Web security

- The application developers recognized the dangers arising from flaws in web application and established a community called OWASP (Online Web Application Security Project). OWASP -a nonprofit organization- is the standards body for web application security. OWASP community includes corporations, educational organizations and individuals around the world.
- After years of testing applications for vulnerabilities, OWASP came up with ten security flaws that make a web application vulnerable.

The ten major threats against web applications are:-

- Injection
- Broken Authentication
- Sensitive data exposure
- XML External Entities (XXE)
- Broken Access control
- Security misconfigurations
- Cross Site Scripting (XSS)
- Insecure Deserialization
- Using Components with known vulnerabilities
- Insufficient logging and monitoring

- **Injection:** SQL injection allows attackers to relay malicious code through web application to other systems. This might trick the application in performing and executing process which were not instructed to be run by a valid user/admin. It can result in data loss or corruption, lack of accountability or denial of access. Injection may sometimes lead to complete host takeover.
- SQL Injection can be classified into three major categories – *In-band SQLi*, *Inferential SQLi* and *Out-of-band SQLi*.

In-band SQLi (Classic SQLi)

In-band SQL Injection is the most common and easy-to-exploit of SQL Injection attacks. In-band SQL Injection occurs when an attacker is able to use the same communication channel to both launch the attack and gather results.

The two most common types of in-band SQL Injection are *Error-based SQLi* and *Union-based SQLi*.

Error-based SQLi

Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site, or logged to a file with restricted access instead.

Union-based SQLi

Union-based SQLi is an in-band SQL injection technique that leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.

Inferential SQLi (Blind SQLi)

Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as dangerous as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the web application and the attacker would not be able to see the result of an attack in-band (which is why such attacks are commonly referred to as "blind SQL Injection attacks"). Instead, an attacker is able to reconstruct the database structure by sending payloads, observing the web application's response and the resulting behavior of the database server.

The two types of inferential SQL Injection are *Blind-boolean-based SQLi* and *Blind-time-based SQLi*.

Boolean-based (content-based) Blind SQLi

Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result.

Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database, character by character.

Time-based Blind SQLi

Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.

Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database character by character.

Out-of-band SQLi

Out-of-band SQL Injection is not very common, mostly because it depends on features being enabled on the database server being used by the web application. Out-of-band SQL Injection occurs when an attacker is unable to use the same channel to launch the attack and gather results.

Out-of-band techniques, offer an attacker an alternative to inferential time-based techniques, especially if the server responses are not very stable (making an inferential time-based attack unreliable).

Out-of-band SQLi techniques would rely on the database server's ability to make DNS or HTTP requests to deliver data to an attacker. Such is the case with Microsoft SQL Server's `xp_dirtree` command, which can be used to make DNS requests to a server an attacker controls; as well as Oracle Database's UTL_HTTP package, which can be used to send HTTP requests from SQL and PL/SQL to a server an attacker controls.

SQL Injection

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

WHAT IS SQL INJECTION

SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.

The impact SQL injection can have on a business is far reaching. A successful attack may result in the unauthorized viewing of user lists, the deletion of entire tables and, in certain cases, the attacker gaining administrative rights to a database, all of which are highly detrimental to a business.

When calculating the potential cost of a SQLI, it's important to consider the loss of customer trust should personal information such as phone numbers, addresses and credit card details be stolen.

While this vector can be used to attack any SQL database, websites are the most frequent targets.

SQL in Web Pages

- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.
- Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):
- Example
- txtUserId = getRequestString("UserId");
  SQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

SQL Injection Based on 1=1 is Always True

•

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: 105 OR 1=1

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

Example

```
SELECT * FROM Users; DROP TABLE Suppliers
```

Look at the following example:

Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

And the following input:

User id: 105; DROP TABLE Suppliers

The valid SQL statement would look like this:

- Result:
- SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

As a result, the entire supplier database could be deleted.

Another way SQL queries can be manipulated is with a UNION SELECT statement. This combines two unrelated SELECT queries to retrieve data from different database tables.

For example, the input http://www.estore.com/items/items.asp?itemid=999 UNION SELECT user-name, password FROM USERS produces the following SQL query:

```
SELECT ItemName, ItemDescription
FROM Items
WHERE ItemID = '999' UNION SELECT Username, Password FROM Users;
```

Using the UNION SELECT statement, this query combines the request for item 999's name and description with another that pulls names and passwords for every user in the database.

