SECURITY, IDENTITY, AND COMPLIANCE IN AWS

Security: Background

In this section, we will discuss the components and fundamental concepts related to infrastructure and software security. The topics to be discussed in this chapter are as follows:

- Importance of security
- Encryption and Decryption
- Credentials and automatic rotation
- DDoS attacks
- Firewalls, and their necessity

If the reader believes themselves to already be familiar with the listed concepts, then they can feel free to skip this section and move on to the rest of the chapter.

Importance of Security

There might be readers who find the concept of securing infrastructure that they do not own to be rather uneasy. After all its their infrastructure, and we are their customer, should not the responsibility of fighting against intrusion and protecting our data be part of the contract?

Well, yes and no. While cloud infrastructure and environments do provide with some basic level of security to the prospective user, this is quite limited and easy to circumvent for any malicious agent worth their salt. Therefore, more often than not, the responsibility of protecting our data and the infrastructure being rented falls upon our own shoulders.

Failure to do so can result in the loss of crucial business data and/or cause the whole system to crash and cease functioning.

Encryption and Decryption

Converting data into an unreadable format, such that even if a malicious agent should get hold of the data, they will not be able to make use of it is called **encryption**, while the process of obtaining the original data from the converted data is called **decryption**. Both processes are performed using a cryptographic token called a **key**, and an **encryption algorithm**, the sequence of operations performed on the data using the key as a variable in order to encrypt and/or decrypt it.

Credentials and Automatic Rotation

Effective credential management is vital to maintaining secure access to infrastructure. Credentials (such as usernames, passwords, API keys, and tokens) are often required to authenticate users or systems, but when mismanaged, they can become points of vulnerability.

Automatic rotation of credentials is a practice that enhances security by periodically updating credentials without manual intervention. Regular rotations help limit the potential exposure window if a credential is compromised. Automations and services that make credential rotation more convenient will be mentioned in later sections.

DDoS Attacks

A Distributed Denial of Service (DDoS) attack aims to overwhelm an infrastructure or service with excessive traffic from multiple sources, causing it to slow down or crash entirely. These attacks are often orchestrated by malicious entities looking to disrupt operations or exploit vulnerabilities for personal gain.

Cloud providers offer DDoS protection services, which can detect unusual traffic patterns, identify DDoS threats, and automatically scale resources to absorb the increased traffic. However, the best DDoS defense strategy combines provider-

based solutions with proper architectural planning, such as deploying services across multiple regions to distribute the load effectively.

Firewalls and Their Necessity

Firewalls act as a critical line of defense by monitoring and controlling incoming and outgoing network traffic based on predefined security rules. They are essential for enforcing boundaries within and outside your cloud infrastructure.

By setting up firewalls, organizations can block unauthorized access while permitting trusted communication, reducing exposure to external threats. Firewalls can operate at various levels, from network firewalls that control access to entire subnets to web application firewalls (WAFs) that specifically filter HTTP/HTTPS traffic for web applications.

AWS Secrets Manager

Accessing AWS services often involves the use of credentials, keys and other information (Usernames, passwords, etc) which need to be kept secure and protected from unauthorized access and distribution. In fact, AWS even has a name for such sensitive pieces of information: **Secrets** and a service called **AWS Secrets Manager** that helps us well, manage them.

To be more specific, AWS Secrets Manager is a service that helps securely store, manage, and retrieve **database credentials**, API keys, and other sensitive secrets. It allows **automatic rotation of credentials**, reducing the risk of exposure and ensuring robust security. By integrating seamlessly with AWS services such as RDS, Redshift, and Lambda, Secrets Manager simplifies managing access to sensitive information, enhancing the security of our AWS environment.

