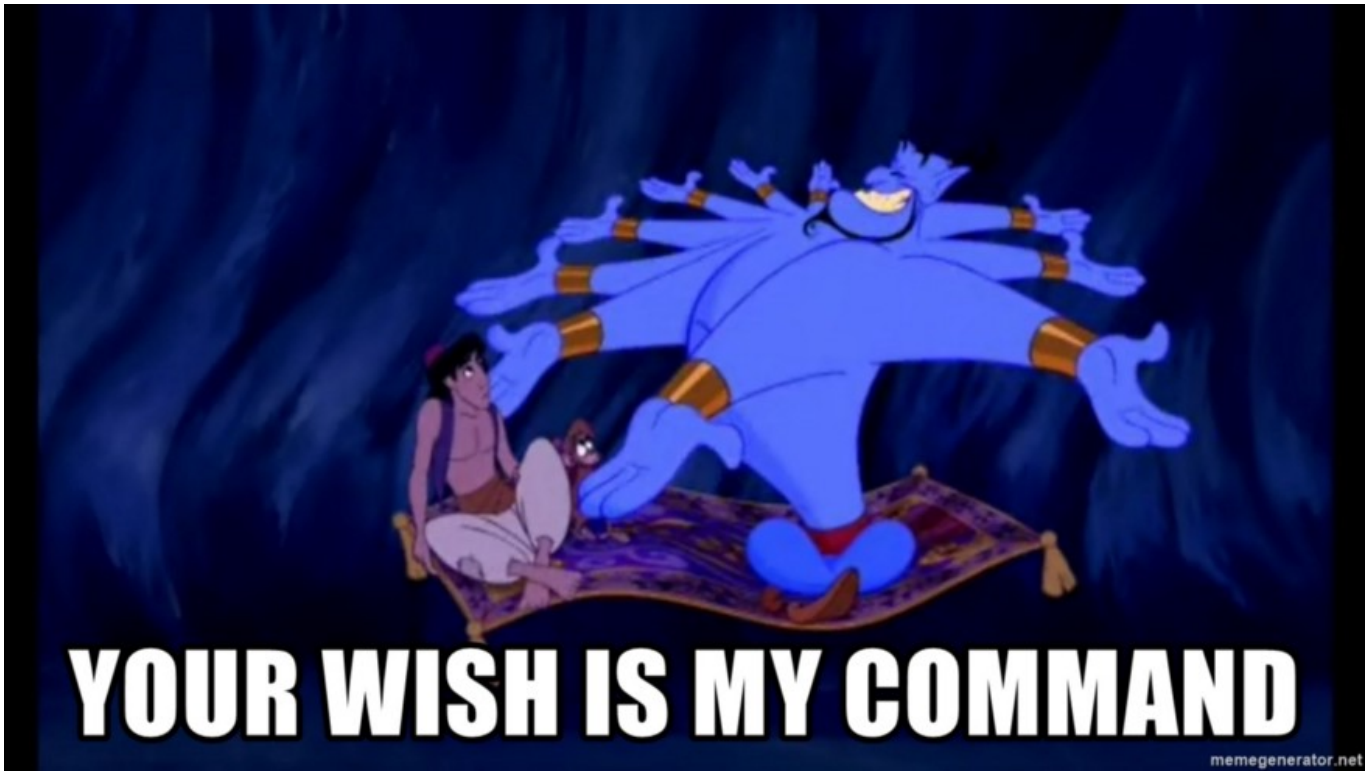
Breaking down — Command Injections

goswamiijaya Follow
Oct 18, 2020 · 6 min read

Command Injection or OS Command Injection is a category of injection vulnerabilities, where an attacker is able to exploit an unsanitized user input further to run default OS commands in the server.

> ***Code Injection:*** *allows the attacker to add their own code that is then executed by the application.*
>
> ***Command Injection:*** *the attacker extends the default functionality of the application, which executes system commands, without injecting code.*



What actually is a Command Injection attack - according to OWASP?

Command injection is an attack in which the goal is the execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user-supplied data (forms, cookies, HTTP headers, etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

Identifying CIs in Application Source-Code: OWASP

1. In **PHP** based applications: Vulnerable Code

```
<?php
print("Specify the file to delete");
print("<p>");
$file=$_GET['filename'];
system("rm $file");
?>
```

In the above code snippet, the application asks for a `filename` value from the user, which is directly supplied to the `system` command for further execution, without sanitization or escaping the values in the `filename` parameter.