Use SQL Parameters for Protection

There are a lot of ways to defend SQL injection. One of the primary defense techniques is “Prepared Statements (Parameterized Queries)”. This technique force the developer to define all the SQL code and then pass in each parameter to the query later. This style allows the database to diffrentiate between code and data, regardless of what user input is supplied.

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. For example, if an attacker enter the userID of ABC or ‘1’=’1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string ABC or ‘1’=’1.

Working:

1. **Prepare:** An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled “?”).
   Example:
   ```
   SELECT count(*)FROM users WHERE username = ? AND password = ?;
   ```
2. **Parse:** The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it.
3. **Execute:** At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values.

- The only proven way to protect a web site from SQL injection attacks, is to use SQL parameters.
- SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

ASP.NET Razor Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);
```

Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

Another Example

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

2. Using Stored Procedures

Stored Procedures adds an extra security layer to your database beside using Prepared Statements. It performs the escaping required so that the app treats input as data to be operated on rather than SQL code to be executed.

The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is written and stored in the database server, and then called from the web app.

If user access to the database is only ever permitted via stored procedures, permission for users to directly access data doesn't need to be explicitly granted on any database table. This way, your database is still safe.

3. Validating user input

Even when you are using Prepared Statements, you should do an input validation first to make sure the value is of the accepted type, length, format, etc. Only the input which passed the validation can be processed to the database. It's like checking who is at the door of your house before you open it and let them in.

But remember, this method can only stop the most trivial attacks, it does not fix the underlying vulnerability.

4. Limiting privileges

Don't connect to your database using an account with root access unless required because the attackers might have access to the entire system. Therefore, it's best to use an account with limited privileges to limit the scope of damages in case of SQL Injection.

5. Hidding info from the error message

Error messages are useful for attackers to learn more about your database architecture, so be sure that you show only the necessary information. It's better to show a generic error message telling something goes wrong and encourage users to contact the technical support team in case the problem persists.

6. Updating your system

