**8**

**Security Considerations**

Security is always at the center of architecture design. Many large enterprises suffer financial losses due to security breaches when their customer data gets leaked. Organizations can therefore not only lose customer trust but also the entire business. There are many industry-standard compliances and regulations out there to make sure your application is *secure* and protects customer-sensitive data. In the previous chapter, you learned about various performance-improvement aspects and technology choices for your architecture. In this chapter, you will gain an understanding of best practices to secure your application and make sure it is compliant with industry-standard regulations.

Security isn't just about getting through the outer boundary of your infrastructure. It's also about ensuring that your environments and their components are secured from each other. For example, in the server, you can set up a firewall that allows you to determine which ports on your instances can send and receive traffic, and where that traffic can come from. You can use firewall protection to reduce the probability that a security threat on one instance will not spread to other instances in your environment. Similar precautions are required for other services such as data and applications. Specific ways to implement security best practices are discussed throughout this chapter.

You will learn about the following best security practices in this chapter:

* Designing principles for architectural security
* Selecting technology for architectural security
* Security and compliance certifications
* The cloud's shared security responsibility model

You will learn about various design principles applicable to secure your solution architecture. Security needs to be applied at every layer and in every component of the architecture. You will get an understanding of the right technology to select to ensure your architecture is secure at every layer.

**Designing principles for architectural security**

Security is all about the ability to protect your system and information while delivering business value for your customers. You need to conduct an in-depth security risk assessment and plan a mitigation strategy for the continuous operation of your business. The following sections talk about the standard design principles that help you to strengthen your architectural security.

**Implementing authentication and authorization control**

The purpose of authentication is to determine if a user can access the system with the provided credentials of user ID and password, while authorization determines what a user can do once they are inside the system. You should create a centralized system to manage your users' authentication and authorization.

Centralized user management system helps you to keep track of users' activity so you can deactivate them if they are no longer a part of the system. You can define standard rules to onboard a new user and remove access for inactive users. The centralized system eliminates reliance on long-term credentials and allows you to configure other security methods such as password rotation and strength.

For authorization, you should start with the **principle of least privilege**—it means users should not have any access to begin with, and are assigned only the required access types according to their job role. Creating an access group according to job role helps to manage the authorization policy in one place and apply authorization restrictions across a large number of users. For example, you can restrict the development team to have full access to the dev environment and read-only access to the production environment. If any new developer joins, they should be added to this dev group, where all authorization policies are managed centrally.

Enabling **single sign-on** (**SSO**) with a centralized users repository helps to reduce the hassle of remembering multiple passwords for your user base and eliminates any risk of password leakage.

Large organizations use centralized user management tools such as **Active Directory** (**AD**) for employee authentication and authorization, to provide them access to internal enterprise applications such as the HR system, the expense system, the timesheet application, and so on.

In a customer-facing application, such as e-commerce and social media websites, you can use an OpenID authentication system to maintain a centralized system. You will learn about large-scale user management tools in more detail in the *OAuth and OpenID Connect* section of this chapter.

**Applying security everywhere**

Often, organizations have a main focus of ensuring the physical safety of their data center and protecting the outer networking layer from any attack. Instead of just focusing on a single outer layer, ensure that security is applied at every layer of the application.

Apply the **defense-in-depth** (**DiD**) approach, and put security at various layers of the application; for example, a web application needs to be secured from external internet traffic by protecting the **Enhanced Data rates for Global Evolution** (**EDGE**) network and **Domain Name System** (**DNS**) routing. Apply security at the load balancer and network layers to block any malicious traffic.

Secure every instance of your application by allowing only required incoming and outgoing traffic in the web application and database layer. Protect operating systems with antivirus software to safeguard against any malware attack. Apply both proactive and reactive measures of protection by putting an **intrusion detection system** (**IDS**) and **intrusion prevention system** (**IPS**) in front of your traffic flow and a **web application firewall** (**WAF**) to protect your application from various kinds of attacks. You will learn more details about the various security tools to use in the *Selecting technology for architectural security* section of this chapter.

**Reducing the blast radius**

While applying security measures at every layer, you should always keep your system isolated in a small pocket to reduce the blast radius. If attackers get access to one part of the system, you should be able to limit a security breach to the smallest possible area of the application. For example, in a web application, keep your load balancer in a separate network from other layers of the architecture, as that will be internet-facing. Further, apply network separation at the web, application, and database layers. In any case, if an attack happens in one layer, it will not expand to other layers of the architecture.

The same rules are applied to your authorization system to give the least privilege to users and provide only the minimum required access. Make sure to implement **multi-factor authentication** (**MFA**) so that even if there's a breach in user access, the attacker always needs a second level of authentication to get into the system.

Provide minimal access to ensure that you are not exposing the entire system and provide temporary credentials to make sure access is not open for a long time. Take caution when providing programmatic access by putting a secure token in place, with frequent key rotation.

**Monitoring and auditing everything all the time**

Put the logging mechanism for every activity in your system and conduct a regular audit. Audit capabilities are often required from various industry-compliance regulations. Collect logs from every component, including all transactions and each API call, to put centralized monitoring in place. It is a good practice to add a level of security and access limitation for a centralized logging account so that no one is able to tamper with it.

Take a proactive approach and have the alert capability to take care of any incident before the user gets impacted. Alert capabilities with centralized monitoring help you to take quick action and mitigate any incident. Monitor all user activity and application accounts to limit the security breach.

**Automating everything**

Automation is an essential way to apply quick mitigation for any security-rule violation. You can use automation to revert changes made against desired configurations and alert the security team—for example, if someone added admin users in your system and an open firewall to an unauthorized port or IP address. You can apply automation to remove such undesired changes in the system. Applying automation in security systems has become popular with the concept of DevSecOps. DevSecOps is about adding security to every part of application development and operation. You will learn more about DevSecOps in *Chapter 12*, *DevOps and Solution Architecture Framework*.

Create secure architectures and implement security control that is defined and managed as code. You can version-control your security as a code template, and analyze changes as required. Automated security mechanisms as software code help you scale security operations more rapidly, in a cost-effective way.

**Protecting data**

Data is at the center of your architecture, and it is essential to secure and protect it. Most of the compliance regulations in place are there to protect customer data and identity. Most of the time, any attack has the intention of stealing the user's data. You should categorize your data as per its sensitivity level and protect it accordingly. For example, customer credit card information should be the most sensitive data and needs to be handled with the utmost care. On the other hand, a customer's first name may not be that sensitive while the card number is sensitive information.

Create mechanisms and tools that minimize the need for direct access to data. Avoid manual processing of data by applying tool-based automation that eliminates human error, especially when handling sensitive data. Apply access restrictions to the data wherever possible to reduce the risk of data loss or data modification.

Once you categorize data sensitivity, you can use the appropriate encryption, tokenization, and access control to protect the data. Data needs to be protected not only at rest but also in motion—when transmitting over the network—as well. You will learn about various mechanisms to protect data in the *Data security* section of this chapter.

**Responding to security incidents**

Keep yourself ready for any security events. Create an incident management process as per your organizational policy requirements. Incident management can differ from one organization to another and from one application to another. For example, if your application is handling **Personally Identifiable Information** (**PII**) of your customers, you need a tighter security measure in your incident response. However, if the application is handling small amounts of sensitive data, such as an inventory management application, then it will have a different approach.

Make sure to simulate the incident response to see how your security team is recovering from the situation. Your team should use automation tools for speed of detection, investigation, and response to any security event. You need to set up the alert, monitor, and audit mechanisms to do **Root Cause Analysis** (**RCA**) to prevent such events from occurring again.

In this section, you learned about the general security principles to apply in your architecture for application security. In the next section, you will learn how to apply these principles using different tools and techniques.

**Selecting technology for architectural security**

The previous section was more focused on the general rules of application security to consider while creating an architecture design, but the question is: *How do we apply these rules to make the application secure during implementation?*. There are various tools and technologies available for each layer of your application to make it secure.

In this section, you will learn in detail about the multiple technology choices to apply in the area of user management and protection of the web, infrastructure, and data of your application. Let's start with the first area, user identity and access management.

**User identity and access management**

User identity and access management are vital parts of information security. You need to make sure only authenticated and authorized users are able to access your system resources in a defined manner. User management could be a daunting task as your organization and product adoption grows. User access management should differentiate and manage access to an organization's employees, vendors, and customers.

Enterprise or corporate users could be the organization's employees, contractors, or vendors. Those are specialist users who have a special privilege to develop, test, and deploy the application. In addition to that, they require access to another corporate system to do their daily job—for example, an **Enterprise Resource System** (**ERP**), a payroll system, an HR system, a timesheet application, and so on. As your organization grows, the number of users can grow from hundreds to thousands.

The end users are the customers who use your applications and have minimal access to explore and utilize the desired feature of the application—for example, players of a gaming application, users of social media applications, or customers of an e-commerce website. The count of these users could be from thousands to millions as the popularity of your product or application grows. The other factor is that user count can grow exponentially, which can add challenges. You need to take special care of security when exposing the application to external-facing internet traffic to protect it from various threats.

Let's talk about corporate user management first. You need to have a centralized repository where you can enforce security policies such as strong password creation, password rotation, and MFA for better user management. The use of MFA provides another means of validating someone's identity, if a password may have already been compromised. Popular MFA providers include Google Authenticator, Gemalto, YubiKey, RSA SecurID, Duo, and Microsoft Authenticator.

