Web Security and Access Management

**A1 Getting started with web application security**

# **1 Introduction**

**What is web application security?**

* Web application security deals with securing
* websites
* mobile apps
* web APIs.
* Hackers subject web applications to different kinds of attacks, with the objective of stealing data or defacing a website.
* Types of attacks differ based on the mode of attack and the attacker’s objective. Common web attacks include
* SQL Injection
* Cross-site scripting attack
* Cross-site Request forgery
* Denial-of-Service attack.

**Why do we need to secure web applications?**

* As many things are moving online, attackers have easier access to information shared on the internet.
* Millions of financial transactions take place over the Internet every day, and large amounts of private data is shared.
* If a web application is not secured, then it can cause a considerable loss to its users.
* To safeguard the interest of its users, the owners of any given website must take the necessary steps to prevent attacks.
* It is the responsibility of the website owners to put a proper system in place so only the intended users can view its data and perform actions on the website.

**2 Cross-site Scripting Attack (XSS)**

**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 2 types**

* Stored Cross-site Scripting Attack
* Reflected Cross-site Scripting Attack

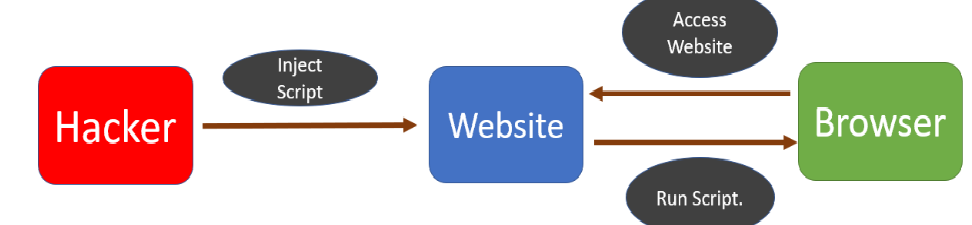
**1. Stored Cross-site Scripting Attack**

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:

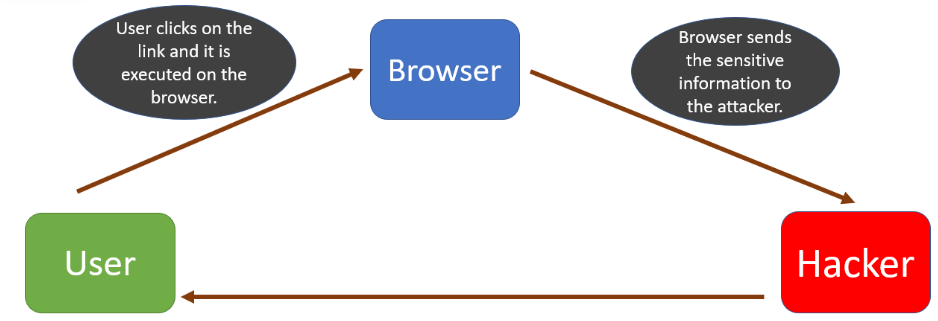
* Installing browser-based Keyloggers to capture keystrokes of the victim. This can be dangerous as the attacker might use this to get access to social media passwords, email passwords, credit card info, banking passwords, etc.
* Capturing session cookies of the user, which can be used to trigger some 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.

**3 Cross-site Request Forgery (CSRF)**

**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.

1. 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.
2. 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**

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 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.

**CSRF attack using POST request**

In an ideal scenario, the GET request is not used for state changes. Normally the state change operations are done through a POST request.

In the case of POST, the user’s browser sends parameters and values in the request body and not the URL, as in the case of a GET request.

Therefore, tricking the user to operate a POST request is a bit difficult. With a GET request, the attacker only needs the victim to send a URL with all the necessary information. In the case of POST, a request body must be appended to the request.

The attacker can create a website and add a JavaScript code to it. They will then send the link of this website to the user through a phishing email.

When the user clicks on the email, the malicious website will send a POST request to the bank application.

The code of malicious website created by the user will be as shown below:

<body onload="document.csrf.submit()">

<form action="http://abcbank.com/transfer" method="POST" name="csrf">

<input type="hidden" name="amount" value="500">

<input type="hidden" name="account" value="attacker">

</form>

As soon as this page loads, the hidden form will be submitted, which will, in turn, send the POST request.

**Methods to prevent CSRF attacks.**

**1. Using CSRF tokens**

This is also known as an anti-CSRF token or synchronizer token. An anti-CSRF token is a type of server-side CSRF protection. It is a random string that is only known to the user’s browser and the web application.

This is how it works:

* When a request is sent to a web server, it generates a token and stores it.
* The token is statically set as a hidden field of the form.
* When the form is submitted by the user the token is included in the POST request data.
* The application compares the token generated and stored by the application with the token sent in the request.
* If these tokens match, the request is valid. Otherwise, the request is considered invalid and is rejected.

