OWASP Top 10 Vulnerabilities:

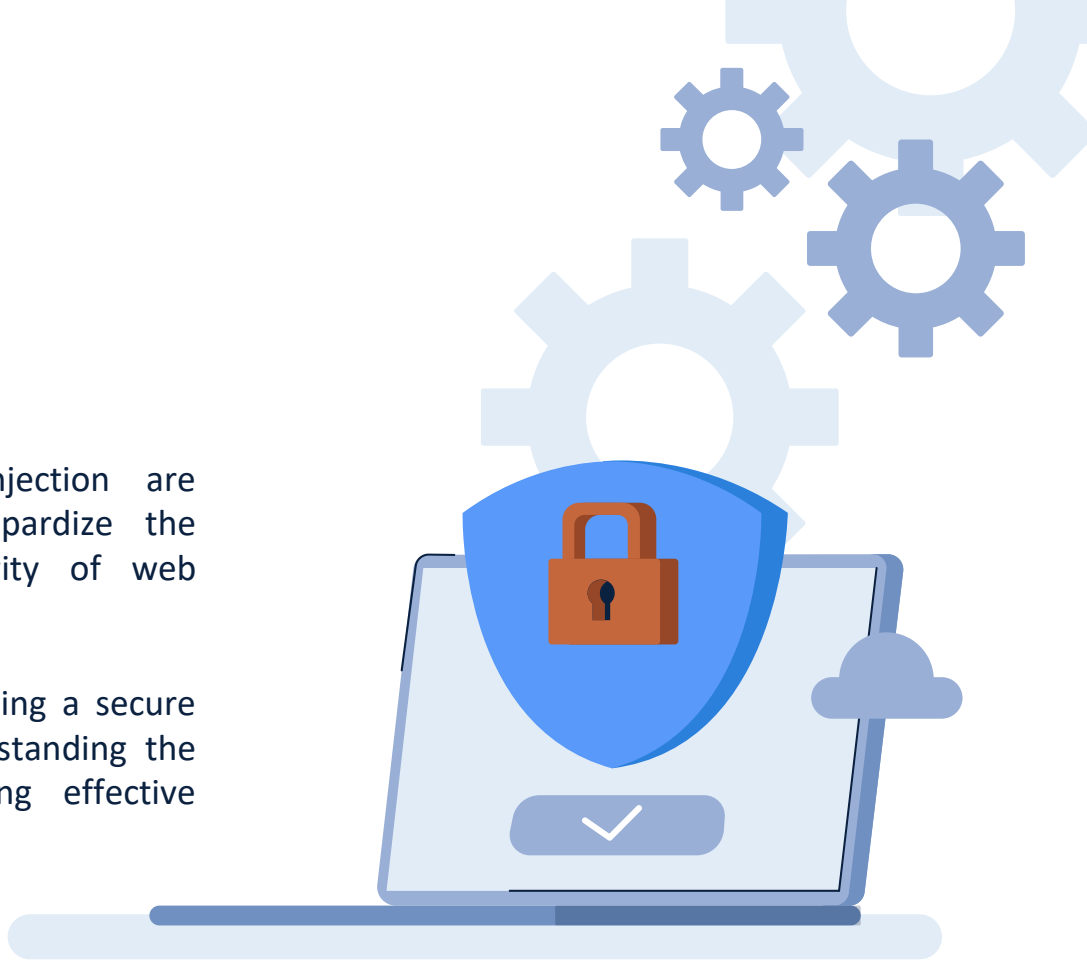
Injection and Insecure Design

Daniela Tomás up202004946@edu.fc.up.pt
Diogo Nunes up202007895@edu.fc.up.pt
João Veloso up202005801@edu.fc.up.pt

Introduction

Insecure design flaws and code injection are vulnerabilities that can seriously jeopardize the confidentiality, availability, and integrity of web applications.

Safeguarding digital assets and maintaining a secure online environment require both understanding the tactics used by attackers and putting effective countermeasures in place.



Injection (A03:2021)

- Injections occur when an application sends untrusted data to an interpreter. This can trick the interpreter into executing unintended commands or accessing data without the required authorization.
- Injection flaws are very prevalent, particularly in legacy code, and are often found in SQL queries, LDAP queries, XPath queries, OS commands, program arguments, etc.
- Injection flaws are normally discovered when there is the ability to examine code directly, but they can also be found via testing, albeit with more difficulty. There are also tools to help discover them, such as fuzzers and scanners.

Exploitation Techniques

01

SQL Injection:

Attackers insert malicious SQL statements into input fields, manipulating the database.

02

Command Injection

Occurs when an attacker can manipulate the input of a system command. By injecting malicious commands, attackers can execute arbitrary code on the underlying operating system. This type of vulnerability is particularly dangerous when user input is directly incorporated into system commands without proper validation or sanitization.

03

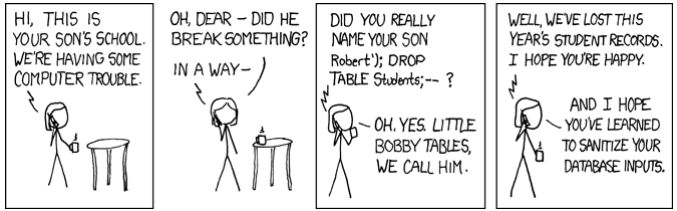
LDAP Injection

Attackers exploit this vulnerability to manipulate the queries, potentially gaining unauthorized access, retrieving sensitive information, or performing unintended operations on the LDAP server.

Examples

01 SQL Injection

In a simple login form where the query is a simple SELECT query to the "users" table, an attacker could simply type *admin'#*, effectively cutting out the part of the query that utilizes the password and gaining admin privileges.