From a user-access perspective, **role-based authentication** (**RBA**) simplifies user management; you can create user groups as per the user's role and assign an appropriate access policy. As illustrated in the following diagram, you have three groups—admin, developer, and tester—with the corresponding access policy applied to the individual group. For example, an admin can access any system, including production, while developer access is limited to the dev environment, and the tester can only access the test environment:

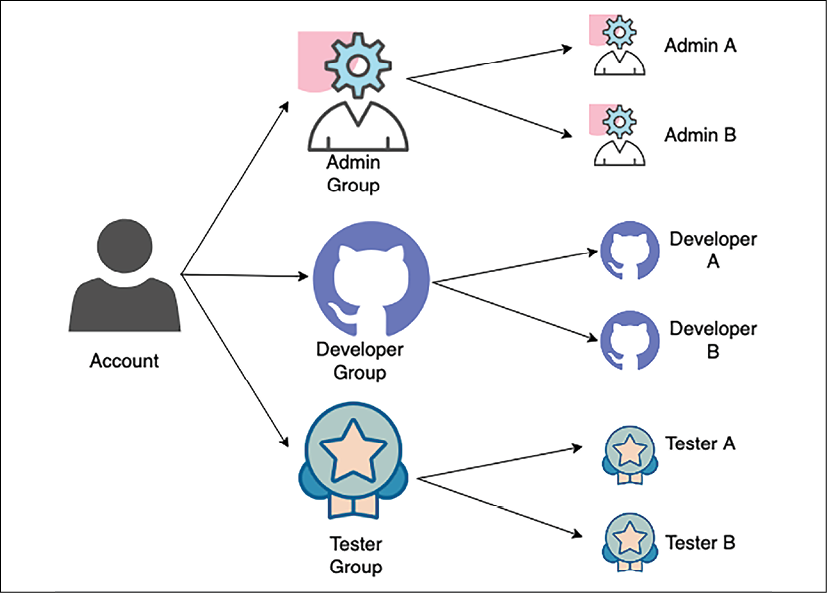


Figure 8.1: User group organization

As shown in the preceding diagram, when any new user joins the team, they get assigned to the appropriate group as per their role. In this way, each user has a defined set of standard access. The user group also helps to update access in case a new development environment gets introduced, and all developers need to have access to that.

**SSO** (**Single Sign-On**) is the standard process to reduce any security lapses and help to automate the system. SSO provides users with a login to the different corporate systems, using a single user ID and password. **Federated Identity Management** (**FIM**) allows users to access the system without a password with a pre-authenticated mechanism. Let's look at some more details.

**Federated identity management and single sign-on**

**FIM** (**Federated identity management**) provides a way to connect the identity management system when user information is stored in the third-party **identity provider** (**IdP**). With FIM, the user only provides authentication information to the IdP, which in turn already has a trusted relationship with the service.

As illustrated in the following diagram, when a user logs in to access a service, the service provider gets credentials from the IdP, rather than getting them directly from the user:

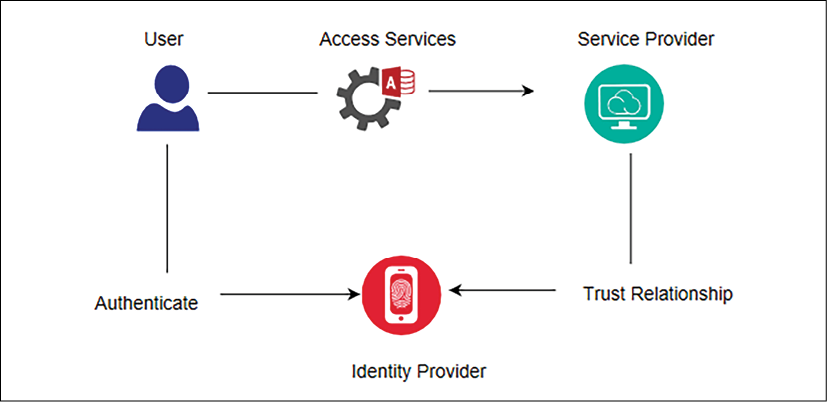


Figure 8.2: FIM authentication flow

SSO allows the use of a single sign-on, with which the user can access multiple services. Here, a service provider could target an environment where you want to log in—for example, a **Customer Relationship Management** (**CRM**) application or your cloud application. An IdP could be a corporate AD. Federation allows authentication similar to an SSO but without a password, as the federation server knows users and allows them to access information.

There are various techniques available to implement FIM and SSO. Let's look at some of the popular **Identify and Access Management** (**IAM**) choices available.

**Kerberos**

Kerberos is an authentication protocol that allows two systems to identify each other in a secure way and helps to implement SSO. It works in the client-server model and uses a ticket system for user identity.

Kerberos has a **Key Distribution Center** (**KDC**), which facilitates authentication between two systems. The KDC consists of two logical parts—the **Authentication** **Server** (**AS**) and the **Ticket-Granting Server** (**TGS**).

Kerberos stores and maintains the secret keys of each client and server in the datastore. It establishes a secure session between two systems during their communication and identifies them with the stored secret key. The following diagram illustrates the architecture of Kerberos authentication:

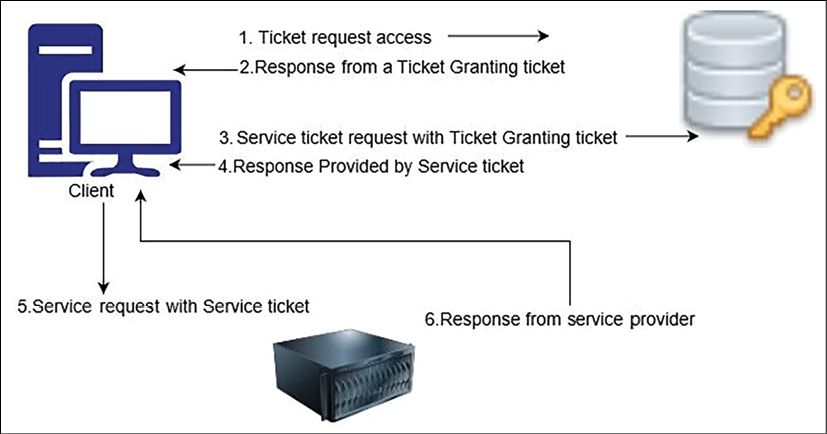


Figure 8.3: Kerberos authentication

As shown in the preceding diagram, when you want to access a service, the following steps are involved:

1. The client sends an access ticket request to the AS as a plaintext request. This request contains the client ID, TGS ID, IP address, and authentication time.
2. The AS checks if your information is available in the KDC database. Once the AS has found your information, it establishes a session between the client request and the TGS. The AS then replies to the client with the **Ticket-Granting Ticket** (**TGT**) and the TGS session key.
3. Now, the TGS session key asks for a password, and, given the correct password, a client can decrypt the TGS session key. However, it cannot decrypt the TGT since the TGS secret key is not available.
4. Now, the client sends the current TGT to the TGS with the authenticator. The TGS contains the session key along with the client ID and **Service Principal Name** (**SPN**) of the resource the client wants to access.
5. Now, the TGS again checks if the requested service address exists in the KDC database. If it does, the TSG will then encrypt the TGT and send a valid session key for the service to the client.
6. The client forwards the session key to the service to prove that the user has access, and the service grants access.

While Kerberos can be very useful, it is an open-source protocol, and generally large enterprises like to use more managed software with robust support, such as AD. Let's look at the working mechanism of one of the most popular user management tools, Microsoft AD, which is based on the **Lightweight Directory Access Protocol** (**LDAP**).

**Microsoft Active Directory**

**AD** (**Active Directory**) is an identity service developed by Microsoft for users and machines. AD has a domain controller, also known as **Active Directory Domain Services** (**AD DS**), which stores the user's and the system's information, their access credentials, and their identity. The following diagram illustrates a simple flow of the necessary authentication process:

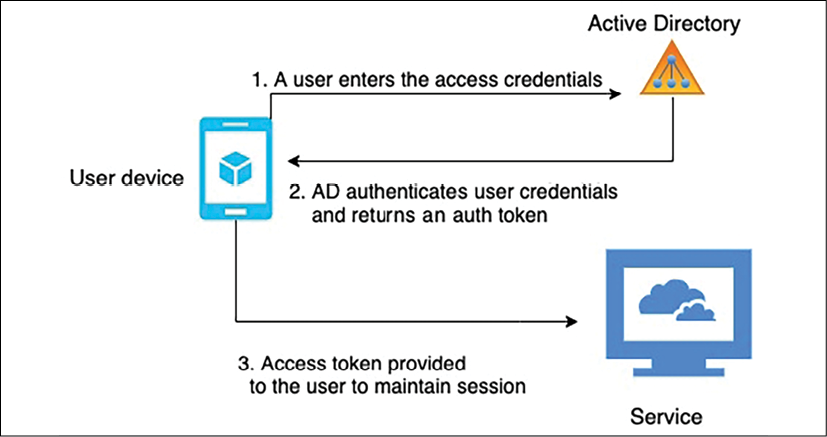


Figure 8.4: AD authentication flow

As shown in the preceding diagram, the user login is managed by AD or any resource on the domain networks. Users first send the request to the domain controller with their credentials and communicate with the **Active Directory Authentication Library** (**ADAL**). The ADAL verifies the user credentials and sends back an access token with a continuous session for the requested service.