Now even if an attacker creates a malicious POST request, it is not possible to add the token as the attacker would not be aware of it.

**2. Same site cookies**

The SameSite flag in cookies is a relatively new method of preventing CSRF attacks and improving web application security. As we see in the method above, an attacker creates a different website that sends a POST request when the user clicks on it through a phishing attack.

If the session cookie is marked as a SameSite cookie, it is only sent along with requests that originate from the same domain. Therefore, even if the user clicks on the hyperlink provided by the attacker, the cookies will not be sent.

**Best practices to avoid a CSRF attack**

There are few things that a user can keep in mind to save itself from a CSRF attack.

* Always log out of the website once work is complete. This will close the session and remove cookies.
* Try not to use multiple websites at the same time. If you are logged in into a website in one browser tab and using a malicious website in another tab, then the chances of CSRF attack increase.
* Do not allow browsers to remember passwords.

**4 Denial-of-Service Attack**

A denial of service or DoS, attack is a type of attack in which the attacker tries to crash an application so that the legitimate users are not able to access the application. The attacker does not gain any benefit from this attack. The main purpose of this attack is to harm an organization, and is often carried out by a competitor or mischievous hacker.

The first DoS attack was done by 13-year old David Dennis in 1974. Dennis wrote a program using the external or ext command that forced some computers at a nearby university research lab to power off.

**Types of Denial of Service Attacks**

The denial of service attacks can be categorized into two types:

* Flood Attack
* Crash Attack

**1. Flood Attack**

In a flood attack, the attacker overwhelms the application with a flood of requests. The application has a limit to the number of requests it can handle per second, and if the number of requests increases exponentially, the server will slow and eventually crash.

There are two types of flood attacks:

* **ICMP flood** – In this attack, the attacker leverages misconfigured network devices by sending spoofed packets that ping every computer on the targeted network, instead of just one specific machine. The network is then triggered to amplify the traffic. This attack is also known as the “smurf attack” or the “ping of death”.
* **A SYN flood** – In this attack, the targeted server receives a request to begin the handshake, but the handshake is never completed. That leaves the connected port occupied and unavailable to process further requests. The attacker continues to send more and more requests, overwhelming all open ports and shutting down the server.

**2. Crash Attack**

This attack is not very common. In this attack, the attacker transmits a bug to the server which then takes advantage of the vulnerabilities of the server.

**Distributed denial of service.**

In a distributed DoS attack, multiple systems orchestrate a synchronized DoS attack to a single target. With this method, the target is attacked from many locations at once instead of being attacked from a single location.

The distributed DoS attack is hard to stop because of the following reasons:

* The attack can be done by systems throughout the world, so it becomes difficult to find the location of the attack.
* It is difficult to counter-attack multiple machines.

**Steps to stop DoS attack**

**1. Black Hole Routing**

If an application owner sees that the application is experiencing an unprecedented load of traffic, then all of the traffic can be routed to a black hole route. In this defense mechanism, the legitimate and illegitimate requests are sent to a black hole, so the application goes down regardless. The benefit is that it gives some time to application owners to look into the origin of the attack and take appropriate action.

**2. Rate Limiting**

Limiting the number of requests a server will accept during a certain window of time is also a way of mitigating denial-of-service attacks. Although this may lead to some valid requests being denied, the benefit of this method is that the system will not be overwhelmed.

**A2 HTTP Basics**

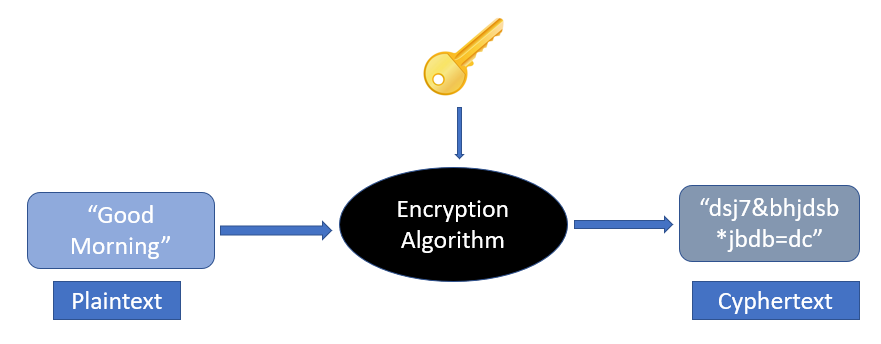
**1 Encryption**

**What is Encryption?**

Today, many people use the Internet for communication, sharing data, banking, shopping, and lots of other stuff. The problem with this level of use is the rapidly increasing threat of cybercrime, and other intrusions on your privacy. A hacker might be listening to our data while it is being transferred over the network. This is troublesome if we are sending sensitive data like passwords.

One of the best ways to protect the data transferred over the Internet is encryption.

