default roles are created in image 6.5. "postgres" is the administrator created by the research team alongside the database. All other roles are common to every database created with GCP.

"cloudsqladmin" is GCP’s superuser role used by Google to manage the database. The tenant’s superuser is "postgres" and belongs to the "cloudsqlsuperuser" role, a custom role created by Google, defining some privileges and removing some compared to "cloudsqladmin".

In order to understand what kind of privileges the "cloudsqlsuperuser" role grants, we can refer to GCP’s official documentation in image 6.6.
### Figure 6.5: List of roles in GCP managed PostgreSQL service. Source: [43]

<table>
<thead>
<tr>
<th>Role name</th>
<th>Attributes</th>
<th>Member of</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudsqladmin</td>
<td>Superuser, Create role, Create DB, Replication, Bypass RLS</td>
<td>()</td>
</tr>
<tr>
<td>cloudsqllagent</td>
<td>Create role, Create DB</td>
<td>{cloudsqlsuperuser}</td>
</tr>
<tr>
<td>cloudsqllserviceaccount</td>
<td>Cannot login</td>
<td>()</td>
</tr>
<tr>
<td>cloudsqlluser</td>
<td>Cannot login</td>
<td>{cloudsqlsuperuser}</td>
</tr>
<tr>
<td>cloudsqllimportexport</td>
<td>Create role, Create DB</td>
<td>{cloudsqlsuperuser}</td>
</tr>
<tr>
<td>cloudsqllreplica</td>
<td>Replication</td>
<td>{pg_monitor}</td>
</tr>
<tr>
<td>cloudsqlsuperuser</td>
<td>Create role, Create DB</td>
<td>{pg_monitor, pg_signal_backend}</td>
</tr>
<tr>
<td>postgres</td>
<td>Create role, Create DB</td>
<td>{cloudsqlsuperuser}</td>
</tr>
</tbody>
</table>

### Superuser restrictions

Cloud SQL for PostgreSQL is a managed service, so it restricts access to certain system procedures and tables that require advanced privileges. In Cloud SQL, customers cannot create or have access to users with superuser attributes.

You cannot create database users that have superuser privileges. However, you can create database users with the `cloudsqlsuperuser` role, which has some superuser privileges, including:

- Creating extensions that require superuser privileges
- Creating event triggers
- Creating replication users
- Creating replication publications and subscriptions
- Full access to the `pg_largeobject` catalog table

### Figure 6.6: Permissions of "cloudsqlsuperuser" role. Source: [50]
Unfortunately, the documentation is vague on the list of exact privileges that the "cloudsqlsuperuser" role has. The research team had to find out relevant privileges through trial and error. Their goal was to discover what kind of privileges this role had, that were originally reserved to the superuser by PostgreSQL in their original implementation, called "non-default capabilities" in the article. The reasoning is that, if PostgreSQL restricted actions only to the superuser, it was probably because owning this privileges entailed consequences such as security issues. Thus, Wiz was hoping, by studying the non-default capabilities, to find a privilege for which GCP overlooked some consequences of owning it, among which of them was privilege escalation or breaking security.

6.2.2.4 ALTER TABLE

Through trial and error, Wiz found that the "cloudsqlsuperuser" role in GCP possessed the ALTER TABLE privilege. This command changes the owner of the targeted table.

```
ALTER TABLE table_name OWNER TO new_owner;
```

which sets the owner of the table `table_name` to `new_owner`.

In the case of GCP, The `new_owner` field was not restricted. This means a default user administrator "postgres" creating a table, whose default owner would be "postgres" according to the documentation, would be able to leverage ALTER TABLE to change it to the user "cloudsqladmin", the managed administrator of GCP.

Used alone, changing the owner of a table, even "cloudsqladmin", does not have significant security consequences if the tenant can not connect as "cloudsqladmin" in the first place. It seems like a practical feature to have, which is probably why GCP authorized it for the user administrator in the first place.

However, in the field of security, experienced pentesters often consider scenarios beyond changing ownership. Usually, services implement features that temporarily allow users to access or execute an object as the owner, such as in the UNIX permission model. Managing to change the owner to a higher privileged user, combined with finding such a feature would let the user execute anything with higher privileges and is a typical privilege escalation workflow.
6.2.2.5 **Index expressions/index functions**

PostgreSQL lets users create indexes. When performing search queries, the engine usually has to go through every entry, unless entries are ordered in a particular way. Creating indexes fulfills exactly this function: ordering data in a certain way to make `SELECT` queries return data faster. Sequential scans become index scans. More explanation about the use of this feature can be found here.

Once created, an index expression as called in the documentation [51] or index function as called in the article [43] is "tied" to the table. Furthermore, the research team found that index functions could be invoked as the owner of the table:

"During our research, we found that performing any of the [INSERT/UPDATE/ANALYZE] commands on a table implicitly invokes the index functions with the table owner’s permissions. This behavior is not detailed in the official PostgreSQL documentation, but references to it can be seen in the PostgreSQL source code."

Index functions are then relevant for the attack plan as they confirm that we can execute arbitrary code as the table owner.

6.2.2.6 **Predefined roles and CREATE ROLE**

PostgreSQL documentation [52] states that:

"PostgreSQL provides a set of predefined roles that provide access to certain, commonly needed, privileged capabilities and information. Administrators (including roles that have the `CREATEROLE` privilege) can `GRANT` these roles to users and/or other roles in their environment, providing those users with access to the specified capabilities and information."

Some predefined roles grant access to the server files and can be seen in table 6.1. Although `CREATEROLE` is a feature that is usually one a service would want to provide to a user, Azure did not disable role creation with these predefined roles for the user administrator. This entailed that a tenant could leverage the user administrator permissions, create a user with a privileged predefined role, and access the underlying system file.

---

4*PostgreSQL Create Index. Available at: https://www.postgresqltutorial.com/postgresql-indexes/postgresql-create-index/
<table>
<thead>
<tr>
<th>Predefined role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pg_read_server_files</code></td>
<td>Allow reading files from any location the database can access on the server with COPY and other file-access functions.</td>
</tr>
<tr>
<td><code>pg_write_server_files</code></td>
<td>Allow writing to files in any location the database can access on the server with COPY and other file-access functions.</td>
</tr>
<tr>
<td><code>pg_execute_server_program</code></td>
<td>Allow executing programs on the database server as the user the database runs as with COPY and other functions which allow executing a server-side program.</td>
</tr>
</tbody>
</table>

### 6.2.3 Summary of the steps of the attack

With all this background information explained, Wiz found two attack plans: one for GCP and one for Azure.

#### 6.2.3.1 Attack plan 1: Google Cloud Platform Cloud SQL [43]

1. Create a new table.

2. Insert some dummy content to the table, so the index function has something to work with.

3. Create a malicious index function (with our code execution payload) on the table.

4. `ALTER` the table owner to `cloudsqladmin`, GCP’s superuser role, used only by Cloud SQL to maintain and manage the database.

5. `ANALYZE` the table, forcing the PostgreSQL engine to switch user-context to the table’s owner (`cloudsqladmin`) and call the malicious index function with the `cloudsqladmin` permissions, resulting in executing our shell command, which we did not have permission to execute before.
6.2.3.2 Attack plan 2: Azure Database for PostgreSQL

1. Create a new role with permissions `pg_read_server_files`, `pg_write_server_files` and `pg_execute_server_program`
2. Log in as the new role
3. Execute arbitrary commands with the COPY statement

6.2.4 Portability on MySQL

The aim of this subsection is to understand our capabilities in MySQL. For this section, we rely principally on documentation as well as information gathering on a live database that we created. A database had been created with a default administrator `myadmin` as shown in figure 6.7. For the majority of the subsections, ideas of attack paths will be presented at the end, after explaining the theory. A summary of the chosen attack paths can be found in the next section.
6.2.4.1 List of reserved usernames

When creating a database, the tenant chooses a username for the administrator account of the database. Some names cannot be chosen. Table 6.2 presents the list of reserved usernames unavailable for naming.

<table>
<thead>
<tr>
<th>Reserved Username</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>Administrator role for newly created users</td>
</tr>
<tr>
<td>ociadmin</td>
<td>OCI Reserved Names</td>
</tr>
<tr>
<td>ocirpl</td>
<td>OCI Reserved Names</td>
</tr>
<tr>
<td>mysql.sys</td>
<td>Default MySQL Reserved Names</td>
</tr>
<tr>
<td>mysql.session</td>
<td>Default MySQL Reserved Names</td>
</tr>
<tr>
<td>mysql.infoschema</td>
<td>Default MySQL Reserved Names</td>
</tr>
</tbody>
</table>

Some of the names are reserved by MySQL for server communication purposes. These names are also reserved in an unmanaged database. However, other names starting with "oci" are OCI reserved names. One or multiple of these reserved accounts are the real root accounts with full privileges on the database.

6.2.4.2 Managed MySQL permission model

Before finding out which account is the managed administrator, we need to understand the permission model. The administrator "myadmin" created alongside the database possess the capabilities described in table 6.3. In particular, compared to the default unmanaged administrator account, the account does not possess the capabilities described in the table 6.4.

The command `SHOW PRIVILEGES;` lists all available privileges [53].

Although it is very clear why some permissions are removed, we hope to find in table 6.3 privileges that we could leverage to do privilege escalation OR flawed implemented removed privileges in table 6.4 (i.e. not completely forbidden to some extent).

Even though the documentation clearly states the capabilities of the user administrator account, related research has shown that the documentation is not always reflecting of the reality of the studied system. The exploit presented in the PostgreSQL managed database for Azure showed that the permissions were not implemented as expected. For this reason, it is still relevant to try using permissions that are indicated as not available from our perspective.
Table 6.3: Administrator Capabilities in OCI’s MySQL Managed Database Service [54]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>Modify the structure of an existing table or database object</td>
</tr>
<tr>
<td>ALTER ROUTINE</td>
<td>Modify the definition of a stored routine</td>
</tr>
<tr>
<td>CREATE</td>
<td>Create new tables or databases</td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td>Create a new user role</td>
</tr>
<tr>
<td>CREATE ROUTINE</td>
<td>Create a new stored routine</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLES</td>
<td>Create temporary tables</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>Create a new user account</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Create a new view</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete rows from a table</td>
</tr>
<tr>
<td>DROP</td>
<td>Remove existing tables or databases</td>
</tr>
<tr>
<td>EVENT</td>
<td>Create, alter, or drop events</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Execute a prepared statement</td>
</tr>
<tr>
<td>INDEX</td>
<td>Create, alter, or drop indexes</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert new rows into a table</td>
</tr>
<tr>
<td>LOCK TABLES</td>
<td>Explicitly lock tables for certain operations</td>
</tr>
<tr>
<td>PROCESS</td>
<td>View or kill currently executing queries</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Create or drop foreign key constraints</td>
</tr>
<tr>
<td>REPLICATION CLIENT</td>
<td>Monitor replication status</td>
</tr>
<tr>
<td>REPLICATION SLAVE</td>
<td>Act as a replication slave server</td>
</tr>
<tr>
<td>SELECT</td>
<td>Retrieve data from one or more tables</td>
</tr>
<tr>
<td>SHOW DATABASES</td>
<td>Show a list of available databases</td>
</tr>
<tr>
<td>SHOW VIEW</td>
<td>Show the definition of a view</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Create, modify, or drop triggers</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Modify existing rows in a table</td>
</tr>
</tbody>
</table>
6.2.4.3 Identifying the OCI managed administrator account, host and user

The table mysql.user contains information about the global privileges of each user.