**LDAP** (**Lightweight Directory Access Protocol**) is the standard protocol that handles the tree-like hierarchical structure of information stored in directories. **Active Directory Lightweight Directory Services** (**AD LDS**) provides an LDAP interface to the directory of users and systems. For file encryption and network traffic encryption, **Active Directory Certificate Services** (**AD CS**) provides the key infrastructure functionality. **Active Directory Federation Services** (**ADFS**) provides access mechanisms for external resources such as web app logins for a large number of users.

As many organizations have started using cloud services, let's learn about the active directory service provided by AWS cloud.

**Amazon Web Services Directory Service**

**Amazon Web Services** (**AWS**) Directory Service helps to connect AWS resources in your account with an existing on-premises user management tool such as AD. It helps to set up a new user management directory in the AWS cloud. AWS Directory Service facilitates a secure connection to the on-premises directory. After establishing the connection, all users can access cloud resources and on-premises applications with their already existing credentials.

AWS AD Connector is another service that helps you to connect the existing Microsoft AD to the AWS cloud. You don't need any specific directory synchronization tool. After setting up an AD connection, users can utilize their existing credentials to log on to AWS applications. Admin users can manage AWS resources, using AWS IAM.

AD Connector helps to enable MFA by integrating with your existing MFA infrastructure, such as YubiKey, a Gemalto token, an RSA token, and so on. For a smaller user base (fewer than 5,000 users), AWS provides Simple AD, which is a managed directory powered by *Samba 4 Active Directory Compatible Server*. Simple AD has common features such as user accounts management, user group management, SSO based on Kerberos, and user group policies.

**Google Identity federation with Active Directory**

Google cloud uses Google Identity for user authentication and authorization. It allows easy user management by federating user identities from the existing identity management system in AD. To implement federation, you can use Google Cloud Directory Sync to synchronize users and groups from an AD service to the Google Cloud domain directory. You can also use ADFS AD for federated authentication within the existing environment.

In this section, you have learned a high-level overview of AD and managed AD services provided by Microsoft and Amazon. The other directory services provided by major technology companies include Okta, Centrify, Ping Identity, and Oracle **Identity Cloud Service** (**IDCS**).

**Security Assertion Markup Language**

Earlier in this section, under *Federated identity management and single sign-on*, you learned about IdPs and SPs. To access a service, the user gets validated from the IdP, which in turn has a trusted relationship with the SP. **Security Assertion Markup Language** (**SAML**) is one of the mechanisms to establish a trusted relationship between an IdP and an SP. SAML uses **extensible markup language** (**XML**) to standardize communication between an IdP and an SP. SAML enables SSO, so users can use a single credential to access multiple applications.

A SAML assertion is an XML document that the IdP sends to the service provider with user authorization. The following diagram illustrates the flow of the SAML assertion:

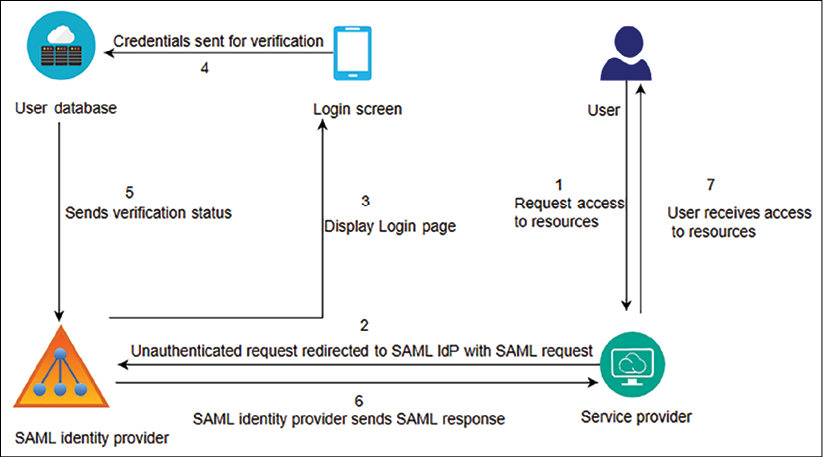


Figure 8.5: User authentication using SAML

As mentioned in the preceding diagram, the following steps are taken to implement user authentication using SAML:

1. A user sends a request to access the service—for example, the Salesforce CRM application—as a service provider.
2. The service provider (a CRM application) sends a SAML request with the user information to the SAML IdP.
3. The SAML IdP pops up the SSO login page, where users enter authentication information.
4. The user access credential goes to the identity store for validation. In this case, the user identity store is an *AD*.
5. The user identity store sends the user validation status to the SAML IdP, with whom the identity store has a trusted relationship.
6. The SAML IdP sends a SAML assertion to the service provider (a CRM application) with information pertaining to user verification.
7. After receiving the SAML response, the service provider allows application access to the user.

Sometimes, service providers can act as an IdP as well. SAML is very popular for establishing a relation between any identity store and service provider. All modern identity store applications are SAML 2.0-compatible, which allows them to communicate with each other seamlessly. SAML allows user identity to be federated and enables SSO for enterprise users.

However, for large user bases such as social media and e-commerce websites, **OAuth** (short for **Open Authorization**) and **OpenID** are more suitable. Let's learn more about OAuth and **OpenID Connect** (**OIDC**).

**OAuth and OpenID Connect**

OAuth is an open standard authorization protocol that provides secure access to an application. OAuth provides secure access delegation. OAuth doesn't share password data but uses the authorization token to establish the identity between service providers and consumers. Users of an application provide access to their information without giving login credentials. While OAuth is mainly for authorization, many organizations have started adding their own mechanisms for authentication. **OIDC** defines the authentication standard on top of **OAuth authorization**.

Large technology companies such as Amazon, Facebook, Google, and Twitter allow users to share information in their accounts with third-party applications. For example, you can log in to a new photo app using your Facebook login and authorize the new app to access only your Facebook photo information.

The following diagram illustrates an OAuth access delegation flow:

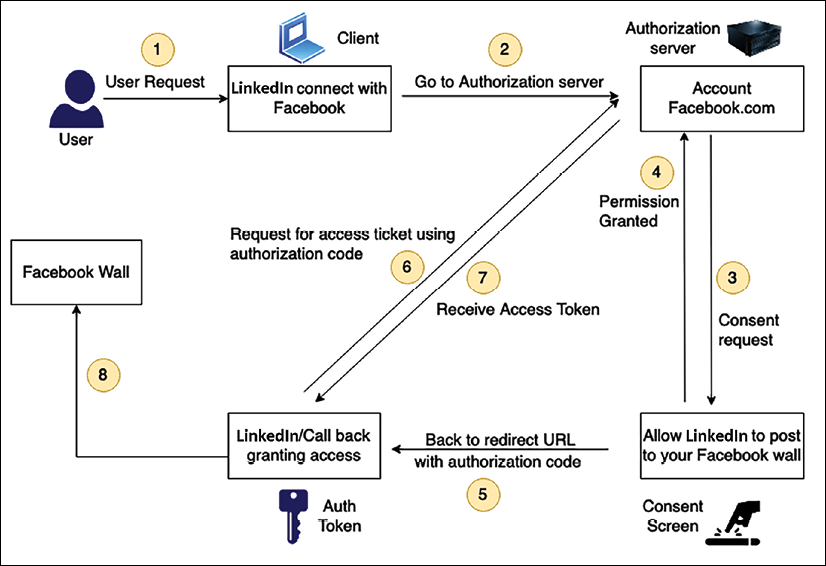


Figure 8.6: User access delegation with OAuth 2.0

As shown in the preceding diagram, the authentication flow follows these steps:

1. In this scenario, you want the LinkedIn app to get your profile photo from Facebook.
2. The LinkedIn app requests authorization to access Facebook profile photos.
3. The authorization server (which is your Facebook account in this case) creates and displays a consent screen to you.
4. You provide your consent to the request for the LinkedIn app to access only your Facebook profile photos.
5. After getting your approval, the authorization Facebook server sends an authorization code back to the requesting LinkedIn app.
6. The LinkedIn app then requests an access token from the authorization server (Facebook account) using the authorization code.
7. The authorization server identifies the LinkedIn app and checks the validity of the authentication code. If the access token is validated, the server issues an access token to the LinkedIn app.
8. The LinkedIn app can now access resources such as Facebook profile photos using the access token.

OAuth 2.0, which is faster than OAuth 1.0 and more comfortable to implement, is now most commonly used. **JSON Web Token** (**JWT**) is a simple and accessible token format that can be used with OAuth and is popular with OpenID.