Encryption is a way of converting plaintext into ciphertext (an encoded text that is not understandable by the third party). Encryption requires the use of an encryption key and an encryption algorithm. The key is used to encrypt/decrypt the plaintext into ciphertext. How that key is used to encrypt the plaintext is defined by the encryption algorithm. Both the receiver and the sender must have the encryption key.

****

**Types of encryption**

**There are two main types of encryption:**

* Symmetric Encryption
* Asymmetric Encryption

**1. Symmetric Encryption**

In symmetric encryption, the same key is used for both encryption and decryption. Here’s a good analogy to explain how symmetric encryption works:

Suppose John needs to send a message to Carl. John will put a message in a box and will lock the box. Carl already has a duplicate copy of the key which was used to lock the box. Carl will receive the box and will unlock it using the key. This is symmetric encryption. Symmetric encryption is much faster than asymmetric encryption and is used to encrypt a large amount of data.

The disadvantage of this encryption type is that the sender and receiver will have to send their key to each other. An attacker may access this key while in transit and will be able to read all the messages. The keys used in symmetric encryption are not very large, as the max length is 256 bits.

**2. Asymmetric Encryption**

In asymmetric encryption, the sender and receiver use a separate key to encrypt and decrypt the message. This is also known as PKI (Public Key Infrastructure). The advantage of this encryption is that the keys are not transferred over the network. So, it is much safer than symmetric encryption. This encryption is achieved through a public-private key model.

The recipient sends a public key to all the senders. The senders then encrypt the messages using this public key. When the receiver receives the message, it uses its private key to decrypt the message. The keys used in asymmetric encryption are fairly large and can be around 2048 bits.

**NOTE**

Since asymmetric encryption is very slow, it is normally used once to exchange the encryption key safely. After that, all the communication is done using symmetric encryption.

**What is an encryption algorithm?**

An encryption algorithm is a mathematical formula used to transform data into ciphertext. An encryption algorithm uses an encryption key to transform plaintext into ciphertext. The ciphertext can be changed back to plaintext using a decryption algorithm and the decryption key.

Below are some of the commonly used encryption algorithms:

* AES
* DES
* Blowfish
* TwoFish
* RC4, RC5, RC6

**What is a brute force attack?**

In a brute force attack, an attacker tries to guess the decryption key. The attacker is not required to do this manually, and there is computer software that performs the same actions. To prevent this from happening, the key should be very strong, so that it becomes impossible for the computer to try all the combinations.

Let’s see what we mean by a strong key. We know that data is represented by bits in computing language. Each bit can have a value of 0 or 1. If a key is 2 bits long then there are four possible combinations, i.e. 00, 01, 10, 11. This is very easy for computers to crack.

Let’s take a key which is 256 bits long. The total number of possible combinations are 2^{256}. A 256-bit private key will have 115, 792, 089, 237, 316, 195, 423, 570, 985, 008, 687, 907, 853, 269, 984, 665, 640, 564, 039, 457, 584, 007, 913, 129, 639, 936 possible combinations. No supercomputer can crack that in any reasonable timeframe. So, to prevent brute force attacks, the key should be of sufficient length.

**2 Understanding SSL certificates**

**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:

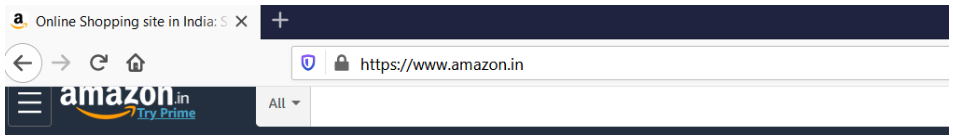
* Name of the entity to which the certificate was issued.
* The public key required for encryption and digital signature verification.
* The digital signature created with the private key of the certificate issuer.

**Note**

SSL is a protocol that is used to secure the HTTP. SSL is deprecated now and Transport Layer Security (TLS) protocol is used instead. Most SSL certificates today also support the Transport Layer Security (TLS) protocol, which is considered to be more secure than SSL.

The application owner should install the SSL certificate on its web server. When an application is secured by an SSL certificate then its URL starts with https instead of http.

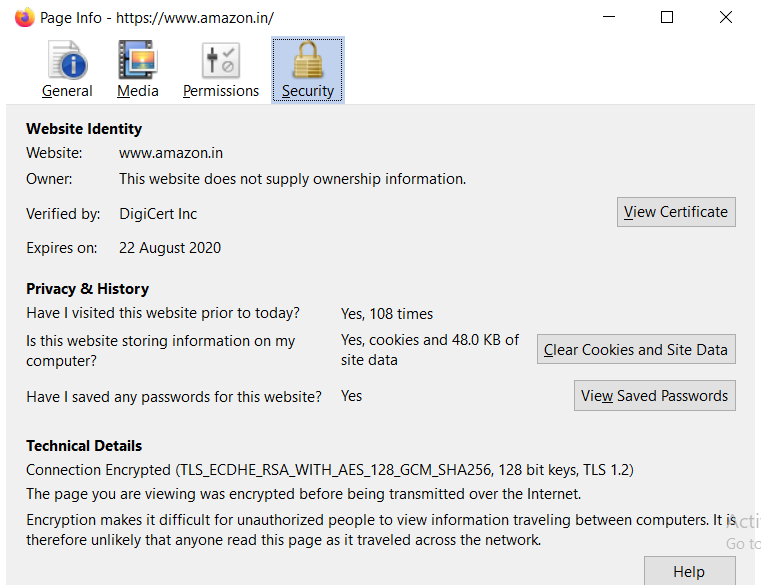
In the below screenshot you might have noticed a lock symbol before the URL. This symbol tells that this website is secured by a certificate.



