1. Suppose a web client makes connection with a web application but it never sends any request or terminate the connection. What are the different types of attack a client would be able to launch?  
     
   **DOS attack**: DOS or Denial of service refers to the technique where a system or group of system send enormous data/information, making the network resources unavailable for all the users connected to the network.  
     
   **Flooding**:

Flooding is an attacking technique in which the attacker is sending large chunk of information, which web server is unable to handle at the given interval by consuming memory and in turn paralyzing it. Few of the flooding techniques are listed below:

**Syn-ack flood:**

Syn-ack flood is a DDos attack that exploits the “three-was handshake” (TCP connection sequence), where a SYN request is initiate a TCP connection, the host must be answered back with a SYN-ACK response and in turned confirmed by ACK response.

Diagram

Description automatically generated  
 [(Anon., www.cloudflare.com)](https://www.cloudflare.com/learning/ddos/what-is-an-ack-flood/)

However, in a SYN flood scenario, the web client sends multiple SYN request. In addition, it either doesn’t response to the host’s SYN-ACK response or send the SYN request from a spoofed IP address. In either case, the host server continues to wait for an acknowledgement for each of the request and occupying the system resources, leading to the denial of service for all the users in the network.

Diagram

Description automatically generated with medium confidence

(Anon., n.d.)

**ICMP (Ping) Flood**: An ICMP flood is a similar flooding attack which overloads the web application with ICMP Echo Request (ping) packets, mostly sending packets immediately, possible without waiting for any replies. This type of attack can consume both outgoing and incoming capacity, since the web application servers will often try attempting to respond with ICMP Echo Reply packets, this would significantly result in overall system slowdown and eventually failure.  
In our case we can assume that the attacker has compromised the web client and hence sending the data to the web application server and flooding with ICMP packets, as shown in the diagram below:  
A picture containing diagram

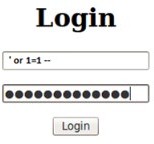
Description automatically generated

(Anon., www.cloudflare.com)  
In addition, these attacks can also be performed by targeting one web application from multiple clients so that the web application gets overloaded with the load it can handle and start denying additional connection or request to be answered and lead to Denial of service.

<https://www.imperva.com/learn/ddos/ddos-attacks/>

https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos#:~:text=There%20are%20two%20general%20methods,the%20most%20common%20DoS%20attack.

1. Consider the figure below, identify the particular type of attack that is illustrated and discuss the effects that suck attack will have. Also, discuss two defense strategies to prevent this attack (illustrate with examples and code instructions as needed).



**SQL injection** :   
  
  
The above figure is an example of SQL injection, which is one of the most common attacking techniques observed.

Here the result will be true every time and return the details of all the users from the Database tables.

The attacker has entered a malicious code into the Login form and the system fails to verify this information which reveal the information to the attacker to which he shouldn’t have the access. In few scenarios, the attacker can also escalate the SQL injection and compromise the server, other infrastructure or perform Denial of service attack.   
 **‘or 1=1 - -**  
Consider the above mentioned code,

The “**or**” operator check for condition and gives the output if either one of the following conditions is true.

The condition **“1=1**” is always true and can result in unauthorized access to sensitive data such are credit cards details, personal user information etc.

The following part of the code is the **“- -"** which is a comment indicator in SQL .i.e that the rest of the code mentioned after this is interpreted as comment.  
(Anon., www.w3schools.com)

The effect of such attacks could lead to leak of user sensitive information such as bank details, user account details, credit card information, user credentials etc. The attacker can abuse the user details and perform actions like transferring money from users account to attackers bank, changing credentials, modifying any sensitive information stored in the database, posing as a user to buy items, identity theft, gaining access to user sensitive information and demanding for ransom etc.

The below defense strategies could help us prevent this kind of attack:  
1. **User Parameterized SQL statement**:  
This is one of the most commonly used technique and are simple to write and easy to understand than dynamic queries, where the query is passes as parameter(prepared statement) rather than as a string concatenation within the query.  
  