**JSON Web Token**

A JWT has a JSON structure that has information about expiration time, issuer, subject, and so on. It is more robust than **Simple Web Token** (**SWT**) and simpler than SAML 2.0. You can see a JWT in the following screenshot:

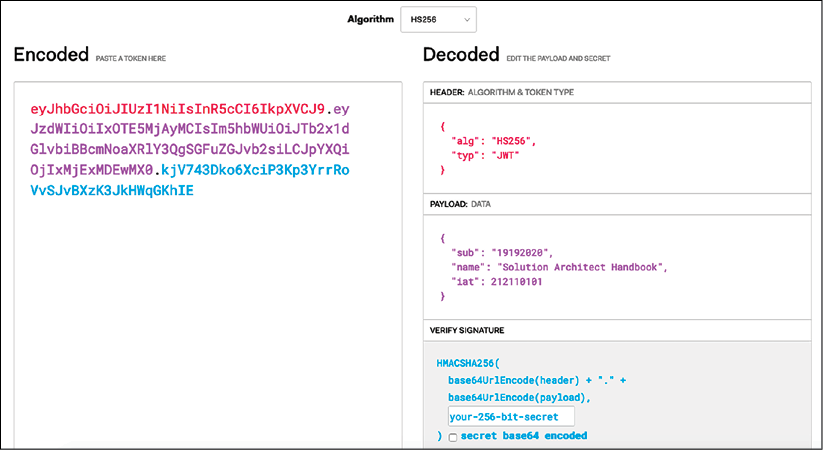


Figure 8.7: Sample JWT

As shown in the preceding screenshot, JWTs consist of three parts separated by dots, as you can see in the **Encoded**section:

* Header: The header consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HS256 or RSA.
* Payload: The payload contains the claims. Claims are statements about the user and any additional data.
* Signature: The signature verifies the message wasn't changed along the way. It can also verify the sender of the JWT.

JSON has a simpler structure than XML, and is also smaller in size, making JWT more compact than SAML. JWT is a good choice to pass information into HTML and HTTP environments.

In this section, you learned about the most common user management tools and services. However, there are various other protocols and services available for user authentication and authorization. Implementation of the protocols mentioned previously can be complicated, and there is a large amount of packaged software available that makes the job easier.

Amazon Cognito is a user access management service provided by AWS that includes standard-based authorization such as SAML 2.0, OIDC, and OAuth 2.0, along with an enterprise user directory that provides the ability to connect with AD. Okta and Ping Identity provide enterprise user management and the ability to communicate with various service provider tools in one place.

Once your application is exposed to the internet, there are always various kinds of attacks that are bound to happen. Let's learn about some of the most common attacks, and how to set up the first layer of defense for web-layer protection.

**Handling web security**

As user demand is changing to require 24/7 availability of services, businesses are evolving to go into online mode and adopting web application models. Web applications also help a company to gain a global customer base. Businesses such as online banking and e-commerce websites are always available, and they deal with customers' sensitive data such as payment information and payer identity.

Now, web applications are central to any business, and these applications are exposed to the world. Web applications can have vulnerabilities, which makes them exposed to cyber-attacks and data breaches. Let's explore some common web vulnerabilities and how to mitigate them.

**Web app security vulnerabilities**

A web application is prone to security breaches as hackers orchestrate cyber-attacks from different locations and by various methods. A web application is more vulnerable to theft than a physical store location. Just as you lock and protect your physical shop, in the same way, your web app needs to protect itself from unwanted activity. Let's explore some standard methods of attack that can cause security vulnerabilities in your web application.

**Denial of service and distributed denial of service attacks**

A **Denial of Service** (**DoS**) attack attempts to make your website unreachable to your users. To achieve a successful DoS attack, the attacker uses a variety of technologies that consume network and system resources, thus interrupting access for legitimate users. The attacker uses multiple hosts to orchestrate the attack against a single target.

A **Distributed Denial of Service** (**DDoS**) attack is a type of DoS attack where multiple compromised systems (typically infected with *Trojans*) are used to target a single system. Victims of a DDoS attack find that all their systems are maliciously used and controlled by the hacker in the distributed attack. As illustrated in the following diagram, a DDoS attack happens when multiple systems exhaust the bandwidth of resources of a targeted system:

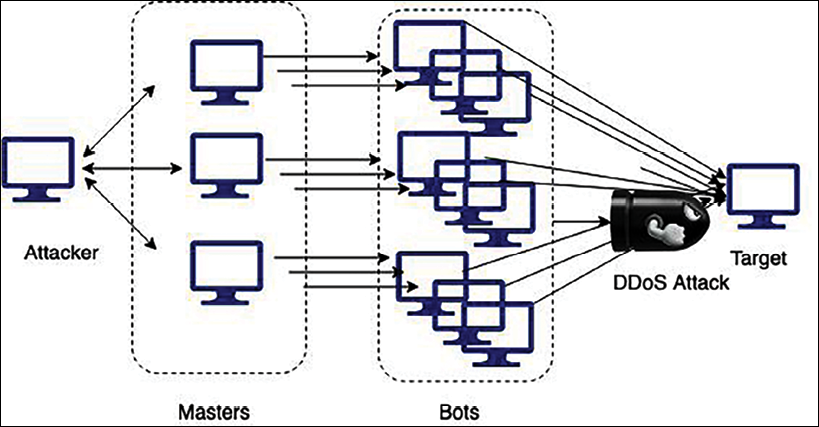


Figure 8.8: DDoS attack

The general concept of a DDoS attack is to leverage additional hosts to amplify the requests made to the target, rendering them overprovisioned and unavailable. A DDoS attack is often the result of multiple compromised systems, whereby a botnet puts a flood of traffic in the targeted system.

The most common DDoS attack happens at the application layer, using either a DNS flood or a **Secure Sockets Layer** (**SSL**) negotiation attack. In DNS floods, attackers exhaust the resources of a DNS server with too many requests. During SSL negotiations, attackers send a large amount of unintelligible data for computationally expensive SSL decryption. The attacker can perform other SSL-based attacks on the server fleet and overburden it with unnecessary task processing.

At the infrastructure layer, a typical DDoS attack happens in the form of the following:

* **User Datagram Protocol**(**UDP**)**reflection**: With UDP reflection, attackers spoof the target server's IP address and make a request that returns amplified significant responses from a hacked reflector server.
* **SYN floods**: With SYN floods, attackers exhaust the target server's **Transmission Control Protocol** (**TCP**) service by creating and abandoning high numbers of connections, blocking legitimate users from accessing the server.

Often, attackers try to get sensitive customer data, and for that purpose, they use a different kind of attack called **SQL injection**(**SQLi**) attacks. Let's learn more about them.

**SQL injection attacks**

As the name suggests, in an SQLi attack, attackers inject malicious **Structure Query Language** (**SQL**) to get control of an SQL database and fetch sensitive user data. The attacker uses SQLi to gain access to unauthorized information, take control of an application, add new users, and so on.

Take an example of a loan-processing web application. You have loanId as a field that customers can use to get all information related to their loan finance. The typical query will look like this: SELECT \* FROM loans WHERE loanId = 117. If proper care is not taken, attackers can execute a query such as SELECT \* FROM loans WHERE loanId = 117 or '1=1' and get access to the entire customer database, as this query will always return the true result.

The other common method to hack user data through script injection is **cross-site scripting** (**XSS**) where a hacker impersonates a legitimate user. Let's learn more about it.

**Cross-site scripting attacks**

You must have encountered phishing emails that have links impersonating a website known to you. Clicking on these links may lead to compromised data through XSS. With XSS, the attacker attaches their code to a legitimate website and executes it when the victim loads the web page. The malicious code can be inserted in several ways, such as in a URL string or by putting a small amount of JavaScript code on the web page.

In an XSS attack, the attacker adds a small code snippet at the end of the URL or client-side code. When you load the web page, this client-side JavaScript code gets executed and steals your browser cookies.

These cookies often contain sensitive information, such as the access token and authentication to your banking or e-commerce websites. Using these stolen cookies, the hacker can get into your bank account and take your hard-earned money.

**Cross-site request forgery attacks**

A **Cross-Site Request Forgery** (**CSRF**) attack takes advantage of user identity by creating confusion. It typically tricks the user with a transaction activity in which the state gets changed—for example, changing the password of a shopping website or requesting a money transfer to your bank.

It is slightly different than an XSS attack as, with CSRF, the attacker tries to forge the request rather than insert a code script. For example, the attacker can forge a request to transfer a certain amount of money from the user's bank and send that link in an email to the user. As soon as users click on that link, the bank gets a request and transfers the money to the attacker's account. CSRF has minimal impact on the individual user account, but it can be very harmful if attackers are able to get into the admin account.

**Buffer overflow and memory corruption attacks**

A software program writes data in a temporary memory area for fast processing, which is called a **buffer**. With a buffer overflow attack, an attacker can overwrite a portion of the memory connected with the buffer. An attacker can deliberately cause a buffer overflow and access connected memory, where an application executable may be stored. The attacker can replace the executable with the actual program and take control of the entire system. Buffer overflow attacks can cause memory corruption with unintentional memory modification, which the hacker can use to inject code.

Looking at the overall application, there are more security threats that exist at the infrastructure layer, network layer, and data layer. Let's explore some standard methods to mitigate and prevent security risks at the web layer.

**Web security mitigation**

Security needs to be applied to every layer, and special attention is required for the web layer due to its exposure to the world. For web protection, important steps include keeping up with the latest security patches, following the best software development practices, and making sure proper authentication and authorization are carried out. There are several methods to protect and secure web applications; let's explore the most common methods.

**Web application firewalls**

WAFs are necessary firewalls that apply specific rules to HTTP and HTTPS traffic (that is, port 80 and 443). WAFs are software firewalls that inspect your web traffic and verify that it conforms to the norms of expected behavior. WAFs provide an additional layer of protection from web attacks.

