

Task- 4

Top 10 Web Application Attacks

The OWASP Top 10 is a regularly-updated report outlining security concerns for web application security, focusing on the 10 most critical risks. The report is put together by a team of security experts from all over the world. OWASP refers to the Top 10 as an 'awareness document' and they recommend that all companies incorporate the report into their processes in order to minimize and/or mitigate security risks.

Below are the security risks reported in the OWASP Top 10 2017 report:

1. Injection

Injection attacks happen when untrusted data is sent to a code interpreter through a form input or some other data submission to a web application. For example, an attacker could enter SQL database code into a form that expects a plaintext username. If that form input is not properly secured, this would result in that SQL code being executed. This is known as an [SQL injection attack](#).

Injection attacks can be prevented by validating and/or sanitizing user-submitted data. (Validation means rejecting suspicious-looking data, while sanitization refers to cleaning up the suspicious-looking parts of the data.) In addition, a database admin can set controls to minimize the amount of information an injection attack can expose.

2. Broken Authentication

Vulnerabilities in authentication (login) systems can give attackers access to user accounts and even the ability to compromise an entire system using an admin account. For example, an attacker can take a list containing thousands of known username/password combinations obtained during a [data breach](#) and use a script to try all those combinations on a login system to see if there are any that work.

Some strategies to mitigate authentication vulnerabilities are requiring [two-factor authentication \(2FA\)](#) as well as limiting or delaying repeated login attempts using [rate limiting](#).

3. Sensitive Data Exposure

If web applications don't protect sensitive data such as financial information and passwords, attackers can gain access to that data and sell or utilize it for nefarious purposes. One popular method for stealing sensitive information is using an [on-path attack](#).

Data exposure risk can be minimized by [encrypting](#) all sensitive data as well as disabling the [caching](#)* of any sensitive information. Additionally, web application developers should take care to ensure that they are not unnecessarily storing any sensitive data.

4. XML External Entities (XEE)

This is an attack against a web application that parses XML* input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An 'external entity' in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker.

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.

5. Broken Access Control

[Access control](#) refers a system that controls access to information or functionality. Broken access controls allow attackers to bypass authorization and perform tasks as though they were privileged users such as administrators. 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.

6. Security Misconfiguration

Security misconfiguration is the most common vulnerability on the list, and is often the result of using default configurations or displaying excessively verbose errors. 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.

7. Cross-Site Scripting

[Cross-site scripting](#) vulnerabilities occur when web applications allow users to add custom code into a url path or onto a website that will be seen by other users. This vulnerability can be exploited to run malicious JavaScript code on a victim's browser. For example, an attacker could send an email to a victim that appears to be from a trusted bank, with a link to that bank's website. This link could have some malicious JavaScript code tagged onto the end of the url. If the bank's site is not properly protected against cross-site scripting, then that malicious code will be run in the victim's web browser when they click on the link.

Mitigation strategies for cross-site scripting include escaping untrusted [HTTP](#) requests as well as validating and/or sanitizing user-generated content. Using modern web development frameworks like ReactJS and Ruby on Rails also provides some built-in cross-site scripting protection.

8. 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.

9. Using Components With Known Vulnerabilities

Many modern web developers use components such as libraries and frameworks in their web applications. These components are pieces of software that help developers avoid redundant work and provide needed functionality; common examples include front-end frameworks like React and smaller libraries that used to add share icons or a/b testing. Some attackers look for vulnerabilities in these components which they can then use to orchestrate attacks. Some of the more popular components are used on hundreds of thousands of websites; an attacker finding a security hole in one of these components could leave hundreds of thousands of sites vulnerable to exploit.

10. Insufficient Logging And Monitoring

Many web applications are not taking enough steps to detect data breaches. The average discovery time for a breach is around 200 days after it has happened. This gives attackers a lot of time to cause damage before there is any response. OWASP recommends that web developers should implement logging and monitoring as well as incident response plans to ensure that they are made aware of attacks on their applications.