In Appendix B is the output of the command `SELECT * FROM mysql.user \G`. The table is constructed as such: the first column is the Host, the second is the User. The following rows are privileges set to Y for "Yes" when they are enabled for the user and N ("No"). We can see, in particular, that our user "myadmin" does not have "File_priv". Other users do not seem to have more privileges than we do except the user "ociadmin" which has "Y" on every privilege, including the "Super_priv" that indicates that this user is a super user. We conclude that "ociadmin" is the managed administrator.

6.2.4.4 Host

The host field shown in the output of the command `SELECT * FROM mysql.user \G` is also relevant to explain (Appendix B).

The Host field specifies the network location from which a user can connect to the MySQL server. "%" means from any source, while 127.0.0.1 and "localhost" are local addresses, which means the MySQL server can only be accessed from within the host where the server is (which we do not have access to) [55].

In particular the documentation explicits that multiple accounts with the same user names can exist [55]:

"MySQL account names consist of a user name and a host name, which enables creation of distinct accounts for users with the same user name who connect from different hosts."

An attack idea could be to create a new user ociadmin with host "%" from the database, although it is supposed to be forbidden by OCI during database creation. We can try to find afterwards what would be the consequences of such a created user.

6.2.4.5 Credentials

Hashed passwords are also stored in the table mysql.user. We can change the password using one of the following commands:

```
ALTER USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
SET PASSWORD FOR 'jeffrey'@'localhost' = 'auth_string';
```
Listing 6.12: Command for changing the password of the user "jeffrey" on MySQL

According to the documentation [56], necessary privilege for it are the privilege CREATE USER or eventually additionally the privilege CONNECTION_ADMIN which we have on some tables (see the listing 6.2.4.7). Password modification seems to be possible to achieve, but some user modification might be restricted to us. As information on this is unclear, we will just test it directly on the database. We hope to be able to modify the "ociadmin" credentials in order to connect as this user.

6.2.4.6 Filesystem access capabilities on the underlying system

Filesystem capabilities are among:

- read
- write
- execute

The success of one of these access prove that we have access to the file system. The permission that grants access to any kind of file operation is "FILE".

6.2.4.6.1 Read MySQL has reading capabilities on the filesystem with the LOAD DATA statement [57]. The LOAD DATA statement is used as follow:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Listing 6.13: Command to read a file from the filesystem in MySQL

The LOCAL key word can also be used in the statement.

To summarize the documentation 5, the user administrator account cannot execute this statement with a non-LOCAL option because it does not have the FILE capability, while the OCI administrator can. Using LOCAL does not interest us because we are already in control of the client filesystem (it is where we connect from). These capabilities will still be tested in order to confirm the documentation.

6.2.4.6.2 Write  MySQL offers write capabilities in the file system with
`SELECT... INTO...` statement, with the following syntax to write files,
according to the documentation [58]:

"The SELECT ... INTO OUTFILE 'file_name' form of SELECT writes
the selected rows to a file. The file is created on the server host,
so you must have the FILE privilege to use this syntax. file_name
cannot be an existing file, which among other things prevents files
such as /etc/passwd and database tables from being modified. The
character_set_filesystem system variable controls the interpreta-
tion of the file name."

According to the OCI and MySQL documentation, the user administrator
account cannot execute this statement because it does not have the FILE ca-
pability, while the OCI administrator can. These capabilities will be tested in
order to confirm the documentation.

6.2.4.6.3 Execute  We are looking for a keyword that executes files or per-
mit command execution, similarly to the COPY FROM PROGRAM in Post-
greSQL. In MySQL, there isn’t a default equivalent of the PostgreSQL COPY
command with the PROGRAM option. However, we can achieve a similar
result using the LOAD DATA INFILE statement along with a user-defined
function (UDF).

In order to achieve this, we need to create an UDF that allows us to execute
shell commands from within MySQL, such as the `lib_mysqludf_sys
UDF which can be found on GitHub. The issue is that the function needs
to exist as a file on the server and be compiled, thus making it impossible for
us to develop such a function only from within the managed database. For
further reading, here is an article on how the UDF function could be written
in the OS directly from the database. However, this method requires FILE
writing privileges, which we do not have.

6.2.4.7 Current privilege & granting capabilities

Unlike PostgreSQL, MySQL does not have a concept of ownership for tables.
The exploit described in the research 3.3 relies on the ability to modify the

---

6https://github.com/mysqludf/lib_mysqludf_sys
owner of a table, which grants certain privileges when executing some statements as the table owner. In order to perform a similar step in MySQL, we need to study how privilege granting is achieved.

In MySQL, privileges can be given, to a user or to a role, or revoked through the GRANT and REVOKE statements [59]. Here is how we would use a GRANT query:

```sql
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
GRANT ALL ON *.* TO 'jeffrey'@'localhost';
```

Listing 6.14: Command to grant the SELECT/ALL privileges to the user jeffrey on the table invoice in the database db2.

The first statement would grant the privilege SELECT to the table "invoice" in the database "db2" to the user "jeffrey". ALL means all privileges according to the documentation, * means any (database or table).

The list of grants can be accessed by using "SHOW GRANTS \G" or for a specific user: "SHOW GRANTS FOR <Username> \G"

We executed SHOW GRANTS \G" connected as the user myadmin. The output is shown in the listing 6.2.4.7:

```
SHOW GRANTS \G
*************************** 1. row
Grants for myadmin@%: GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPPLICATION SLAVE, REPPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE ON *.* TO 'myadmin'@'%' WITH GRANT OPTION
*************************** 2. row
Grants for myadmin@%: GRANT APPLICATION_PASSWORD_ADMIN, AUDIT_ADMIN, BACKUP_ADMIN, CONNECTION_ADMIN, FLUSH_OPTIMIZER_COSTS, FLUSH_STATUS, FLUSH_TABLES, FLUSH_USER_RESOURCES, REPPLICATION_APPLIER, ROLE_ADMIN, XA_RECOVER_ADMIN ON *.* TO 'myadmin'@'%' WITH GRANT OPTION
*************************** 3. row
Grants for myadmin@%: REVOKE INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE
```
Every row are permissions that were granted to us through a GRANT query. In particular, row 1 and 2 shows the permissions that we have for all the databases that we create (*.* means any <database>.<table>). Row 3 shows that we have some additional capabilities on the mysql database (including the mysql.user table we mentioned earlier). Row 4 shows we have some additional capabilities on the the sys database and row 6 indicates that we have the administrator role. GRANT OPTION indicates that the privileges can be granted by us to another user. ADMIN OPTION achieves the same purpose for roles.

The GRANT keyword is not a banned privilege of the user administrator account, as the administrator is supposed to be able to create roles and grant them privileges to other users or roles. However, statements that might be banned are GRANT statements associated to certain sensitive privileges. If the user administrator manages to grant himself or another user more permissions than they are initially allowed to have, then the service will be considered exploited. Because of how privilege granting works in MySQL and the previous explanation, using an exploit such as the one presented for Azure PostgreSQL (creating a user with more privileges than us) does not seem possible.
6.2.4.8 Impersonation capabilities of the administrator

This paragraph will study our admin impersonation capabilities.

Unfortunately, since MySQL does not have table owners, no equivalent query of INSERT/UPDATE/ANALYZE was found, as in, a function that would execute the content of a table as the table owner.

However, MySQL has the notion of stored programs [60]:

"Stored programs (procedures, functions, triggers, and events) and views are defined prior to use and, when referenced, execute within a security context that determines their privileges"

In other words, one can execute a program with the privileges of the account that wrote it. The account who wrote the program is called the "definer". The SET_USER_ID is a privilege that would let us change the definer to any account [61]: "SET_USER_ID Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account as the DEFINER attribute of a view or stored program. Stored programs execute with the privileges of the specified account, so ensure that you follow the risk minimization guidelines listed in Section 25.6, “Stored Object Access Control”.

Unfortunately SET_USER_ID is not a privilege that is granted to us. Thus, we cannot write programs and execute them with the superuser context once created. Another way would be to find an adequate stored procedure with a superuser context that would be interesting to use. However, we have tried to read the source code of procedures written by a super user like so:

```sql
SQL > SHOW PROCEDURE CODE gen_range;
ERROR: 1289 (HY000): The 'SHOW PROCEDURE|FUNCTION CODE' feature is disabled; you need MySQL built with '--with-debug' to have it working
```

Listing 6.16: Attempt to read source code of procedures

And failed. Finding an existing adequate stored program to leverage with admin rights would seem complex and time inefficient due to the voluntary lack of documentation about OCI’s defined procedures. Due to lack of time and our lack of understanding of the service in black-box, we decided to give up this lead.

The last method for managed administrator access we imagined is to change the password of the administrator account. However, even if we managed to
change the password, we would still not be able to connect to the admin account from the client as it only accepts connexion from 127.0.0.1 (the local machine that hosts the database). Could we spoof the entering IP so we can trick the server into thinking that we connect from the local host? It is possible only if we can control the network we are in.

6.2.4.9 Localhost spoofing

When a tenant creates a database, they are given an IP to connect to which is the address of the MySQL server. The MySQL server accepts connections on port 3306 or 33060 only. The IP is private and therefore needs to be accessed from a compute instance within the private network. Since we are the owner of the tenancy, we own the virtual cloud network (private network) on which we have total control. Table 6.5 recapitulates our addresses on the private network.

Just like a regular router with a routing table, we could define a rule that forwards traffic from 10.0.0.103 on a port of our choice to 10.0.0.103 on port 3306. This way we are hoping that the incoming connection will be forwarded to itself and appear as localhost from the point of view of the server. We could leverage this to effectively connect as users with hosts "localhost" on the MySQL server. An explanation on how we can spoof clients IP is detailed here\(^8\), and we are applying this method with the server IP as the spoofed IP.

6.2.4.10 MySQL configuration file & variables

The MySQL configuration file determines the server’s global behavior. Statements can be enabled or disabled, regardless of individual permissions of users. If an user possess the appropriate permissions but the corresponding variable is set to OFF, the query will still be disabled. The settings in the configuration file are necessary for the user’s permissions to be valid. If a user has the permission to do an operation, but the server bans the operation through the configuration file, the operation cannot succeed.

For example, the variable `local-infile` determines if the statement `LOAD DATA INFILE` is authorized \([62]\). We can check within the MySQL server the value of that variable with:

Listing 6.17: List variables similar to 'local_infile'. This command also shows the value of the outputed variables.

In our case, the value of local-infile is set to ON. We can see the values of all the existing variable through this command:

```
SHOW VARIABLES;
```

Listing 6.18: Command to show all existing variables and their values in MySQL

Variables can help learn more about the system in general of the file system. For instance, file paths can appear as values of variables, which can give us examples of valid paths on the file system. Some variables are set as disabled by default by MySQL. Variables set to ON can thus give hope to find an exploitable path.

6.2.4.11 Other attack paths considerations

The version of the MySQL server is 8.0.33. According to Common Vulnerabilities and Exposure (CVE), there is no known attack path for this version of MySQL.

6.3 Attack plan

To summarize what have been mentioned in this section, here follows a list of attacks that are relevant to try for all the studied services. Attacks that could violate Oracle’s policy are still mentioned but will not be executed or be executed with parameters that do not go against the policies. Relevant parameters that cannot be used will be mentioned in the result section.

6.3.1 IAM

6.3.1.1 Documentation

In light of previous explanations, three attacks are going to be performed:

1. The first one aims to find out if creating a restrictive policy with "!=" works
2. The second one determines how priority is given to contradictory statements.
3. The third tests if a policy can be validated with an invalid OCID.