We can click on the lock symbol to get more details about the certificate as shown in the image below. We can see the certificate and also see which Certification Authority (CA) issued this certificate.

**Note:**

A Certification Authority (CA) is a company or an organization which is trusted to sign, issue and revoke digital certificates. Some of the most popular certification authorities are Sectigo SSL, Symantec SSL, RapidSSL, GeoTrust SSL, and Thawte SSL.



**SSL certificates validation level**

Before a Certification Authority (CA) issues an SSL certificate to an organization, they have to validate the organization. The CA needs to validate that the organization is doing legal business and owns the domain. This is what’s known as SSL certificate validation.

**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.

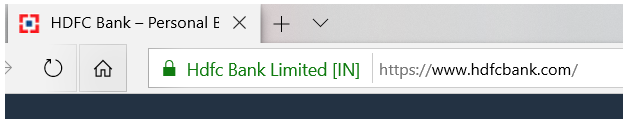
The OV SSL provides a way for customers to check the verified business information in the certificate details section. This is not available in Domain Validation Certificates.

**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.

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.



**Note:**

The green address bar is visible only in Internet Explorer. In Firefox and Chrome, the EV and DV certificates are displayed similarly with just a lock icon.

**Types of SSL certificates**

The SSL certificates can be divided into three types based on the number of domains it protects.

**1. Single-name SSL certificates**

A single-name SSL certificate can only be used on a single domain or IP. It is not applicable on sub domains. For example, if www.mysite.com has a certificate then it is not applicable to blog.mysite.com.

This is the default SSL certificate type and it is available at all validation levels.

**2. Wildcard SSL certificates**

A wildcard SSL certificate is applicable to a domain and all its subdomains. For example, if \*.mysite.com has a wildcard certificate then this certificate will be applicable on all its sub-domains like contact.mysite.com or mail.mysite.com will also have the certificate.

**3. Multi-Domain SSL Certificates**

A multi-domain SSL certificate lists multiple distinct domains on one certificate. With an MDC, domains that are not subdomains of each other can share a certificate. For example, domains like www.mysite.com, www.example.com, and www.abc.com can share the same certificate.

# **3) What is HTTPS, and How Does it Work?**

**What is HTTPS?**

Since the beginning of the Internet, HyperText Transfer Protocol (HTTP) has been used to transfer information between the client and the server. The data is transferred in plain text format through HTTP. This means an attacker can get access to the data that is being transferred over the Internet. In the initial years of the Internet, the data that was transferred was mostly static HTML pages and did not raise any security concerns. As the Internet became more widely used for sensitive tasks like banking, more people felt the need for a secure way to transfer their data.

In 1994, Netscape Communications enhanced HTTP with some encryption. They introduced a new encryption protocol named Secure Socket Layer (SSL) and added it on top of HTTP. This combination of HTTP and SSL is called HTTPS (HTTP Secure).

In 1999, Transport Layer Security (TLS) protocol was introduced. This was more efficient than the SSL protocol, so SSL is no longer used.

**Benefits of using HTTPS over HTTP**

There are few benefits of using HTTPS instead of HTTP:

**1. Authenticity**

HTTPS ensures that the client is talking to the intended website. It is not possible for an attacker to respond to the client’s requests. In HTTPS, the website provides its identity to the client.

**2. Confidentiality**

HTTPS ensures that the data that is being transferred between the client and server is secure and an attacker cannot read it. This is achieved by encrypting the data in HTTPS.

**3. Message Integrity**

HTTPS ensures that the data is not modified by an attacker while it is being transferred over the Internet. It gives the client and server a way to verify that the data has not been tampered with.

**How does HTTPS work?**

In HTTPS, before transferring any data, the client first verifies that the server it is talking to is the correct server and how the data will be encrypted. This process is called the TLS handshake.

TLS handshakes are a series of datagrams, or messages, exchanged by a client and a server. A TLS handshake involves multiple steps, as the client and server exchange the information necessary for completing the handshake and making further conversation possible.

The exact steps within a TLS handshake will vary depending upon the kind of key exchange algorithm used and the cipher suites supported by both sides.

Now, we will look at each step of the TLS Handshake with TLS 1.2 and the RSA algorithm.