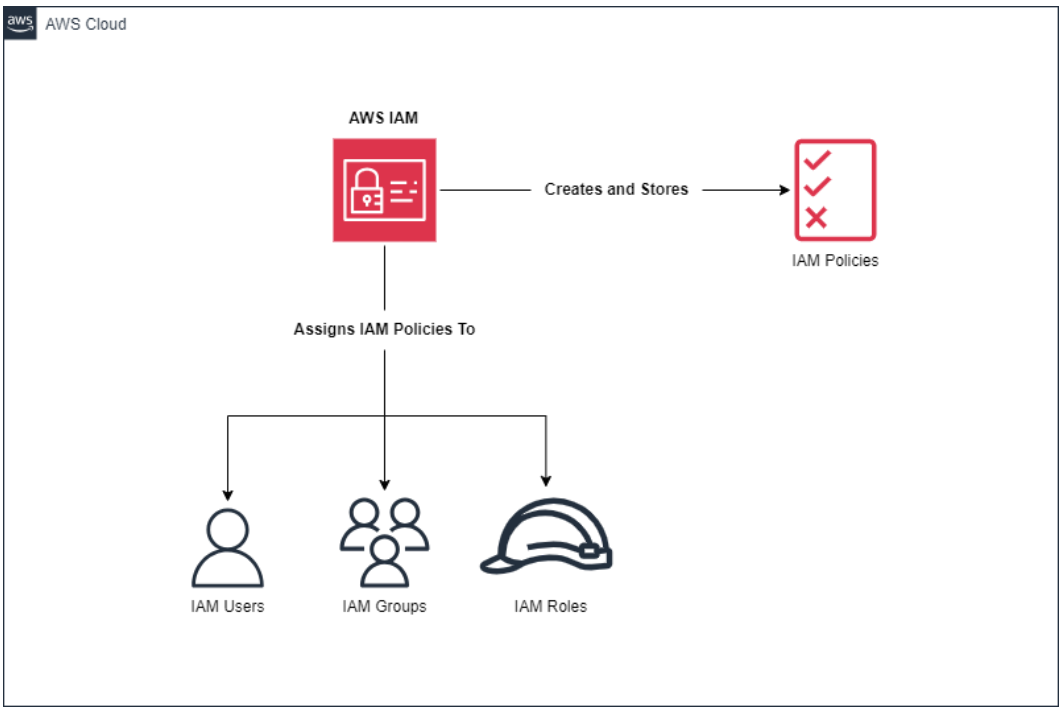
It also offers fine-grained access control using AWS Identity and Access Management (IAM), a cornerstone service for security in AWS (discussed in next section), ensuring that only authorized users or services can retrieve secrets. Additionally, Secrets Manager encrypts secrets using AWS KMS, providing an extra layer of protection.

Note however that AWS Secrets Manager is not the end-all be-all service for storing sensitive information and that certain other AWS services (some of which will be discussed later) may be better suited for specific types of secrets. When storing encryption keys for example, it might be better to use the AWS Key Management Service and when storing string parameters it might be better to use AWS Systems Manager Parameter Store.

IAM and IAM Roles

The principle of least privilege has already been discussed in the background section, but how exactly do we ensure that the people and teams using our AWS environment are made to follow it? How do we micro-manage permissions for individual users, groups and all the different agents that may be utilizing our AWS resources? The answer is **AWS Identity and Access Management**, or IAM for short.

AWS Identity and Access Management (IAM) is a key service that enhances security and access control within the Amazon Web Services ecosystem. It allows cloud administrators to manage users and groups, setting and assigning fine-grained permissions to them. IAM helps us ensure that only authorized users can perform specific actions, maintaining robust security and compliance.



The versatility of IAM, its wide-ranging use cases and multitude of features and sub-features make it a hard service to summarize properly. But most of what IAM does can be encapsulated as being responsible for **A)** Creating and storing IAM Policies and **B)** Assigning them to different identity types (IAM Users, Groups and Roles). Both factors have been illustrated succinctly in the diagram above.

Now, from that above statement we can craft the following questions, answering which will help us get a more complete understanding of IAM:

- What are IAM Policies?
- What is an identity type, and how do IAM Users, Groups and Roles differ from one another?