6.3.1.1.1 Restrictive policy with "!'="

1. Write the following policy intended to allow all users except the specified user to read buckets:
   
   ```plaintext
   Allow group 'THESIS\_USERS' to read buckets in
   compartment Pwn where request.user.id!=’ocid1.
   user.oc1...[...].hurwta’
   ```

   Listing 6.19: Policy for allowing anyone but the restricted user to access buckets

2. Test if the user can read the bucket and conclude.

6.3.1.1.2 Priority on complementary policy statements

1. Write the following policy statements in different order:
   
   ```plaintext
   Allow group 'THESIS\_USERS' to read buckets in
   compartment Pwn where request.user.id!=’ocid1.
   user.oc1...[...].hurwta’
   
   Allow group 'THESIS\_USERS' to read buckets in
   compartment Pwn where request.user.id='ocid1.
   user.oc1...[...].hurwta’
   ```

   Listing 6.20: Set of complementary policies to test bucket access for

2. Test if the user can read the bucket and conclude.

6.3.1.1.3 Invalid parameter in a policy

1. Write the following policy intended to restrict the user. A 'z' was added by "mistake" at the end of the policy. More erroneous policies will be tested.

   ```plaintext
   allow THESIS\_USERS to read object-family in
   compartment ObjectStorage where request.user.id
   !=’ocid1.user.oc1...[...].hurwtaz’
   ```

   Listing 6.21: Policy statement with an added character by error at the end
2. Testing if the user can read the bucket and conclude

6.3.1.2 **ABAC**

6.3.1.2.1 **Case 1: Accessing the private bucket through a computing instance** (see Figure 6.8)

1. Creating a computing instance with the tag "priv-tags.bucket-access"
2. Executing the following payload in Annex A.
3. The result should show if we have access to the bucket or not
4. If we have access to the bucket, use the enumerators to see if they can detect such a bucket or extra privileges.

6.3.1.2.2 **Case 2: Accessing the private bucket through a serverless function** (see Figure 6.9)

1. Creating a serverless function (Annex A) with the tag "priv-tags.bucket-access"
2. Execute the serverless function
3. The result should show if we have access to the bucket or not
4. If we have access to the bucket, use the enumerators to see if they can detect such a bucket or extra privileges.

6.3.2 MySQL

6.3.2.1 Replacing the administrator account with a banned username

1. Create another username with the same name
2. If the previous step works, renaming our user with name "ociadmin"

6.3.2.2 Accessing the file system

1. Test the default capabilities of file system access (SELECT INTO and LOAD DATA)
2. If they do work, we have access to the file system
3. If success, further exploitation is possible (prompting a shell for instance) but will not be executed

6.3.2.3 Accessing the administrator account via localhost spoofing

Even if we do not manage to succeed at all the steps, they are still going to be reviewed as the success of one of them could have potential security implica-
tions that we will discuss.

1. Changing the password of "ociadmin" in the mysql.user table, so that we know the credentials.

2. Adding a route from 10.0.0.103 to 10.0.0.103. Incoming connections would then go from 10.0.0.202 (us) to 10.0.0.103 to 10.0.0.103 and appear as localhost (see Table 6.5)

3. Connecting as "ociadmin" with our modified password from a remote mysql client on 10.0.0.202.
<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELOAD</td>
<td>Reload or refresh server settings</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>Shutdown the server</td>
</tr>
<tr>
<td>FILE</td>
<td>Access the file system</td>
</tr>
<tr>
<td>SUPER</td>
<td>Perform administrative functions</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>Create tablespaces</td>
</tr>
<tr>
<td>AUDIT_ADMIN</td>
<td>Manage audit log</td>
</tr>
<tr>
<td>BINLOG_ADMIN</td>
<td>Manage binary logs</td>
</tr>
<tr>
<td>BINLOG_ENCRYPTION_ADMIN</td>
<td>Manage binary log encryption</td>
</tr>
<tr>
<td>CLONE_ADMIN</td>
<td>Manage clone plugin</td>
</tr>
<tr>
<td>CONNECTION_ADMIN</td>
<td>Manage connections</td>
</tr>
<tr>
<td>ENCRYPTION_KEY_ADMIN</td>
<td>Manage encryption keys</td>
</tr>
<tr>
<td>FLUSH_OPTIMIZER_COSTS</td>
<td>Flush optimizer costs</td>
</tr>
<tr>
<td>FLUSH_USER_RESOURCES</td>
<td>Flush user resources</td>
</tr>
<tr>
<td>GROUP_REPLICATION_ADMIN</td>
<td>Manage Group Replication</td>
</tr>
<tr>
<td>INNODB_REDO_LOG_ARCHIVE</td>
<td>Archive InnoDB redo logs</td>
</tr>
<tr>
<td>INNODB_REDO_LOG_ENABLE</td>
<td>Enable InnoDB redo logs</td>
</tr>
<tr>
<td>PERSIST_RO_VARIABLES_ADMIN</td>
<td>Manage persistent read-only variables</td>
</tr>
<tr>
<td>REPLICATION_SLAVE_ADMIN</td>
<td>Manage replication slaves</td>
</tr>
<tr>
<td>RESOURCE_GROUP_ADMIN</td>
<td>Manage resource groups</td>
</tr>
<tr>
<td>RESOURCE_GROUP_USER</td>
<td>Use resource groups</td>
</tr>
<tr>
<td>ROLE_ADMIN</td>
<td>Manage roles</td>
</tr>
<tr>
<td>SERVICE_CONNECTION_ADMIN</td>
<td>Manage service connections</td>
</tr>
<tr>
<td>SESSION_VARIABLES_ADMIN</td>
<td>Manage session variables</td>
</tr>
<tr>
<td>SET_USER_ID</td>
<td>Set user ID</td>
</tr>
<tr>
<td>SHOW_ROUTINE</td>
<td>Show stored routines</td>
</tr>
<tr>
<td>SYSTEM_USER</td>
<td>Perform actions as the system user</td>
</tr>
<tr>
<td>SYSTEM_VARIABLES_ADMIN</td>
<td>Manage system variables</td>
</tr>
<tr>
<td>TABLE_ENCRYPTION_ADMIN</td>
<td>Manage table encryption</td>
</tr>
</tbody>
</table>

Table 6.4: List of ungranted privileges to the administrator account [53]

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.103</td>
<td>Address of the MySQL server on our private network</td>
</tr>
<tr>
<td>10.0.0.206</td>
<td>Address of our compute instance on our private network</td>
</tr>
</tbody>
</table>

Table 6.5: IP addresses on our virtual cloud network
Chapter 7

Results
7.1 IAM

7.1.1 Documentation

7.1.1.1 Restrictive policy with "!=

7.1.1.1.1 Tested policy

allow THESIS_USERS to read object-family in compartment Pwn where request.user.id!='ocid1.user.oc1...[...]hurwta'

Listing 7.1: Tested policy

7.1.1.1.2 Result  The policy is well enforced as we notice that the user does not have access to the bucket. If the
request.user.id!='ocid1.user.oc1...[...]hurwta' condition is removed, the user has access to the bucket.

However, we have noticed that the policies take some time (2 - 5 min) to enter into effect, when issuing a change from allowing to restricting. In this context, we managed to get such a result shown in figure 7.1.

This figure was obtained while having an enforced policy which stated that our user should not have had access to the bucket. We can see on the figure that OCI is enforcing it, as the warning messages show that this bucket should be unavailable and we should not know this bucket exists. However, we can still see an OCID, proving that this resource exist and providing us a way to access it. An attacker that wants to successfully attack a live running environment might take advantage of time during policy changes in this way.

7.1.1.2 Policy precedence

As mentioned before, this test is relevant to make, as "!=" policies are indeed not allowing the user that is restricted in the "!=" statement, as shown above.

7.1.1.2.1 Case 1

Allow group 'THESIS_USERS' to read buckets in
    compartment Pwn where request.user.id='ocid1.user.oc1...
    aaaaaaaaaazqmip7mikqken7wkwx3agegi64dc361e5fzg2fca4raohurwta'
Figure 7.1: View of the console from the user THESIS_USER after a change in policy within about 5 minutes.
Allow group 'THESIS\_USERS' to read buckets in
  compartment Pwn where request.user.id='ocid1.user.oc1
  ..
  aaaaaaaaazbqmip7mikqken7wkwx3agegi64dc36le5fzg2fca4raohurwta

Listing 7.2: "Complementary" policies affecting the group THESIS\_USERS

7.1.1.2.2 Result The first policy simply allows all the other users than our
user to access the bucket, while the second allows our user. The result of this
policy is that the targeted user can see the bucket. This is conform with our
expectations.

7.1.1.2.3 Case 2

Allow group 'THESIS\_USERS' to read buckets in
  compartment Pwn where request.user.id='ocid1.user.oc1
  ..
  aaaaaaaaazbqmip7mikqken7wkwx3agegi64dc36le5fzg2fca4raohurwta

Allow group 'THESIS\_USERS' to read buckets in
  compartment Pwn where request.user.id!='ocid1.user.
oc1..
  aaaaaaaaazbqmip7mikqken7wkwx3agegi64dc36le5fzg2fca4raohurwta

Listing 7.3: "Complementary" policies affecting the group THESIS\_USERS
written in the opposite order

7.1.1.2.4 Result The result of this policy is the same as above. We con-
clude that order in which the policies are written does not matter, and that
attempting to restrict a particular user from attempting to read a bucket in this
way does not work.

7.1.1.2.5 Discussion If case 2 is used in a scenario where one wants to
restrict a user to access the bucket, too wide privileges could be granted in this
manner to this user as the policies does not work as their logic would expect.
We think that this could cause misconfigurations and unsafeness in a tenancy.
7.1.1.3 Invalid parameter in a policy

In our second case, the administrator attempts to make a policy but fails to
type the OCID of the user correctly.

7.1.1.3.1 Tested policies

allow THESIS_USERS to read object-family in compartment
   Pwn where request.user.id!=’ocid1.user.oc1..[...]
   hurwtaz’
allow THESIS USERS to read object-family in compartment
   Pwn where request.user.ids!=’ocid1.user.oc1..[...]
   hurwtaz’
allow THESIS USERS to read object-family in compartment
   Pwn where request.users.id!=’ocid1.user.oc1.[...]
   hurwtaz’

Listing 7.4: Sample of tested policies, one by one. Each of them contains a
spelling error

We have also tried to find out if the following statement could be parsed
as correct and enforced. If, by mistake, a wildcard character is entered in
the policy statement, we imagine it could make the variable match a pattern.
For comparison purposes, the second statement is the correct way, according
to the documentation [63], to match make a variable match a pattern. It is
surrounded with slashes. Because of this reason, we think it is unlikely that
the right syntax will be achieved by mistake.

allow ’THESIS USERS’ to read object-family in
   compartment Pwn where request.user.id!=’ocid1.user.
   ocl..[...]hurwt*’
allow ’THESIS USERS’ to read object-family in
   compartment Pwn where request.user.id= /ocid1.user.
   ocl..[...]hurwt*/

Listing 7.5: Tested policies, one after the other. Those policies test the
behavior for pattern matching in strings.

7.1.1.3.2 Result The first three policies validated and we see that the user
has access to the bucket which means the policy was not enforced. We con-
clude that spelling mistakes are not checked by OCI and could lead to policies
not reflecting an adequate level of access to an unaware administrator.
The first statement of the second batch of policies with the parameter `request.user.id!=ocid1.user.octl...[...]hurt*` let the user access the bucket. The second statement effectively did not allow the user to access the bucket. We conclude that it is unlikely that an administrator will ban or authorize more resources or users than intended due to a mistakenly added *.

### Discussion

The impact of a spelling mistake is different whether the condition is constructed with `"="` or `"!="`. Policy statements are created to allow. A statement with a condition constructed with `"="` is therefore meant to allow matching a specific resource or user. If the condition is misspelled, the user or resource is not granted access. By default, a mistake restricts.

However, a statement with a condition constructed with `"!="` is therefore meant to allow a group of users or resources but a specific resource or user. If the condition is misspelled, the user or resource meant to be excluded is granted access. By default, a mistake allows.

