Lab Report

1. Name: 林靖昀 Student ID: B12902116

2. Please briefly describe where the vulnerabilities are and how you exploited it for the following challenges. (Please attach the screenshot of the flag to prove your work as well.)
   - Magic number
   - 3 stages

   Magic number

   With radare2, we can step into main and see the assembly code. we see that after printing “Give me the magic :)”, there is a read syscall, which reads 4 bytes from stdin into the buffer [var_24h], then there is a check:

   ```
   mov eax, dword [var_24h] (Our input)
   cmp eax,0x79487ff
   je 0x40096f                    (The address we want to go to)
   ```

   Thus in pwntools, we can use “r.send(p32(0x79487ff))” to pass the first check.

   After giving the magic number, we see:

   Hacker can complete 1000 math problems in 60s, prove yourself.

   39143 + 5998 = ?

   We can simply write a python script to parse and solve all these questions for us:

   ```
   for i in range(1000):
       problem = r.recvuntil("?")
       r.send(str(eval(problem[:-4])) + '\n')
   ```

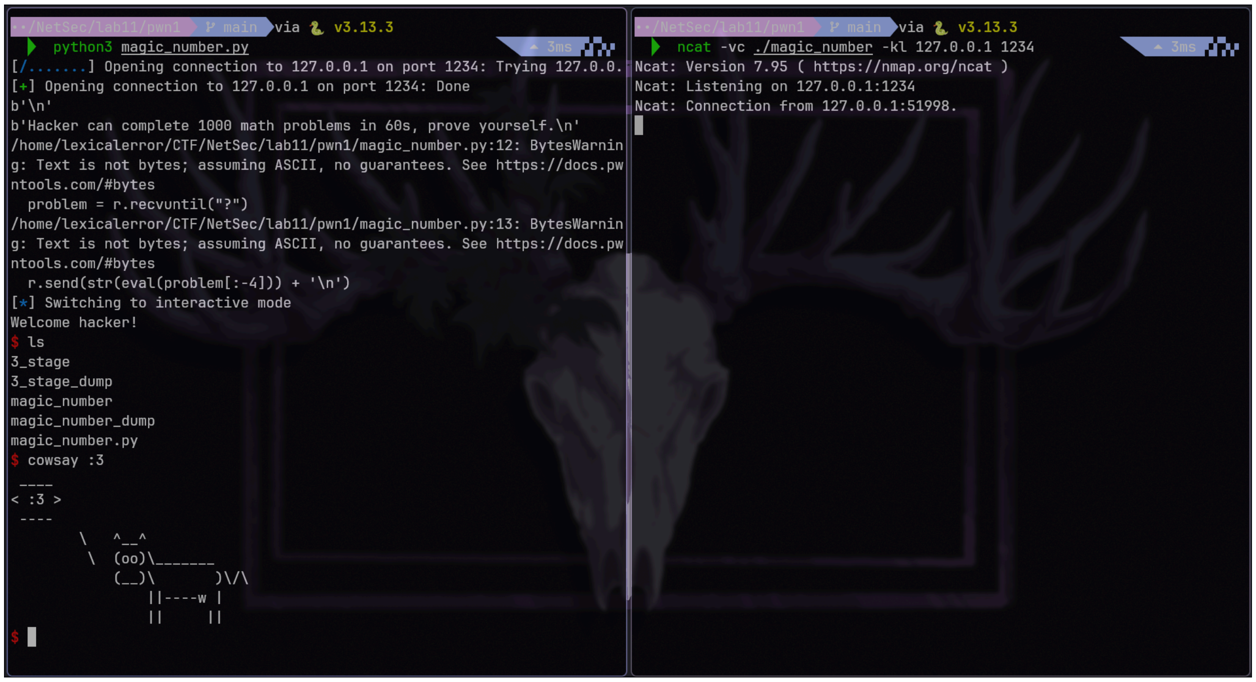
   Thus the final python script is:

   ```
   from pwn import *
   r = remote("127.0.0.1", 1234)
   r.recvuntil(b":)")

   # First input: 0x79487ff
   r.send(p32(0x79487ff))
   print(r.recvline())
   print(r.recvline())

   for i in range(1000):
       problem = r.recvuntil("?")
       r.send(str(eval(problem[:-4])) + '\n')
   r.interactive()
   ```

Screenshot:



```
··/NetSec/lab11/pwn1  main via 🐍 v3.13.3
 python3 magic_number.py                                          3ms
[/.......] Opening connection to 127.0.0.1 on port 1234: Trying 127.0.0.
[+] Opening connection to 127.0.0.1 on port 1234: Done
b'\n'
b'Hacker can complete 1000 math problems in 60s, prove yourself.\n'
/home/lexicalerror/CTF/NetSec/lab11/pwn1/magic_number.py:12: BytesWarnin
g: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pw
ntools.com/#bytes
  problem = r.recvuntil("?")
/home/lexicalerror/CTF/NetSec/lab11/pwn1/magic_number.py:13: BytesWarnin
g: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pw
ntools.com/#bytes
  r.send(str(eval(problem[:-4])) + '\n')
[*] Switching to interactive mode
Welcome hacker!
$ ls
3_stage
3_stage_dump
magic_number
magic_number_dump
magic_number.py
$ cowsay :3
 ____
< :3 >
 ----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
$ 
```

```
··/NetSec/lab11/pwn1  main via 🐍 v3.13.3
 ncat -vc ./magic_number -kl 127.0.0.1 1234                       3ms
Ncat: Version 7.95 ( https://nmap.org/ncat )
Ncat: Listening on 127.0.0.1:1234
Ncat: Connection from 127.0.0.1:51998.
```

3 stages:

With radare2, we can step into main and see the assembly code. We see that it calls three functions, sym.stage1, sym.stage2, sym.stage3. First we look at stage1:

In stage1 we see the scanf("%d", &var_ch) call that takes user input, then there are 3 checks:

1. mov eax, dword [var_ch]   (Our input)

   and eax, 1

   test eax, eax

   je 0x4007bb   (The failure message function)

2. mov eax, dword [var_ch]   (Our input)

   and eax, 0x100000

   test eax, eax

   je 0x4007bb   (The failure message function)

3. mov eax, dword [var_ch]   (Our input)

   and eax, 0x200

   test eax, eax

jne 0x4007bb   (The failure message function)

So we want the first two “and eax <x>” to be non-zero, and the last one to be zero.

