**What is a Cross-site Scripting attack?**

The Cross-site Scripting attack, also known as XSS attack, is a kind of attack in which a malicious script is added to a website.

When a user accesses this website then they accidentally run this malicious script, compromising their data as the attacker gets

control of the user’s browser.

**Types of Cross-site Scripting Attack**

The XSS attack can be categorized into two types:

**1. Stored Cross-site Scripting Attack**

This is the most dangerous type of XSS attack because it is very easy for the attacker to inject a malicious script through this method.

These attack targets websites that allow user input and store it in the database, e.g. comments.

The attacker writes the malicious script inside the input box, for example, a comment box.

When the attacker clicks submit, the malicious script is saved as a comment in the website database.

When a user opens this website, the malicious script runs on the browser as soon as the comments load.

The malicious script can attack the user through the following methods:

- Installing browser-based Keyloggers to capture keystrokes of the victim. This can be dangerous as the attacker might use this to get

access to social media passwords, email passwords, credit card info, banking passwords, etc.

- Capturing session cookies of the user, which can be used to trigger some other kind of attack, like a CSRF attack.

- Redirecting users to other malicious websites.

**2. Reflected Cross-site Scripting Attack**

In this kind of attack, the attacker tricks the user into clicking a link that contains the malicious script.

This kind of attack is a bit more difficult to execute than stored XSS.

The user may receive the malicious link through email, search results, or advertisements on another website.

As soon as the user clicks on the link, the malicious script is executed on the user’s browser. This script can then steal browser data, such as cookies, and send it to the attacker.

**How to prevent XSS**

- Since XSS works by injecting malicious code into a website, the website owners should make sure that all user inputs are validated

before they are stored into the database. The theory here is to treat all data or input as malicious until they pass certain criteria,

like type and length requirements.

- Sanitizing user input is another mitigation method that essentially requires all user data to be cleaned of potentially

dangerous symbols that are usually used in HTML markup and JavaScript code. There are lots of tools available that blacklist

HTML tags in user input. If your website allows HTML in user input then you can use these tools to blacklist the tags that build malicious code.

**What is CSRF?**

Cross-site Request Forgery (CSRF), is an attack that tricks a web browser into executing an unwanted action in an application after a user logs in.

It allows an attacker to force a logged-in user to act without their consent or knowledge.

In a CSRF attack, the attacker cannot access the data because the attacker does not have access to the response.

This can be devastating, as the attacker can force the user to transfer funds from a banking website or share sensitive information.

**How does CSRF work?**

To perform a CSRF attack, a few conditions should be met.

Cookie-based session handling – The user has already logged in into the website that is being attacked, and

the website relies on cookies to identify the user.

No unpredictable request parameters – The requests that perform the malicious action do not contain any parameters

whose values the attacker cannot determine or guess. For example, when tricking a user into transfering funds,

the attacker must not be required to know the password of the user.

CSRF attack using a GET request

**Scenarion:**

Let’s look at an example of a CSRF attack using a GET request. Suppose a user, Alex, is a customer of ABC bank. He is logged into the bank website.

This means the session is currently active, and login information is maintained in the cookies.

Now, suppose a request to transfer funds looks like this:

GET http://abcbank.com/transfer.do?acct=Bob&amount=＄500 HTTP/1.1

The attacker will create a request in which the account details of the attacker will be provided.

GET http://abcbank.com/transfer.do?acct=attacker&amount=＄500 HTTP/1.1

The attacker has created the request. The only thing they need to do now is trick the user into sending this request from their own browser.

The attacker can create a promotional email which it will send to the user. This email will contain a hyperlink as shown below:

<a href="http://abcbank.com/transfer.do?acct=attackerA&amount=＄500">Get the offer!</a>

If the user clicks on the hyperlink then the transaction will go through and money will be transferred to the attacker’s account.

As you can see, the user must already be logged in for this attack to be successful. Otherwise, the user will get a

login prompt and become skeptical of the link.

**HTTPS:**

**What are SSL certificates?**

When a user accesses a website, data is transferred between the client (browser) and the server (website).

This data is not safe to send in the clear because it may be read by an attacker.