In case of mistake, the first case is more desirable than the second. It is better to restrict than allow by default. In the first case, users that are meant to have access will contact the administrator in order to retrieve their access and fix the policy. The second case breaks confidentiality.

Therefore, in order to restrict a user, if a spelling mistake is made on a `"!="` condition and the OCID as shown in our tested policies, the restriction is going to fail. It is also difficult to understand from a glimpse who the administrator is restricting as human readable usernames are not valid variables for this purpose. This is most probably why this method is not recommended by OCI.

We couldn’t find official resources from OCI that specified that policies could contain policies error. It is rather surprising, as spelling errors on strings before the WHERE clause trigger a failure to validate the policy creation. One could then incorrectly assume that the whole policy is spelled check, while it is not. An administrator could realize too late or never realize that the policy is wrong, worst-case scenario being that the policy grants too wide permission because of a misspelled `"!="` condition.

### Suggested fixes

The reason for not performing resource validation during policy creation is to leave the possibility for resources to be created post-definition of policies. For this reason, we do not think it is relevant to enforce spellchecking on every part of the policy.
However, we think that the OCI documentation should explicit which part of the policy statement is spell checked and which ones are not. For instance, the documentation [63] states that strings are case insensitive. We suggest that such a warning would be put in a similar place.

To the administrators, we suggest creating policies with Infrastructure as Code tools such as Terraform for instance. Terraform allows for the use of variables based on the declared infrastructure when creating policies, which can reduce the number of spelling errors and make policies easier to track. We also suggest to not use policies that are allowing by default with a more fine grained WHERE clause that restricts certain items (with “!=”) but rather to specify explicitly what is allowed with “=”.

7.1.2 ABAC

7.1.2.1 Accessing the private bucket through a computing instance

7.1.2.1.1 Payload  The attacker creates a compute instance tagged with the tag “priv-tags.bucket-access”. The attacker has access to the compute instance with root access, as they are its creator.

The attacker writes and executes a payload on the compute instance. The payload ([37] or Annex A) is written in python and works as follow (see Figure 7.2):

- We import the OCI python SDK
- We authenticate the instance
- We read the content of the bucket of the summary as the instance and generate a report containing the names of all the files
- We upload the report to the bucket

These steps constitute the report issuer, which is executed as a service every 30 seconds on the compute instance.

7.1.2.1.2 Results  On the compute instance, we can check the status of the report issuer.

```
[opc@app-vm ~]$ sudo systemctl status reportissuer
reportissuer.service - Issue Report Job
        Loaded: loaded (/etc/systemd/system/reportissuer.
              service; enabled; vendor preset: disabled)
```
Figure 7.2: Illustration of the report issuer steps
Active: active (running) since Thu 2023-07-27 18:04:10 GMT; 5min ago
Main PID: 8461 (python3)
  Tasks: 1
Memory: 4.8M
  CGroup: /system.slice/reportissuer.service
        8461 python3 /home/opc/reportissuer.py

Jul 27 18:06:47 app-vm reportissuer.py[8461]: Creating a report
Jul 27 18:06:47 app-vm reportissuer.py[8461]: ### Report generated 2023-07-27 18:06:47.909397
Jul 27 18:07:17 app-vm reportissuer.py[8461]: Creating a report

Listing 7.6: Status check of the report issuer on the compute instance.

We can see that the report issuer has been running a few times without error and generated reports of the content of the bucket. We can check with an administrator view that the report was generated on figure 7.3. We cannot check that the report was correctly added to the bucket from the developer view of the console. However the developer can retrieve all information about the bucket from the compute instance and adapting the script.

Here is the content of the downloaded summary.txt from the bucket:
generated/summary.txt (0.1103515625K)
secret.txt (0.0849609375K)
### Report generated 2023-07-27 18:35:38.857269

Listing 7.7: Content of the file generated/summary.txt

This proves that our reportissuer from the compute instance that we control as the attacker has access to all the content of the bucket thesis-report and can also upload content to the bucket.

Note that the compute instance, the bucket and the tags do not have to exist in the same compartment for this to work, as long as the object namespace is known for the attacker. This means multiple administrators writing policies in their own compartment could inadvertently conjointly create an unsafe tenancy.
Figure 7.3: View of the console from the administrator account. The file "summary.txt" has been uploaded in the folder "generated".

7.1.2.2 Accessing the private bucket through a serverless function

7.1.2.2.1 Payload The attacker creates a payload function and uploads it to the image repository. The payload [37] or Annex A is written in python and works as follow (see Figure 7.4):

- We import the OCI python SDK
- We authenticate the instance
- In the targeted bucket, we read the content of the file secret.txt
- We upload a new file, "secret.processed.txt" which contains a custom message and the content of the file "secret.txt"

After uploading the function to the repository, the attacker adds the tag "priv-tags.bucket-access" to the function.

The attacker deploys the function by entering

```
fn invoke <appname> <app function> --content-type application/json
```

Listing 7.8: Command to deploy and execute a serverless function.
Figure 7.4: Illustration of the serverless function steps
7.1.2.2 Results  Our file "secret.processed.txt" is correctly uploaded to the bucket, as shown in figure 7.5 which is a view obtained from the administrator console. The developer still cannot see the bucket from the console. Retrieving it's content shows our message with the content of the file "secret.txt". We conclude that we had access in read and write to our bucket.

A very secret message

Listing 7.9: Content of the original file "secret.txt"

I have access to the bucket, the file said:A very secret message

Listing 7.10: Content of the retrieved file "secret.processed.txt"

7.1.2.3 Execution of the enumerators

We choose to apply the enumerators with all the unsafe policies added. Therefore, policies simulating scenario 1 and 2 are all in place. Our aim is to find out if the enumerators can detect privilege escalation possibilities of the type described above.
7.1.2.3.1 oci-enum We first decide to run the tool as the superuser, with no specified compartment. The aim of this step is to see the amount of information that we can obtain with very little initial information as an auditor. By default the tool places itself at the tenancy level.

OCI Enumerator - INFO: Enumerating compute instances...

OCI Enumerator - INFO: Enumerating IAM

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Default domain</td>
<td><a href="https://idcs-8">https://idcs-8</a> bcc76e8364d4bde9b677a254d6878bd.identity.oraclecloud.com:443</td>
</tr>
</tbody>
</table>

oci-enum -c ~/.oci/config -p DEFAULT --region eu-stockholm-1 --all

Name        | Compartment ID | Description | State          |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pwn</td>
<td>ocid1.tenancy.oc1..[...]bnfa</td>
<td>On purposed misconfigured test compartment</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

OCI Enumerator - INFO: Enumerating Object Storage

Listing 7.11: Result of oci-enum, superuser credentials and no specified compartment (tennacy scoped), truncated.

The tool has an awareness of the domain which contains the policy although they are not explicitly presented to us.

However, the enumerator does not find any of our compute instances or object storage. This is due to the fact that the enumerator uses OCI APIs such as "list_bucket" that searches for resources within a compartment only and not the sub-compartments. OCI does not have APIs that searches for resources at compartment and sub-compartment included. Resources are shown at the specified compartment which is the tenancy, as none is specified. In order to
enumerate everything, an attacker would have to launch the enumerator with each of the specified compartments manually, script it or modify the enumerator.

We then decide to specify the OCID of the "Pwn" compartment, where all our resources (compute instances, buckets) reside.

```
oci-enum -c ~/.oci/config -p DEFAULT --region eu-stockholm-1 --all --compartment-id ocid1.compartment.
oc1..[...]hxra
```

OCI Enumerator - INFO: Enumerating compute instances...

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Public IPs</td>
</tr>
<tr>
<td>dev-vm</td>
<td>ocid1.instance.oc1.eu-stockholm-1.zlmda 129.XXX 10.0.9.17 STOPPED</td>
</tr>
<tr>
<td>instance-20230708-1808</td>
<td>ocid1.instance.oc1.eu-stockholm-1.wrgla 129.XXX 10.0.0.206 RUNNING</td>
</tr>
</tbody>
</table>

OCI Enumerator - INFO: Enumerating Object Storage

<table>
<thead>
<tr>
<th>Bucket Name</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>thesis-reports</td>
<td>axsqbv0a4cus</td>
</tr>
</tbody>
</table>

Listing 7.12: Result of oci-enum, superuser credentials and Pwn compartment, truncated.

The enumerator retrieves the bucket and the instances correctly.

From the standpoint of the developer, we now know that in order for them to get valuable information, the compartment of interest needs to be specified.

```
oci-enum -c ~/.oci/config -p DEFAULT --region eu-stockholm-1 --all --compartment-id ocid1.compartment.
oc1..[...]hxra
```

OCI Enumerator - INFO: Enumerating compute instances...

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Public IPs</td>
</tr>
<tr>
<td>dev-vm</td>
<td>ocid1.instance.oc1.eu-stockholm-1.zlmda 129.XXX 10.0.9.17 STOPPED</td>
</tr>
<tr>
<td>instance-20230708-1808</td>
<td>ocid1.instance.oc1.eu-stockholm-1.wrgla 129.XXX 10.0.0.206 RUNNING</td>
</tr>
</tbody>
</table>
OCI Enumerator - INFO: Enumerating IAM

OCI Enumerator - ERROR: Authorization failed or requested resource not found Request: list_domains

OCI Enumerator - ERROR: Authorization failed or requested resource not found Request: list_users

<table>
<thead>
<tr>
<th>Name</th>
<th>Compartment ID</th>
<th>Description</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>child-comp</td>
<td>ocid1.compartment.oc1..xra</td>
<td>Child</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>comp of pwn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-compartment</td>
<td>ocid1.compartment.oc1..xra</td>
<td>Sub compartment to test heritage</td>
<td>ACTIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OCI Enumerator - INFO: Enumerating Object Storage

OCI Enumerator - ERROR: You do not have authorization to perform this request, or the requested resource could not be found. Request: list_buckets

OCI Enumerator - INFO: Enumerating Load Balancers

OCI Enumerator - ERROR: Authorization failed or requested resource not found. Request: list_load_balancers

Listing 7.13: Result ofoci-enum, developer credentials and Pwn compartment, truncated.

Also note that the compartment ID is not the one of the compartment on the row but the one of the parent, which questions on the utility of the column as we already know this information... An attacker can not retrieve the OCID of the compartment of interest easily from this output and therefore dive deeper in the compartment tree.

As expected, the enumerator returns that our developer does not have access to the buckets but has access to some compute instances. The tool also has no awareness of the policies as the developer. Thus, an attacker or an auditor would not be able to find from this output that the tested credentials have
extra hidden permissions.

However, this tool also checks permissions for instance principals. The attacker can run the tool with the credentials of the compute instance they have access to.

The tool works with a instance metadata file from the version 1, which is now deprecated by OCI.

We retrieve the metadata file by executing the following command on the instance:

```
```

Listing 7.14: Command to retrieve the identity file for the compute instance.

For unknown reasons, the tool returns an error message "Invalid authentication details provided" when fed with the retrieved json file as it is. We did not manage to fix the metadata file or the tool to retrieve the permissions of the compute instance. If the authentication was successful, we would have expected the tool to detect the bucket as existing and available, as read access to the bucket also marks it as available when listing resources.