**Step 1 -> The ‘client hello’ message**

The client initiates the handshake by sending a “hello” message to the server. The message will include:

* The TLS version the client supports.
* The cipher suites supported.
* A string of random bytes known as the client random.

**NOTE** :

A cipher suite is a set of encryption algorithms that are supported by the client. The client and server need to agree upon which cipher suite will be used for each handshake.

**Step 2 -> The ‘server hello’ message**

If the capabilities of the client and server match, then it replies to the client with the following information:

* The cipher suite it has selected for communication.
* An SSL Certificate.
* A randomly selected prime number is called a **server random.**

**Step 3 -> Certificate Validation**

The client verifies that the SSL certificate provided by the server is valid. The client verifies the following properties of the certificate:

* The certificate’s digital signature.
* The certificate has not expired or revoked.
* The certificate contains the correct domain name.

**Note:**

The client browser has a trust store where it stores certificates from some of the most popular Certificate Authorities.

**Step 4 -> Pre-master secret**

The client generates a random string of bytes called a pre-master secret. The client encrypts it with the server’s public key and transmits it to the server. The client gets the server’s public key from the SSL certificate.

**Note:**

The client encrypts the pre-master secret with the server’s public key. This can be decrypted with only a private key which is present with the server.

**Step 4 -> Pre-master secret**

The client generates a random string of bytes called a pre-master secret. The client encrypts it with the server’s public key and transmits it to the server. The client gets the server’s public key from the SSL certificate.

**Note:**

The client encrypts the pre-master secret with the server’s public key. This can be decrypted with only a private key which is present with the server.

**Step 5 -> Session Key creation**

Now both the client and the server calculate the session key using the client random, server random, and the pre-master secret. They both will obtain the same session key.

**Step 6 -> Client Finished**

The client sends a finished message encrypted by the session key.

**Step 7 -> Server Finished**

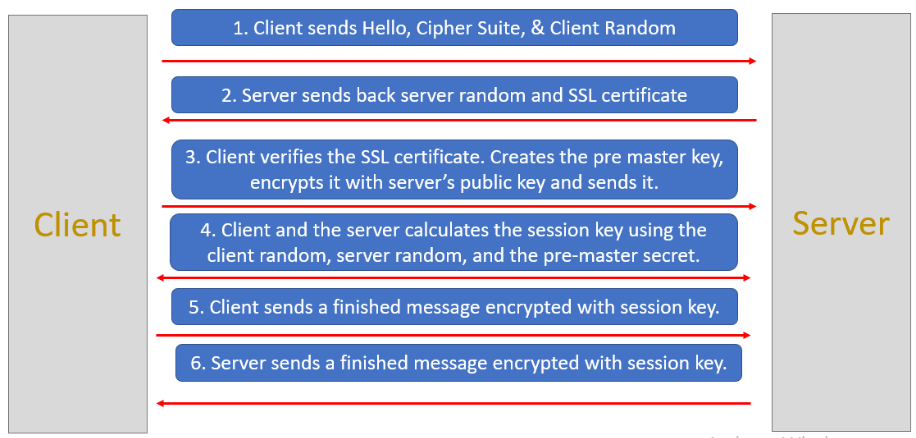
The server sends the finished message encrypted by the session key.

**Step 8 -> Symmetric encryption successful**

Now the client and the server will encrypt their messages with the same session key. Secure communication is achieved.

**Note:**

Let’s suppose an attacker is listening to all the messages while a connection is being established. The attacker can get client random and server random, but it cannot get the pre-master secret. This is because the pre-master secret is encrypted using the server’s public key. Only the server can decrypt it, using its private key. So, the attacker will not be able to calculate the session key.



# **4) What is HTTPS, and How Does it Work?**

**What are Cookies**

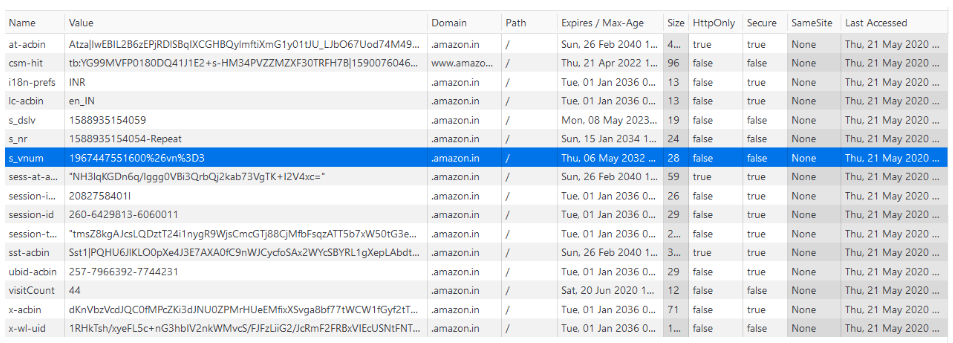
HTTP cookies, or web cookies, are small text files that store small pieces of information. They are created by the websites we visit and are stored on our browser. Cookies are limited to 4kb in size, which means they cannot store large amounts of data.