Parameterized queries helps the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied. For example:

**String query = "SELECT \* FROM products WHERE category = '"+ input + "'";**

**Statement statement = connection.createStatement();**

**ResultSet resultSet = statement.executeQuery(query);**  
  
The above-mentioned code has vulnerabilities related to SQL injection because user input is concatenated directly into the query.  
 This code could be easily written in a way that prevents the user input from interfering with query structure:  
  
**PreparedStatement statement = connection.prepareStatement("SELECT \* FROM products WHERE category = ?");**

**statement.setString(1, input);**

**ResultSet resultSet = statement.executeQuery();**  
Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. In the safe example below, if an attacker were to enter the userID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'='1.  
  
(Anon., portswigger.net)  
  
  
2. **Allow -list Input validation:**  
Various parts of SQL queries aren't legal locations for the use of bind variables, such as the names of tables or columns, and the sort order indicator (ASC or DESC). In such situations, input validation or query redesign is the most appropriate defense. For the names of tables or columns, ideally those values come from the code, and not from user parameters.

But if user parameter values are used for targeting different table names and column names, then the parameter values should be mapped to the legal/expected table or column names to make sure unvalidated user input doesn't end up in the query. Please note, this is a symptom of poor design, and a full rewrite should be considered if time allows.

Here is an example of table name validation.  
String tableName;

switch(PARAM):

  case "Value1": tableName = "fooTable";

                 break;

  case "Value2": tableName = "barTable";

                 break;

  ...

 default      : throw new InputValidationException("unexpected value provided"

+ " for table name");

The tableName can then be directly appended to the SQL query since it is now known to be one of the legal and expected values for a table name in this query. Keep in mind that generic table validation functions can lead to data loss as table names are used in queries where they are not expected.

For something simple like a sort order, it would be best if the user supplied input is converted to a boolean, and then that boolean is used to select the safe value to append to the query. This is a very standard need in dynamic query creation.  
  
public String someMethod(boolean sortOrder) {

 String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" : "DESC");` ...

(Anon., cheatsheetseries.owasp.org)

1. Consider a scenario in which an attacker steals a TLS-enabled website's private key while avoiding detection. What is the attacker capable of with the private key?

This scenario can be referred to Man in the middle attack.

Private key is the most important part of TLS/SSL certificate, which is used to decrypt or encrypt information. If compromised, the attacker with the private can decrypt the information and can gain access to user sensitive data and intercept the information.  
In this scenario where the attacker has successfully stolen the private key without being detected, he can perform a serious damage to the user, as the legitimate user is unaware that his/her information has been compromised. This form of attack is a type of eaves dropping where an attacker intercepts and exploits the information and the data can be gleaned.  
 The attacker can also impersonate as a legitimate user and perform malicious activities like transferring amount to the attackers bank account, changing user credentials, modifying any sensitive information stored by the user in the database, targeting users the connections list and sending malicious content to them in order to enhance his attack, posing as a user to buy items online, identity theft, gaining access to user sensitive information and demanding for ransom etc.

Additionally, it can also be used to gain a foothold inside a secured perimeter during the infiltration stage of an advanced persistent threat (APT) assault.  
  
  
 Traffic Eavesdropping :

Since the attacker has gained access to the private key of the user without being detected, it becomes very difficult to spot on. The attacker can use all the user’s sensitive information(like user credentials, email access, account access etc.)that the user presumes are secure. In most cases the interception occurs in real time.

To the attacker, everything being sent over TLS can be decrypted and should consequently be thought of as cleartext. This means passwords, credit card numbers, and other Personal and Private Information is vulnerable to being either harvested or leveraged against the user as the attack escalates. Since the users entire session is visible to the attacker. In some cases, the attacker can also alter the data that is being transmitted between the user and the web application, the figure below illustrates a clear example of such attack. Diagram

Description automatically generated

Diagram

Description automatically generated   
  
<https://www.venafi.com/blog/finding-private-key-tlsssl-certificate>

<https://www.internetsociety.org/deploy360/tls/basics/>

https://www.cisa.gov/uscert/ncas/alerts/TA14-098A

<https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/>

<https://www.techtarget.com/iotagenda/definition/man-in-the-middle-attack-MitM>  
<https://security.stackexchange.com/questions/16685/what-can-an-attacker-do-with-a-stolen-ssl-private-key-what-should-the-web-admin#:~:text=Part%20of%20the%20reason%20to,server%20is%20modified%20in%2Dtransit>.