7.1.2.3.2 scout-suite

The following command executes the tool with the administrator credentials:

```
(venv) shanly@WSW70VTHW3:$ scout oci --profile DEFAULT
2023-08-02 15:30:27 WSW70VTHW3 scout[4298] INFO Launching Scout
2023-08-02 15:30:27 WSW70VTHW3 scout[4298] INFO Authenticating to cloud provider
2023-08-02 15:30:28 WSW70VTHW3 scout[4298] INFO Gathering data from APIs
2023-08-02 15:30:28 WSW70VTHW3 scout[4298] INFO Fetching resources for the Identity service
2023-08-02 15:30:28 WSW70VTHW3 scout[4298] INFO Fetching resources for the Object Storage service
2023-08-02 15:30:28 WSW70VTHW3 scout[4298] INFO Fetching resources for the KMS service
2023-08-02 15:30:29 WSW70VTHW3 scout[4298] ERROR objectstorage.py L16: Failed to get Object Storage namespace: 'str' object has no attribute 'items'
2023-08-02 15:30:29 WSW70VTHW3 scout[4298] ERROR objectstorage.py L36: Failed to get Object Storage buckets: Parameter namespaceName cannot be None, whitespace or empty string
```
Listing 7.15: Result from the execution of ScoutSuite straight after its download. The execution fails.

The first execution of the tool required a fix from us, as the tool could not retrieve the Object Storage namespace properly. Our first task was thus to modify the source code of the tool. The issue was in the function "get_namespace".

```python
async def get_namespace(self):
    try:
        response = await run_concurrently(
            lambda: list_call_get_all_results(self._client.get_namespace))
        # for some reason it returns a list of chars instead of a string
        return ''.join(response.data)
    except Exception as e:
        print_exception(f'Failed to get Object Storage namespace: {e}')
    return None
```

Listing 7.16: Code of origin for the function get_namespace in ScoutSuite

A commentary in the code says "for some reason it returns a list of chars instead of a string". We think execution fails because the API does not return a list of characters anymore. We implemented the following fix.

```python
async def get_namespace(self):
    try:
        response = await run_concurrently(
            lambda: self._client.get_namespace()
        )
        return response.data
    except Exception as e:
        print_exception(f'Failed to get Object Storage namespace: {e}')
    return None
```

Listing 7.17: Our suggested fix for the function get_namespace in ScoutSuite

A pull request on the open source repository was opened\(^1\).

Figure 7.6: Report of the tenancy from ScoutSuite, view of the Identity Dashboard.

Figure 7.7: Report of the tenancy from ScoutSuite, view of the policies.

Figure 7.8: Report of the tenancy from ScoutSuite, view of the warning "Policy Affects User".
After a successful execution of the tool thanks to our patch, the identity dashboard shows us items which could make our tenancy unsafe and evaluate the overall security of our tenancy as shown in figure 7.6.

The report shows all of our policies (see Figure 7.7). Although the identity dashboard give us warnings, the useful warning shown in figure 7.8 revealed itself to be a false positive, because the concerned policy is a default policy added by OCI. It was also very difficult to pinpoint the flagged policy by the warning, as there was no way for us to find the flagged policy unless we read the content of each of them and compared it to the warning message. In addition, there was no warning on the fact that a user might have too wide permissions compared to the policy statements.

For the same reasons as described for "oci-enum", buckets are not detected. However, there is no possibility of specifying the compartment as we execute the tool. In order to specify a compartment, we need to modify the configuration file used for the oci cli. This is not recommended as it makes it difficult to script in order to explore the discovered compartments. In addition, the configuration file is used for the oci cli which might disturb a user used to his default configuration of set profiles when using the official querying tool.

We now specify the compartment as an option in the command.

As we specified the compartment in our profile now, the tool finds the bucket (see Figure 7.9). However, it has no awareness of the policies defined
at the tenancy level anymore as the tenancy and compartment reports are separated. This tool does not prove itself useful to detect our misconfiguration, and reveals itself to be quite unpractical for OCI auditing in general.

7.1.2.4 Discussion

Using enumerators relying on OCI API fails because OCI itself does not have an endpoint to check for this sort of permissions. Moreover, the OCI API does not have an easy awareness of privileges associated to a user or all the resources contained in the tenancy. This makes it difficult for any tool or human to find a misconfiguration of this type. The discussion will tackle how administrators and Oracle can work conjointly to limit and prevent the surface attack of these misconfigurations.

7.1.2.4.1 Administrator posture If an organisation decides to implement ABAC, administrators have choices to make in terms of the organisation of tags and tag namespaces. If the administrator wants to give the developer access to certain defined tags, while some other defined tags are used for matching rules for dynamic groups and therefore not available to the developer, multiple solutions could be studied:

- The defined tag that is used for the dynamic group is in the same namespace as the other privilege-less tags, for example because they relate organisationally

- A tag namespace is dedicated to the tags that define matching rules for dynamic groups, for example because it is easier to keep track of all the tags that grant rights if they are together

- Each defined tag that is used for a dynamic group belong to its own separated tag namespace, for example to limit the listing ability and discovery of an user that is allowed to used a defined tag.

All these choices have advantages and disadvantages. The choice depends on how the organisation wants the tag name-spaces to be organized, similarly to compartments. As tags are management/organizational tools, it is important to have an organization of tags that makes sense for the company.

Whatever the final choice is, it is important to restrict access in a where clause. As we discussed in the part 7.1.1.3.4, it is safer to design policies with "=" conditions than "!=", which is a constraint that could determine how
tag namespaces could be organised. For example, let's consider the scenario where there are 10 non sensitive tag namespaces and 1 sensitive tag namespace, that are related by organisational meaning (for example departments or teams, and so cannot be regrouped further). In order to grant a group access to the 10 non sensitive tag namespaces, the names will have to be enumerated one after the other with a "=" condition. With a "!=" condition, only the one sensitive tag namespace needs to be written. This choice makes the policy unsafe is the sensitive tag namespace happens to be misspelled.

We intuit that such a policy, because it is faster and more intuitive to write for an administrator will be a go-to choice by default. We advise to not write such policies but instead write explicitly the name of allowed resources, even in the case they are numerous.

7.1.2.4.2 Oracle's posture Documentation code is likely to be copy pasted, so it is important that Oracle's default posture is as secure as possible. An example scenario of tagging is presented in OCI’s documentation "Managing tags and tag namespaces" [64] which aims to give an overview and introductory tutorial of tagging. The section "Required Permissions for Working with Defined Tags" aims to inform the administrator about policy building for users to be able to use tags.

• To allow a group to simply view the tag namespaces in the tenancy (or in a compartment) requires inspect access:
  Allow group GroupA to inspect tag-namespaces in tenancy

• To allow a group to read the tag definitions contained in tag namespaces requires read access:
  Allow group GroupA to read tag-namespaces in tenancy

• To allow a group to apply, update, or remove a defined tag for a resource requires the use access on the tag namespace
  Allow group GroupA to use tag-namespaces in tenancy

These policies are the first three policies explained in the documentation, which make them likely to be copied by an administrator and implemented. These
are unsafe in a context where the administrator aims to implement ABAC policies, as there could be tag namespaces associated to permissions. These policies allow users to use any tags, which is unsafe. Stakes of such a policy are not discussed on this page of the documentation.

The documentation though further explains how to restrict the policy:

"To allow usage of a specific tag namespace or namespaces, use a where clause with the target.tag-namespace.name variable. For example:

Allow group GroupA to use tag-namespaces in tenancy where target.tag-namespace.name='Operations'

But does not explains the stakes of why such an implementation is more safe and important, which might be overlooked by an administrator.

The first occurrence of such a warning appears in "Managing access with tags" [65]:

"If your organization chooses to create policies that use tags to manage access, then ensure that you have appropriate controls in place to govern who can apply tags. Also, after policies are in place, keep in mind that applying tags to a group, user, or resource has the potential to confer access to resources.

Before you create a policy that specifies a tag on either a target or a requestor, ensure that you are aware of:

• all the potential requestors (users, groups, dynamic groups) that carry the tag
• all the resources that carry the tag

Before you apply a tag to a resource, ensure that you are aware of any policies in place that include the tag and could impact who has access to the resource."

This proves that this issue is a known issue for OCI. One could argue that although the warning is clear as OCI attempts to address the issue, it is hard to find unless we are specifically looking for it, as it does not appear in general pages on tagging. Warning such as these does not appear in the console for example when creating policies. We think that, in addition to prevention, OCI should put in place appropriate tools to ease policy and tag rights tracking, such as permission listing APIs.
7.1.3 Others

Other pitfalls were found to be worthy to be noted and evaluated.

7.1.3.1 Buckets vs Objects

During our study we fell into the following pitfall: we found that the access for the metadata of the bucket and the content of the bucket itself is governed by two different key words.

- "bucket", representing the bucket containing the objects
- "objects", the contained objects themselves.

object-family include both bucket and objects. For a detailed explanation, authorizing "read buckets" instead of "object-family" results in:

- All the buckets are listable in the compartment
- All information about the buckets are available (OCID, object count, size of the bucket)
- The content (objects) is not available. Instead the warning message displays "Either the bucket does not exist or you are not authorized to access it" (see Figure 7.10). However, the bucket evidently exists as we can see its OCID and the associated metadata. Metrics for it are even available for us (see Figure 7.12)
- However, no operation on the bucket is executable as the bucket is considered unreachable by us. For example deletion operations do not succeed (see 7.11).

This resulted in misconfigurations during testing as we thought that bucket designated the whole object storage and were surprised to see we could not access the content of the bucket. We thought it was worthwhile to note as other people could make the same mistakes.

We shall now evaluate the consequences of such a misconfiguration:

- In the case where the metadata is authorized instead of the metadata and the content, resulting permissions are too restrictive compared to what is intended. This misconfiguration does not result in particular risk of unauthorized access, although functionalities will be limited for users.
Figure 7.10: View of the bucket from the console. The metadata is readable but the content is not accessible.

Figure 7.11: Failed attempt at deletion of the bucket
In the case where the content is authorized instead of the metadata only, such a misconfiguration indeed results in a confidentiality breach. However, as we think that authorizing metadata is non-standard behavior and more intentional, an administrator might look up how to configure such a policy in the first place. Thus, however more risk entailing, we think that this scenario is less likely to happen.

To conclude, such a misconfiguration entails no particular risk. Awareness was risen for debugging purposes.

7.1.3.2 Case insensitiveness for policies

As we mentioned in the background, policies are case insensitive, while resources are not. We believe it is worthwhile to remind as this could be the cause for unsafe misconfigurations. An example of misuse is given in the OCI documentation:

"Condition matching is case insensitive. This is important to remember when writing conditions for resource types that allow case-sensitive naming. For example, the Object Storage service allows you to create both a bucket named "BucketA" and a bucket named "bucketA" in the same compartment. If you write a condition that specifies "BucketA", it will apply also to "bucketA", because the condition matching is case insensitive."

We shall now evaluate the severity of such a misconfiguration. The worst case is applying a policy affecting both buckets, when only one of them is intended to be targeted. This results in permissions being too permissive as writing policies can only grant more rights, not restricting them. Each written
policy statement affecting one bucket will open more permissions for the other bucket. Such a misconfiguration could then result in a threat for confidentiality and is estimated to be important due to the easiness of such a mistake.

### 7.1.4 Conclusion

In our security assessment of the IAM service, it has been found that, in general, it is difficult for administrators to tell who has access to what based on the policies for several reasons:

1. The policies, as we saw on our "complementary" policies, can not have the effect we expected in terms of authorization.

2. Spelling errors can be made and are difficult to detect. They have to be reviewed manually.

3. The most notable reason is that there is no overview of policies and consequential permissions provided by OCI. Therefore, the full responsibility of keeping track of policies falls on the administrator.

4. Third-party enumerators/auditors fall short to help administrator into having a good overview of permissions.

Furthermore, we found a real necessity to enhance the enumerators and make them efficient, such as including recursive searching for compartments and their sub-compartments. This shortcoming is also due to the result returned by the OCI API itself.

All these reasons contribute to making policy writing unsafe. We believe that solutions, such as automated tools whether they are third-party tools or provided by Oracle, writing more extensive documentation, implementing console warning or safeguards against spell checks, selecting only existing resources, would help administrators secure their tenancy in a more straightforward way.

