A1: Injection

Injection flaws are a set of security vulnerabilities which occur when suspicious data is inserted into an app as a command or query. Known injection attacks include SQL, OS, XXE, and LDAP.

All the input fields or the data source can be an injection vector. These types of attacks occur when the attacker can send maliciously crafted data through an input field to a web application. If there is a no proper validation of the input methods, attacker can cause the web application to malfunction and cause data loss, corruption, denial of service or even pawn the entire host.

These types of attacks are found in SQL, LDAP, SMTP headers, XML, ORM queries etc and are easy to discover an exploit using automated scanners and fuzzing software.

A2: Broken Authentication

When an application’s functions are not implemented correctly, the attack surface is open for criminals to easily break in and compromise passwords, session IDs, and exploit other flaws using stolen credentials. Sessions should be unique to each individual user, and without some necessary session management, an attacker can sneak in, disguised as a user to steal tokens and passwords to gain the access it is after.

Authentication in web applications is mostly used to control users' access to specific information. The most common security risks related to authentication and session management are: password theft, stealing session tokens and impersonating legitimate users. Security flaws related to authentication are typically identified in password reset functionality by tampering with cookies, session IDs, or similar.

Attackers will have access to thousands of username and password combinations which can be used for easily brute forcing a username password combination in a web application lacking bruteforce prevention methods. Moreover, if the session tokens and cookies are not expired after a particular time, valid sessions can be created by an attacker and gain access to confidential data.

Consider the case of a public computer. A person uses this computer to access a website. Instead of login out, the user simply closes the web browser and walks away. If proper session timeout is not employed in the web application, another person using the same computer can access the previous sessions becuase the previous user is still authenticated.

A3 – Sensitive Data Exposure

Sensitive data exposures may occur when security controls on the data are not implemented correctly, thus leaving a hole for attackers to steal sensitive information such as passwords, payment information, IDs, addresses, and anything else you may have stored which can be of value. Applications should ensure that access be authenticated and data be encrypted. A failure of such may lead to a major privacy violation.

Sensitive data stored in databases (or anywhere else) should be well protected. Credit card details, social security numbers and other sensitive customer details should be encrypted when stored in a database, even if they are not directly accessible through a web application. The same applies for sensitive data that is transmitted to and from the web application, such as credentials or payment details. Such information should be also be transmitted over a secure and encrypted layer.

Data should be always encrypted; in storage as well as in transit. When cryptography is employed we should make sure that there is strong key generation and management, strong algorithm, good password hashing techniques and proper protocol and cypher usage.

A4 – XML External Entities (XXE) [NEW]

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

XML Entities are like variables: they will expand to a defined value once they are processed by the XML parser. Even many who are not familiar with XML might have seen such an entity in a different place, such as HTML code. For example Instead of copy and pasting characters like the copyright symbol (©) you can conveniently write &copy; and the browser will display it correctly. These are predefined and can't be changed using HTML code.

However, XML gives you a way to define your own entities in order to make coding and configuration easier. An example (in an XML file) where this is necessary is in a customizable configuration file in a web application. For example you need to use a website’s name in multiple locations, such as the page title or footer, or for email templates. Instead of setting the name in each and every single XML tag, it's easier to define the XML Entity called &sitename; in which to hold this information and use it where necessary. So whenever you want to change the website name, you need only change it in the entity definition once, not in every single XML tag.

XML has other advantages too. For instance, XML allows you to define external entities. These are entities that may contain the content from a remote website or API endpoint. When called the XML parser will automatically fetch the external entity and include it in the XML file. That way, you can easily change the meaning of multiple documents without having to edit them manually.

However, there is one problem. When it is possible to pass XML documents to a parser that supports external entities, attackers can retrieve content from a website behind a firewall, issue requests to certain services, and even disclose the content of files stored on the server. Because entities can be referenced within entity definitions, attackers can craft an XML document that contains only 10 entities, but that will eventually expand to a billion entities once it is parsed. This is also known as the 'Billion Laughs Attack'.

Attackers can easily exploit vulnerabilities in XML processors, by uploading malicious XML files that may contain unwanted codes and thus exploiting the vulnerable code, dependencies and integration processes.

These flaws results in data extraction, remote code execution, DOS attacks and sometimes complete system compromise.

A5 – Broken Access Control

A flawed access control may be caused by unenforced user restrictions and this allows attackers to exploit and access unauthorized functionality or data. Access control is meant to control what “authorized” users are allowed and not allowed to do within an app, and to establish proper access control, the app must ensure that it is performing solid authorization checks and that proper authentication is in place to tell which users are privileged and which are in fact random internet users.

Broken access control may be due to that developers often fall to the difficulty of implementing a proper access control with all of the rules in place. And in various cases, access control rules are placed throughout the code and the collection of rules becomes scattered and nearly impossible to follow and understand.

Broken Access Control refers to restrictions that are not properly enforced. It occurs for example when authenticated users without administrative authority can create new administrator accounts.

A6 – Security Misconfiguration

According to OWASP, Security Misconfiguration is the most commonly seen issue. Strong security requires a good and secure configuration set and deployed for apps, frameworks, servers, database, and custom code, and all should be kept up to date. Otherwise, the flaws to come as a result can be exploited by attackers and will allow them to access privileged data. Proper configuration of an application’s entire environment needs to be defined, implemented, and regulated or it may lead to severe security holes.

Web application security is not just about secure web application coding. To ensure the security of a web application, it is also important to:

* secure the configuration of the web server,
* secure the operating system of the web server,
* [**ensure that the server is always updated with the latest security patches**](https://www.netsparker.com/blog/web-security/web-application-security-basics-keeping-software-up-to-date/).

A7 – Cross-Site Scripting (XSS)

Following a broad disagreement with the previous A7 (which was “Insufficient Attack Protection”), OWASP updated the list and placed Cross-Site Scripting as the updated A7. Cross-Site Scripting, commonly known as XSS, is a vulnerability that is often found in web apps. XSS allows attackers to inject client-side scripts into public facing web pages and, in many cases, can be used by attackers to work their way past access controls.

This is done by tricking a browser so that it accepts data from an untrusted source, and this typically happens when attackers use familiar code (such as JavaScript, for example) as developers don’t scrub out these characters.

Apps that allow user input without having full control over output may be highly at risk to XSS attacks. When an XSS attack is successful, attackers are able to cause serious damage to websites and have the ability to drag users on to other websites (often hosting more malicious code). Other known kinds of XSS attacks are Stored XSS, DOM Based XSS, and Reflected XSS.

A [**cross-site scripting (XSS)**](https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/) vulnerability allows hackers to inject malicious client-side script, in a website or web application, that is later executed by the victims.

A8 – Insecure Deserialization [New]

According to OWASP, “Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process – taking data structured from some format, and rebuilding it into an object.”

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

A9 – Using Components with Known Vulnerabilities

Component, including libraries and frameworks, may be taken from the open source community and should be used with caution in case vulnerabilities are lurking. As a vulnerable component is exploited, attackers can leverage it and cause the app serious damage and a major loss of data that can undermine the app, and perhaps even the organization.

By taking using components with known vulnerabilities, attackers can take advantage of that are easily attempt an SQLi and an XSS (among other attack methods) to attempt to takeover an occupy the app.

A10 – Insufficient Logging & Monitoring

According to OWASP, insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Insufficient Logging and Monitoring refers to the inability to log and detect hacking attempts and breaches.