SQL injection vulnerability is a frequent programming error and it's discovered regularly, so it's vital to apply patches and updates your system to the most up-to-date version as you can, especially for your SQL Server.

7. Keeping database credentials separate and encrypted

If you are considering where to store your database credentials, also consider how much damaging it can be if it falls into the wrong hands. So always store your database credentials in a separate file and encrypt it securely to make sure that the attackers can't benefit much.

Also, don't store sensitive data if you don't need it and delete information when it's no longer in use.

8. Disabling shell and any other functionalities you don't need

Shell access could be very useful indeed for a hacker. That's why you should turn it off if possible. Remove or disable all functionalities that you don't need too.

Final thought

The key to avoiding being the victim of the next SQL Injection Attack is always be cautious and trust nobody. You don't know when the bad guy is coming so hope for the best and prepare for the worst, validate and sanitize all user interactions.

2. Broken authentication and session management: Broken authentication and session management allows attackers to gain details like password, tokens and session IDs which then enable them to login to that users account and impersonate them to carry out transactions. **Accounts may be hijacked by attackers by using the active session IDs exposed in URLs**.To avoid such attacks user credentials must be passed through encrypted connections and stored using concepts of hashing and encryptions. Timeout of session IDs and authentication must be carried out in order to discourage such threats. Re-authentication in case of forgot password must be done to verify the identity of the user.

3. Cross Site Scripting (XSS): This occurs when a user enters data without input user validation. and this **untrusted data is sent to the application without carrying out proper validation (browser side scripting)**. Next time when other user visits the site or runs the same application, he witnesses the application behaves in a manner in which the attacker wanted it to behave. The malicious script can access cookies, session tokens or any other sensitive information retained by the browser and used with that site. These scripts can also rewrite the content of the HTML page.

Reflected css/Cross Site Request Forgery (CSRF): Also known as one click attack or session riding. This flaw allows the attacker to force an end user to **execute unwanted actions on a web application in which they are currently authenticated** and send a forged HTTP request along with user authentication details to a vulnerable website. It actually results in an undesired function on the victim's behalf.*Example*: Most of the suspicious links you receive via suspicious or unidentified sources through mails.

- Other ways to protect your site include but are not limited to the following:
- a) utilise built in controls such as Anti-Forgery Tokens, HTTPOnly Cookies, Access-Control-Allow-Origin Headers