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.

**Are third party cookies harmful?**

Third-party cookies are created by advertisers, marketers, and social media platforms to track your browsing history. Based on your browsing history, these organizations suggest advertisements to the user.

If you search for a product on Amazon and choose not to buy it, then suddenly you may start seeing the ads for that product on many of the websites you visit. The reason you are now seeing these ads is that your web browser stored a third-party cookie and is using this information to send you targeted advertisements.

If you visit a website and try to create an account, then you may provide certain information like name, address, phone number, and email address. If the website uses third-party cookies, then your contact information may get revealed to other parties in order to send you spam.

According to advertising agencies, third-party cookies are a good thing as they help the advertiser to show relevant ads to the user. From the perspective of a user, though, they might be an attack on privacy.

**A3 JSON Web Tokens**

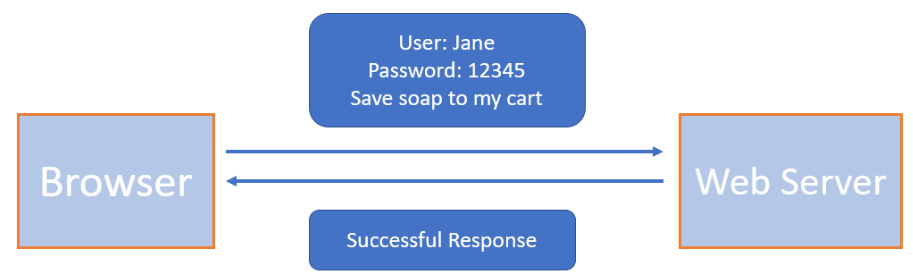
**1 Session-based Authentication**

**HTTP is a stateless protocol**

HTTP is a stateless protocol. This means that each HTTP request is considered an independent request and no information from the previous request is saved. If the application is static and it is available to everyone, then we don’t have any problems. We just need to inform the server which page we want to access, and we will get the result. If the application is dynamic, then we may need to send additional information regarding who is accessing the page.

Let’s say, for example, that we are shopping on Amazon. If we add certain items to our cart, then we should be able to see all the items even after we navigate to a different page. In this case, each time a request is sent to the Amazon server from the client, the client needs to send its identity.

In the below image Jane has sent a request to a web server to add an item to her cart. Jane is sending her credentials along with the request.

****

Jane is sending a second request but since HTTP is stateless, she needs to send her credentials again



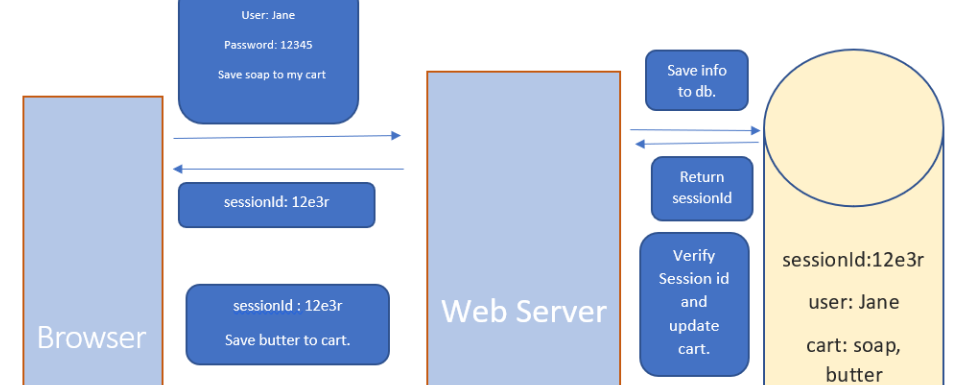
**Introducing Session-based authentication**

If you look at the above example, we need to send our login information each time an HTTP request is made to the server.

This is not a good practice and can be frustrating for the user. To solve this issue, session-based authentication comes into the picture. It is also known as cookie-based authentication.

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

1. 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.
2. 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.
3. 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.
4. The web server looks at the **sessionId** and checks if it has any info regarding this **sessionId**.
5. If the **sessionId** is valid then the web server recognizes the user and returns the requested information.