02 Command Injection

In a simple web application that allows a user to ping a server by inserting a hostname or IP, the user can provide the following input: *8.8.8.8; ls -la*, which will lead to the ls command being run after the ping.

03 LDAP Injection

In an application that uses LDAP to authenticate users against a directory server. The application constructs an LDAP query to validate user credentials. An attacker, instead of providing a valid username and password, enters the following input for the username: **)(uid=*))(|(uid=**.
In this case, the attacker's input manipulates the query to match all users, bypassing the authentication process. The attacker can potentially gain unauthorized access to the application.

Countermeasures

- To create a countermeasure against injection attacks, first understand their workings and potential consequences;
- Conduct a code review to identify vulnerabilities, including user input or external data incorporated without proper validation or sanitization;
- Use trusted APIs with well-documented interfaces to avoid using interpreters;
- Regular security testing helps identify and mitigate injection vulnerabilities;
- Implement security headers and access controls to protect against various attacks;
- Ensuring users can only access authorized data and actions is crucial.

Insecure Design (A04:2021)

- Insecure design vulnerabilities result from inadequate system architecture or design choices that provide security flaws. Attackers take advantage of these flaws to compromise the system’s availability, confidentiality, or integrity. Injection vulnerabilities can result from insecure design practices.

Exploitation Techniques

01

Weak Authentication Design

Attackers can obtain unauthorized access to sensitive data and increase their privileges within a system by circumventing weak access control or authentication mechanisms.

02

Inadequate Web Server Design

Web servers that cannot manage a huge amount of incoming HTTP requests might be overwhelmed with excessive requests, resulting in performance degradation or system failure.

03

Detailed Error Messages

Providing detailed error messages to users or exposing sensitive information such as the internal state of the system in error messages.

Examples

01 Weak Authentication Design

If an application has a weak or easily guessable password encryption scheme, attackers can crack passwords using brute force tactics or dictionary attacks and compromise user accounts.

02 Inadequate Web Server Design

Flooding a website with a large amount of HTTP requests, using server resources, and perhaps slowing or making the website unavailable.

03 Detailed Error Messages

An attacker can guess valid credentials if a login error message indicates if a username or password is wrong.

Countermeasures

- Ensuring development teams are knowledgeable in security best practices, identifying vulnerabilities, and building features with security in mind;
- Creating a library of secure design patterns and pre-built components can save time and ensure consistent security precautions;
- Threat modeling is a proactive approach to security, identifying potential weaknesses in areas like authentication, access control, business logic, and data flows;
- Error messages should only include relevant information to the target audience;
- Implementing resource consumption limits ensures no one user or service can overload the system, preventing system security compromise.

State-of-the-Art

The latest trends, technologies, and best practices related to countering injection and insecure design vulnerabilities include advancements in web application firewalls, machine learning-based threat detection, secure coding frameworks, automated testing, and a shift-left approach to security.

Injection vulnerabilities, such as SQL Injection and Command Injection, occur when untrusted data is improperly included in the code. The state-of-the-art includes parameterized queries, ORMs and Security Libraries, Web application firewalls, content security policies, static analysis and dynamic scanning tools, regular expression limitations, and API security to combat injection risks.

Insecure design encompasses security flaws within an application's architecture and development decisions. The state-of-the-art includes practices such as thread modeling, secure development frameworks, security by default, microservices and serverless architectures, Continuous Integration/Continuous Deployment (CI/CD), and a Secure API Design.

References

- https://owasp.org/Top10/A04 2021-Insecure Design/
- https://owasp.org/Top10/A03 2021-Injection/
- The MITRE Corporation, Common Weakness Enumeration, “CWE-89: Improper Neutralization of Special Elements used in na SQL Command (’SQL Injection’)”. https://cwe.mitre.org/data/definitions/89.html, 2023.
- https://xkcd.com/327/
- The MITRE Corporation, Common Weakness Enumeration, “CWE-78: Improper Neutralization of Special Elements used in na OS Command (’OS Command Injection’)”. https://cwe.mitre.org/data/definitions/78.html, 2023.
- The MITRE Corporation, Common Weakness Enumeration, “CWE-90: Improper Neutralization of Special Elements used in an OS Command (’OS Command Injection’)”. https://cwe.mitre.org/data/definitions/90.html, 2023.
- The MITRE Corporation, Common Weakness Enumeration, “CWE-257: Storing Passwords in a Recoverable Format”. https://cwe.mitre.org/data/definitions/257.html, 2023.
- The MITRE Corporation, Common Weakness Enumeration, “CWE-799: Improper Control of Interaction Frequency”. https://cwe.mitre.org/data/definitions/799.html, 2023.
- The MITRE Corporation, Common Weakness Enumeration, “CWE-209: Generation of Error Message Containing Sensitive Information”. https://cwe.mitre.org/data/definitions/209.html, 2023.
- https://owasp.org/www-project-top-ten/
- https://sectigostore.com/blog/what-is-owasp-what-are-the-owasp-top-10- vulnerabilities/
- Dafydd Stuttard and Marcus Pinto, The Web Application Hacker’s Handbook, 2nd ed, 2011.

Conclusion

We discuss the importance of addressing vulnerabilities like injection attacks and insecure design flaws in cybersecurity. It emphasizes the need for understanding attacker exploitation techniques, implementing countermeasures, and identifying attack vectors.

Proactive approaches include secure architecture reviews, security training, and coding practices. Access controls, security audits, and security by design are also crucial. This holistic approach ensures a secure digital ecosystem.