We have managed to perform privilege escalation when unsafe ABAC policies were written. Our work also made apparent that compromising compute instance poses as much risks issues as compromising user accounts, as compute instances can also be given permissions. An access to the SSH key of the compute instance could lead to permission escalation. It’s important to note that the compromise of SSH keys is a prevalent error that occurs due to their frequent exposure on public repositories. The widespread mishandling of SSH keys adds to the complexity of securing IAM.
We also fell into other pitfalls which were esteemed worthwhile to share as they seemed easy to make and could have security consequences. The aim is to alert and sensitize administrators.

Our conclusion for this service is to proceed with caution.
7.2 Oracle MySQL Managed Database

7.2.1 Attack 1: Replacing the administrator account with a banned username

7.2.1.1 Through the console during the database creation

Attempting to create a database with an administrator name such as "ociadmin", "ociadmin*" and '*ociadmin" in the console will fail as presented in figures 7.13, 7.14 and 7.15. Although the documentation says that "ociadmin" only is reserved, we can conclude from these tries that anything fitting the syntax '*ociadmin* is banned. It is also true for other banned usernames in table 6.2 such as administrator.

It is unclear why the documentation does not reflect accurately the usernames that are unavailable and why '*ociadmin* and others are all banned. A probable explanation is to safeguard against eventual field injections that could evaluate to ociadmin after being parsed.

7.2.1.2 Through CREATE USER in the database

The following command creates an user in the database called "ociadmin" with host "%".

```
CREATE USER 'ociadmin'@'%' IDENTIFIED BY 'Ppassword1234 %';
```

Query OK, 0 rows affected (0.0032 sec).
Figure 7.14: View from the console at database creation. The specified username for the user administrator contains a banned username.

Figure 7.15: View from the console at database creation. The specified username for the user administrator contains a banned username.
Listing 7.18: Command to create a user "ociadmin" on host % in MySQL

The command succeeds because the hosts are not in conflict. Unlike restrictions in the console, we can actually create arbitrary usernames with the host "%" just like the user administrator in the console.

However the user administrator "myadmin" possess privileges that are not transferable. Indeed, listing 6.2.4.7 shows on row 3 and 4 that granted privileges are not grantable to the created user.

In order to simulate the fact that an user administrator was created from the console with the "ociadmin" username, we will thus need to find a way to rename the user administrator itself.

7.2.1.3 **Renaming the original user administrator account**

First we need to delete the former ociadmin user on host % as two users with the same name cannot be defined on the same host.

```sql
SQL > DROP USER 'ociadmin'@'%';
Query OK, 0 rows affected (0.0035 sec).
SQL > RENAME USER myadmin TO ociadmin ;
Query OK, 0 rows affected (0.0049 sec).
```

Listing 7.19: Command to rename the user "myadmin" to "ociadmin" on host % in MySQL

We can check with a forbidden query how the server sees us:

```sql
UPDATE mysql.user SET File_priv = 'Y' WHERE User = 'ociadmin';
ERROR: 1142: UPDATE command denied to user 'ociadmin'@'10.0.0.206' for table 'user'
```

Listing 7.20: Executing a forbidden query in order to get an error message. It attempts to set our FILE privileges to Yes.

The server indeed sees us as ociadmin connecting from 10.0.0.206. We can also check our privileges:

```sql
SQL > SHOW GRANTS \G
*************************** 1. row
Grants for ociadmin@%: GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, PROCESS, REFERENCES, INDEX,
```
ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE ON *.* TO 'ociadmin'@'%' WITH GRANT OPTION

*************************** 2. row
Grants for ociadmin@%: GRANT APPLICATION_PASSWORD_ADMIN, AUDIT_ADMIN, BACKUP_ADMIN, CONNECTION_ADMIN, FLUSH_OPTIMIZER_COSTS, FLUSH_STATUS, FLUSH_TABLES, FLUSH_USER_RESOURCES, REPLICATION_APPLIER, ROLE_ADMIN, XA_RECOVER_ADMIN ON *.* TO 'ociadmin'@'%' WITH GRANT OPTION

*************************** 3. row
Grants for ociadmin@%: REVOKE INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, CREATE VIEW, CREATE ROUTINE, ALTER ROUTINE, EVENT, TRIGGER ON 'mysql'.* FROM 'ociadmin'@'%

*************************** 4. row
Grants for ociadmin@%: REVOKE CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, CREATE VIEW, CREATE ROUTINE, ALTER ROUTINE, EVENT, TRIGGER ON 'sys'.* FROM 'ociadmin'@'%

*************************** 5. row
Grants for ociadmin@%: GRANT PROXY ON ''@'' TO 'ociadmin'@'%' WITH GRANT OPTION

*************************** 6. row
Grants for ociadmin@%: GRANT 'administrator'@'%' TO 'ociadmin'@'%' WITH ADMIN OPTION

6 rows in set (0.0006 sec)

Listing 7.21: Listing grants for the current user in MySQL

We have the same privileges as before and our username was correctly updated.
7.2.1.4 Results

We see from the result of this command that OCI policies on naming conventions are not enforced within the database as we succeeded to create an administrator with the name `ociadmin`.

7.2.1.5 Discussion & suggested fixes

MySQL has no native solution for forbidding username creation matching a certain string.

In order for this to be enforced, we suggest two solutions

- using **ADD CONSTRAINT**. A constraint is a rule or condition that is applied to a table column or a group of columns. Inserted rows requests that make the constraint false are rejected. We could add such a constraint to prevent insertion of reserved usernames with host "%" like so:

```
ALTER TABLE mysql.user ADD CONSTRAINT check_unbanned_username CHECK (User != 'ociadmin' AND Host != '%');
```

Listing 7.22: Example of constraint to enforce banned usernames to not be picked by users.

As we cannot alter the table `mysql.user` as a user admin, this code cannot be validated for user creation and we are unsure if such a constraint would effectively make forbidden user creation fail. But such a constraint could also not be removed by any user administrator created with the database. The following code was tested on a table of our creation:

```
SQL > ALTER TABLE users ADD CONSTRAINT
   check_unbanned_username CHECK (User != 'ociadmin' OR Host != '%');
Query OK, 0 rows affected (0.0203 sec)

Records: 0 Duplicates: 0 Warnings: 0
MySQL 10.0.0.103:33060+ ssl pwn SQL > INSERT INTO users(user, host) VALUES ('ociadmin', '%');
ERROR: 3819: Check constraint 'check_unbanned_username' is violated.
```

---

Listing 7.23: Tested query in an unprivileged database to check the efficiency of the constraint.

The results show that entry with "ociadmin" and "%" as user and host are banned, but not other entries. The constraint can be modified to include all reserved usernames.

- Modifying the source code or creating a plugin. It is uncertain whether Oracle decided to create their managed service by relying only on built-in functions, modifying the source code or programming plug-ins on top of the MySQL source code. Given that we do not know, we will present some pseudo code that could be added to the existing version of MySQL during username changing or username creation:

  1. Creating a list containing the reserved usernames
  2. Comparing the inputted username to the list of the reserved usernames
  3. Rejecting the user creation/modification of the database if a match is found

As during creation, the database is not up and running yet, such a check cannot be performed by the MySQL server and has to be done by OCI directly, which is why the error message is displayed only in the console, which is considered sufficient. After database creation, OCI cannot "see" anymore what is happening in the MySQL server as the tenant is not using the OCI API or console anymore, which is why such checks should be implemented.

7.2.2 Attack 2: Accessing the file system

7.2.2.1 Reading files in the filesystem

In order to test our access to the file system, we could try to read the content of an existing file. How to find a file of interest? Some files are common to
every UNIX system. Since we know that a mysql server is running, we also know what kind of files could exist in the file system.

Running the command

```
SHOW VARIABLES;
```

Listing 7.24: Command to show all variables in MySQL

can also help uncover existing paths or interesting files in the system as they can be the value of some variables. In the following work, the file paths will be left unspecified as we do not want to risk accessing illegal files.

The following commands are executed as the user "myadmin":

```
SQL > LOAD DATA LOCAL INFILE '' INTO TABLE results;
ERROR: 2068 (HY000): LOAD DATA LOCAL INFILE file request rejected due to restrictions on access.

SQL > LOAD DATA INFILE '' INTO TABLE results;
ERROR: 1045 (28000): Access denied for user 'myadmin'@'%' (using password: YES)
```

Listing 7.25: Executed commands to prove read access in the filesystem

The first command fails as expected since the user does not have the FILE permission which is necessary with the LOCAL keyword. Why the second statement fails is unclear. It was not of crucial interest as this statement does not concern the filesystem where the database is in, in any case.

### 7.2.2.2 Writing files in the filesystem

The following command writes a file at path "/tmp/myfile.txt". Every UNIX system possesses a temporary (tmp) folder, which is writable by most users and cleared after the system stops. This folder is usually empty. By attempting to create such a file, we are sure not to replace any existing file in the file system, if the operation succeed.

```
SELECT * FROM table INTO OUTFILE '/tmp/myfile.txt'
```

ERROR 1227 (42000): Access denied; you need (at least one of) the FILE privilege(s) for this operation

Listing 7.26: Executed commands to prove write access in the filesystem

The operation failed.
7.2.2.3 Conclusion

Conforming to the documentation, these operations did not succeed. The FILE privilege is correctly enforced by the server. Moreover, the error messages do not give hints about existence of files. Usually, error messages will differ according to if the file does exist ("error: this file does not exist") or if the file exist and the user does not have the right to access it ("error: you do not have the sufficient permission to access this file"). By voluntarily not making the error message explicit, Oracle prevents file discovery through error messages.

7.2.3 Attack 3: compromission of the admin account locally

7.2.3.1 Changing the password

The following commands attempt to change the password of the "ociadmin" user.

```sql
SQL > ALTER USER 'ociadmin'@'127.0.0.1' IDENTIFIED BY '
   Ppassword1234%';
ERROR: 1227: Access denied; you need (at least one of)
   the SYSTEM_USER privilege(s) for this operation

SQL > SET PASSWORD FOR 'ociadmin'@'127.0.0.1' = '
   Ppassword1234%';
ERROR: 1044: Access denied for user 'myadmin'@'%' to
database 'mysql'
```

Listing 7.27: Attempts to change the managed administrator password.

With this error message in the first statement, we learn that we require the SYSTEM_USER privilege for such an operation to succeed. According to the documentation [66], we need to this privilege to modify users that are granted the SYSTEM_USER privilege, including "ociadmin". This feature is a way of preventing cases were it would be convenient to grant password permissions in general for an user administrator, except on a few select sensitive accounts such as "ociadmin". This way, we can still change the password of all the users that we create, while not being able to modify the managed administrator.

It is unclear why the second statement fails. The documentation [67] states that: "Setting the password for a named account (with a FOR clause) requires the UPDATE privilege for the mysql system schema.", which is a privilege that we have in our grants (see row 3 of listing 6.2.4.7).
SQL > CREATE USER 'testuser'@'test' IDENTIFIED BY 'Ppassword1234%';
Query OK, 0 rows affected (0.0049 sec)
SQL > SET PASSWORD FOR 'testuser'@'test' = 'Ppassword1234%';
ERROR: 1044 (42000): Access denied for user 'myadmin'@'%' to database 'mysql'

Listing 7.28: Attempts to change the password of an unprivileged user

Here we see that trying to modify the password, through this method, of a regular user is also denied. This confirms that the issue is a general privilege access to the mysql.user table, rather than a specific issue linked to the privileged status of "ociadmin".

7.2.3.2 Spoofing the localhost ip

In order to add a portforward rule from 10.0.0.103:ANY to 10.0.0.103:3306, the closest option we found on the OCI console was to create a routing rule in the VCN shown in figure 7.16.

Unfortunately, the attack fails right at the beginning with the error message "The Private IP is attached to a VNIC whose SRC/DST check is enabled".