IAM Policies

An IAM policy is a document which decides the permissions (i.e. what it can and cannot do) of any AWS resource or IAM identity associated with it. Discussing all the different types of policies is beyond the scope of this book, limiting the discussion to identity-based policies as shown in the diagram.

Identity based policies are documents written in JSON that can assign and control the permissions associated with any IAM User, Group or Role it is associated with. An example of an identity-based IAM Policy that enabled any resource/service it is attached to the ability to access an S3 bucket named `iamBucket`:

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

Note however that it is not necessary for a cloud administrator to write JSON documents every time they wish to assign permissions related to a service. For most widely used tasks, AWS provides us with **Managed Policies**, which are JSON documents created and administered by AWS on our behalf. We can simply pick them up and attach them to the desired resource, bypassing the hurdle of having to write one ourselves.

IAM Identities

We have mentioned IAM identities a great deal, but what are they exactly? In simple words, an IAM Identity is any agent or collection of agents that can interact with resources and services in AWS. Said agent could be a human or it could be a computer, with different identity types being used accordingly.

IAM identities are yes, used to define permission boundaries (what an agent can or cannot do), but it is also used for easy identification, authentication and verification of the agent it relates to, helping us ensure that the agent is "legitimate". The three different identity types are very briefly explained below:

- **IAM Users**: An IAM User represents a single person, a single human being within our AWS environment. When we first create an AWS account for example, a single identity that has complete access to all AWS services and resources in the account is created. This identity is called the AWS account ***Root User***.

- **IAM Groups:** An IAM Group represents a collection of IAM Users, allowing us to easily organize groups of IAM Users together.

- **IAM Roles:** An IAM Role is a broader identity type, and can be assumed by any entity that needs it, be it human or computer. It is most widely used to bestow permissions upon specific applications and/or AWS Services. For example, if we wished for an EC2 instance to be able to access DynamoDB, we would assign an IAM Role to it and write a policy that grants access to DynamoDB before attaching it to said IAM Role.

Note for Beginners

People toying around with AWS through the Management Console may be regularly utilizing the Root User Email to login to the console, however, this is not accepted as a good practice due to various security vulnerabilities. Instead, it is considered better to create an IAM User, assign an AWS Managed Policy called `AdministratorAccess` to it, and login through the credentials of said IAM User credentials.

IAM Roles when using AD

Active Directory (AD) is a directory service developed by Microsoft for Windows domain networks. AD enables organizations to enforce security policies, manage access to resources, and maintain a hierarchical structure of the network in a centralized way. They perform the tasks of making sure each person is who they claim to be (Authentication), usually by checking the user ID and password they enter, and allow them to access only the data they're allowed to use (Authorization). The two methods available to us if we wish to use IAM roles alongside AD are as follows: **IAM Roles** and **AWS AD Connector**.

A keen reader might've noticed that both authentication and authorization are tasks usually performed using IAM in an AWS environment. Integrating Active Directory with IAM therefore makes logical sense, and AWS allows us to perform this using IAM Roles, usually done by first granting the roles with necessary permissions and then using a language called SAML (Security Assertion Markup Language) to create a relationship between our Active Directory service and desired IAM Roles.

Another way of using AD and AWS together is through the **AWS AD Connector,** a rather straightforward service that does exactly what it sounds like, it acts as a proxy between the AD service and AWS, connecting them without the need to replicate the AD infrastructure within our AWS environment.

Multi-Factor Authentication

Multi-factor Authentication (MFA) is a widely used method of authentication where users are required to provide two or more types of verification factors to gain access to a system, application or account. This is usually done in order to make it harder for unauthorized individuals to access sensitive resources, even if they have compromised the user's password.

Now, IAM, which is responsible for handling the signing in of accounts to the AWS Console and granting access to AWS services, provides us with out-of-the-box MFA integration as one its key features, allowing us to add an additional layer of security to our AWS account in the form of an One-Time-Password (OTP) from a mobile app, a hardware token, or an SMS.

