1. **SQL Injection**

**Description**

Injection is a security vulnerability that allows an attacker to alter backend[SQL](https://www.guru99.com/sql.html)statements by manipulating the user supplied data.

Injection occurs when the user input is sent to an interpreter as part of command or query and trick the interpreter into executing unintended commands and gives access to unauthorized data.

The SQL command which when executed by web application can also expose the back-end database.

**Implication**

* An attacker can inject malicious content into the vulnerable fields.
* Sensitive data like User Names, Passwords, etc. can be read from the database.
* Database data can be modified (Insert/Update/ Delete).
* Administration Operations can be executed on the database

**Vulnerable Objects**

* Input Fields
* URLs interacting with the database.

**Examples:**

* SQL injection on the Login Page

Logging into an application without having valid credentials.

Valid userName is available, and password is not available.

Test URL: **http://demo.testfire.net/default.aspx**

User Name: sjones

Password: 1=1' or pass123

SQL query created and sent to Interpreter as below

SELECT \* FROM Users WHERE User\_Name = sjones AND Password = 1=1' or pass123;

**Recommendations**

1. White listing the input fields
2. Avoid displaying detailed error messages that are useful to an attacker.

## Cross Site Scripting

**Description**

Cross Site Scripting is also shortly known as XSS.

XSS vulnerabilities target scripts embedded in a page that are executed on the client side i.e. user browser rather then at the server side. These flaws can occur when the application takes untrusted data and send it to the web browser without proper validation.

Attackers can use XSS to execute malicious scripts on the users in this case victim browsers. Since the browser cannot know if the script is trusty or not, the script will be executed, and the attacker can hijack session cookies, deface websites, or redirect the user to an unwanted and malicious websites.

XSS is an attack which allows the attacker to execute the scripts on the victim's browser.

**Implication:**

* Making the use of this security vulnerability, an attacker can inject scripts into the application, can steal session cookies, deface websites, and can run malware on the victim's machines.

**Vulnerable Objects**

* Input Fields
* URLs

**Examples**

1. **http://www.vulnerablesite.com/home?"<script>alert("xss")</script**>

The above script when run on a browser, a message box will be displayed if the site is vulnerable to XSS.

The more serious attack can be done if the attacker wants to display or store session cookie.

2. **http://demo.testfire.net/search.aspx?txtSearch <iframe**> <src = **http://google.com** width = 500 height 500></iframe>

The above script when run, the browser will load an invisible frame pointing to **http://google.com**.

The attack can be made serious by running a malicious script on the browser.

**Recommendations**

1. White Listing input fields
2. Input Output encoding

## Broken Authentication and Session Management

**Description**

The websites usually create a session cookie and session ID for each valid session, and these cookies contain sensitive data like username, password, etc. When the session is ended either by logout or browser closed abruptly, these cookies should be invalidated i.e. for each session there should be a new cookie.

If the cookies are not invalidated, the sensitive data will exist in the system. For example, a user using a public computer (Cyber Cafe), the cookies of the vulnerable site sits on the system and exposed to an attacker. An attacker uses the same public computer after some time, the sensitive data is compromised.

In the same manner, a user using a public computer, instead of logging off, he closes the browser abruptly. An attacker uses the same system, when browses the same vulnerable site, the previous session of the victim will be opened. The attacker can do whatever he wants to do from stealing profile information, credit card information, etc.

A check should be done to find the strength of the authentication and session management. Keys, session tokens, cookies should be implemented properly without compromising passwords.

**Vulnerable Objects**

* Session IDs exposed on URL can lead to session fixation attack.
* Session IDs same before and after logout and login.
* Session Timeouts are not implemented correctly.
* Application is assigning same session ID for each new session.
* Authenticated parts of the application are protected using SSL and passwords are stored in hashed or encrypted format.
* The session can be reused by a low privileged user.

**Implication**

* Making use of this vulnerability, an attacker can hijack a session, gain unauthorized access to the system which allows disclosure and modification of unauthorized information.
* The sessions can be high jacked using stolen cookies or sessions using XSS.

**Examples**

1. Airline reservation application supports URL rewriting, putting session IDs in the URL:

**http://Examples.com/sale/saleitems;jsessionid=2P0OC2oJM0DPXSNQPLME34SERTBG/dest=Maldives** (Sale of tickets to Maldives)

An authenticated user of the site wants to let his friends know about the sale and sends an email across. The friends receive the session ID and can be used to do unauthorized modifications or misuse the saved credit card details.

1. An application is vulnerable to XSS, by which an attacker can access the session ID and can be used to hijack the session.
2. Applications timeouts are not set properly. The user uses a public computer and closes the browser instead of logging off and walks away. The attacker uses the same browser some time later, and the session is authenticated.

**Recommendations**

1. All the authentication and session management requirements should be defined as per OWASP Application Security Verification Standard.
2. Never expose any credentials in URLs or Logs.
3. Strong efforts should be also made to avoid XSS flaws which can be used to steal session IDs.

## Insecure Direct Object References

**Description**

It occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key as in URL or as a FORM parameter. The attacker can use this information to access other objects and can create a future attack to access the unauthorized data.

**Implication**

* Using this vulnerability, an attacker can gain access to unauthorized internal objects, can modify data or compromise the application.

**Vulnerable Objects**

* In the URL.

**Examples:**

Changing "userid" in the following URL can make an attacker to view other user's information.

**http://www.vulnerablesite.com/userid=123** Modified to **http://www.vulnerablesite.com/userid=124**

An attacker can view others information by changing user id value.

**Recommendations:**

1. Implement access control checks.
2. Avoid exposing object references in URLs.
3. Verify authorization to all reference objects.

## 5. Cross Site Request Forgery

**Description**

Cross Site Request Forgery is a forged request came from the cross site.

CSRF attack is an attack that occurs when a malicious website, email, or program causes a user's browser to perform an unwanted action on a trusted site for which the user is currently authenticated.

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.

A link will be sent by the attacker to the victim when the user clicks on the URL when logged into the original website, the data will be stolen from the website.

**Implication**

* Using this vulnerability as an attacker can change user profile information, change status, create a new user on admin behalf, etc.

**Vulnerable Objects**

* User Profile page
* User account forms
* Business transaction page

**Examples**

The victim is logged into a bank website using valid credentials. He receives mail from an attacker saying "Please click here to donate $1 to cause."

When the victim clicks on it, a valid request will be created to donate $1 to a particular account.

**http://www.vulnerablebank.com/transfer.do?account=cause&amount=1**

The attacker captures this request and creates below request and embeds in a button saying "I Support Cause."

**http://www.vulnerablebank.com/transfer.do?account=Attacker&amount=1000**

Since the session is authenticated and the request is coming through the bank website, the server would transfer $1000 dollars to the attacker.

**Recommendation**

1. Mandate user's presence while performing sensitive actions.
2. Implement mechanisms like CAPTCHA, Re-Authentication, and Unique Request Tokens.