This is a problem if we are sending sensitive data like credit card details, passwords, or personal information over the Internet.

SSL (Secure Sockets Layer) certificates create an encrypted environment between a client and a server.

A Secure Sockets Layer certificate (SSL certificate) is a small data file installed on a Web server that allows for a secure

connection between the server and a web browser.

The certificate is base64 encoded and contains the following information:

- Name of the entity to which the certificate was issued.

- The public key required for encryption and digital signature verification.

- The digital signature created with the private key of the certificate issuer.

**There are three main types of validations:**

**1. Domain Validation Certificates**

Domain Validation SSL or DV SSL is the most basic type of SSL certificate. This type of certificate can be obtained in a

few minutes and is not very expensive. This certificate is suitable for websites that just need encryption and nothing more.

To obtain this certificate, an applicant needs to prove their control over the domain name only. The issued certificate contains

the domain name that was supplied to the certification authority within the certificate request.

**2. Organization Validation Certificates**

To acquire this certificate, the applicant needs to prove that their company is a registered and legally accountable business.

Getting this certificate may take 3-4 days, as the business is vetted to confirm that it is a legal business.

This type of certificate is suitable for sites that need the user to authenticate.

The OV SSL provides a way for customers to check the verified business information in the certificate details section. This is not available in Domain Validation Certificates.

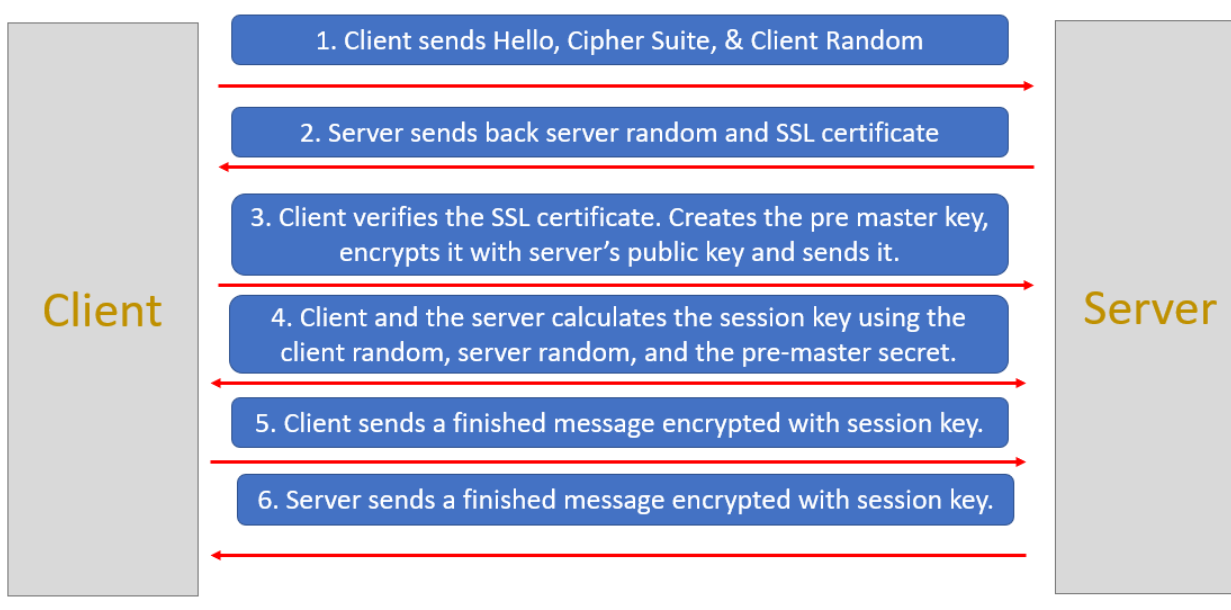
**3. Extended Validation Certificates**

This certificate is very expensive and takes some time, as a lot of vetting is done before this certificate is issued.

This is suitable for applications that ask for confidential details of users like credit card numbers.

This certificate can be easily distinguished from the other two certificates, as the URLs of websites with this certificate have a

green address bar containing the company name.