In fact, not utilizing MFA when using IAM credentials as the primary method of accessing the AWS Management Console or CLI is generally frowned upon, with AWS designating Multi-Factor Authentication as an **IAM Best Practice**.

It is accepted as an IAM best practice for system administrators to utilize MFA (Multi-Factor Authentication) for more secure access to AWS Services.

KMS and Multi-Region Keys

We discussed secrets and AWS Secrets Manager in a previous section, but one of the types of secrets that the mentioned service is not suited for are encryption keys, the cryptographic signatures used to secure stored data.

AWS Key Management Service (KMS) is a vital service for managing cryptographic keys and controlling their use across a wide range of AWS services and applications. In AWS, said keys are usually utilized in order to encrypt/decrypt data either at rest, or during transmission from one service to another. KMS provides us with centralized management of encryption keys, enhancing data security both at rest and in transit. A service that is quite well-integrated into the AWS ecosystem, KMS ensures seamless encryption and decryption across a vast array of AWS services, facilitating strong security measures without adding operational complexity.

KMS supports several types of keys, including:

- **Customer Managed Keys (CMKs)**: Created and managed by the user, offering full control over permissions and lifecycle.
- **AWS Managed Keys**: Automatically created and managed by AWS services such as S3, RDS, and EBS, providing ease of use without manual key management.
- **AWS Owned Keys**: Managed entirely by AWS and used across multiple AWS accounts, offering a simplified, cost-effective solution for less sensitive data.

Also, when utilizing KMS for info stored across AWS regions, we must use the **Multi-Region KMS key**, a variant of the normal KMS key which is designed specifically for dealing with data that spans multiple regions and availability zones.

Certificate Authority (CA)

A **Certificate Authority (CA)** is a trusted entity that issues digital certificates used to verify the identity of individuals, organizations, and servers, either over the internet (i.e., public certificates like SSL/TLS) or within private networks (i.e., private certificates). These certificates are essential for establishing secure communication by encrypting data and confirming the authenticity of the communicating parties. There are two services in AWS primarily used when dealing the type of digital certificates mentioned above, **AWS Private Certificate Authority (AWS Private CA)** and **AWS Certificate Manager (ACM)**. Both services are discussed briefly below:

AWS Private Certificate Authority (AWS Private CA) is a proprietary service offered by AWS that helps manage and issue digital certificates within a private AWS environment. It plays a crucial role in securing communications between AWS resources, ensuring encrypted data transfer, and verifying the identities of the parties involved in internal transactions.

Now, while AWS Private CA primarily focuses on issuing and managing private certificates for AWS resources, they are not the dominant type of digital certificates, and we may face situations where might need to integrate external certificates (e.g., SSL/TLS certificates issued by third-party providers) with our pre-existing AWS resources.

AWS allows us to import these external certificates via **AWS Certificate Manager (ACM)** rather than AWS Private CA. However, these imported certificates have limitations. For example, while AWS-issued certificates can be automatically renewed, external certificates imported into AWS must be manually rotated, and requiring more effort to maintain in general.

There are ways though to make the process of handling external certificates in ACM slightly more convenient, for example AWS Config has a managed rule named `acm-certificate-expiration-check` which can automatically check for expiring certificates, reminding us to rotate the certificate manually whenever the expiration date of the certificate approaches near.

AWS CloudHSM

A **Hardware Security Module (HSM)** is a physical device that provides a secure environment for generating, storing, and managing cryptographic keys. Think of it as a hardware-based counterpart of AWS KMS, though it operates on a more isolated and secure level. HSMs safeguard sensitive information, such as private keys, by isolating them within tamper-resistant hardware, providing robust physical security.

By adding a physical component to the security model, HSMs ensure the integrity of cryptographic operations and help organizations meet stringent security and compliance standards, particularly important for highly regulated industries like finance, government, and healthcare, where data security is paramount.