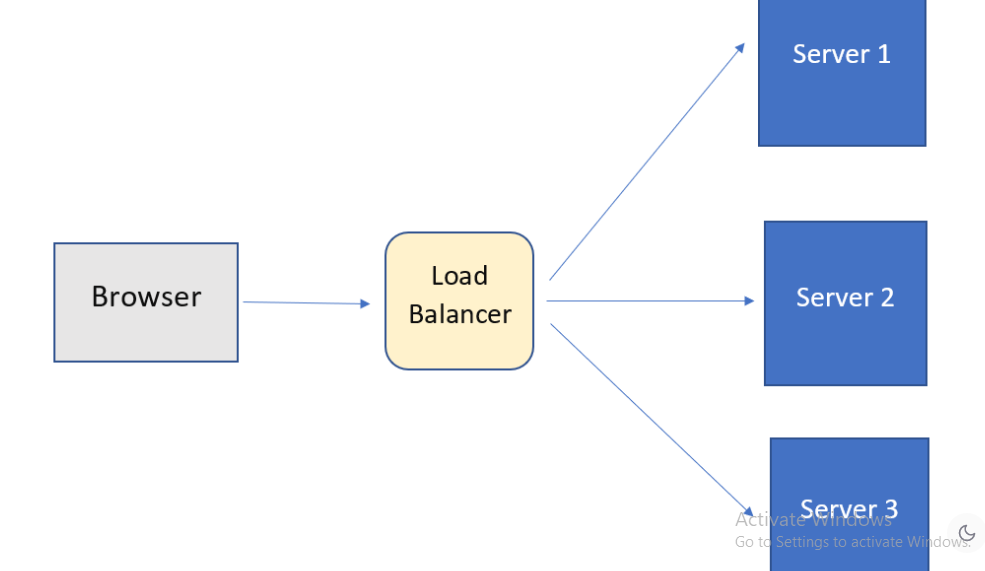
**Limitations of Session-based authentication**

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.

In this case, we can’t use session-based authentication as we can’t save the session info on both servers.



**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.

**2 Token Based Authentication**

**Revisiting session-based authentication**

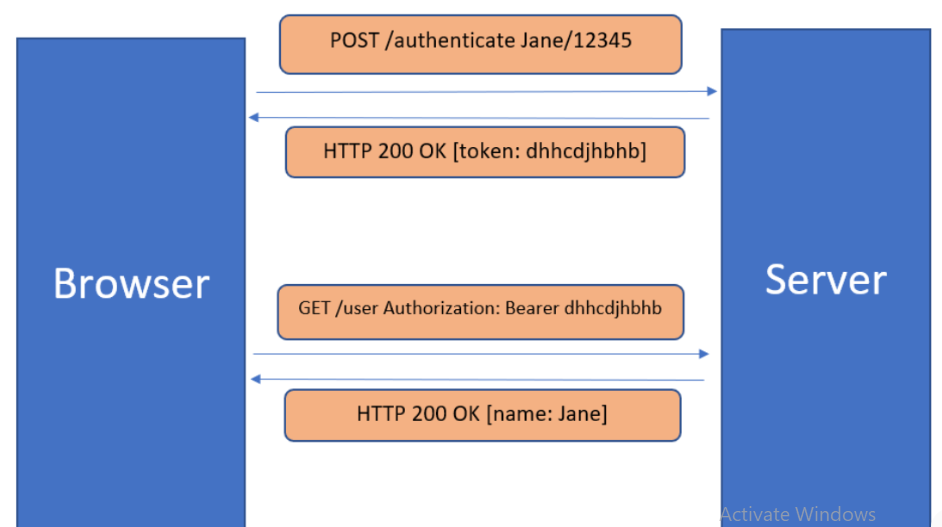
In the previous lesson, we discussed that in session-based authentication, the user information is stored on the server. This resulted in lots of issues related to performance and scalability. But what if we don’t want to save the user information on our server? We can’t save the user information in cookies as they have a size limit and also it is not safe.

**Token-based authentication**

We have an alternative to cookies, and that alternative is tokens. A token can store all the user information in an encrypted format and this token can be stored on the client-side.

Here is the basic flow of token-based authentication:

1. The client sends a request to the server with a username/password.
2. The application validates the credentials and generates a secure, signed token for the client.
3. The token is sent back to the client and stored there.
4. When the client needs to access something new on the server, it sends the token through the HTTP header.
5. The server decodes and verifies the attached token. If it is valid, the server sends a response to the client.
6. 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.

**Types of Tokens**

There are basically two token types:

1. Access Tokens
2. Refresh Tokens

**Access tokens** are used to grant access to a protected resource. When a client first authenticates it is given both types of tokens, but the access token is set to expire after a short period. By doing this, even if someone manages to get access to your token, it can be used only for some time.

**Refresh tokens** are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with an identical or narrower scope. It does not need the credential information again. The refresh token is also valid for some duration, but it is much more than an access token.

**Note:** Although the refresh token does not need the user’s credentials again to generate an access token, it still requires the **client id** and **client secret** (we will look at these terms in OAuth lessons). So even if attackers get a refresh token, they will not be able to get the access token.

**3 JWT - JSON Web Token**

**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.

JWT is one way of creating an access token. There are few alternatives to JWT such as Branca, Pasito, and Macaroon.

By **signing the JWT**, its integrity will be maintained. Other parties will be able to see the data in the JWT but will not be able to modify it.

