# Injection

**Injection vulnerabilities** occur when **untrusted input** is inserted into a command or query in such a way that the interpreter (SQL, OS, LDAP, etc.) **executes unintended commands** or **modifies program behavior**. The attacker **injects code into the application's data stream**, and the system blindly trusts and executes it.

# Common Injection Targets

| Target | Description |
| --- | --- |
| SQL | Inject into database queries |
| OS Command | Inject into shell/system commands |
| LDAP | Modify directory queries |
| XPath | Tamper XML data queries |
| NoSQL | Bypass or manipulate NoSQL logic (MongoDB, etc.) |
| SMTP Header | Inject into email headers |
| XML (XXE) | Exploit XML parser through DTD |

# Impact:

| Type | Description |
| --- | --- |
| Authentication Bypass | Login without credentials |
| Data Exposure | Read sensitive records (PII, credentials) |
| Data Manipulation | Modify/delete records |
| RCE (Remote Code Execution) | In OS command injection |
| DoS (Denial of Service) | Crash app or overload DB |
| Lateral Movement | Pivot to other systems/databases |

# Types of Injection Vulnerabilities

1. **SQL Injection (SQLi)**
2. **Command Injection**
3. **LDAP Injection**
4. **NoSQL Injection**
5. **XPath Injection**
6. **XML External Entity (XXE) Injection**

# Detection Techniques

| Method | Description |
| --- | --- |
| Manual Testing | Insert payloads in input fields, URLs, headers |
| Fuzzing | Use Burp Intruder with common injection payloads |
| Logic Observation | Input ', --, ;, ` |
| Boolean-Based Checks | Try input like 1=1, 1=2 and compare responses |
| Blind Injection | Observe time delay or behavioral differences |
| Error-Based | Look for DB errors like syntax error, unexpected token |
| Source Code Audit | Look for unsanitized input passed into sensitive functions |

# Tools for Injection Testing

| Tool | Use |
| --- | --- |
| Burp Suite | Manual & automated injection discovery |
| SQLMap | Auto-detect and exploit SQL injection |
| NoSQLMap | Exploit NoSQL injection (MongoDB) |
| Commix | Command injection scanner |
| ZAP | Passive/active scan for injection points |
| wfuzz / ffuf | Wordlist-based fuzzing |
| Postman | For manual request crafting and tampering |

# Mitigation

## 1. Input Validation

* Allowlist expected characters/values (e.g., email format)
* Block dangerous characters (', ", ;, --, |, <, >, etc.)

## 2. Use Safe APIs / Parameterized Queries

| Language | Safe Practice |
| --- | --- |
| PHP (PDO) | prepare(), bindParam() |
| Java | PreparedStatement |
| Python | cursor.execute("SELECT \* FROM users WHERE id = %s", (id,)) |
| .NET | SqlCommand.Parameters.Add() |

## 3. Escape Input (Context-Aware)

| Context | Escape Method |
| --- | --- |
| SQL | Parameterized query (avoid escaping manually) |
| HTML | Encode <, >, " |
| Shell | Avoid user input in shell or escape special characters |

## 4. Least Privilege Principle

* DB users should not have DROP, DELETE, or xp\_cmdshell access unless needed

## 5. Disable Dangerous Functions

* Remove or restrict access to:
  + eval, system, exec, shell\_exec
  + xp\_cmdshell in SQL Server

## 6. Web Application Firewall (WAF)

* Helps detect/block known attack patterns
* Not a substitute for secure code

# Points

“Injection exploits occur when **user input reaches a parser unsanitized** — goal is to **isolate commands from data**.”

“Always prefer **parameterized queries over string concatenation** in database calls.”

“Each injection vector (SQL, NoSQL, OS, LDAP) requires **contextual protection and output encoding**.”

# Real-World Examples

| Org | Vulnerability |
| --- | --- |
| Sony (2011) | SQL injection led to full database dump |
| TalkTalk (2015) | SQL injection caused massive data breach |
| Equifax (2017) | Apache Struts RCE → command injection via vulnerable input parsing |
| GitHub, Google (bounty reports) | Repeated NoSQL and command injection discoveries |