1. Would the following code running on [https://hacker.com](https://hacker.com/) be allowed to print out the contents of the Quercus homepage, which include the currently logged-in student’s grades? Why or why not?

<script>

const result = await fetch(' https://sis.ncirl.ie/apex/f?p=1202:LOGIN') const data = await result.body.text()

console.log(data) // I have your grades!

</script>

You can assume that [https://sis.ncirl.ie/apex/f?p=1202:LOGIN](https://sis.ncirl.ie/apex/f?p=1202%3ALOGIN) does not send any special HTTP headers such as Access-Control-Allow-Origin, which are also known as "CORS" headers.  
  
The following code would not return the student grades and attacker will not get the data as CORS( Cross Origin resource sharing)is not included.  
CORS or Cross origin resource sharing is a HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should be allowed to load its resources.

For example, if we want to access an API hosted at https://api.myworld.com from our client-side frontend application that is hosted at https://globe.com, the browser will not allow this request to complete as there are not from the same origin.

CORS can be considered during the following scenario:

a. API is accessed by the browser.  
 b. API is hosted on a separate domain.

The browser has a security feature known as Same Origin Policy(SOP) which allows one web page to access the data of the second web page only if they are from same origin. The browser checks for the following conditions:  
**Scheme name**: It is a protocol that is used to access the data on the internet. Most commonly used protocols are http://, https://, ftp:// etc.

**Host name**: A hostname is a domain name assigned to a host computer. It is the address of the host where the resource is located. This is usually a combination of the host’s local name with its parent domain’s name.

E.g. [www.hacker.com](http://www.hacker.com) consist of host machine name as www and domain name has “hacker.com”.

**Port number**: It is the way to identify a specific process to which an internet or other network message is to be forwarded when it arrives at a server. Some of the commonly used ports are : 80, 443, 3306 etc.

<https://www.educba.com/types-of-networking-protocols/>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

<https://www.educative.io/answers/how-cors-cross-origin-resource-sharing-works>

1. Suppose an attacker includes the following code on their site [https://hacker.com](https://hacker.com/) that makes GET request to a vulnerable bank server and transfer money into the attacker’s account.

<img src=“<https://vulnerablebank.com/transfer?amt=1000&to=attacker>“ />

The attacker is hoping that the victim had visited the vulnerable bank website before they visit the [https://hacker.com](https://hacker.com/) and send the GET request to the bank. The attacker entices the victims to visit their site by including hundreds of the images. Explain how the bank can modify their server code to protect its customers from this attack.  
  
  
The following steps could be taken from the bank’s server end in order to protect its users:

CSRF or Cross Site request forgery refers to the technique where the attacker tries to manipulate or force user to carry out unintentional activity on a web application on which they are currently authenticated and perform actions such as transferring funds, changing credentials, email addresses etc  
Diagram

Description automatically generated  
  
The bank can enforce the below necessary security measures to mitigate from such attacks:

**Session time out** – The bank can ensure the session times out after a short period of time. This reduces the timeframe that the attacker has to perform the attack and hence protecting the users. This also ensures that the application will logout the user if there is no input from his end within a specified time frame.

**Unique CSRF tokens** – The most common method to prevent CSRF attacks is to append random challenge tokens .The user should get a random CSRF token every time he logs into the application. In all subsequent requests, this token should be passed and validated in the server. This effectively disable the malicious URLs as they cannot be authenticated by the server, which checks for the unique CSRF tokens. An example of this could be google authenticator, Microsoft authenticator etc.

**Tokens per request** – The aforementioned CSRF tokens are usually given out per session but can be given out for each request separately for ensuring that the request is valid and not coming from a source other than the user.Since the tokens has been established during each interval, the attacker will not able to know what is the token of the page, because with every loading of the page the token will change to other random number\string.  
  