In order to disable this, we would need to enable "Skip Source/Destination Check" on the VNIC that the Private IP is assigned to. As we do not have control over the compute instance, we are also unable to activate this feature.

7.2.3.3 Conclusion

We have failed to change the password of the managed administrator. We have also failed to spoof a local connection to the MySQL server. The attack did not succeed.

7.2.4 Conclusion

In general, attacks against the MySQL managed database showed that the service was robust to our attack attempts.

Using MySQL instead of PostgreSQL seems to be a design choice that made the service invulnerable to the studied attack paths.

The lack of discoverable information on the service made the penetration testing very difficult to achieve. A notable challenge was the voluntary obfuscation of certain features of the administrator privileges. Wiz team in the
Figure 7.16: View of the console from the administrator account during Route Rule creation.
related research also had to face this problem, as they had to read the source code of PostgreSQL in order to find extra capabilities.

Privilege escalation was found to be impossible in all tested cases as MySQL properly enforces authorizations and default commands were well chosen and restricted by OCI.

Moreover, the network considerations prevent us to bypass the security of the MySQL server.

However, we managed to go against OCI’s database creation policy and create an user administrator with a reserved name. This attack did not seem to pose particular risks for the service.
Chapter 8

Conclusions

The goal of this work is to give a comprehensive security review of Oracle Cloud Infrastructure (OCI) through its provided services. The motivation for this research arises from the limited attention given to OCI in existing literature, despite its growing impact in the environment of cloud services.

To answer our research question "How secure are OCI services against misconfiguration vulnerabilities, past vulnerabilities discovered on other cloud providers, and accessible penetration testing?", two services were selected for study: Identity and Access Management (IAM) and Oracle Managed MySQL Databases. This selection was made after considering research applied to other major cloud providers, prevailing challenges in Cloud Security, and, finally, legal and regulatory constraints. The short time window of 6 months was also a limiting factor for our work.

Both misconfigurations and inherent design flaws were considered for attack paths in this work.

Regarding IAM, privilege escalation was found to be possible through the tagging of compute instances and serverless functions. Common pitfalls such as misspelling in policies or case insensitivity were evaluated and found to be impacting certain situations and difficult to correct. We found that, in general, it was very difficult for administrators to have an accurate overview of the permissions of the existing users, as OCI does not provide an API for this purpose, preventing straightforward policy-making and verification. Existing cloud audit third-party tools fall short of detecting misconfigurations of the explained sort and do not help the administrator get a clearer overview of granted permissions.

The managed MySQL database service was found to be robust to most of our attacks. Authorizations were correctly enforced on the server and network considerations made by OCI prevented us from accessing the database.
locally. However, we have noted that the documentation, notably for banned usernames, was different from the observed reality, and we successfully managed to create a user administrator with a banned name. This issue did not pose a particular further risk. Overall, we have found that existing documentation on the managed MySQL server was voluntarily undisclosed to prevent attacks, and made our study difficult.

In general, for both services, we have found that documentation was incomplete, not reflecting reality, or lacked appropriate warnings. Overall consequences can be mitigated with a good security posture from administrators. However, we deplore that responsibilities fall on the administrator when easy tools could be put in place to prevent oversights.

This work has also permitted the following side contributions as a consequence: existing open-source tools for cloud auditing of OCI were used and evaluated and patched in some cases (see our pull request\(^1\)). Both "ScoutSuite" and "oci-enum" seemed insufficient for professional and efficient use.

Two services were studied in total. We humbly admit that it can seem like a restrictive scope to give an overview of the general security of OCI. However, this work was still found to be of significant importance as we believe it to be the first step into securing OCI services. Studying services is difficult work in itself as diving into a specific service requires one to be very knowledgeable about the service of interest. Furthermore, given the number of services that a cloud provider can offer, it was found to be impossible to cover a wide range of services by ourselves in 6 months. Much work is left to review OCI's security exhaustively, in an academic manner.

Looking towards the future, we suggest several areas for further exploration:

- Improve oci-enum and ScoutSuite to make a recursive search of resources on compartments and their sub-compartments.
- Study the untackled services in this work
- Develop tools for a comprehensive view of permissions and associated users. Given that OCI uses a syntax close to language for policymaking, it might seem like a good idea to explore possibilities with Natural Language Processing AI tools.

In conclusion, this research makes a substantial contribution to the field of cloud security, as it is, to our knowledge, the first of its kind to attempt to

give a comprehensive survey of a cloud service provider’s security landscape, in an academic context. This work is also remarkable in that it studies Oracle Cloud Infrastructure, minor in the shares that it owns in the market, but a promising cloud provider. By delving into the complexities of IAM and Oracle Managed MySQL Databases, we have uncovered valuable insights that shed light on existing vulnerabilities and areas for improvement. The challenges encountered throughout this study highlight the need for continued efforts to enhance both the technological infrastructure and the documentation supporting it. This work not only advances our understanding of cloud security but also provides a foundation upon which future research and practical developments can be built.
Bibliography


[12] Olga Murphy. “Adoption of Infrastructure as Code (IaC) in Real World; lessons and practices from industry”. In: (2022).


Appendix A

IAM

#!/usr/bin/env python3
import datetime
import time
import uuid
import os
import sys
import oci

def prepare_report_entries(storage_namespace,
bucket_name, object_prefix, client):
    report_entries = []
    objects = client.list_objects(storage_namespace,
bucket_name, fields="name,size", prefix=object_prefix
    ).data.objects
    for obj in objects:
        entry = str(obj.name)+' ('+str(obj.size/1024)+'K
    report_entries.append(entry)
    return report_entries

def upload_report(report_entry_list, tmp_directory,
storage_namespace, bucket_name, object_name, client):
    tmp_report_filename = tmp_directory+'/bucket_report
    with open(tmp_report_filename, 'w') as stream:
        for entry in report_entry_list:
            stream.write(entry+'
')
report_timestamp_str = '### Report generated ' +
    str(datetime.datetime.now()) + '
'  
stream.write(report_timestamp_str)
print(report_timestamp_str)
with open(tmp_report_filename, 'r') as stream:
    client.put_object(storage_namespace, bucket_name,
                       object_name, stream)
    os.remove(tmp_report_filename)

if __name__ == '__main__':
    bucket_name = os.environ['APP_BUCKET_NAME']
    summary_object_name = os.environ['APP_OBJECT_NAME']
    object_prefix = os.environ['APP_OBJECT_PREFIX']
    polling_interval_seconds = int(os.environ['APP_POLLING_INTERVAL_SECONDS'])
    tmp_directory = os.environ['HOME']
    while True:
        print('Creating a report')
        signer = oci.auth.signers.InstancePrincipalsSecurityTokenSigner()
        client = oci.object_storage.ObjectStorageClient(config={}, signer=signer)
        storage_namespace = client.get_namespace().data
        report_entry_list = prepare_report_entries(
            storage_namespace, bucket_name, object_prefix, client
        )
        upload_report(report_entry_list, tmp_directory,
                      storage_namespace, bucket_name, summary_object_name,
                      client)
        time.sleep(polling_interval_seconds)


import io
import os
import json
import uuid
import oci

from fdk import response
from io import StringIO
```python
def load_object_content(client, storage_namespace, bucket_name, object_name):
    res = client.get_object(storage_namespace, bucket_name, object_name)
    # Response uses oci._vendor.urllib3.response.HTTPResponse
    httpresponse = res.data.raw
    data_bytes = b''
    for chunk in httpresponse.stream(4096):
        data_bytes += chunk
    return data_bytes.decode('UTF-8')

def prepare_function_response(ctx, res_dict, headers_dict):
    res_str = json.dumps(res_dict)
    rsp = response.Response(ctx, response_data=res_str, headers=headers_dict)
    return rsp

def handler(ctx, data: io.BytesIO=None):
    res_dict = {}
    headers_dict={'Content-Type': 'application/json'}

    try:
        # Process input
        bucket_name = "thesis-reports"
        object_name = "secret.txt"
        # Process only .raw.csv
        signer = oci.auth.signers.get_resource_principals_signer()
        client = oci.object_storage.ObjectStorageClient(config={}, signer=signer)
        storage_namespace = client.get_namespace().data
        processed_object_name = object_name.replace('.txt', '.processed.txt')
        new_content = load_object_content(client, storage_namespace, bucket_name, object_name)
        client.put_object(storage_namespace, bucket_name, processed_object_name, "I have access to the bucket, the file said:"+new_content)
```
res_dict['object_name'] = object_name
res_dict['result'] = "success"
return prepare_function_response(ctx, res_dict, headers_dict)

except (Exception, ValueError) as ex:
    res_dict['result'] = "error"
    res_dict['reason'] = str(ex)

    return prepare_function_response(ctx, res_dict, headers_dict)

---

Listing A.2: Code of the serverless function
Appendix B

SQL

SQL > SELECT * FROM mysql.user \
*************************** 1. row 
***************************
                     Host: %
        User: administrator
    Select_priv: Y
        Insert_priv: Y
        Update_priv: Y
        Delete_priv: Y
    Create_priv: Y
        Drop_priv: Y
        Reload_priv: N
    Shutdown_priv: N
        Process_priv: Y
        File_priv: N
    Grant_priv: Y
    References_priv: Y
        Index_priv: Y
        Alter_priv: Y
    Show_db_priv: Y
        Super_priv: N
Create_tmp_table_priv: Y
        Lock_tables_priv: Y
        Execute_priv: Y
    Repl_slave_priv: Y
    Repl_client_priv: Y
    Create_view_priv: Y
Show_view_priv: Y  
Create_routine_priv: Y  
Alter_routine_priv: Y  
Create_user_priv: Y  
Event_priv: Y  
Trigger_priv: Y  
Create_tablespace_priv: N  
ssl_type:  
ssl_cipher: 0x  
x509_issuer: 0x  
x509_subject: 0x  
max_questions: 0  
max_updates: 0  
max_connections: 0  
max_user_connections: 0  
plugin: caching_sha2_password  
authentication_string:  
password_expired: Y  
password_last_changed: 2023-07-08 16:38:53  
password_lifetime: NULL  
account_locked: Y  
Create_role_priv: Y  
Drop_role_priv: Y  
PasswordReuseHistory: NULL  
PasswordReuseTime: NULL  
PasswordRequireCurrent: NULL  

*************************** 2. row

Host: %  
User: myadmin  
Select_priv: Y  
Insert_priv: Y
Update_priv: Y
Delete_priv: Y
Create_priv: Y
Drop_priv: Y
Reload_priv: N
Shutdown_priv: N
Process_priv: Y
File_priv: N
Grant_priv: Y
References_priv: Y
Index_priv: Y
Alter_priv: Y
Show_db_priv: Y
Super_priv: N
Create_tmp_table_priv: Y
Lock_tables_priv: Y
Execute_priv: Y
Repl_slave_priv: Y
Repl_client_priv: Y
Create_view_priv: Y
Show_view_priv: Y
Create_routine_priv: Y
Alter_routine_priv: Y
Create_user_priv: Y
Event_priv: Y
Trigger_priv: Y
Create_tablespace_priv: N

ssl_type:
ssl_cipher: 0x
x509_issuer: 0x
x509_subject: 0x
max_questions: 0
max_updates: 0
max_connections: 0
max_user_connections: 0

plugin: caching_sha2_password
authentication_string: $A$005$W6/
QNqYeSA.3EQ7FSTX0BiP5FZufZTLoIpfcPowpHeAhB
password_expired: N
password_last_changed: 2023-07-08 16:38:53
password_lifetime: NULL
account_locked: N
Create_role_priv: Y
Drop_role_priv: Y
Password_reuse_history: NULL
Password_reuse_time: NULL
Password_require_current: NULL
User_attributes: {
"Restrictions": [
}
*************************** 3. row
***************************
Host: %
User: ocirpl
Select_priv: N
Insert_priv: N
Update_priv: N
Delete_priv: N
Create_priv: N
Drop_priv: N
Reload_priv: N
Shutdown_priv: N
Process_priv: N
File_priv: N
Grant_priv: N
References_priv: N
Index_priv: N
Alter_priv: N
Show_db_priv: N
Super_priv: N
Create_tmp_table_priv: N
Lock_tables_priv: N
Execute_priv: N
Repl_slave_priv: Y
Repl_client_priv: N
Create_view_priv: N
Show_view_priv: N
Create_routine_priv: N
Alter_routine_priv: N
Create_user_priv: N
Event_priv: N
Trigger_priv: N
Create_tablespace_priv: N

ssl_type:
ssl_cipher: 0x
x509_issuer: 0x
x509_subject: 0x
max_questions: 0
max_updates: 0
max_connections: 0
max_user_connections: 0

plugin: caching_sha2_password

password_expired: N
password_last_changed: 2023-07-08 16:38:53
password_lifetime: NULL
account_locked: N
Create_rolePriv: N
Drop_rolePriv: N
Password_reuse_history: NULL
Password_reuse_time: NULL
Password_require_current: NULL
User_attributes: NULL