WAF rate limiting is the ability to look at the amount or type of requests sent to your service and define a threshold that caps how many requests are allowed per user, session, or IP address. Approved and un-approved lists allow you to allow or block users explicitly. AWS WAF helps you to secure your web layer by creating and applying rules to filter web traffic. These rules are based on conditions that include HTTP headers, user geolocation, malicious IP addresses, or custom **Uniform Resource Identifiers** (**URIs**), and so on. AWS WAF rules block common web exploits such as XSS and SQLi.

AWS WAF provides a centralized mechanism in the form of rules that can be deployed across multiple websites. This means that you can create a single set of rules for an environment that has various websites and web applications running. You can reuse rules across applications instead of recreating them.

Overall, WAF is a tool that applies a set of rules to HTTP traffic. It helps to filter web requests based on data such as IP addresses, HTTP headers, HTTP bodies, or URI strings. It can be useful for mitigating DDoS attacks by offloading illegitimate traffic. Let's learn more about DDoS mitigation.

**DDoS mitigation**

Resilient architecture can help to prevent or mitigate DDoS attacks. A fundamental principle in keeping your infrastructure secure is reducing the potential number of targets that an attacker can hit. In short, if an instance doesn't need to be public, then don't make it public. An application-layer attack can spike monitoring metrics such as network utilization for your **content distribution network** (**CDN**), load balancer, and server metrics due to HTTP flood. You can apply various strategies to minimize the attack surface area:

* Wherever possible, try to reduce the number of necessary internet entry points. For example, open incoming internet access to your load balancer, not web servers.
* Hide any required internet entry points from untrusted end users so that they cannot access them.
* Identify and remove any non-critical internet entry points—for example, expose file-share storage for vendors to upload data with limited access, rather than exposing it to worldwide internet traffic.
* Isolate the access point and apply a specific restrictions policy for end user traffic compared to application management traffic.
* Create a decoupled internet entry point to minimize the attack surface.

Your primary goal is to mitigate DDoS attacks at the edge location of the CDN. It's more challenging and costly to handle DDoS attacks if they get through to your application servers. The following diagram illustrates a DDoS mitigation example for an AWS cloud workload:

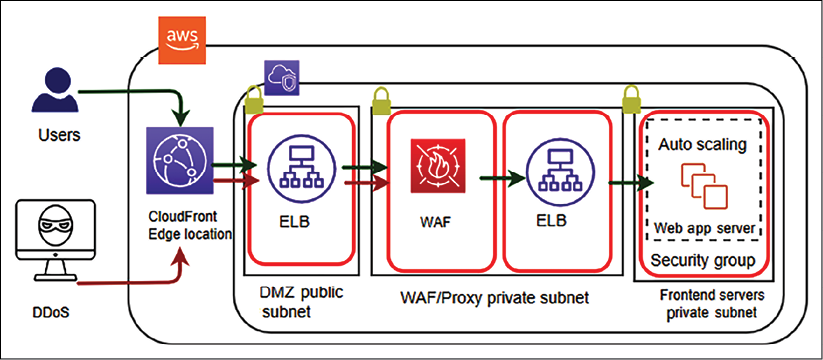


Figure 8.9: DDoS WAF sandwich mitigation strategy

The preceding diagram illustrates a **WAF sandwich architecture**, where the WAF appliance is staged between two load balancers to handle a DDoS attack. Frequent DDoS attacks come from attacking strategies such as SYN floods and UDP reflection, which Amazon CloudFront prevents by only accepting well-formed connections before the attacking strategy can reach your application servers. CDNs such as Amazon CloudFront help to tackle DDoS attacks by isolating them at a geographically isolated location and preventing the traffic from affecting other locations. Network firewall security helps you to control incoming and outgoing traffic at an individual server level.

As mentioned in the previous section, WAFs are used to protect web applications against exploit attacks such as XSS and SQLi attacks. In addition to this, WAFs also help to detect and prevent DDoS attacks at the web application layer.

To handle a DDoS attack, you can apply either horizontal or vertical scaling. You can take advantage of scaling in the following way:

1. First, select the right server size and configuration for your web application.
2. Second, apply a load balancer to distribute traffic among the fleet of servers and add auto-scaling to add/remove servers as required.
3. Finally, use the CDN and DNS server, as they are built to handle traffic at scale.

Scaling for DDoS attacks is an excellent example of why it's essential to set reasonable maximum counts for your servers. A DDoS attack could scale your servers out to a count that would be extremely costly, while still potentially not being able to avoid becoming unavailable. Having reasonable maximum limits for expectations of regular traffic spikes would prevent a DDoS attack from costing your company too much money.

In this section, you learned about various security risks and vulnerabilities at the web layer and some standard methods to protect them. As security needs to be applied to every layer, let's explore more about the protection of the infrastructure layer.

**Securing an application and its infrastructure**

In the previous section, you learned about securing the web layer. As security needs to be applied at every layer of your workload, let's learn about securing the application and network layers.

**Application and operating system hardening**

It is not possible to entirely eliminate vulnerabilities in your application, but you can limit system attacks by hardening your application's operating system, filesystem, and directory. Once attackers can get into your application, they can get root access and orchestrate an attack on the entire infrastructure. It is essential to limit attacks to the application level by restricting the directory by *hardening permission*. At the process level, restrict memory and CPU utilization to prevent a DoS attack.

Set the right permission at the file, folder, and file partition levels, which is the only requirement for the application to execute. Avoid giving root privilege to the application or its users. You should create a separate directory with only required access for each application so only the required user has application access. Don't use common access for all applications.

Automate application restart by using tools and avoid a manual approach, whereby users need to log in to the server to start.

You can use process control tools such as **DAEMON Tools** and **Supervisord** to automate an application restart. For a Linux operating system, a utility such as **systemd** or **System V init** scripts help to start/stop the application.

**Software vulnerabilities and secure code**

It is always recommended to apply the latest security patch to your operating system provided by your operating system vendor. This helps to fill any security holes in the system and protect your system from vulnerabilities where attackers are able to steal your security certificate or run arbitrary code. Make sure to integrate secure coding best practices into your software development process, as recommended by the **Open Web Application Security Project** (**OWASP)**, details about which can be found here: <https://owasp.org/www-project-top-ten/>.

Keeping your system up to date with the latest security patch is very important. It is better to automate the process of the most recent patch installation as soon as it becomes available. However, sometimes, running a security patch may break your working software, so it's better to set up a **Continuous Integration** **and** **Continuous Deployment** (**CI/CD**) pipeline with automated test and deployment. You will learn more about the CI/CD process in *Chapter 12*, *DevOps and Solution Architecture Framework*.

The AWS cloud provides a system manager tool that allows you to apply security patches and monitoring of your server fleet in the cloud. You can use a tool such as **auto-updates** or **unattended-upgrades** to automate security patch installation.

**Network, firewall, and trusted boundary**

When it comes to protecting your infrastructure, securing the network comes into consideration first. The physical security of your IT infrastructure in the data center is to be taken care of by providers. In the case of cloud-like AWS providers, they take the utmost care of the physical security of your infrastructure. Let's talk about ensuring network security, which is your responsibility as an application owner.

To understand it better, let's take an example from a public cloud provider such as AWS. You can apply the same example to your on-premises or private cloud network infrastructure as well.

As illustrated in the following diagram, you should apply security at every layer and define trusted boundaries around each layer, with minimal access:

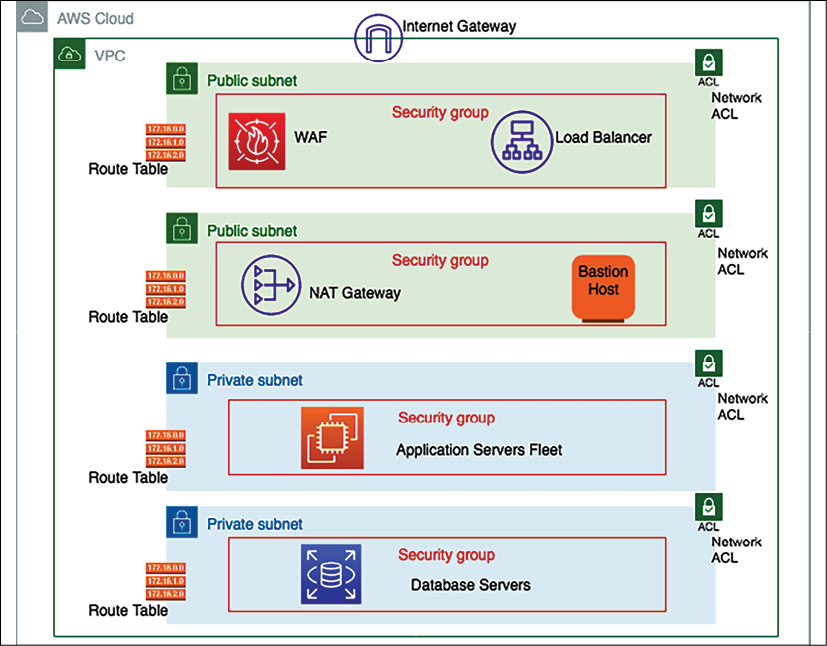


Figure 8.10: Network configuration for infrastructure security

In the preceding diagram, the load balancer is in a public subnet, which can accept internet traffic and distribute it to the application server fleet. WAF filter traffic is based on set rules and protects your application from various attacks, as you learned in the previous section. The application server fleet and database servers are in a private subnet, which means direct internet access is not reachable to expose them to the public internet. Let's dive deep into the preceding architecture diagram and walk through each layer, as follows:

* Amazon**Virtual Private Cloud** (**VPC**) provides you with logical network isolation of your infrastructure. Amazon VPC is your network environment in the cloud. It's where you will launch many of your resources. It's designed to provide greater control over the isolation of your environments and their resources from each other. You can have multiple VPCs in each account or region.
* When you create a VPC, you specify its set of IP addresses with **Classless Inter-Domain Routing** (**CIDR**) notation. CIDR notation is a simplified way of showing a specific range of IP addresses. For example, 10.0.0.0/16 covers all IPs from 10.0.0.0 to 10.0.255.255, providing 65,535 IP addresses to use.
* Subnets are segments or partitions of a network divided by the CIDR range. They create trusted boundaries between private and public resources. Rather than defining your subnets based on the application or functional tier (web/app/data), you should organize your subnets based on internet accessibility. A subnet allows you to define clear, subnet-level isolation between public and private resources.
* In this environment, all of your resources that require direct access to the internet (public-facing load balancers, **Network Address Translation** (**NAT**) instances, bastion hosts, and so on) would go into the public subnet, while all other instances (such as database and application resources) would go into your private subnet. Use subnets to create layers of separation between *tiers* of resources, such as putting your application instances and your data resources into separate private subnets.
* The majority of resources on AWS can be hosted in private subnets, using public subnets for controlled access to and from the internet as necessary. Because of this, you should plan your subnets so that your private subnets have substantially more IPs available compared to your public subnets.
* While subnets can provide a fundamental element of segregation between resources using **Network** **Access Control List** (**NACL**) rules, security groups can give an even more fine-grained level of traffic control between your resources, without the risk of overcomplicating your infrastructure and wasting or running out of IPs.
* A routing table contains a set of rules, called **routes**. Routes determine which application servers are to receive network traffic. For better security, use a custom route table for each subnet.
* Security groups are the virtual firewalls that control inbound and outbound traffic for one or more instances from the CIDR block range, or another security group, as designated resources. As per the principle of least privilege, deny all incoming traffic by default and create rules that can filter traffic based on TCP, UDP, and **Internet Control Message Protocol** (**ICMP**) protocols.
* An NACL is an optional virtual firewall that controls inbound and outbound traffic at the subnet level. An NACL is a stateless firewall that is compared to the security group, which is stateful. This means that if your incoming request is allowed, then the outbound request does not have to be inspected or tracked. While stateless, you have to define both inbound and outbound traffic rules explicitly.
* Internet traffic is routed through an **internet gateway** (**IGW**) to make a subnet public. By default, internet accessibility is denied for internet traffic in your environment. An IGW needs to be attached to your VPC, and the subnet's route table should define the rules to the IGW.
* A private subnet blocks all incoming and outgoing internet traffic, but servers may need outgoing internet traffic for software and security patch installation. A NATgateway enables instances in a private subnet to initiate outbound traffic to the internet and protects resources from incoming internet traffic.
* A bastion host acts like a jump server, which allows access to other resources in the private subnet. A bastion host needs to be hardened with tighter security so that only appropriate people can access it. To log in to the server, always use *public-key cryptography* for authentication rather than a regular user ID and password method.

Many organizations typically collect, store, monitor, and analyze network flow logs for various purposes, including troubleshooting connectivity and security issues and testing network access rules. You need to monitor traffic flow to your system VPC, which includes recording incoming and outgoing traffic information from your network. **VPC Flow Logs** enables you to capture that information, along with accepted and rejected traffic information for the designated resource to understand traffic patterns better.

Flow logs can also be used as a security tool for monitoring traffic that is reaching your instance. You can create alarms to notify you if certain types of traffic are detected. You can also create metrics to help you to identify trends and patterns. You can create a flow log for a VPC, a subnet, or a network interface. If you create a flow log for a subnet or VPC, each network interface in the VPC or subnet is monitored.

As you can see, there are multiple layers for security available at the network layer that can help to protect your infrastructure. Keeping resources in their isolated subnet helps to reduce the blast radius. If an attacker can penetrate one component, you should be able to restrict them to limited resources. You can use an IDS and an IPS in front of your infrastructure to detect and prevent any malicious traffic. Let's learn more about them.

**Intrusion detection system and intrusion prevention system**

An IDS detects any cyber-attack happening through network traffic by recognizing an attack pattern. An IPS goes a step further and helps to stop malicious traffic proactively.

The IPS sits behind the firewall and provides a layer of analysis for dangerous content such as malicious packets dropping, blocking traffic from the source address, and connection resetting.

The IPS has two major detection methods, signature-based detection and statistical anomaly-based detection, for finding exploits.**Signature-based detection** is based on a dictionary of uniquely identifiable patterns of each exploit. Each exploit signature is stored in a continuously growing dictionary of signatures to determine a pattern. **Statistical anomaly detection** defines a baseline performance parameter. It takes samples of network traffic randomly and compares them to the baseline performance level. If network traffic activity is outside the parameters, the IPS takes action.

You need to determine the applicability of the IDS/IPS system as per your application's requirements. An IDS can be host-based or network-based.

**Host-based IDS**

In an IDS, a host- or agent-based IDS is running on each host of your environment. It can review the activity within that host to determine if an attack has occurred and has been successful. It can do this by inspecting logs, monitoring the filesystem, monitoring network connections to the host, and so on. The software or agent then communicates with a central/command application about the health or security of the host it is monitoring.

The pros of host-based solutions include that they can deeply inspect the activity inside each host. They can horizontally scale as far as required (each host gets its own agent), and do not need to impact the performance of running applications. The cons include the additional configuration management overheads that can be introduced if managing agents on many servers, which are burdensome for an organization.

As each agent is operating in isolation, widespread/coordinated attacks can be harder to detect. To handle coordinated attacks, the system should respond immediately across all hosts, which requires the host-based solution to play well with the other components, such as the operating system and the application interface, deployed on the host.

**Network-based IDS**

A network-based IDS inserts an appliance into the network, through which all traffic is routed and inspected for attacks.

The pros include a simple/single component that needs to be deployed and managed away from the application hosts. Also, it is hardened or monitored in a way that might be burdensome across all hosts. An individual/shared view of security exists in a single place so that the big picture can be inspected for anomalies/attacks.

However, a network-based IDS includes the performance hit of adding a network hop to applications. The need to decrypt/re-encrypt traffic to inspect it is both a massive performance hit and a security risk that makes the network appliance an attractive target. Any traffic that the IDS is unable to decrypt cannot inspect/detect anything.

An IDS is a detection and monitoring tool and does not act on its own. An IPS detects, accepts, and denies traffic based on set rules. IDS/IPS solutions help to prevent DDoS attacks due to their anomaly-detection capabilities that make them able to recognize when valid protocols are used as an attack vehicle. An IDS and an IPS read network packets and compare contents to a database of known threats. Continuous auditing and scanning are required for your infrastructure to proactively secure it from any attack, so let's learn more about this.

In this section, you learned all about securing your infrastructure from various types of attacks. The goal of these attacks is to get hold of your data. You should secure your data in such a way that an attacker is not able to acquire sensitive information even after getting hold of the data. Let's learn about data protection using security at the data layer, encryption, and backup.

**Data security**

In today's digital world, every system revolves around data. Sometimes, this data may contain sensitive information such as customer health records, payment information, and government identity. Securing customer data to prevent any unauthorized access is most important. There are many industries that place stress on data protection and security.

Before architecting any solution, you should define basic security practices as per the application objective, such as complying with regulatory requirements. There are several different approaches used when addressing data protection. The following section describes how to use these approaches.

**Data classification**

One of the best practices is to classify your data, which provides a way to categorize and handle organizational data based on levels of sensitivity.

According to data sensitivity, you can plan data protection, data encryption, and data access requirements.

By managing data classification as per your system's workload requirements, you can create the data controls and level of access needed for the data.

For example, content such as a user rating and review is often public, and it's fine to provide public access, but user credit card information is highly sensitive data that needs to be encrypted and put under very restricted access.

At a high level, you can classify data into the following categories:

* **Restricted data**: This contains information that could harm the customer directly if it got compromised. Mishandling of restricted data can damage a company's reputation and impact a business adversely. Restricted data may include customer PII data such as social security numbers, passport details, credit card numbers, and payment information.
* **Private data**: Data can be categorized as confidential if it contains customer-sensitive information that an attacker can use to plan to obtain their restricted data. Confidential data may include customer email IDs, phone numbers, full names, and addresses.
* **Public data**: This is available and accessible to everyone, and requires minimal protection—for example, customer ratings and reviews, customer location, and customer username if the user made it public.

You can have a more granular category depending on the type of industry and the nature of the user data. Data classification needs to balance data usability versus data access. Setting different levels of access, as mentioned previously, helps to restrict only the necessary data and make sure sensitive data is not exposed. Always avoid giving direct human access to data and add some tools that can generate a read-only report for users to consume in a restrictive manner.

**Data encryption at rest**

Data at rest means it is stored somewhere such as a **storage area network** (**SAN**) or **network-attached storage** (**NAS**) drive, or in cloud storage. All sensitive data needs to be protected by applying symmetric or asymmetric encryption, explained in this section, with proper key management.

Data encryption is a method to protect your data whereby you convert your data from plaintext to encoded ciphertext format using an encryption key. To read these ciphertexts, they first need to be decrypted using the encryption key, and only authorized users will have access to those decryption keys.