- b) request users to re-authenticate when performing important actions (e.g. authorising a payment to another bank account)

CROSS SITE SCRIPTING: (XSS)

WHAT IS CROSS SITE SCRIPTING (XSS)

Cross site scripting (XSS) is a common attack vector that injects malicious code into a vulnerable web application. XSS differs from other web attack vectors (e.g., SQL injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.

A successful cross site scripting attack can have devastating consequences for an online business's reputation and its relationship with its clients.

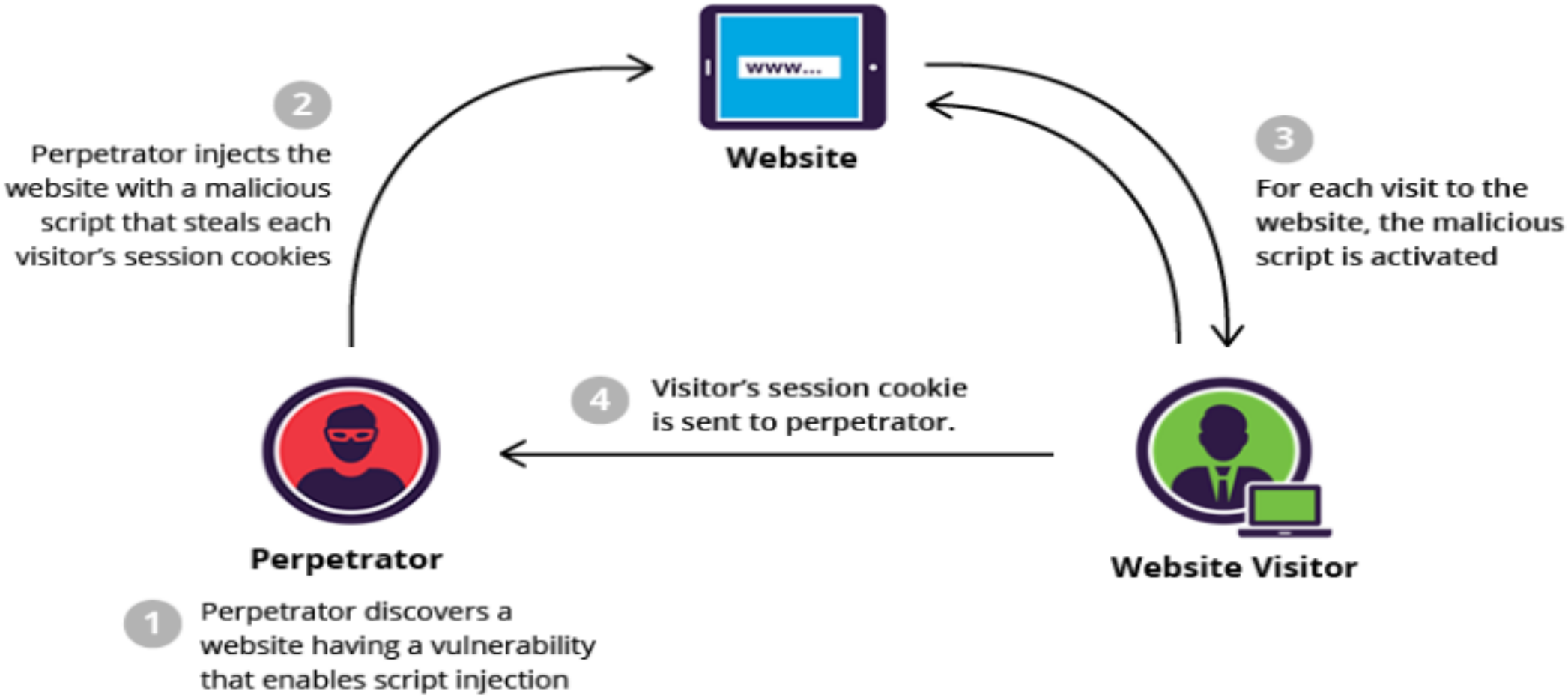
Depending on the severity of the attack, user accounts may be compromised, Trojan horse programs activated and page content modified, misleading users into willingly surrendering their private data. Finally, session cookies could be revealed, enabling a perpetrator to impersonate valid users and abuse their private accounts.

Cross site scripting attacks can be broken down into two types: stored and reflected.

Stored XSS, also known as persistent XSS, is the more damaging of the two. It occurs when a malicious script is injected directly into a vulnerable web application.

Reflected XSS involves the reflecting of a malicious script off of a web application, onto a user's browser. The script is embedded into a link, and is only activated once that link is clicked on.

To successfully execute a stored XSS attack, a perpetrator has to locate a vulnerability in a web application and then inject malicious script into its server (e.g., via a comment field).



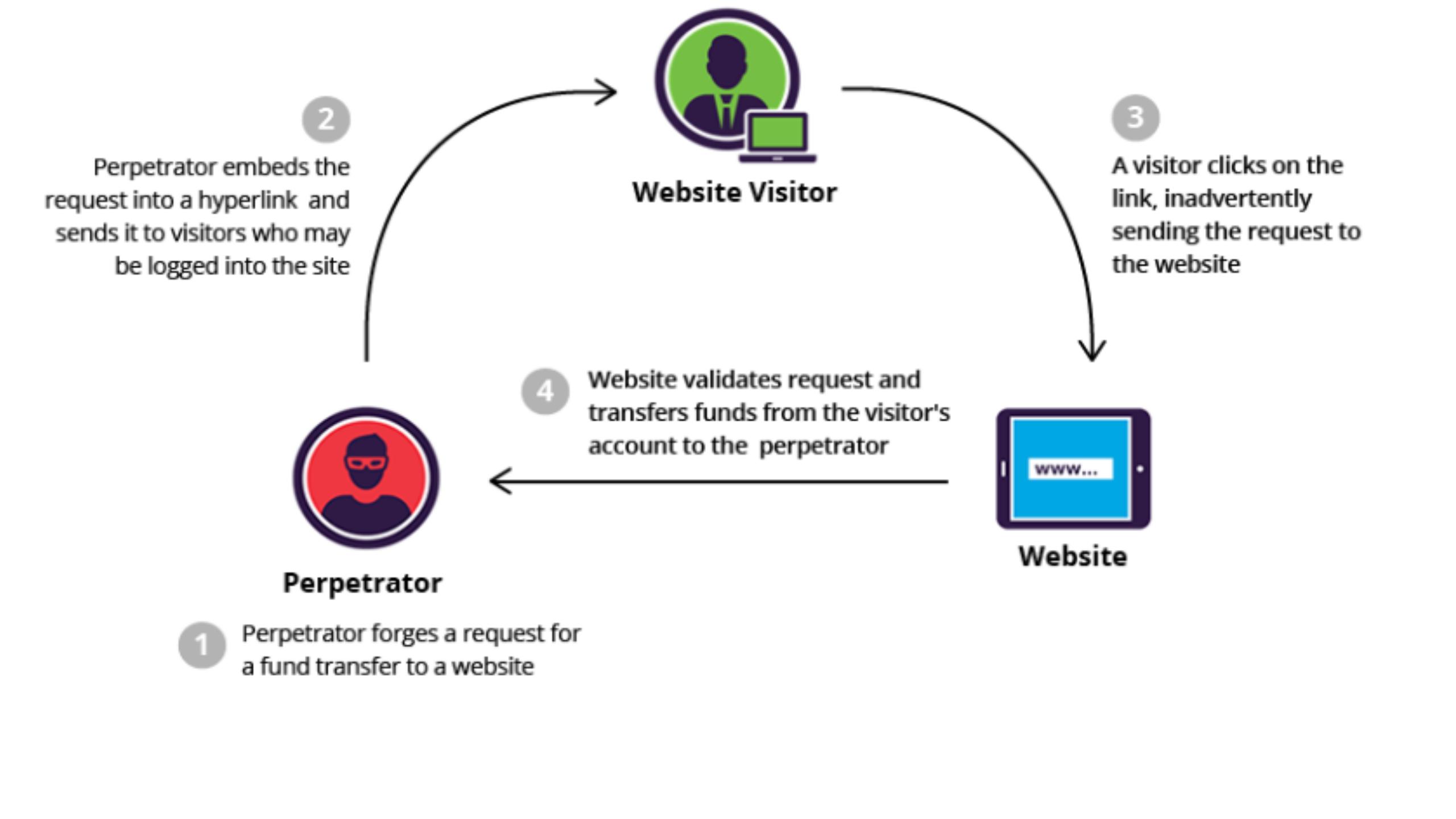
- Using the session cookie, the attacker can compromise the visitor's account, granting him easy access to his personal information and credit card data. Meanwhile, the visitor, who may never have even scrolled down to the comments section, is not aware that the attack took place.
- Unlike a reflected attack, where the script is activated after a link is clicked, a stored attack only requires that the victim visit the compromised web page. This increases the reach of the attack, endangering all visitors no matter their level of vigilance.

One of the most frequent targets are websites that allow users to share content, including blogs, social networks, video sharing platforms and message boards. Every time the infected page is viewed, the malicious script is transmitted to the victim’s browser.

STORED XSS ATTACK PREVENTION/MITIGATION

- A web application firewall (WAF) is the most commonly used solution for protection from XSS and web application attacks.
- WAFs employ different methods to counter attack vectors. In the case of XSS, most will rely on signature based filtering to identify and block malicious requests.

Eg of reflected xss


Website Visitor
2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

CSRF EXAMPLE

Before executing an assault, a perpetrator typically studies an application in order to make a forged request appear as legitimate as possible.

For example, a typical GET request for a $100 bank transfer might look like:

```
GET http://netbank.com/transfer.do?acct=PersonB&amount=$100 HTTP/1.1
```

A hacker can modify this script so it results in a $100 transfer to their own account. Now the malicious request might look like:

```
GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100 HTTP/1.1
```

A bad actor can embed the request into an innocent looking hyperlink:

```
<a href="http://netbank.com/transfer.do?acct=AttackerA&amount=$100">Read more!</a>
```

Next, he can distribute the hyperlink via email to a large number of bank customers. Those who click on the link while logged into their bank account will unintentionally initiate the $100 transfer.

Note that if the bank's website is only using POST requests, it's impossible to frame malicious requests using a <a> href tag. However, the attack could be delivered in a <form> tag with automatic execution of the embedded JavaScript.

This is how such a form may look like:

```
<body onload="document.forms[0].submit()">
 <form action="http://netbank.com/transfer.do" method="POST">
  <input type="hidden" name="acct" value="AttackerA"/>
  <input type="hidden" name="amount" value="$100"/>
  <input type="submit" value="View my pictures!"/>
 </form>
</body>
```

METHODS OF CSRF MITIGATION

- Best practices include:
- Logging off web applications when not in use
- Securing usernames and passwords
- Not allowing browsers to remember passwords
- Avoiding simultaneously browsing while logged into an application

4. Insecure direct object reference: This occurs when a reference (object, file or database) is **exposed to be viewed/ modified/used by a user who is not verified for access (authorization).** These flaws enable attackers to compromise all the data linked to the modified parameter. Once an attacker finds a way into the application, most likely he'll will figure a way to dig deeper and compromise any data possible.
Example: Access given to all users when only admin should have the access right to view/use/modify data.

- This type of attack allows malicious users to access otherwise unauthorised data by manipulating known information such as a URL parameter. This attack is possible because there are inadequate security guards in place to ensure that only authorised users should have access to specific pieces of information. For example, just because my bank account URL is https://mysecurebank/account?id=12345 I shouldn't be able to see someone else's account by changing the Id parameter value. And yet, what OWASP is telling us is that this type of attack is still possible! To protect your resource, you should implement thorough Access Control and even go as far as introducing Indirect Reference Maps to hide away real-life keys.

- **Insecure Deserialization**
- This threat targets the many web applications which frequently serialize and deserialize data. Serialization means taking objects from the application code and converting them into a format that can be used for another purpose, such as storing the data to disk or streaming it. Deserialization is just the opposite: converting serialized data back into objects the application can use. Serialization is sort of like packing furniture away into boxes before a move, and deserialization is like unpacking the boxes and assembling the furniture after the move. An insecure deserialization attack is like having the movers tamper with the contents of the boxes before they are unpacked.
- An insecure deserialization exploit is the result of deserializing data from untrusted sources, and can result in serious consequences like DDoS attacks and remote code execution attacks. While steps can be taken to try and catch attackers, such as monitoring deserialization and implementing type checks, the only sure way to protect against insecure deserialization attacks is to prohibit the deserialization of data from untrusted sources.

5. Security misconfiguration: The application, application server, web server, database server should all have secure configurations. Developers *and* system administrators need to work together to ensure that the *entire* stack is configured properly. If a system is compromised through faulty security configurations, data can be stolen or modified slowly over time.

- For instance, an application could show a user overly-descriptive errors which may reveal vulnerabilities in the application. This can be mitigated by removing any unused features in the code and ensuring that error messages are more general.

- Some examples of security misconfiguration that would allow an attacker to mount an attack:
- a) Insufficiently secured web server logs
- b) exposing internal code through improperly handled exceptions
- c) exposing server/platform information through http headers
- This information can be used by an attacker to penetrate the system using known vulnerabilities or data exposed through the attack. Due to the fact that this attack is so broad, it's hard to provide mitigation advice but I've included some suggestions below:
- a) ensure all OS and software is up-to-date
- b) adopt Secure Development Lifecycle principles
- c) make security part of your Change Management process
- d) take advantage of Penetration Testing