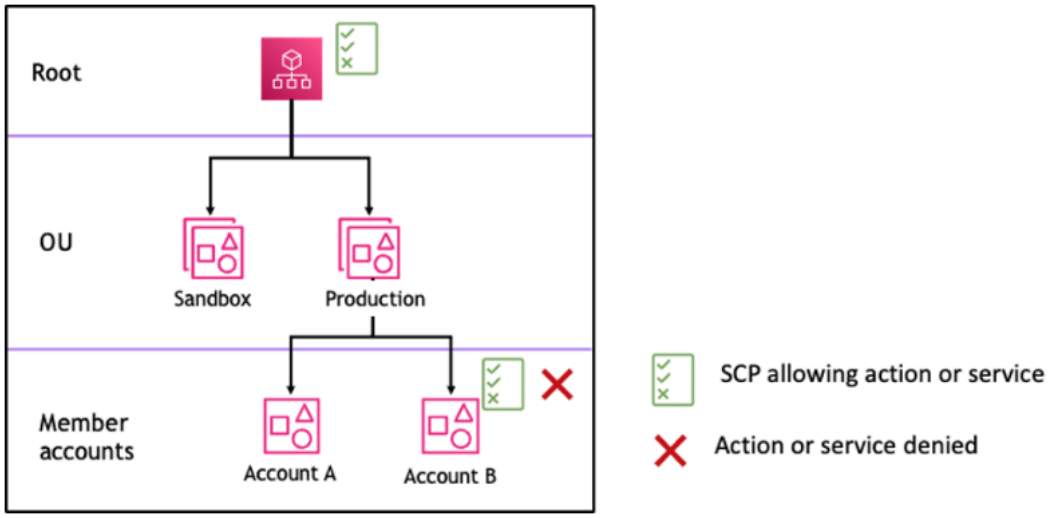
AWS CloudHSM extends the same physical security component to the cloud, allowing organizations to generate and use encryption keys within a dedicated, highly secure hardware device. It gives users complete control over their keys, from creation to management and cryptographic operations, a level of control not provided by KMS, Secrets Manager or other AWS services designed to handle sensitive data.

Now, while AWS CloudHSM can be integrated with other AWS services, it is best understood as an isolated offering, independent from most other AWS services. This is because the primary function of CloudHSM is to secure cryptographic keys and perform cryptographic operations, a function which is jeopardized by unnecessary connections and external dependencies that could introduce vulnerabilities. For example, integrating CloudHSM with a service like AWS CloudTrail (which logs and monitors AWS API calls) might expose unnecessary details about sensitive cryptographic operations to unwanted sources.

Service Control Policies (SCPs)

We have already discussed AWS Organizations and how we can use it to manage a multi-account environment, but there may be situations where enforcing governance and compliance standards across all accounts is critical. This is typically achieved using **Service Control Policies (SCPs)**, which allow administrators to define fine-grained permission boundaries at the organizational level.

SCPs in AWS can be attached to Organizational Units (OUs), individual accounts, or even the root of the organization. They ensure that the member accounts under the defined structure cannot exceed the permissions set by the SCPs, providing centralized control over what actions and services can and cannot be used within each account. It may therefore be thought of as being used to set a kind of upper limit for the entities attached to it, overriding any and all permission policies that go against it. A diagram that illustrates an organization hierarchy where an entity on each level has an SCP associated with it is shown below:



Note again that the purpose of SCPs is to define permission boundaries and not to grant permissions to the constituent identities of an organization. To grant permissions to the accounts within an organization, identity and/or role based policies like the ones discussed in the IAM section should be used instead. SCPs are built not to provide but rather to limit an organization's permission boundary, it is meant for setting guardrails and creating hard red lines for an organization making sure that unauthorized actions do not take place.

AWS Shield

AWS Shield is a managed Distributed Denial of Service (DDoS) protection service that safeguards applications running on AWS, and it can be applied to critical AWS resources such as load balancers, CloudFront distributions, and Route 53. Now AWS Shield comes in two flavors, both of which are pretty self-explanatory: **AWS Shield Standard** and **AWS Shield Advanced**.