So, here an attacker can modify the request to run system commands like `id` as in the below request:

Request

```
http://127.0.0.1/delete.php?filename=bob.txt;id
```

Therefore, an attacker can extract the private values of `id` via his modified request as shown in the below response:

Response

```
  Please specify the name of the file to delete

  uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

2. Similarly, In **Perl** based applications: Vulnerable Code

```
use CGI qw(:standard);

$name = param('name');

$nslookup = "/path/to/nslookup";

print header;

if (open($fh, "$nslookup $name|")) {

while (<$fh>) {

print escapeHTML($_);

print "<br>\n";

}

close($fh);

}
```

Suppose an attacker provides a domain name like this:

Attack code- `cwe.mitre.org%20%3B%20/bin/ls%20-l`

The "%3B" sequence decodes to the ";" character, and the %20 decodes to a *space*. The `open()` statement would then process a string like this:

```
/path/to/nslookup cwe.mitre.org ; /bin/ls -l
```

As a result, the attacker executes the " `/bin/ls -l` " command and gets a list of all the files in the program's working directory — CWE.

3. And, In **Java** based applications: Vulnerable Code

The below code reads the name of a *shell script* to execute from the system properties. It is subject to the second variant of OS command injection.

```
String script = System.getProperty("SCRIPTNAME");

if (script != null)

System.exec(script);
```

What you as an attacker can do?

An attacker can upload malicious programs or files, obtain passwords, gain backdoor access, escalate privileges, execute unintended commands, compromise any sensitive files placed on the server, or escalate attack to the network interface.

Steps to exploit Command Injections:

1. Use the `ping` command to **trigger a time delay** by causing the server to *ping* its loopback interface for a specific period.
2. Induce a *30-second* time delay on the platform (Windows or Unix) if no filtering is in done:

```
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1
```

3. **Monitor the time taken** for the application to respond, for each of the induced time delays:

```
| ping –i 30 127.0.0.1 |
| ping –n 30 127.0.0.1 |
& ping –i 30 127.0.0.1 &
& ping –n 30 127.0.0.1 &
; ping 127.0.0.1 ;
%0a ping –i 30 127.0.0.1 %0a
` ping 127.0.0.1 `
```

4. If a time delay occurs, the application **may be vulnerable** to command injection.

5. Repeat the test case several times to confirm that the delay was **not** the result of "**network latency** or other anomalies".

6. Try changing the value of the `-n` or `-i` parameters and confirming that the **delay experienced varies systematically** with the value supplied.

1. If successful try injecting commands like `ls` or `dir` . Check whether you can **retrieve the results of the command to your browser**.

2. If you are unable to retrieve results directly: attempt to **open an out-of-band channel back to your computer**. Try using ***TFTP*** to copy tools up to the server, using `telnet` or `netcat` to create a reverse shell back to your computer, and using the mail command to send command output via ***SMTP***.

3. You can **redirect the results of your commands to a file** within the *webroot*, which you can then retrieve directly using your browser.

For example: `dir > c:\inetpub\wwwroot\foo.txt`

4. When you have found a means of injecting commands and retrieving the results, **determine your privilege** level; use — `whoami` or attempt to write a harmless file to a protected directory.

5. You may then seek to **escalate privileges, gain backdoor access to sensitive application data, or attack other hosts reachable from the compromised server.**

How to identify a Command Injection vulnerability in WebApps?

In a web application where the filename is shown in the URL.

Perl — append the Pipe symbol `|` onto the end of the filename.

URL before alteration:

```
http://sensitive/cgi-bin/userData.pl?doc=profile.txt
```

URL modified:

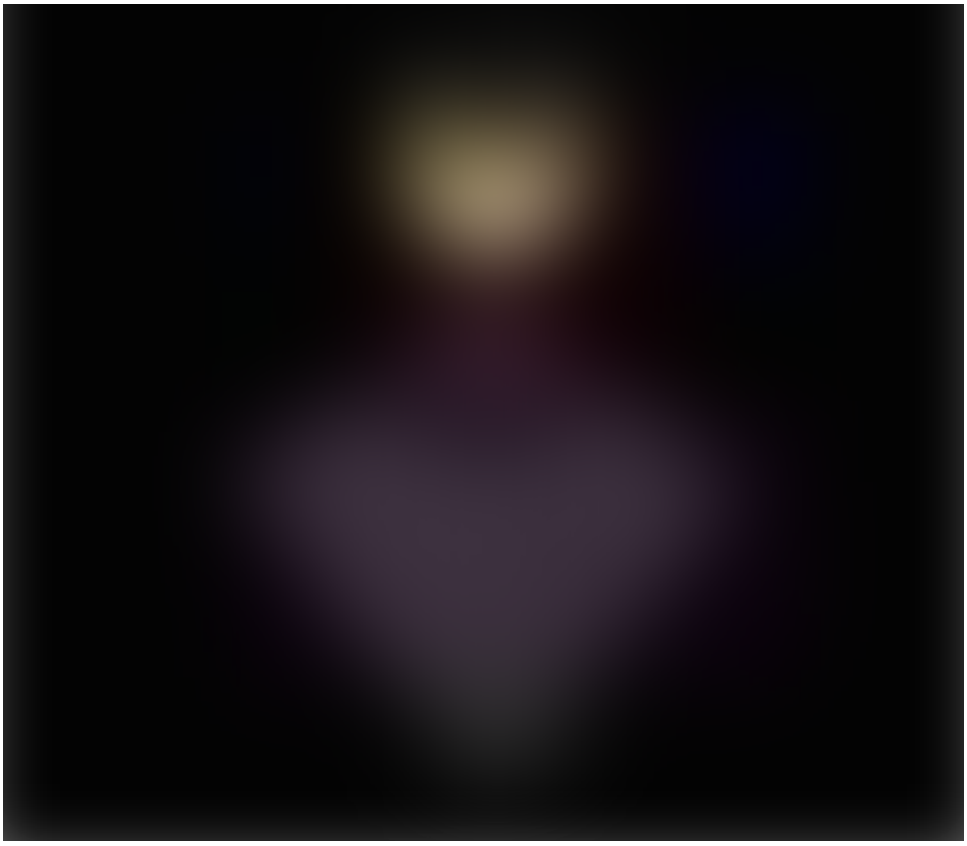
```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

This will execute the command `/bin/ls`.

PHP — appending a semicolon `;` to the end of a URL followed by an OS command. ; is %3B in URL encoding.

URL modified:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd
```



Understanding the role of Special Characters in Command Injection

Combining Special characters with the user input, lets you modify or diverge the application to perform unintended actions. As in Command Injection, a request to perform a command 2 which is ordered dynamically by the attacker. The following special character can be used for command injection such as `|` `;` `&` `$` `>` `<` `'` `!`

> *`cmd1|cmd2` : Use of `|` will make command 2 to be executed independent of the fact whether command 1 is executed or not.*
>
> *`cmd1;cmd2` : Use of `;` will make command 2 to be executed independent of the fact whether command 1 is executed or not.*
>
> *`cmd1||cmd2` : Command 2 will only be executed if command 1 fails to execute.*
>
> *`cmd1&&cmd2` : Command 2 will only be executed if command 1 succeeds in execution.*
>
> *`$(cmd)` : For example, `echo $(whoami)` or `$(touch test.sh; echo 'ls' > test.sh)`*
>
> ***cmd** : It's used to execute specific command. For example, whoami*

> *>(cmd): >(ls)*
>
> *<(cmd): <(ls)*

How to bypass the mitigations?

Try using some characters as shown below to bypass the checks or escaping done by the application.

For *command injection* try characters or combination of characters as below to test the defense implemented by the application: `| ; & $ > < ' \ ! >> #`

Escape or filter special characters for ***Windows***, `( ) < > & * ‘ | = ? ; [ ] ^ ~ ! . ” % @ / \ : + , ``

Escape or filter special characters for ***Linux***, `{ } ( ) > < & * ‘ | = ? ; [ ] $ – # ~ ! . ” % / \ : + , ``

Further exploitation

Have a keen eye for below system commands that are more likely to be vulnerable for Command Injection. Check if you can run these commands directly.

Java - `Runtime.exec()`

C/C++ -

```
system
exec
ShellExecute
```

Python -

```
exec
eval
os.system
os.popen
subprocess.popen
subprocess.call
```

PHP -

```
system
```

```
shell_exec

exec

proc_open

eval
```

Ciao!

References:

WSTG - Latest

This article describes how to test an application for OS command injection. The tester will try to inject an OS command...

owasp.org

The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws

The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws eBook: Stuttard, Dafydd, Pinto, Marcus...

www.amazon.in