A learning path for Injection Vulnerabilities

Injection ranked as 3rd in the 2021 OWASP top 10 list.
Over 500,000 applications were tested for some form of injection with the average incidence rate of 3.37% with a maximum of 19% and 274k occurrences. [1]

Intermediate Level

This attack type is considered a major problem in web security. In fact, injection attacks, such as SQLi and XSS, are very dangerous and also widespread, especially in legacy applications.

Performing injection allows an attacker to inject code into a program or query to execute remote commands that can read or modify a database or change data on a website.
Therefore, injection is an attacker's attempt to send data to an application in a way that will change the meaning of commands being sent to an interpreter. There are lots of interpreters in the typical web environment. Anything that combines user supplied data into a command is susceptible to injection.
This kind of flaws are prevalent particularly in legacy code and are easy to find when examining code. There are also tools like scanners and fuzzers that can help attackers find injection flaws.
Injection can compromise both backend systems as well as other clients connected to the vulnerable application.

An application is vulnerable to injection attacks when:

- User supplied data is not validated, filtered or sanitized by the application.
- Dynamic queries or non-parametrized calls without context-aware escaping are

used directly in the interpreter.
- Hostile data is used within object-relational mapping search parameters to extract additional sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands or stored procedures. [1]

Cross site scripting attacks are a type of injection, in which malicious script are injected into otherwise trusted websites.
XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different user.

Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates, without validating or encoding it.
Therefore, an attacker can use XSS to send a malicious script to an unsuspecting user and gain elevated access privileges to sensitive data maintained by the browser on behalf of the victim user.
A Cross Site Scripting vulnerability may be used by attackers to bypass access controls and same origin policy.
XSS effects vary in range from minor nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and on the nature of any security mitigation implemented by the site's owner network.

There is no single, standardized classification of Cross Site Scripting flaws but most experts agree to divide XSS attacks into two categories: stored (persistent) and reflected (non persistent). There is also a third type, less popular, called DOM based XSS.

Stored XSS
Stored attacks are those where the injected script is permanently stored on the target servers, for example in a database, forum, comment field. The victim retrieves the

malicious script from the servers when it requests the stored information. This kind of XSS is often referred to as Persistent XSS. [2]

The data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users during regular browsing, without proper HTML escaping. This vulnerability arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read and the application does not perform any processing of the data.

Persistent XSS vulnerabilities can be more significant than other types because an attacker's malicious script is rendered automatically, without the need to individually target victims or lure them to a third-party website. The code could also be further designed to self-propagate across accounts, creating a client-side worm.

Blind Cross Site Scripting

It is a form of Persistent XSS. It generally occurs when the attacker’s payload is saved on the server and reflected back to the victim from the backend application. For example, in feedback forms, an attacker can submit the malicious payload using the form, and once the backend user or admin of the application will open the attacker’s submitted form via the backend application, the malicious payload will be executed. [2]

Reflected XSS:

Reflected XSS, also called Non-Persistent XSS is, the most basic type of Cross Site Scripting vulnerability.

These holes show up when the data provided by a web client, most commonly in HTTP requests parameters, is used immediately by server-side scripts to parse and display a page of results for and to the user, without properly sanitizing the content.

The injected script is reflected off the web server in responses that include some or all of the input sent to the server as part of the request. [2]

Any non-validated user-supplied data, included in the resulting page without proper HTML encoding, may lead to markup injection.

Reflected attacks are delivered to victims via various ways, such as e-mail messages, or on some other website.

The bait is an innocent-looking URL, pointing to a trusted site but containing the XSS vector.
When a user is tricked into clicking on a malicious link, the injected code travels to the vulnerable web site which reflects back the attack to the victim's browser. The browser executes the code because it comes from a "trusted" server.
At this point, the script can carry out any action, and retrieve any data, to which the user has access.

DOM based XSS:
It is an XSS attack where the attack payload is executed as a result of modifying the DOM environment on the victim's browser used by the original client-side script, so that the client-side code runs in an unexpected manner. The page itself does not change, but the client-side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment. This is in contrast to other XSS attacks, where the attack payload is placed in the response page. [3]
The HTTP response sent from the server does not contain the attacker's payload. This payload manifests itself at the client-side script at runtime.
This type of XSS arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing data back to the DOM.
Prevention strategies for DOM-based XSS include very similar measures to traditional XSS prevention strategies but implemented in JavaScript code and contained in web pages.

SQL injection is a code injection technique used to attack data-driven applications.
A SQL injection attack consists of insertion or "injection" of a malicious SQL query via the input data from the client to the application. [4]

SQL injection attacks allow attackers to spoof identity, cause repudiation issues, allow the complete disclosure of all data on the system, destroy data or make it otherwise unavailable, and become administrators of the database server performing privilege escalation.

It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.
Therefore, the severity of SQL Injection attacks is limited by the attacker's skill and imagination. [4]
Unfortunately, this has become a common issue with database-driven web sites and consequences can affect all aspects of data.
A SQL database usually holds sensitive data: loss of confidentiality is a frequent problem with SQL injection.
If poor SQL commands are used to check usernames and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password, affecting authentication.
If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.
Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack, breaking data integrity. [4]

SQL injection can be performed in different parts of a query.
Most SQL injection vulnerabilities arise within the WHERE clause of a SELECT query. But SQL injection vulnerabilities can in principle occur at any location within the query, and within different query types. The most common other locations where SQL injection arises are:

- In UPDATE statements, within the updated values or the WHERE clause.
- In INSERT statements, within the inserted values.
- In SELECT statements, within the table or column name.
- In SELECT statements, within the ORDER BY clause. [5]

It is important to consider that SQL injection attacks can be performed using any controllable input that is processed as a SQL query by the application. For example, some websites take input in JSON or XML format and use this to query the database.

Weak countermeasures implementations often just look for common SQL injection keywords within the request, so attackers could be able to bypass these filters by simply encoding or escaping characters in the prohibited keywords.

Bibliography

[1] «A03:2021 - Injection,» [Online]. Available: https://owasp.org/Top10/A03_2021-Injection. [Consulted 2023].

[2] «Cross Site Scripting,» [Online]. Available: https://owasp.org/www-community/attacks/xss/. [Consulted 2023].

[3] «DOM Based XSS,» [Online]. Available: https://owasp.org/www-community/attacks/DOM_Based_XSS. [Consulted 2023].

[4] «SQL Injection,» [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection. [Consulted 2023].

[5] «What is SQL injection,» [Online]. Available: https://portswigger.net/web-security/sql-injection. [Consulted 2023].