**What are Cookies**

HTTP cookies, or web cookies, are small text files that store small pieces of information. They are created by the websites we visit and are stored on our browser. Cookies are limited to 4kb in size, which means they cannot store large amounts of data.

**A cookie generally contains:**

name - A website or a third-party server identifies a cookie using its name.

value - A random alphanumeric character, and it stores data like a unique identifier to identify the user and other information.

attribute – A set of characteristics such as the expiration date, domain, path, and flags.

**Types of Cookies**

Based on the source, the cookies can be classified into two types:

**First-party cookies:**

These are installed by the website that the user is currently on.

They are normally used to determine whether a user is logged in.

**Third-party cookies**

These are installed by other websites or third-party servers that are not being viewed by the users.

Third-party cookies are used to track users’ browsing patterns and interests to show relevant advertisements.

You may have noticed that when you search for a product on an eCommerce website, then you start seeing the ads for that product on other websites.

This is achieved through third party cookies.

**Based on the validity, the cookies can be classified into two types:**

**Session cookies:**

Session cookies are created for a single session and vanish once you close the browser.

These cookies are created by the website and the user cannot disable them from the browser.

These cookies are used to save session information while users browse a website. As soon as we close the browser, these cookies expire.

**Permanent cookies:**

Permanent cookies don’t expire even after we close the browser or even shut down the computer.

They have a specific expiration date set by the website and remain valid until then.

Suppose we log in to a website and after a few days, when we try to login again, then we don’t need to re-enter the username and password.

This becomes possible because of permanent cookies.

Since these cookies store sensitive information, it’s not safe and can be risky if people with malicious intentions somehow get access to

our computer.

**Below are the steps to create a session between a user and a web server.**

The user (normally a browser) sends a request to the server. The request contains the login credentials of the user and the info it is requesting.

The web server authenticates the user. It creates a session and stores all the information about the user in memory or a database and returns a sessionId to the user.

This sessionId is stored by the user in browser cookies. The next time the user makes a request it sends the cookies as well in the HTTP header.

The web server looks at the sessionId and checks if it has any info regarding this sessionId.

If the sessionId is valid then the web server recognizes the user and returns the requested information.

**Limitations of Session-based authentication**

There are a few limitations of the session-based authentication. We will discuss them here:

**1. Problems faced in Distributed Systems**

We know that in session-based authentication, the session details are saved on the server. However, in a distributed system,

it is not necessary that a request from a given user will always go to the same server. It’s quite possible that one request is

handled by one particular server and the next request is handled by another server.

**2. Performance Issue**

Storing and retrieving the session information from the database or memory is a costly process.

Each time a new user authenticates, we need to store their information.

And whenever a user sends a sessionId with the request then we need to validate it from the database or memory. This leads to a lot of back and forth.

**3. Cookie Fraud**

It is possible that a malicious user or a website could gain access to your cookies and then perform some malicious operations on a website.

This is also known as CSRF attack, which we have discussed earlier.

**Here is the basic flow of token-based authentication:**

The client sends a request to the server with a username/password.

The application validates the credentials and generates a secure, signed token for the client.

The token is sent back to the client and stored there.

When the client needs to access something new on the server, it sends the token through the HTTP header.

The server decodes and verifies the attached token. If it is valid, the server sends a response to the client.

When the client logs out, the token is destroyed.

**Benefits of token-based authentication**

The following are the benefits of using token-based authentication:

**1. Scalability and Statelessness**

The token-based authentication is truly stateless. The server does not store any client information. Each time a request is made,

the client sends the complete information through a token. This is very useful if our application is deployed on multiple servers.

**2. More Secure**

The token is encrypted, so it is much more secure than cookies. Also, the token expires after some time so the user will have to login again.

**3. Can be generated anywhere**

It is not necessary that the server or application which is validating the token should also generate it. The token generation process can be

done on a separate server or by a different company.

**4. Helpful in Authorization**

Within the token payload, you can easily specify what resources a user can access. For example, if a third-party API wants to access my

Gmail account then I can provide a token that will allow that API to collect only my contact information from Gmail. It will not be able to access other resources.

