Casper wargame Solutions

Francesca Del Nin
student number: 0734630

December 2018

1 Overview

Level	Password	Time spent
casper4	zssEylQhyfOdX0H7OOKFxEXG0iY9Y7PL	
casper41	qQftts3uK2DjrNxfEPzXTyOpB20vMakY	
casper6		
casper60	EoWZv5u7pnBkchLK30vXhiyH1sDexpCB	
casper8	AAqtUL09LWefJlJFvTg0SpVg0j89DGe2	
casper80	5CS80xbUrxgjbrU3BBS5YslY00qoUd5J	
casper83	MABIQpR4omrMCmdc7PwxbRNm5EPHzYX5	
casper10	BpI02agB8PmOKDI1LRvMeQJWi2GrzgWA	

2 Casper4 solution

2.1 Description

Program casper4 takes an input from the user and prints a greeting and the input from the user. If no input is provided it prints an explanation on how to use the program and exit.

2.2 Vulnerability

This program is vulnerable because it calls `strcpy` function passing the (char) pointer `s` without doing a bound check on the length of the variable pointed, so it can be longer than 666 byte, which is the length of the `buf` array in which the content will be copied.

2.3 Exploit description

We provide as input a string of 678 byte in total composed as following:

1. the shellcode (21 byte),
2. some padding, in this case ”A”’s but can also be NOP (653 byte),
3. the address of the buffer, retrieved through gdb by placing a breakpoint inside the `greetUser` function and using the command `p &buf` on the server.

In this way we fill the whole buffer and we overwrite the frame pointer and the return address of the function, so when the execution of greetUser is done the instruction inside the buffer (so the shellcode) will be executed instead of returning to main.

2.4 Mitigation

2.5 Advanced level casper41

This exploit works also with casper41 because it only checks for environment variables which are not used for the exploit. I just need to update the address of the buffer, found using gdb on the binary file of casper41.

3 casper6

This program prints a greeting to the user when provided an input, it uses a struct composed by a buffer array and a function pointer.

3.1 Description

3.2 Vulnerability

3.3 Exploit description

3.4 Mitigation

4 casper8

4.1 Description

This level prints a greeting to the user as the other levels, but in this case the stack is non-executable so it's not possible to do a buffer overflow.

4.2 Vulnerability

In this case the vulnerability is

4.3 Exploit description

To make this exploit work I first looked for how many input characters make the program go in segmentation fault, then I verified that I was overwriting also EIP register using the "info frame" command after the execution of strcpy inside greetUser and found out that I need a total of 682 byte to complete overwrite the return address of greetUser.

Then I searched for the address of the `system` function (which execute a shell command) inside libc, which is loaded because it's included in the program. I used gdb command `p &system` while running the program and I found that the address is 0xb7e62310. Since I want to pass as argument to this function the string "/bin/xh", which is not present inside the loaded library, I created a new environment variable (through export MYSHELL=/bin/xh). To find out the address in which is stored I used the command `x/s *((char **)environ)` and went through the variables until I found MYSHELL at *((char **)environ+19) at the address 0xbfffff3f, so to make it point just to "/bin/xh" without "MYSHELL" the address is 0xbfffff47.

To make it a cleaner attack I also searched for the address of the `exit` function inside the library to pass it as return address of the system function. I used the same command as above and found it at 0xb7e55260.

To make the exploit work I passed a string composed as following:

- 682-4=678 A's;
- `system` function address (endian encoded);
- `exit` function address (endian encoded);
- the address of the string "/bin/xh".

This attack worked inside gdb but not outside because the address of the environmental variables can change a bit, the error displayed was telling "bin/xh: not found" so the string passed was missing a "/". Changing the address of the string to 0xbfffff46 did the trick.

4.4 Mitigation

4.5 Advanced level casper80

The same approach work also for this advanced level because it only checks for NOP in the input. To make it work I checked again the distance from the buffer to the return address of `greetUser` function, this time it has changed to 682 byte (so 686 to overwrite it). I also changed the address of the string because the address of the environmental variable MYSHELL changes a bit, to make it work I used 0xbfffff44. To get this value I read the error message displayed with the address used in the other exploit and there where two characters missing.

4.6 Advanced level casper82

The same approach works also for casper82, because it checks for ASCII characters just for the length of the buffer, so if I fill the whole buffer with A's and only after use the ascii characters to overwrite the return address the attack will still work. The exploit in this case is exactly the same as casper80.