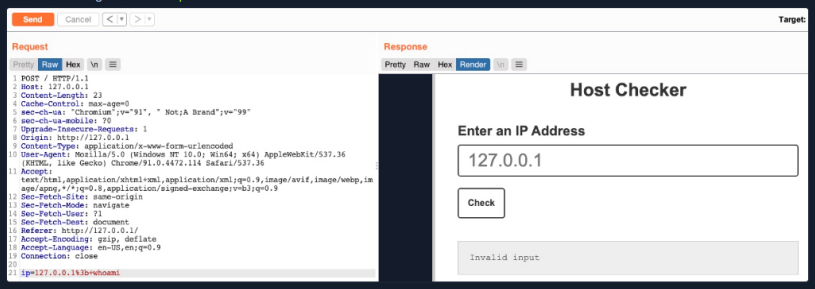
Identifying Filters

As we have seen in the previous section, even if developers attempt to secure the web application against injections, it may still be exploitable if it was not securely coded. Another type of injection mitigation is utilizing blacklisted characters and words on the back-end to detect injection attempts and deny the request if any request contained them. Yet another layer on top of this is utilizing Web Application Firewalls (WAFs), which may have a broader scope and various methods of injection detection and prevent various other attacks like SQL injections or XSS attacks.

This section will look at a few examples of how command injections may be detected and blocked and how we can identify what is being blocked.

Filter/WAF Detection

Let us start by visiting the web application in the exercise at the end of this section. We see the same `Host Checker` web application we have been exploiting, but now it has a few mitigations up its sleeve. We can see that if we try the previous operators we tested, like (`;`, `&&`, `||`), we get the error message `invalid input`:



This indicates that something we sent triggered a security mechanism in place that denied our request. This error message can be displayed in various ways. In this case, we see it in the field where the output is displayed, meaning that it was detected and prevented by the `PHP` web application itself. `If the error message displayed a different page, with information like our IP and our request, this may indicate that it was denied by a WAF.`

Let us check the payload we sent:

Code: bash

```bash
127.0.0.1; whoami
```

Other than the IP (which we know is not blacklisted), we sent:

1. A semi-colon character `;`
2. A space character
3. A `whoami` command

So, the web application either `detected a blacklisted character` or `detected a blacklisted command`, or both. So, let us see how to bypass each.

Blacklisted Characters

A web application may have a list of blacklisted characters, and if the command contains them, it would deny the request. The `PHP` code may look something like the following:

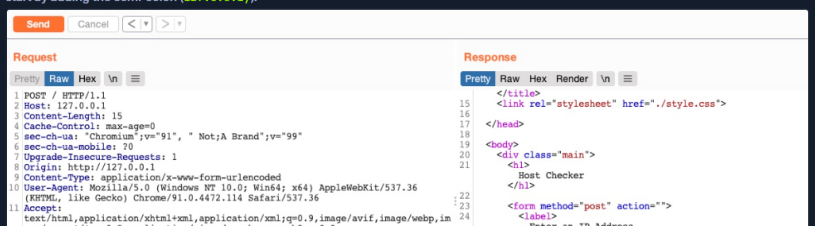
Code: php

```php
$blacklist = ['&', '|', ';', ...SNIP...];
foreach ($blacklist as $character) {
    if (strpos($_POST['ip'], $character) !== false) {
        echo "Invalid input";
    }
}
```

If any character in the string we sent matches a character in the blacklist, our request is denied. Before we start our attempts at bypassing the filter, we should try to identify which character caused the denied request.

Identifying Blacklisted Character

Let us reduce our request to one character at a time and see when it gets blocked. We know that the (`127.0.0.1`) payload does work, so let us start by adding the semi-colon (`127.0.0.1;`):



```
  age/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%3b
```

```
      Enter an IP Address
    </label>
25  <input type="text" name="ip" placeholder="127.0.0.1"
26  <button type="submit">
      Check
    </button>
27  </form>
28
29  <p>
30    <pre>
31      Invalid input
```

We still get an `invalid input`, error meaning that a semi-colon is blacklisted. So, let's see if all of the injection operators we discussed previously are blacklisted.

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

Pwnbox Location

UK 137ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

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

Cheat Sheet

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

+1 Try all other injection operators to see if any of them is not blacklisted. Which of (new-line, &, |) is not blacklisted by the web application?

new-line

Submit Hint

Previous Next

Mark Complete & Next