0x1   is 000000000000000000000001 in binary

0x100000   is 000100000000000000000000 in binary

0x200   is 000000000000001000000000 in binary

Thus the number: 000100000000000000000001 (1048577 in decimal)

Should allow us to pass the stage 1.

In stage2, we see the scanf(“%d %d %d”, &var_14h, &var_10h, &var_ ch) call that takes user input, then there are 3 checks:

1. mov eax, dword [var_14h]   (Our first input)

   cmp eax, 0x64   (100 in decimal)

   jne 0x40084d   (The failure message function)

2. mov eax, dword [var_10h]   (Our second input)

   cmp eax, 0x100   (256 in decimal)

   jne 0x40084d   (The failure message function)

3. mov eax, dword [var_ch]   (Our third input)

   cmp eax, 0xfaceb00c   (4207849484 in decimal)

   jne 0x40084db   (The failure message function)

Thus the input 100 256 4207849484 should allow us to pass stage 2.

In stage3, we see the scanf(“%ld”, &var_10h) call that takes user input, then there is one check:

mov rax, qword [var10h]   (Our input)

cmp rax, obj.im_a_global_variable

jne 0x4008c9   (The failure message function)

In gdb/objdump we see:

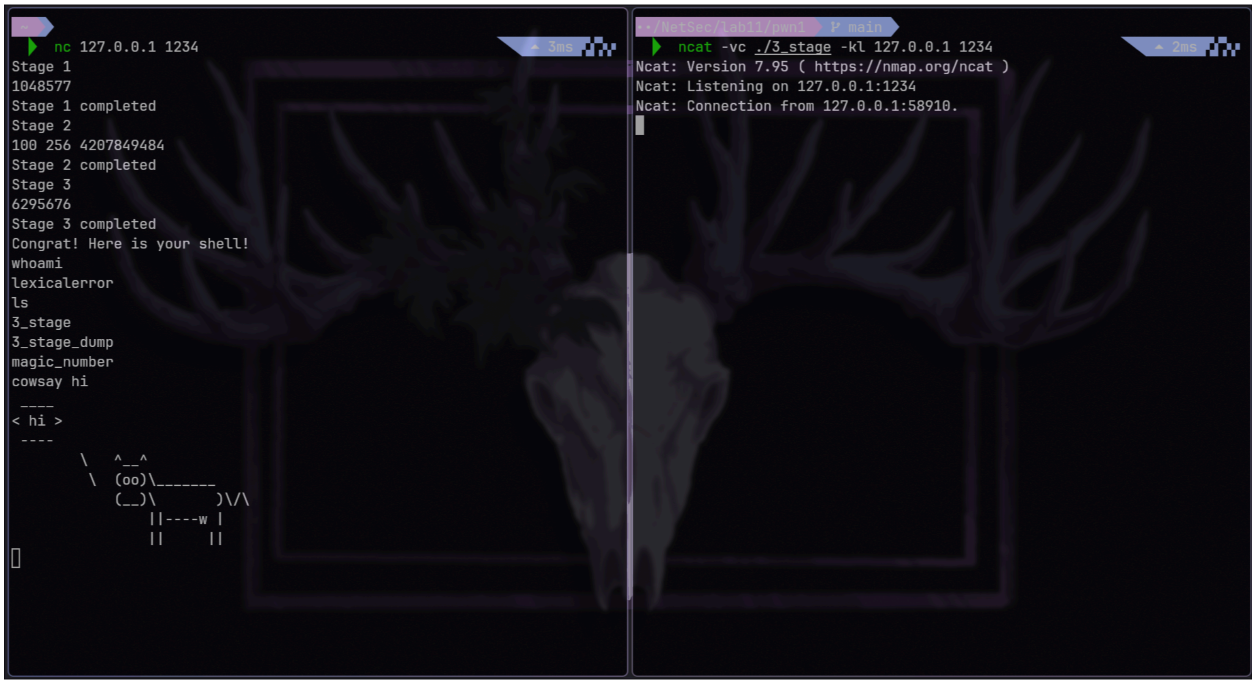
mov rax,QWORD PTR [rbp-0x10] (Our input)

cmp rax,0x60107c

jne 4008c7 (The failure message function)

Thus the input 0x60107c (6295676 in decimal) should allow us to pass stage 3.

Screenshot:



```
~
▶ nc 127.0.0.1 1234                                    3ms
Stage 1
1048577
Stage 1 completed
Stage 2
100 256 4207849484
Stage 2 completed
Stage 3
6295676
Stage 3 completed
Congrat! Here is your shell!
whoami
lexicalerror
ls
3_stage
3_stage_dump
magic_number
cowsay hi
 ____
< hi >
 ----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
~/NetSec/lab11/pwn1  main
▶ ncat -vc ./3_stage -kl 127.0.0.1 1234                2ms
Ncat: Version 7.95 ( https://nmap.org/ncat )
Ncat: Listening on 127.0.0.1:1234
Ncat: Connection from 127.0.0.1:58910.
```