6. Sensitive data exposure: Sensitive data like credit card details may not be protected in the best possible manner. Attackers might break into such databases and carry out transactions using these data. This includes data and backup of that data. Attackers cannot break the encryption directly, they steal keys, cause middle man attacks or steal clean text data from server or browser. The only way to protecting such sensitive data is by using strong encryption algorithms.

- There are a few things you can do in order to mitigate against these risks:
- a) encryption at rest and in transit
- b) use SSL throughout your site
- c) use strong encryption algorithms
- d) implement a secure encryption key management

7.Broken access control/ Missing function level access control: Function level authorization must be present on both application and on servers. Most of the web applications verify function level access rights before making that functionality accessible to the user. However, if the access control checks are not performed on the server, hackers will be able to penetrate into the application without proper authorization. **Requests need to be verified on both ends to hinder the forgery of requests to access functionality without valid authorization.***Example*: Anonymous users access private functionality or regular users may use a privileged function if function level access control is not implemented.

- For example a web application could allow a user to change which account they are logged in as simply by changing part of a url, without any other verification.
- Access controls can be secured by ensuring that a web application uses authorization tokens and sets tight controls on them.
- Many services issue authorization tokens when users log in. Every privileged request that a user makes will require that the authorization token be present. This is a secure way to ensure that the user is who they say they are, without having to constantly enter their login credentials.

- So this risk is about the absence of proper authorisation which could enable attackers to utilise code that they should not be able to execute. This is fairly wide security risk and there are many ways to protect against it such as:
- a) deny access by default and only open up access in a piecemeal fashion.
- b) do not write your own security features
- c) test your site using an unauthorised user

8.

XML External Entities (XXE)

Often, applications need to receive and process XML documents from users. Old or poorly configured XML parsers can enable an XML feature known as external entity references within XML documents, which **when evaluated will embed the contents of another file**. Attackers can abuse this to **read confidential data, access internal systems, and even shut down the application in a Denial of Service (DoS) attack**.

For example, an XML document containing this:

<!ENTITY xxe SYSTEM “file:///etc/passwd”>]>&xxe;

would include the contents of the password file within the XML document.

This can be prevented by simply disabling DTD and External entity evaluation in the parser, or upgrading to a modern parser library that is not vulnerable.

- The best ways to prevent XEE attacks are to have web applications accept a less complex type of data, such as JSON**, or at the very least to patch XML parsers and disable the use of external entities in an XML application.
- **JavaScript Object Notation (JSON) is a type of simple, human-readable notation often used to transmit data over the internet. Although it was originally created for JavaScript, JSON is language-agnostic and can be interpreted by many different programming languages.

9. Using components with known vulnerabilities: Some tools that are used to develop an application may have security flaws, **which are already a known fact and common knowledge to attackers.** Implementing such a tool is not advisable as it might in-turn result in giving rise to most of the flaws mentioned above. The attacker knows the weakness of one component so he might have knowledge of what other specific areas of application are vulnerable because of this flaw.Virtually every application has these issues because most development teams don't focus on ensuring their components/ libraries are up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse.

- So how do you mitigate against this threat
- a) identify and catalogue all components being utilised within an application stack.
- b) check for updates and common vulnerability and exposures (CVEs) to the components on a regular basis
- c) automate integration testing of these components
- d) patch components

- 10. insufficient logging and monitoring:
- While 100% security is not a realistic goal, there are ways to keep your website monitored on a regular basis so you can take immediate action when something happens.
- Not having an efficient logging and monitoring process in place can increase the chances of a website compromise.
- An audit log is a document that records the events in a website so you can spot anomalies and confirm with the person in charge that the account hasn't been compromised.
- We know that it may be hard for some users to perform audit logs manually. If you have a WordPress website, you can use our free Security Plugin which can be downloaded from the official WordPress repository.