AWS Shield Standard, which is automatically included at no extra cost for all AWS customers, provides protection against common, smaller-scale network and transport layer DDoS attacks, ensuring continuous availability of your applications.

AWS Shield Advanced on the other hand, is designed for DDoS attacks of a greater sophistication and larger scale and offers more enhanced features compared to AWS Shield Standard, including more advanced attack detection, near-real-time attack visibility, and tailored mitigations to reduce application downtime. Additionally, opting into the service also provides us with **24/7 access to the AWS DDoS Response Team (DRT)**, a team of DDoS and AWS experts who can engage in real time to assist with threat mitigation and offer detailed post-attack diagnostics.

TLDR;

AWS Shield provides us with protection against Distributed Denial of Service (DDoS) attacks.

AWS WAF

Short for **Web Application Firewall**, AWS WAF helps protect web applications against malicious actors by providing us with a firewall that filters and monitors HTTP requests, allowing us to block **SQL injection**, **cross-site scripting (XSS)**, and other common web exploits. It can also be used to restrict access to our website by IP addresses, geographic locations, or other criteria, making it an essential layer of defense for our web resources.

Note that AWS WAF operates on the application layer of the OSI model, and is a different service from AWS Network Firewall (discussed in a later section) which operates primarily on the network layer (consult the Background on Networking if a refresher on the OSI layer model is needed).

Additionally, AWS WAF also has an easy-to-configure feature called **rate-based limiting**, which detects source IP addresses that make large numbers of HTTP requests within a 5-minute time span and automatically blocks requests from the offending source IP until the rate of requests falls below a set threshold. This can help us mitigate against DDoS style attacks on the application layer, an area not covered by services like AWS Shield, which focus primarily on network and transportation layer protection.

Now, AWS WAF is an account-specific service and is limited to being managed by a single account though some of the service's shortcomings can be circumvented using **AWS Firewall Manager**, a service which allows us to manage firewalls across multiple AWS accounts, allowing us to create and enforce WAF policies across a multi-account AWS environment.

GuardDuty

Amazon GuardDuty is a managed, continuous **threat detection service** designed to help you identify malicious or unauthorized activity in your AWS accounts, workloads, and services. GuardDuty exists to address the growing need for proactive and automated security monitoring across cloud environments, where manual threat detection can be both time-consuming and ineffective against increasingly sophisticated attacks.

The service leverages machine learning, anomaly detection, and integrated threat intelligence from sources like AWS Security Hub, AWS Web Application Firewall (WAF), and global threat databases to detect a wide range of security issues. These include unusual API calls, unauthorized access attempts, suspicious network traffic patterns, and potential data exfiltration.

Basically, GuardDuty is designed to reduce the complexity of threat detection by providing organizations with real-time security findings without the need to manage or deploy additional infrastructure.

Additionally, GuardDuty can also be used to enable fast and/or automated responses to potential threats, with integration support for event-driven workflows and initiating lambda functions for the purposes of automated remediation or prevention of the detected threads.

AWS Network Firewall

AWS Network Firewall is a managed service that allows us to deploy network firewalls to the VPCs within our AWS environment, providing them with scalable network protection. It enables **traffic inspection and filtering** at the network and transport level of the OSI model using customizable rulesets, with each rule relating to one of the following five factors:

- **IP addresses** (for example, block all traffic from a specified IP range)
- **Protocols** (for example, allow only HTTPS traffic through the firewall)
- **Port** (for example, block all incoming traffic from non-standard ports)
- **Domain name** (for example, block access to all requests from a specific domain name)
- **Pattern-matching** (for example, block requests with malicious patterns in its payload)

Note also that the Network Firewall service integrates seamless with AWS Firewall Manager, enabling us to centrally apply the firewall policies and rulesets across both multiple VPCs and multiple accounts. A similar arrangement was also discussed in a previous section on AWS WAF.