**10 Most Common Web Security Vulnerabilities**

OWASP or Open Web Security Project is a non-profit charitable organization focused on improving the security of software and web applications.

The organization publishes a list of top web security vulnerabilities based on the data from various security organizations.

The web security vulnerabilities are prioritized depending on exploitability, detectability and impact on software.

* **Exploitability –**

What is needed to exploit the security vulnerability? Highest exploitability when the attack needs only web browser and lowest being advanced programming and tools.

* **Detectability –**

How easy is it to detect the threat? Highest being the information displayed on URL, Form or Error message and lowest being source code.

* **Impact or Damage –**

How much damage will be done if the security vulnerability is exposed or attacked? Highest being complete system crash and lowest being nothing at all.

The main aim of OWASP Top 10 is to educate the developers, designers, managers, architects and organizations about the most important security vulnerabilities.

**The Top 10 security vulnerabilities as per OWASP Top 10 are:**

* [SQL Injection](https://www.guru99.com/web-security-vulnerabilities.html#1)
* [Cross Site Scripting](https://www.guru99.com/web-security-vulnerabilities.html#2)
* [Broken Authentication and Session Management](https://www.guru99.com/web-security-vulnerabilities.html#3)
* [Insecure Direct Object References](https://www.guru99.com/web-security-vulnerabilities.html#4)
* [Cross Site Request Forgery](https://www.guru99.com/web-security-vulnerabilities.html#5)
* [Security Misconfiguration](https://www.guru99.com/web-security-vulnerabilities.html#6)
* [Insecure Cryptographic Storage](https://www.guru99.com/web-security-vulnerabilities.html#7)
* [Failure to restrict URL Access](https://www.guru99.com/web-security-vulnerabilities.html#8)
* [Insufficient Transport Layer Protection](https://www.guru99.com/web-security-vulnerabilities.html#9)
* [Unvalidated Redirects and Forwards](https://www.guru99.com/web-security-vulnerabilities.html#10)

**SQL Injection**

[](https://www.guru99.com/images/cassandra/021216_0701_10MostCommo1.jpg)

**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.

**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.

**Security Misconfiguration**

**Description**

Security Configuration must be defined and deployed for the application, frameworks, application server, web server, database server, and platform. If these are properly configured, an attacker can have unauthorized access to sensitive data or functionality.

Sometimes such flaws result in complete system compromise. Keeping the software up to date is also good security.

**Implication**

* Making use of this vulnerability, the attacker can enumerate the underlying technology and application server version information, database information and gain information about the application to mount few more attacks.

**Vulnerable objects**

* URL
* Form Fields
* Input fields

**Examples**

1. The application server admin console is automatically installed and not removed. Default accounts are not changed. The attacker can log in with default passwords and can gain unauthorized access.
2. Directory Listing is not disabled on your server. Attacker discovers and can simply list directories to find any file.

**Recommendations**

1. A strong application architecture that provides good separation and security between the components.
2. Change default usernames and passwords.
3. Disable directory listings and implement access control checks.

**Insecure Cryptographic Storage**

**Description**

Insecure Cryptographic storage is a common vulnerability which exists when the sensitive data is not stored securely.

The user credentials, profile information, health details, credit card information, etc. come under sensitive data information on a website.

This data will be stored on the application database. When this data are stored improperly by not using encryption or hashing\*, it will be vulnerable to the attackers.

(\*Hashing is transformation of the string characters into shorter strings of fixed length or a key. To decrypt the string, the algorithm used to form the key should be available)

**Implication**

* By using this vulnerability, an attacker can steal, modify such weakly protected data to conduct identity theft, credit card fraud or other crimes.

**Vulnerable objects**

* Application database.

**Examples**

In one of the banking application, password database uses unsalted hashes \* to store everyone's passwords. An SQL injection flaw allows the attacker to retrieve the password file. All the unsalted hashes can be brute forced in no time whereas, the salted passwords would take thousands of years.

(\*Unsalted Hashes – Salt is a random data appended to the original data. Salt is appended to the password before hashing)

**Recommendations**

1. Ensure appropriate strong standard algorithms. Do not create own cryptographic algorithms. Use only approved public algorithms such as AES, RSA public key cryptography, and SHA-256, etc.
2. Ensure offsite backups are encrypted, but the keys are managed and backed up separately.

**Failure to restrict URL Access**

**Description**

Web applications check URL access rights before rendering protected links and buttons. Applications need to perform similar access control checks each time these pages are accessed.

In most of the applications, the privileged pages, locations and resources are not presented to the privileged users.

By an intelligent guess, an attacker can access privilege pages. An attacker can access sensitive pages, invoke functions and view confidential information.

**Implication**

* Making use of this vulnerability attacker can gain access to the unauthorized URLs, without logging into the application and exploit the vulnerability. An attacker can access sensitive pages, invoke functions and view confidential information.

**Vulnerable objects:**

* URLs

**Examples**

1. Attacker notices the URL indicates the role as "/user/getaccounts." He modifies as "/admin/getaccounts".
2. An attacker can append role to the URL.

**http://www.vulnerablsite.com** can be modified as **http://www.vulnerablesite.com/admin**

**Recommendations**

1. Implement strong access control checks.
2. Authentication and authorization policies should be role-based.
3. Restrict access to unwanted URLs.

**Insufficient Transport Layer Protection**

**Description**

Deals with information exchange between the user (client) and the server (application). Applications frequently transmit sensitive information like authentication details, credit card information, and session tokens over a network.

By using weak algorithms or using expired or invalid certificates or not using SSL can allow the communication to be exposed to untrusted users, which may compromise a web application and or steal sensitive information.

**Implication**

* Making use of this web security vulnerability, an attacker can sniff legitimate user's credentials and gaining access to the application.
* Can steal credit card information.

**Vulnerable objects**

* Data sent over the network.

**Recommendations**

1. Enable secure HTTP and enforce credential transfer over HTTPS only.
2. Ensure your certificate is valid and not expired.

**Examples:**

1. An application not using SSL, an attacker will simply monitor network traffic and observes an authenticated victim session cookie. An attacker can steal that cookie and perform Man-in-the-Middle attack.

**Unvalidated Redirects and Forwards**

**Description**

The web application uses few methods to redirect and forward users to other pages for an intended purpose.

If there is no proper validation while redirecting to other pages, attackers can make use of this and can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

**Implication**

* An attacker can send a URL to the user that contains a genuine URL appended with encoded malicious URL. A user by just seeing the genuine part of the attacker sent URL can browse it and may become a victim.

**Examples**

1.**http://www.vulnerablesite.com/login.aspx?redirectURL=ownsite.com**

Modified to

**http://www.vulnerablesite.com/login.aspx?redirectURL=evilsite.com**

**Recommendations**

1. Simply avoid using redirects and forwards in the application. If used, do not involve using user parameters in calculating the destination.
2. If the destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user.

***This article is contributed by Prasanthi Eati***