Commonly used key-based encryption falls into one of two categories of cryptography:

* **Symmetric-key encryption**: With symmetric encryption algorithms, the same key is used to encrypt and decrypt the data. Each data packet is self-encrypted with a secret key. Data is encrypted while saving and decrypted during retrieval. Earlier, symmetric encryption used to be applied as per the **Data Encryption Standard** (**DES**), which used a 56-bit key. Now, the **Advanced Encryption Standard** (**AES**) is heavily used for symmetric encryption, which is more reliable as it uses a 128-bit, 192-bit, or 256-bit key.
* **Asymmetric-key encryption**: With the help of asymmetric algorithms, two different keys can be used, one to encrypt and one to decrypt. In most cases, the encryption key is a public key and the decryption key is a private key. Asymmetric key encryption is also known as **public-key encryption**. The public and private keys are *unidentical*, but they are paired together. The private key is only available to one user, while the public key can be distributed across multiple resources. Only the user who has a private key can decrypt the data. **Rivest–Shamir–Adleman** (**RSA**) is one of the first and most popular public key encryption algorithms used to secure data transmissions over a network.

Data encryption and decryption come with a performance price as it adds an additional layer for processing. You need to make a careful trade-off while choosing data for encryption. You may want to apply encryption only where it's really necessary to avoid performance and key management overhead.

If you are encrypting your data with an AES 256-bit security key, it's become almost impossible to break the encryption. The only way to decrypt is by getting your hands on the encryption key, which means you need to secure your code and keep it in a safe place. Let's learn about some essential management methods to safeguard your encryption key.

**Encryption key management**

Key management involves controlling and maintaining your encryption key. You need to make sure that only authorized users can create and access the encryption key. Any encryption key management system handles storage, rotation, and destruction of the key in addition to access management and key generation. Key management differs depending on whether you are using a symmetric or asymmetric algorithm. The following methods are popular for key management.

**Envelope encryption**

Envelope encryption is a technique to secure your data encryption key. Data encryption keys are symmetric keys to increase the performance of data encryption. Symmetric encryption keys work with an encryption algorithm such as AES and produce ciphertext that you can store safely, as it is not readable by a human. However, you need to save the symmetric encryption data key along with data to use it for data decryption, as needed. Now, you need to further protect the data key in isolation, which is where envelope encryption helps you to protect it. Let's understand it in more detail with the help of the following diagram:

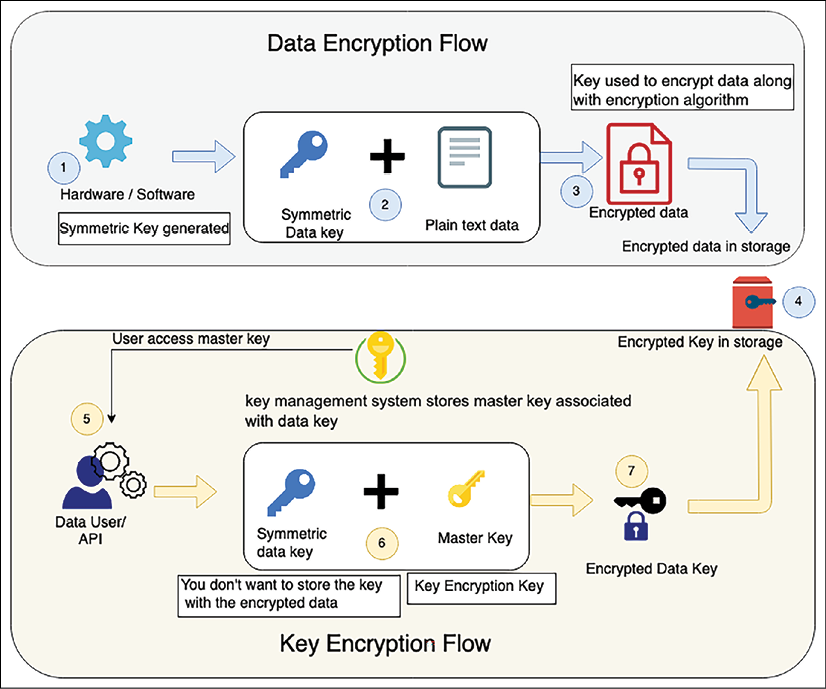


Figure 8.11: Envelope encryption

The preceding diagram illustrates the following flow to explain envelope encryption:

1. The symmetric key is generated from software or hardware.
2. The generated symmetric data key is used to encrypt plaintext data.
3. The key encrypts data using an algorithm such as AES and generates encrypted ciphertext data.
4. The encrypted data is stored in targeted storage.
5. As the data key needs to be stored with ciphered data, the data key needs to be encrypted further. The user gets the customer master key stored in the key management systems to encrypt the data key.
6. The data key is encrypted using the master key. Your master key is the main encryption key as it encrypts the data encryption key. Only the master key can encrypt multiple data keys, and it is securely stored in the key management systems, with restricted access.
7. The master key encrypts the data key, and the encrypted data key is stored along with ciphered data in storage, while the master key is securely stored in the key management system with restricted access.

If a user wants to decrypt data, then they first need a master key that has, in turn, an encrypted data encryption key. This master key can be stored in a separate access system such as a hardware security module or software-based key management service provided by cloud providers such as AWS. Let's look into this in more detail.

**AWS Key Management Service**

AWS **Key Management Service** (**KMS**) uses envelope encryption whereby a unique data key encrypts customer data, and KMS master keys encrypt data keys. You can bring your key material to AWS KMS and manage user access, key distribution, and rotation from a centralized place. You can also disable unused keys, and a low number of keys helps to improve the performance of the application and encourage better key management.

AWS KMS is designed to limit access and protect master keys. KMS helps you to implement key security best practices by never storing plaintext master keys on disk or in memory. KMS also gracefully rotates the master keys to secure your data further.

AWS KMS is a multitenancy key management module; a customer wants to have a dedicated key management module due to compliance. In the same line, other cloud vendors provide a key management system, such as the GCP-provided Cloud Key Management, and Microsoft provides Azure Key Vault.

Sometimes, customers want to have their key management system due to industry regulatory reasons for multi-tenancy. They can choose to store the key in an HSM in such a situation. Cloud providers such as AWS also provide stores, such as AWS CloudHSM. You can choose your own HSM vendor as well. Let's explore HSM in more depth.

**Hardware security module**

A **hardware security module** (**HSM**) is a device that is designed to secure encryption keys and associated cryptographic operations. An HSM is designed with physical mechanisms to protect the keys, which includes tamper detection and response. In case of any key tampering, the HSM destroys the keys and prevents any security compromise.

An HSM includes logical protections to limit access controls. Logical separation helps the HSM appliance administrator to manage the device securely. Access restriction applies rules for users who can connect it to the network and provision the IP address. You can create a separate role for everyone, including security officers, appliance admins, and users.

As the loss of a key can make your data useless, you need to make sure of high availability for your HSM by maintaining at least two HSMs in different geographic locations. You can use other HSM solutions, such as SafeNet or Voltage. To protect your key, choose a managed HSM provided by cloud services such as AWS CloudHSM or CipherCloud.

**Data encryption in transit**

Data in transit means data in motion and transferred over the network. You may encrypt data at rest in the source and destination, but your data transfer pipeline needs to be secure when transferring data. When transferring data over an unencrypted protocol such as HTTP, it can get leaked by an attack such as an **eavesdropping attack** or **man-in-the-middle** (**MITM**) attack.

In an eavesdropping attack, the attacker captures a small packet from a network and uses it to search for any other type of information. A MITM attack is a tampering-based attack, where the attacker secretly alters the communication to start communicating on behalf of the receiver. These kinds of attacks can be prevented by transferring data over SSL, using a strong protocol such as **Transport Security Layer** (**TSL**).

You will observe that most websites now use HTTPS protocol for communication, which encrypts data using SSL. By default, HTTP traffic is unprotected. SSL/TLS protection for HTTP traffic (HTTPS) is supported by all web servers and browsers. HTTP traffic is also applicable to service-oriented architectures such as **Representational State Transfer** (**REST**)- and **Simple Object Access Protocol** (**SOAP**)-based architectures.

SSL/TSL handshakes use certificates to exchange a public key using asymmetric encryption and then use the public key to exchange a private key using symmetric encryption. A security certificate is issued by an acceptable **Certification Authority** (**CA**) such as Verisign. Procured security certificates need to be secured using a **Public Key Infrastructure** (**PKI**). The following is a standard SSL handshake with RSA key exchange:

1. **Client Hello**:The client sends a message to the server to communicate with the client via SSL. Information includes the SSL version number, cipher settings, and user session-specific data.
2. **Server Hello**: Server sendsinformation back to the client, which requires using SSL. The server confirms the SSL version number and certificate with the public key.
3. **Authentication and pre-master secret**: The client authenticates the server certificate with the details such as common name, date, and issuer. The client creates the pre-master secret for the session as per their cipher, encrypts with the server's public key, and sends the encrypted pre-master secret to the server.
4. **Decryption and master secret**: The server uses its private key to decrypt the pre-master secret. Both the server and client perform steps to generate the master secret with the agreed cipher.
5. **Encryption with session key**: Both the client and server exchange messages to inform that future messages will be encrypted. It is called a shared secret. Once shared, the client and server exchange messages to confirm message encryption and decryption. From there, both protect their communication for the rest of the session.