By **encrypting the JWT,** its secrecy is maintained between two parties. Other parties will not be able to see the data, but if they change anything then we will not be able to find out.

In this lesson, and in further lessons, we will be discussing signed JWTs. Encrypted JWT is a more complex topic and is out of the scope for this course.

**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)



We will discuss **each part of the JWT** and see what they mean.

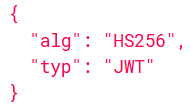
**1. Header**

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:

* **alg**: the algorithm used to sign or encrypt the JWT.
* **typ**: the content that is being signed or encrypted.

The header JSON looks like as shown below.



Now when we encode it to **base64encode**, we get the first part of our JSON web token.



This is just encoding and not encryption. Anyone can easily decode this string and get the JSON.

**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.

**Note:**

Suppose a JSON web token is assigned to a user with jti as 1234. This user will use this token once, and suppose an attacker gets access to the token. When the attacker sends this token again, then the server will check the jti field and it will get to know that the token has been used earlier. It will not allow the token to be used again. Another benefit of the jti field is that it can be used to blacklist the tokens.

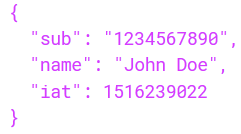
**b) Public Claim Names**

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.

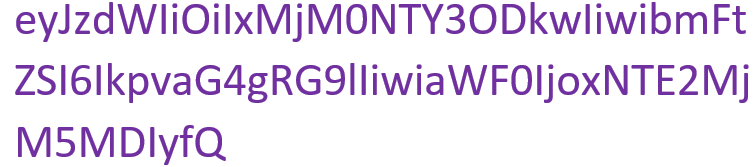
**c) Private Claim Names**

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.

The payload JSON looks like as shown below.

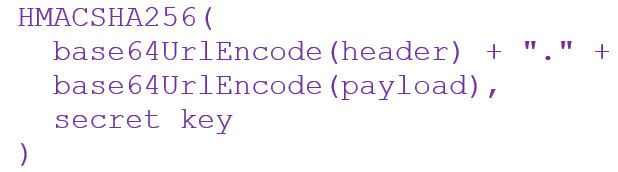


Now when we encode it to base64encode, we get the second part of our JSON web token.



**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.



Let’s say our secret key is **12345**. In that case, the signature will be:



Since this is a signature, if someone tries to decode it they will not be able to get the JSON.

**4 JWT Validation**

In this lesson, we will look at how JWTs can be used as an authentication and authorization mechanism. As mentioned in the previous lesson, we will be discussing signed JWTs.

Here is the basic flow of JWT authentication:

1. The client sends a request to the server with user credentials.
2. The server generates a signed JWT for the client if the credentials are valid.
3. The server sends the token back to the client which is stored in the browser.
4. For every subsequent request, the client sends the token back to the server.
5. The server validates the token, and if it is valid then grants access to the client.

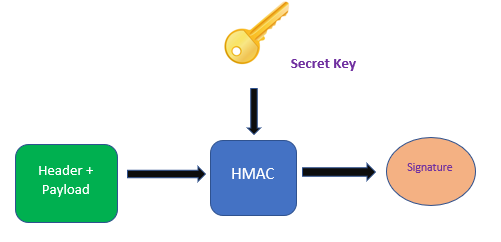
**How tokens are signed**

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

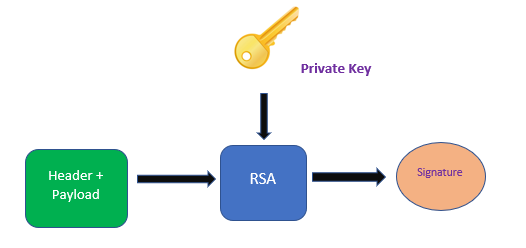


**Asymmetric Signatures**

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.

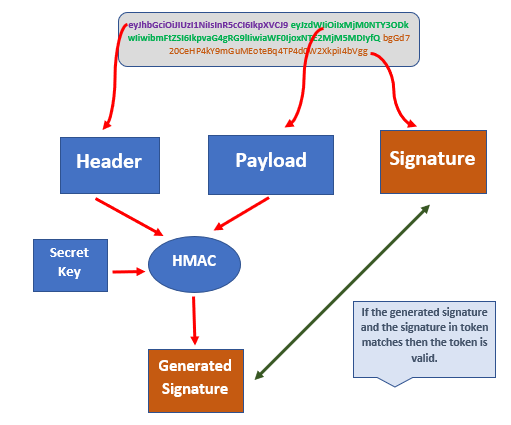


**How is token validated**

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.



**Claims validations**

It is not sufficient to just validate the signature of the token. There are a few other security properties that need to be validated as discussed below:

1. Check if the token is still valid. This can be validated through exp claim.
2. Validate that the token is actually meant for you through the aud claim.
3. Check if the token can be used at this time using the nbf claim. NBF stands for not before which means that this token should not be used before a particular time.