Hackers methodology

- Fingerprinting/ reconnaissance
- Scanning
- Gaining access
- Maintaining access
- Clearing the tracks.

- Reconnaisance:This is the first step of Hacking. It is also called as Footprinting and information gathering Phase. This is the preparatory phase where we collect as much information as possible about the target. We usually collect information about three groups,
- Network
- Host
- People involved
- There are two types of Footprinting:
- **Active:** Directly interacting with the target to gather information about the target. Eg Using Nmap tool to scan the target
- **Passive:** Trying to collect the information about the target without directly accessing the target. This involves collecting information from social media, public websites etc.

- Scanning:
- Three types of scanning are involved:
- **Port scanning:** This phase involves scanning the target for the information like open ports, Live systems, various services running on the host.
- **Vulnerability Scanning:** Checking the target for weaknesses or vulnerabilities which can be exploited. Usually done with help of automated tools
- **Network Mapping:** Finding the topology of network, routers, firewalls servers if any, and host information and drawing a network diagram with the available information. This map may serve as a valuable piece of information throughout the hacking process.

- 3. Gaining Access:
- This phase is where an attacker breaks into the system/network using various tools or methods. After entering into a system, he has to increase his privilege to administrator level so he can install an application he needs or modify data or hide data.
- 4. Maintaining Access:
- Hacker may just hack the system to show it was vulnerable or he can be so mischievous that he wants to maintain or persist the connection in the background without the knowledge of the user. This can be done using Trojans, Rootkits or other malicious files. The aim is to maintain the access to the target until he finishes the tasks he planned to accomplish in that target.

- 5. Clearing Track:
- No thief wants to get caught. An intelligent hacker always clears all evidence so that in the later point of time, no one will find any traces leading to him. This involves modifying/ corrupting/deleting the values of Logs, modifying registry values and uninstalling all applications he used and deleting all folders he created.

Types of hackers

- Black hat
- White hat
- Grey hat

Web Application Hacking Methodology

- Web application hacking is not just about using automated tools to find common vulnerabilities. It is indeed a methodological approach that, if followed, would help reveal many more flaws and potential security vulnerabilities. The following section describes the systematic approach and process to be followed for testing the security of web applications.

1. Analyzing web applications

The first step is to understand and analyze the target application. Unless and until sufficient details about the target application are known, one cannot proceed with further testing. Some of the information that needs to be gathered is:

- What is the purpose of the target application?
- Who is the audience of the target application?
- How critical is the application from a business perspective?
- What technology platform has been used to develop the application?
- What are the important workflows in the target application?

2. Identifying the entry and exit points

The next step is to identify entry and exit points. This gives an idea of how an attacker might try to intrude into the application. The entry point may be a login screen or a registration form.

- **3. Breaking down the components**
- This step involves breaking down application components. It’s vital to know what components are used within the application, whether it involves an additional application server and/or database server.

- **4. Testing manually for vulnerabilities**
- Using tools like BurpSuite, Paros, ZAP, and others, manual security testing can be performed. This mainly involves intercepting potential HTTP requests, modifying and tampering with the parameter values, and then analyzing the application's response.

- **5. Automated security scanning**
- Tools like IBM AppScan, Fortify, and Acunetix are some of the commercial tools available for automated web application security testing. They perform a comprehensive scan on input parameters across the application and check for various vulnerabilities.

- **6. Removing false positives**
- Automated scanning tools may produce false positives. Hence it is important to manually verify any vulnerability found during scanning and remove false positives if any.

- **7. Reporting with remediation**
- A certain security issue might appear as merely a missing functionality to a developer. Hence, it is critical to prepare a vulnerability report with all necessary artifacts and proof-of-concepts in order to make the developer community understand the severity of the vulnerabilities identified. It is also necessary to suggest a fix recommendation for any identified vulnerability.

- THE DIFFERENT WEB HACKING TOOLS
- Nmap
- Openssh
- Wireshark
- Nessus
- Kali linux
- Angry ip scanner
- Cain and abel
- Ettercap
- Burp suite
- Metasploit
- Etc