**What is JWT?**

A JSON Web Token (JWT) is a standard that defines a safe, compact, and self-contained way of transmitting information between a client and a

server in the form of a JSON object. A JWT can either be signed (JWS) or encrypted (JWE) or both.

If a JWT is neither signed nor encrypted, then it is called an insecure JWT.

**Common use cases for using JWT**

Here are some use cases in which JSON Web Tokens are useful:

**1) Authorization**

One of the most important use cases of JWT is the authorization. Suppose we are using an application that needs some data from our Gmail.

We can authenticate ourselves on the Gmail authentication server by providing credentials. Gmail will provide us with a JWT, which our

application can use to get data from Gmail. The token will contain information regarding what data can be accessed.

**2) Information Exchange**

JWT can also be used to share certain information between two parties secretly.

JWT Structure

A JSON Web Token is basically three strings separated by a . (dot)

**1. Header**

This is the first part of JWT. It is also known as the JOSE header (JSON Object Signing and Encryption). This header describes what algorithm is used to sign or encrypt the data contained in the JWT.

The header defines two attributes:

a) alg: the algorithm used to sign or encrypt the JWT.

b) typ: the content that is being signed or encrypted.

**2) Payload**

This is the second part of JWT. It contains the main information that the server uses to identify the user and permissions. The payload consists of claims. Claims are statements about an entity (typically the user) and additional data.

There are three types of claims.

a) Registered Claim Names

These are reserved names that provide a starting point for a set of useful, interoperable claims.

iss: identifies the principal that issued the JWT.

sub: identifies the principal that is the subject of the JWT.

aud: identifies the recipients that the JWT is intended for.

exp: identifies the expiration time at or after which the JWT MUST NOT be accepted for processing.

nbf: identifies the time before which the JWT MUST NOT be accepted for processing.

iat: identifies the time at which the JWT was issued.

jti: The JWT ID is a unique identifier for the JWT. The identifier value MUST be assigned in a manner that ensures that there

is a negligible probability that the same value will be accidentally assigned to a different data object. It can be used to

prevent the JWT from being replayed. This is helpful for a one-time use token.

b) Public Claim Names

Public claim names are JSON Web Token Claims that can be defined at will by those using JWTs.

However, in order to prevent collisions, any new claim name SHOULD either be defined in the

IANA Registry, JSON Web Token Claims Registry, or be defined as a URI that contains a collision resistant namespace.

c) Private Claim Names

A producer and consumer of a JWT may agree to any Private claim name that is not a

Reserved claim name or a Public claim name. Unlike Public claim names, these Private claim names are subject to

collision and should be used with caution.

**3) Signature**

The third and final part of JWT is the signature. It is created by combining the header and payload parts of JWT and then

hashing them using a secret key.

**How tokens are signed**

**There are two mechanisms to sign a token:**

**Symmetric Signatures:**

When a JWT is signed using a secret key, then it is called a symmetric signature. This type of signature is done when there is only one server that signs and validates the token. The same secret key is used to generate and validate the token. The token is signed using HMAC. HMAC stands for Hashing for Message Authentication Code. It’s a message authentication code obtained by running a cryptographic hash function (like MD5, SHA1, and SHA256) over the data (to be authenticated) and a shared secret key.

**Asymmetric Signatures:**

This signature is suitable for distributed scenarios. Suppose there are multiple applications that can validate a given JWT.

If we use a secret key to sign a JWT, then these applications will need that key to validate the token.

It is not possible to share the secret key amongst all the applications, as it may get leaked. To solve this issue,

asymmetric signing is done. Asymmetric signing uses a private-public key pair for signing. There is one server that has the private key.

This server generates the tokens, signs them using the private key, and shares it with the client.

Now the client can send this token to any application and they can validate it using the public key. This signature is done using RSA.

It is asymmetric encryption and a digital signature algorithm.

**How is token validated**

Let us now look at how a server validates a JWT. We already know that a JWT has three parts: a header, a payload, and a signature.

When a server receives a token, it fetches the header and payload from that token. It then uses the secret key or

the public key (in the case of asymmetric signing) to generate the signature from the header and payload.

If the generated signature matches the signature provided in the JWT, then it is considered to be valid.