<https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html>

<https://www.valencynetworks.com/blogs/csrf-for-banking-apps-final-updated/>

<https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/?utm_source=google&utm_medium=cpc&utm_campaign=sw-waf-IN&utm_content=&utm_term=csrf&gclid=Cj0KCQiA37KbBhDgARIsAIzce14dxkT1d1klvUSMAJ1dTM9RU_DU-m0_L2iG62gBukT9YFXluzewhpUaAkUsEALw_wcB>

1. The following use case diagram is for a Car Rental System, and is focused on the functionality of the system.

Identify and specify the security requirements that should be considered in relation to these use cases. Your answer should include at least 2 functional and 3 non-functional security requirements (one of each type: Security property requirements, Constraint/ Negative requirements, or Security Assurance Requirements)  
  
Functional security requirements as per the user case diagram:  
1. The Web application must ensure the validation from user input and that it doesn’t exceed the limit allocated for type of input.   
2 The Web application must also have defense in place against denial-of-service attack.  
Non-functional security requirement as per the use case diagram:

1.Security Property Requirement : The application must check the integrity of the user account information.

2.Constraint/negative requirement : The application must not return data to the user to which he doesn’t have access to or not authorized.

3.Security Assurance Requirement : The development processes must comply with SSE-CMM capability level 3 or above.

* 1. Provide an example of an abuse case (Outsiders trying to breach the system); update the diagram accordingly and provide its basic flow description

**Abuse case : SQL injection**

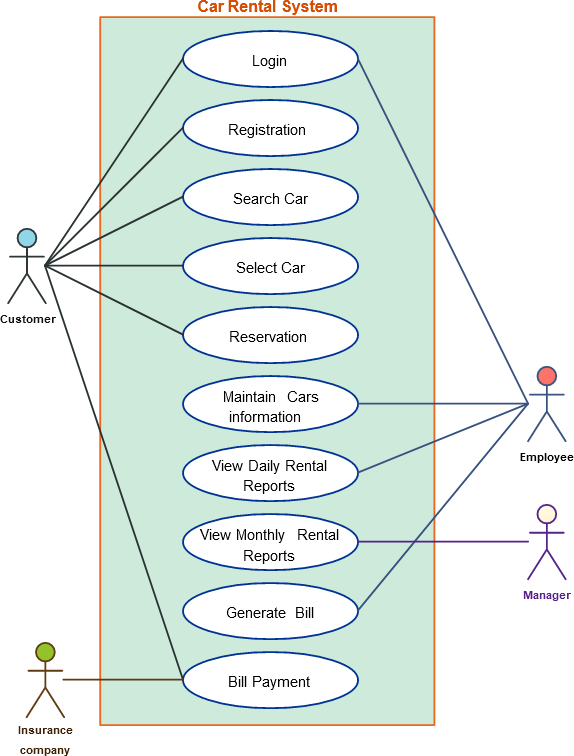
Diagram

Description automatically generated

Users data

‘or1=1--

In this scenario the attacker is trying to perform a SQL injection by passing “ ‘or 1=1—“ which is always true and return all the users data. The attacker can then use this data to perform malicious activity. The attacker can also impersonate as a legitimate user and perform malicious activities like modifying payment details and to transfer money to attackers bank account, changing user credentials, altering the data stored in the database etc.

* 1. Provide an example of a misuse case (Insiders doing inappropriate tasks intentionally); update the diagram accordingly and provide its basic flow description  
       
     Misuse case : Information modification by the employee.  
       
     Diagram

     Description automatically generated  
       
     In this scenario the Bad employee is trying to misuse his privileges by changing the daily rental report details and stores a false information into the data base and also generate fake bills that could leverage his sales target.

1. Suppose you're a web developer working for a MY news. The product manager at MY informs you that management has made a decision to ban users of the BAD web browser. The reason is BAD web browser has ad-blocking capabilities that are negatively affecting revenue of MY news. Now, as a web developer, it is your duty to implement this functionality. BUT! you found, BAD browser uses the same User-Agent header value as the Chrome browser, so it's not possible to distinguish BAD browser users by merely looking at this header value. Propose and describe a method that you could use to distinguish BAD users from other browser users. It's okay if your method has some false positives, as long as it recognizes all BAD users.

Scenario 1:

assuming my new is using google ad sense for their adverts,  
<https://www.whatismybrowser.com/>

<https://developers.whatismybrowser.com/>

<https://www.labnol.org/code/19818-detect-adblock-javascript>

<https://www.youtube.com/watch?v=x5KpgaaJ8Vo&t=365s>   
<https://tms-outsource.com/blog/posts/how-do-websites-detect-adblock/>

<https://www.detectadblock.com/>

<https://easylist-downloads.adblockplus.org/easylist.txt>

<https://www.publift.com/blog/ad-blockers>

its uses client headers as well as client hints,t hey also offer api which allow broswer detection , using this api we can find out whcih rowser we are currently using.