*************** 4. row
***************

Host: 127.0.0.1
User: ociadmin

Select_priv: Y
Insert_priv: Y
Update_priv: Y
Delete_priv: Y
Create_priv: Y
Drop_priv: Y
Reload_priv: Y
Shutdown_priv: Y
Process_priv: Y
File_priv: Y
Grant_priv: Y
References_priv: Y
Index_priv: Y
Alter_priv: Y
Show_db_priv: Y
Super_priv: Y
Create_tmp_table_priv: Y
Lock_tables_priv: Y
Execute_priv: Y
Repl_slave_priv: Y
Repl_client_priv: Y
Create_view_priv: Y
Show_view_priv: Y
Create_routine_priv: Y
Alter_routine_priv: Y
Create_user_priv: Y
Event_priv: Y
Trigger_priv: Y
Create_tablespace_priv: Y
  ssl_type: X509
  ssl_cipher: 0x
  x509_issuer: 0x
  x509_subject: 0x
  max_questions: 0
  max_updates: 0
  max_connections: 0
  max_user_connections: 0
    plugin: caching_sha2_password
  authentication_string: $A$005$ dy4^ "/F s@P`8+kobmIVZJCs.HU5NNDmBGPzUBc.SCrLL4WGwjY8EVPu4
  password_expired: N
  password_last_changed: 2023-07-08 16:38:53
  password_lifetime: NULL
  account_locked: N
  Create_role_priv: Y
  Drop_role_priv: Y
Password_reuse_history: NULL
Password_reuse_time: NULL
Password_require_current: NULL
User_attributes: NULL
*************************** 5. row
***************************
Host: 127.0.0.1
User: ocidbm
Select_priv: Y
Insert_priv: N
Update_priv: N
Delete_priv: N
Create_priv: N
Drop_priv: N
Reload_priv: N
Shutdown_priv: N
Process_priv: Y
File_priv: N
Grant_priv: N
References_priv: N
Index_priv: N
Alter_priv: N
Show_db_priv: Y
Super_priv: N
Create_tmp_table_priv: N
Lock_tables_priv: N
Execute_priv: N
Repl_slave_priv: N
Repl_client_priv: Y
Create_view_priv: N
Show_view_priv: N
Create_routine_priv: N
Alter_routine_priv: N
Create_user_priv: N
Event_priv: N
Trigger_priv: N
Create_tablespace_priv: N
  ssl_type:
    ssl_cipher: 0x
    x509_issuer: 0x
    x509_subject: 0x
max_questions: 0
  max_updates: 0
max_connections: 0
max_user_connections: 0
    plugin: caching_sha2_password
authentication_string: $A$005$ eC’ CY*pU g
EtZ ~JXw3UZMESvRlfeOPWtpvCUpxxKg.CRlxBwhTjDmdq7
password_expired: N
password_last_changed: 2023-07-08 16:40:41
password_lifetime: NULL
account_locked: N
Create_role_priv: N
Drop_role_priv: N
Password_reuse_history: NULL
Password_reuse_time: NULL
Password_require_current: NULL
User_attributes: NULL

*************************** 6. row
***************************
Host: localhost
User: mysql.infoschema
Select_priv: Y
Insert_priv: N
Update_priv: N
Delete_priv: N
Create_priv: N
Drop_priv: N
Reload_priv: N
Shutdown_priv: N
Process_priv: N
File_priv: N
Grant_priv: N
References_priv: N
Index_priv: N
Alter_priv: N
Show_db_priv: N
Super_priv: N
Create_tmp_table_priv: N
Lock_tables_priv: N
Execute_priv: N
Repl_slave_priv: N
Repl_client_priv: N
Create_view_priv: N
Show_view_priv: N
Create_routine_priv: N
Alter_routine_priv: N
Create_user_priv: N
Event_priv: N
Trigger_priv: N
Create_tablespace_priv: N
ssl_type:
ssl_cipher: 0x
x509_issuer: 0x
x509_subject: 0x
max_questions: 0
max_updates: 0
max_connections: 0
max_user_connections: 0
plugin: caching_sha2_password
authentication_string:
$A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBEUSED

password_expired: N
password_last_changed: 2023-07-08 16:38:47
password_lifetime: NULL
account_locked: Y
Create_role_priv: N
Drop_role_priv: N
PasswordReuse_history: NULL
PasswordReuse_time: NULL
PasswordRequire_current: NULL
User_attributes: NULL

*************************** 7. row

Host: localhost
User: mysql.session
Select_priv: N
Insert_priv: N
Update_priv: N
Delete_priv: N
Create_priv: N
Drop_priv: N
Reload_priv: N
Shutdown_priv: Y
Process_priv: Y
File_priv: N
Grant_priv: N
References_priv: N
Index_priv: N
Alter_priv: N
Show_db_priv: N
Super_priv: Y
<table>
<thead>
<tr>
<th>Privilege</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create_tmp_table_priv</td>
<td>N</td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>N</td>
</tr>
<tr>
<td>Execute_priv</td>
<td>N</td>
</tr>
<tr>
<td>Repl_slave_priv</td>
<td>N</td>
</tr>
<tr>
<td>Repl_client_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_view_priv</td>
<td>N</td>
</tr>
<tr>
<td>Show_view_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_routine_priv</td>
<td>N</td>
</tr>
<tr>
<td>Alter_routine_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_user_priv</td>
<td>N</td>
</tr>
<tr>
<td>Event_priv</td>
<td>N</td>
</tr>
<tr>
<td>Trigger_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_tablespace_priv</td>
<td>N</td>
</tr>
<tr>
<td>ssl_type</td>
<td></td>
</tr>
<tr>
<td>ssi_cipher</td>
<td>0x</td>
</tr>
<tr>
<td>x509_issuer</td>
<td>0x</td>
</tr>
<tr>
<td>x509_subject</td>
<td>0x</td>
</tr>
<tr>
<td>max_questions</td>
<td>0</td>
</tr>
<tr>
<td>max_updates</td>
<td>0</td>
</tr>
<tr>
<td>max_connections</td>
<td>0</td>
</tr>
<tr>
<td>max_user_connections</td>
<td>0</td>
</tr>
<tr>
<td>plugin</td>
<td>caching_sha2_password</td>
</tr>
<tr>
<td>authentication_string</td>
<td></td>
</tr>
<tr>
<td>$A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED</td>
<td></td>
</tr>
<tr>
<td>password_expired</td>
<td>N</td>
</tr>
<tr>
<td>password_last_changed</td>
<td>2023-07-08 16:38:47</td>
</tr>
<tr>
<td>password_lifetime</td>
<td>NULL</td>
</tr>
<tr>
<td>account_locked</td>
<td>Y</td>
</tr>
<tr>
<td>Create_role_priv</td>
<td>N</td>
</tr>
<tr>
<td>Drop_role_priv</td>
<td>N</td>
</tr>
<tr>
<td>Password_reuse_history</td>
<td>NULL</td>
</tr>
<tr>
<td>Password_reuse_time</td>
<td>NULL</td>
</tr>
<tr>
<td>Password_require_current</td>
<td>NULL</td>
</tr>
<tr>
<td>User_attributes</td>
<td>NULL</td>
</tr>
</tbody>
</table>

*** 8. row

---

Host: localhost
User: mysql.sys
Select_priv: N
Insert_priv: N
Update_priv: N
<table>
<thead>
<tr>
<th>Authorization</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_priv</td>
<td>N</td>
</tr>
<tr>
<td>Drop_priv</td>
<td>N</td>
</tr>
<tr>
<td>Reload_priv</td>
<td>N</td>
</tr>
<tr>
<td>Shutdown_priv</td>
<td>N</td>
</tr>
<tr>
<td>Process_priv</td>
<td>N</td>
</tr>
<tr>
<td>File_priv</td>
<td>N</td>
</tr>
<tr>
<td>Grant_priv</td>
<td>N</td>
</tr>
<tr>
<td>References_priv</td>
<td>N</td>
</tr>
<tr>
<td>Index_priv</td>
<td>N</td>
</tr>
<tr>
<td>Alter_priv</td>
<td>N</td>
</tr>
<tr>
<td>Show_db_priv</td>
<td>N</td>
</tr>
<tr>
<td>Super_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_tmp_table_priv</td>
<td>N</td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>N</td>
</tr>
<tr>
<td>Execute_priv</td>
<td>N</td>
</tr>
<tr>
<td>Repl_slave_priv</td>
<td>N</td>
</tr>
<tr>
<td>Repl_client_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_view_priv</td>
<td>N</td>
</tr>
<tr>
<td>Show_view_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_routine_priv</td>
<td>N</td>
</tr>
<tr>
<td>Alter_routine_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_user_priv</td>
<td>N</td>
</tr>
<tr>
<td>Event_priv</td>
<td>N</td>
</tr>
<tr>
<td>Trigger_priv</td>
<td>N</td>
</tr>
<tr>
<td>Create_tablespace_priv</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSL Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl_type:</td>
<td></td>
</tr>
<tr>
<td>ssl_cipher:</td>
<td>0x</td>
</tr>
<tr>
<td>x509_issuer:</td>
<td>0x</td>
</tr>
<tr>
<td>x509_subject:</td>
<td>0x</td>
</tr>
<tr>
<td>max_questions:</td>
<td>0</td>
</tr>
<tr>
<td>max_updates:</td>
<td>0</td>
</tr>
<tr>
<td>max_connections:</td>
<td>0</td>
</tr>
<tr>
<td>max_user_connections:</td>
<td>0</td>
</tr>
<tr>
<td>plugin:</td>
<td>caching_sha2_password</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication_string:</td>
<td>$A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBEUSED</td>
</tr>
<tr>
<td>password_expired:</td>
<td>N</td>
</tr>
<tr>
<td>password_last_changed:</td>
<td>2023-07-08 16:38:47</td>
</tr>
<tr>
<td>password_lifetime:</td>
<td>NULL</td>
</tr>
<tr>
<td>account_locked:</td>
<td>Y</td>
</tr>
<tr>
<td>Create_role_priv: N</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>Drop_role_priv: N</td>
<td></td>
</tr>
<tr>
<td>Password_reuse_history: NULL</td>
<td></td>
</tr>
<tr>
<td>Password_reuse_time: NULL</td>
<td></td>
</tr>
<tr>
<td>Password_require_current: NULL</td>
<td></td>
</tr>
<tr>
<td>User_attributes: NULL</td>
<td></td>
</tr>
</tbody>
</table>

Listing B.1: Result of the command SELECT * FROM mysql.user