Non-web transmission of data over the network should also be encrypted, and this includes **Secure Shell** (**SSH**) and **Internet Protocol Security** (**IPsec**) encryption. SSH is most prevalent while connecting to servers, and IPsec is applicable to securing corporate traffic transferred over a **virtual private network** (**VPN**). File transfer should be secured using **SSH File Transfer Protocol** (**SFTPS**) or **FTP Secure** (**FTPS**), and email server communication needs to be secured by **Simple Mail Transfer Protocol Secure** (**SMTPS**) or **Internet Message Access Protocol** (**IMAP**).

In this section, you learned about various methods to secure data at rest and in motion with different cryptographic techniques. Data backup and recovery is an important aspect of protecting your data in the case of any unforeseen incidents. You will learn more about data backup in *Chapter 9*, *Architectural Reliability Considerations*, in the *Disaster recovery planning* section.

There are many governing bodies available that publish compliance, which is a set of checklists to ensure customers' data security. Compliance also makes sure that organizations comply with industry and local government rules. Let's learn more about various compliance measures in the next section.

**Security and compliance certifications**

There are many compliance certifications depending on your industry and geographical location to protect customer privacy and secure data. For any solution design, compliance requirements are among the critical criteria that need to be evaluated. The following are some of the most popular industry-standard compliances:

* Global compliance includes certifications that all organizations need to adhere to, regardless of their region. These include ISO 9001, ISO 27001, ISO 27017, ISO 27018, SOC 1, SOC 2, SOC 3, and CSA STAR for cloud security.
* The US government requires various kinds of compliance to handle the public sector workload. These include FedRAMP, DoD SRG Level-2, 4, and 5, FIPS 140, NIST SP 800, IRS 1075, ITAR, VPAT, and CJIS.
* Industry-level compliance of an application applies to a particular industry. These include PCI DSS, CDSA, MPAA, FERPA, CMS MARS-E, NHS IG Toolkit (in the UK), HIPAA, FDA, FISC (in Japan), FACT (in the UK), Shared Assessment, and GLBA.
* Regional compliance certification applies to a particular country or region. These include EU GDPR, EU Model Clauses, UK G-Cloud, China DJCP, Singapore MTCS, Argentina PDPA, Australia IRAP, India MeitY, New Zealand GCIO, Japan CS Mark Gold, Spain ENS and DPA, Canada Privacy Law, and US Privacy Shield.

As you can see, there are many compliance certifications available from different regulatory bodies as per industry, region, and government policy. We are not going into the details of compliance, but you need to evaluate your application with compliance requirements before starting your solution design. Compliance requirements influence the overall solution design heavily. You need to decide what kind of encryption is required, as well as the logging, auditing, and location of your workload based on your compliance needs.

Logging and monitoring help to ensure robust security and compliance. Logging and monitoring are essential. If an incident occurs, your team should be notified immediately and be ready to respond to incidents. You are going to learn more about monitoring and alert methods in *Chapter 10*, *Operational Excellence Considerations*.

There are several compliance industries depending on your application geolocation, industry, and government rules. You learned about the various categories of compliance and some common compliance standards appropriate for each group. Many organizations are moving to the cloud, so it's vital to understand security in the cloud.

**The cloud's shared security responsibility model**

As the cloud is becoming the norm and many organizations are moving their workload to a public cloud such as AWS, **Google Cloud Platform** (**GCP**), and Azure, the customer needs to understand the cloud security model. Security in the cloud is a joint effort between the customer and the cloud provider. Customers are responsible for what they implement using cloud services and for the applications connected to the cloud. In the cloud, customer responsibility for application security needs depends upon the cloud offerings they are using and the complexity of their system.

The following diagram illustrates a cloud security model from one of the largest public cloud providers (AWS), and it's pretty much applicable to any public cloud provider, such as Azure, GCP, Oracle, IBM, or Alibaba:

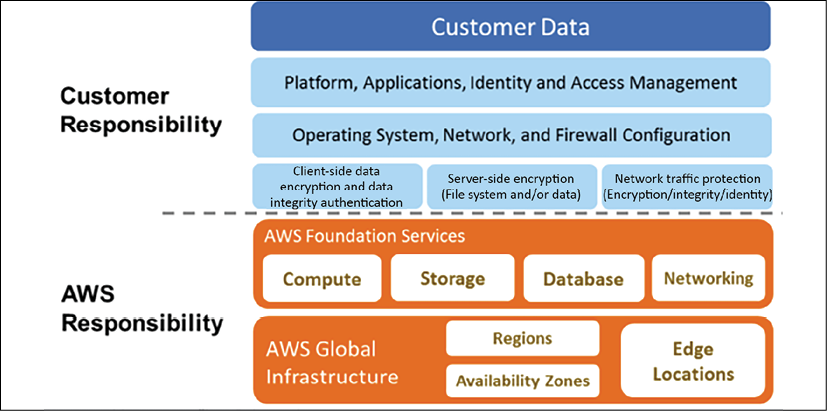


Figure 8.12: AWS cloud shared security responsibility model

As shown in the preceding diagram, AWS handles the security of the cloud, specifically the physical infrastructures that host your resources.

This includes the following:

* **Data centers**: Non-descript facilities, 24/7 security guards, two-factor authentication, access logging and review, video surveillance, and disk degaussing and destruction
* **Hardware infrastructure**: Servers, storage devices, and other appliances that rely on AWS services
* **Software infrastructure**: Host operating systems, service applications, and virtualization software
* **Network infrastructure**: Routers, switches, load balancers, firewalls, cabling, and so on. Also includes continuous network monitoring at external boundaries, secure access points, and redundant infrastructure

The customer handles the **security in the cloud**, which includes the following:

* **Server's operating system**: The operating system installed on the server could be vulnerable to attacks. Patching and maintenance of the operating system is the customer's responsibility, as software applications depend heavily upon it.
* **Application**: Every application and its environments, such as dev, test, and prod, are maintained by the customer. So, handling password policies and access management is the customer's responsibility.
* **Operating system/host-based firewalls**: Customers need to protect their entire system from external attacks. However, the cloud provides security in that area, but customers should consider an IDS or an IPS to add an extra layer of security.
* **Network configuration and security group**: The cloud provides tools to create a network firewall, but it depends on the application requirements as to which traffic needs to be stopped or allowed to go through. Customers are responsible for setting up firewall rules to secure their systems from external and internal network traffic.
* **Customer data and encryption**: Data handling is the customer's responsibility, as they are more aware of the data protection that's needed. The cloud provides tools to apply for data protection by using various encryption mechanisms, but it's the customer's responsibility to apply those tools and secure their data.

The public cloud also provides various compliance certifications that apply to the hardware portions managed by them. To make your application compliant, you need to handle and complete audits for application-level complaints. As a customer, you get an additional advantage by inheriting security and compliance provided by the cloud provider.

Try to automate security best practices wherever possible. Software-based security mechanisms improve your ability to scale more rapidly, cost-effectively, and securely. Create and save a custom baseline image of a virtual server, and then use that image automatically on each new server you launch. Create an entire infrastructure that is defined and managed in a template to replicate best practices for the new environment.

The cloud provides all kinds of tools and services to secure your application in the cloud, along with in-built security at the IT infrastructure level. However, it's up to the customer how they want to utilize those services and make their application secure in the cloud. The overall cloud provides better visibility and centralized management for your IT inventory, which helps to manage and secure your system.

Security is the priority for any solution, and a solution architect needs to make sure their application is secure and protected from any attack. Security is a continuous effort. Each security incident should be treated as an improvement opportunity for the application. A robust security mechanism should have authentication and authorization controls. Every organization and application should automate responses to security events and protect infrastructure at multiple levels.

**Summary**

In this chapter, you learned about various design principles to apply security best practices for your solution design. These principles include key considerations during solution design to protect your application by putting in the appropriate access control, data protection, and monitoring. You need to apply security at every layer. Starting with user authentication and authorization, you learned about applying security at the web layer, application layer, infrastructure layer, and database layer. Each layer is vulnerable to a different kind of attack, and you learned various methods to protect your application with the available technology choices.

For user management, you learned about using FIM and SSO to handle corporate users, and various methods for implementation of user authentication and authorization. These choices include enterprise management services such as Microsoft's AD and AWS Directory Service. You also have options to handle millions of users, using OAuth 2.0.

At the web layer, you learned about various attack types such as **DDoS**, **SQLi**, and **XSS**. You learned about how to protect those attacks, using different DDoS prevention techniques and network firewalls. You learned various techniques to protect code at the application layer and ensure the security of your infrastructure. You dived deep into different network components and methods to build trusted boundaries to limit the attack radius.

You learned about data protection by putting proper data classification in place and tagged your data as confidential, private, or public data. You learned about symmetric and asymmetric algorithms and how they differ from each other. You learned about using key management to protect the public/private encryption key. Data can be in motion or sitting in storage. You learned about how to protect data in both modes. In the end, you learned about various compliance and shared security responsibility models applicable to a cloud workload.

While this chapter is about applying security best practices, reliability is another essential aspect of any solution design. To make your business successful, you need to create a reliable solution that should always be available and able to handle workload fluctuation. In the next chapter, you will learn about the best practices to make your application reliable with the available technology choices. You will learn various disaster recovery and data replication strategies to make your application more reliable.