8. Compare and contrast XSS vs CSRF; provide defenses and discuss each of them. Illustrate with examples in the discussion.  
  
  
**XSS (cross site scripting)** refers to a technique in which an attacker inserts a malicious code into the webserver and gains users’ information. This is done by exploiting trust a browser has in the data sent to the legitimate website. Here an attacker executes an arbitrary java script within the victim’s browser. XSS can also be described as “two way” in which the attacker can inject malicious script which could be an arbitrary request , read the responses and exfiltrate data to an external domain of the attackers preference.   
It’s estimated that more than 60% of web applications are susceptible to XSS attacks, which eventually account for more than 30% of all web application attacks.  
  
https://www.securecoding.com/blog/xss-attacks/#:~:text=It's%20estimated%20that%20more%20than,of%20all%20web%20application%20attacks.  
  
The consequences of XSS are often more serious than CSRF, both technique exploits trust that a site has on user’s browser.  
  
**CSRF ( cross site request forgery)** refers to the technique in which the attacker tries to manipulate the user to perform action which they never intended to perform on a web application where they are currently authenticated. Here java script is not required to perform the attack and the malicious code is stored in the third-party sites.  
In most common cases, the attacker performs the request like transferring funds, changing email address, changing user account password, update address or phone number etc.  
  
Here the attacker tries to misuse the same origin policy feature which is designed to prevent different websites from interfering with each other. .   
  
XSS defences with examples:  
  
1. **Validate the input and escape the untrusted data based on context and in correct order**: All the data being parsed must be validated against the web applications logic before processing the information or moving it to the storage.  
Ensure that the executable portion of any untrusted users is removed that could appear in the HTML pages.  
At the point where user controllable data is output in HTTP responses, encode

the output to prevent it from being interpreted as active content. Depending on the output context,

this might require applying combinations of HTML, URL, JavaScript, and CSS encoding etc.  
E.g.: If a user submits a URL that will be returned in responses, validating that it starts with a safe protocol such as HTTP and HTTPS.

Graphical user interface

Description automatically generated with medium confidence

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://owasp.org/www-pdf-archive//Xenotix\_XSS\_Protection\_CheatSheet\_For\_Developers.pdf

2. **Use <!DOCTYPE HTML>:** DOCTYPE (DTD or Document Type Declaration) tells the browser to follow the standard in rendering HTML, CSS as well as how to execute script. Always use<doctype html> before <html>

Graphical user interface, text, application

Description automatically generated  
  
  
<https://www.w3schools.com/tags/tag_doctype.asp>

**3.Follow whitelist approach**: Create a whitelist of all the allowed tags and tags that a web application should accepts from a user and hence blacklist can be easily bypassed.  
  
4. **Prevent CRLF Injection /HTTP response Splitting:** Sanitize and encode all user supplied data properly before passing out through HTTP headers. CRLF Injection can destroy and bypass all the security headers like CSP, X-XSS Protection etc.

**5.Use appropriate response headers**: To prevent XSS in HTTP responses that aren't intended to

contain any HTML or JavaScript, we can use the Content Type and X Content Type Options headers to

ensure that browsers interpret the responses in the way we intend to.

6. **Content security policy**: As a last line of defence, we can also implement Content Security Policy (CSP) to

reduce the severity of any XSS vulnerabilities that still occur.

<https://portswigger.net/web> security/cross site scripting

https://owasp.org/www

CSRF defences with examples:  
**1.Employ anti-forgery tokens (request verification tokens) :** CSRF token can be prevented CSRF since the attacker cannot validate the request with the server. For Synchronised Token Pattern, CSRF tokens should not be transmitted using cookies. The CSRF token can be transmitted to the client as part of a response payload, such as a HTML or JSON response.  
 Eg.

<form action="/transfer.do" method="post">

<input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZMGYwMGEwOA==">

[...]

</form>  
  
Inserting the CSRF token in the custom HTTP request header via JavaScript is considered more secure than adding the token in the hidden field form parameter because it uses custom request headers  
<https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#synchronizer-token-pattern>

**2.Identifying Source Origin : ( via Origin and referrer header)**

**Verify the referrer header** : If the Origin header is not present, verify the hostname in the Referrer header matches the target origin. This method of CSRF mitigation is also commonly used with unauthenticated requests, such as requests made prior to establishing a session state, which is required to keep track of a synchronization token.

**Verify Origin Header:** If the Origin header is present, verify that its value matches the target origin. Unlike the Referrer, the Origin header will be present in HTTP requests that originate from an HTTPS URL.

In both cases, make sure the target origin check is strong. For example, if the site is hacker.com make sure hacker.org.attacker.com does not pass the origin check (i.e, match through the trailing / after the origin to make sure we are matching against the entire origin).If neither of these headers are present, we can either accept or block the request. It is recommended to block.

https://portswigger.net/web-security/csrf/xss-vs-csrf#:~:text=What%20is%20the%20difference%20between,they%20do%20not%20intend%20to.  
  
--------------------------------------------------------------------------------------------------------------------  
  
What the attacks have in common is that one site tries to act withtheprivileges of another site, hence the phrasecross-site.

**XSS**works by exploiting the trust a browser has in data sent to it from a legitimate website.

othe attacker triesto manipulate**what the site sends to thebrowser**.

**CSRF**exploits the trust a website has in data sent from a semi-trustedbrowser.

othe attacker tries to manipulate **what the browser sends to thesite**.

In short, it's all aboutexploiting trust.

The right defense is toreduce the trust as much as possible. In particular, by using inputvalidation.

XSS defences with examples  
  
CSRF defences with examples

3 different types of XSS  
  
Reflected  
Stored  
DOM – script is inserted into an element within HTML code modifying the DM environment in the browser  
  
Defenses:  
XSS

Defenses Summary

1.

Validate the input and Escape untrusted data based on context and in correct order . All

the untrusted data should be validated against the web application’s logic before processing or moving

it into storage. Remove all executable portions of untrusted user provided content that could

appear in HTML pages. At the point where user controllable data is output in HTTP responses, encode

the output to prevent it from being interpreted as active content. Depending on the output context,

this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

2.

Follow a whitelist approach . Make a whitelist of allowed tags and attributes that the web

application should accept from the user. Blacklists can be easily bypassed.

3.

Use appropriate response headers . To prevent XSS in HTTP responses that aren't intended to

contain any HTML or JavaScript, you can use the Content Type and X Content Type Options headers to

ensure that browsers interpret the responses in the way you intend.

4.

Content Security Policy . As a last line of defense, you can use Content Security Policy (CSP) to

reduce the severity of any XSS vulnerabilities that still occur.

https://portswigger.net/web

security/cross site scripting

https://owasp.org/www

pdf archive//Xenotix\_XSS\_Protection\_CheatSheet\_For\_Developers.pdf  
  
1 . Validate the input and Escape untrusted data based on context and in

correct order

Examples:

•

If a user submits a URL that will be returned in responses, validating that it starts with a safe protocol such as HTTP and H TTP S.

•

If a user supplies a value that it expected to be numeric, validating that the value actually contains an integer

•

Validating that input contains only an expected set of characters.



Escape submitted user input/output so that the browser interprets it only as data, not as code



A server can remove all executable portions of untrusted user provided content that could

appear in HTML pages.

o

For example, it might look for script tags and filter them out/replace them.

o

Then, instead of running the script, the browser will print it in the document.

o

HTML escaping: converting <script>...</script> into lt;script&gt lt script&gt



Such filtering is often done in the comment sections of blogs ( e.g.Wordpress

**2.Follow a whitelist approach**

Make a whitelist of allowed tags and attributes that the web application should accept from the user. Blacklists can be easily bypassed.

Unlike XSS, which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user'sbrowser.

CSRF  
Graphical user interface, text, application

Description automatically generated

CSRFDefenses

**1.Employ anti-forgery tokens (request verification tokens)**

**2.Validate the referrer**

**3.Other defenses**

<https://www.geeksforgeeks.org/difference-between-xss-and-csrf/#:~:text=XSS%20stands%20for%20Cross%2DSite,side%20script%20in%20a%20website>.

# Bibliography

Anon., n.d. *www.imperva.com.* [Online]   
Available at: https://www.imperva.com/learn/ddos/syn-flood/