Week2- Write-up

Vishnu Vardhan Ciripuram
N14912012
vc2499

Challenge – Quasar

The challenge involved connecting to a remote server, but before guessing the black hole's mass, we needed to reverse-engineer a binary file to identify the correct value.

Solution Steps:

- The binary file likely contained the logic for generating or storing the mass value I needed. I opened this binary file in **Binary Ninja** to reverse-engineer it and find the mass.
- I began by analyzing the **main()** function to understand the program flow. Using Binary Ninja's disassembly view, I traced through the code and noticed a function that referenced a specific value related to the mass.

```
00001289  int32_t main(int32_t argc, char** argv, char** envp)

00001295      int32_t argc_1 = argc
00001298      char** argv_1 = argv
000012a1      set_buffering_mode()
000012b0      puts(str: "\n\tThis quasar is the brightest…  ")
000012bf      puts(str: "\tCan you guess the mass of the …  ")
000012d3      printf(format: &data_207a)
000012d3
000012f4      if (read_input() != 0x3f5476a00)
00001320          puts(str: "\n\tThat's not the correct mass!…  ")
00001325          return 1
00001325
00001300      puts(str: "\n\tYeah! You're right!")
0000130a      print_flag()
0000130f      return 0
```

- While examining the function calls, I identified a key function that seemed to be responsible for printing or loading the mass. Upon further inspection, I found a **global variable** storing a hexadecimal value, which appeared to be the mass in question.
- The mass was found in the binary as 0x3f5476a00
- I submitted to and received the flag
  flag{sc4la4r_v4lu3s_h1dd3n_1n_pl41ns1ght!_defae1fb2b45fd49}

```
┌──(kali㉿kali)-[~]
└─$ nc offsec-chalbroker.osiris.cyber.nyu.edu 1250
Please input your NetID (something like abc123): vc2499
hello, vc2499. Please wait a moment ...

        This quasar is the brightest object in the universe.
        Can you guess the mass of the black hole at its center?

        > 0x3f5476a00

        Yeah! You're right!

        Here's your flag, friend: flag{sc4la4r_v4lu3s_h1dd3n_1n_pl41ns1ght!_defae1fb2b45fd49}
```

Challenge – Secrets

The challenge involved connecting to a remote server and guessing a one-byte key (between 0 and 255) to retrieve a flag.

Solution Steps:

- I opened the provided binary file in **Binary Ninja** to understand the logic behind the key verification process. The binary likely contained logic that compared the user-provided key with some internal value.
- I began by analyzing the **main()** function to trace the program's workflow. Using Binary Ninja's disassembly view, I noticed that the program read the key and XORed it with a hidden secret value.

```
00001269  int32_t main(int32_t argc, char** argv, char** envp)

00001279      int32_t argc_1 = argc
0000127f      char** argv_1 = argv
00001286      char** envp_1 = envp
0000128d      void* fsbase
0000128d      int64_t rax = *(fsbase + 0x28)
000012a1      set_buffering()
000012b0      void var_a8
000012b0      read_message(&var_a8)
000012c4      printf(format: "\nCan you guess the key?\n> ")
000012ce      char rax_4 = read_key()
000012d9      int32_t var_ac = 0
00001348      int32_t result
00001348
00001348      while (true)
00001350          if (sx.q(var_ac) u>= strlen(&var_a8))
0000135c              puts(str: "\nGood job! that's the right key…  ")
00001366              read_flag()
0000136b              result = 0
0000136b              break
0000136b
00001316          if (zx.d(*(sx.q(var_ac) + &secret) ^ rax_4) != sx.d(*(&var_a8 + sx.q(var_ac))))
00001322              puts(str: "\nTry again!\n\n")
00001327              result = 1
0000132c              break
0000132c
0000132e          var_ac += 1
0000132e
00001374      *(fsbase + 0x28)
00001374
0000137d      if (rax == *(fsbase + 0x28))
00001389          return result
00001389
0000137f      __stack_chk_fail()
0000137f      noreturn
```

- While investigating further, I found that although the key was processed, the secret value used in the XOR operation was not directly visible in the binary. After exploring related functions like **read_key()**, I confirmed that the key was expected to be a one-byte value ranging from 0 to 255 (0xff is 255 in decimal).

```
0000138a  int64_t read_key()

00001396      void* fsbase
00001396      int64_t rax = *(fsbase + 0x28)
000013b8      void var_15
000013b8      fgets(buf: &var_15, n: 5, fp: stdin)
000013ce      int64_t result = strtol(nptr: &var_15, endptr: nullptr, base: 0xa)
000013ce
000013e6      if (result s< 0 || result s> 0xff)
000013f2          puts(str: "\nThe key is out of range :( Try…  ")
000013fc          exit(status: 1)
000013fc          noreturn
000013fc
00001409      *(fsbase + 0x28)
00001409
00001412      if (rax == *(fsbase + 0x28))
0000141a          return result
0000141a
00001414      __stack_chk_fail()
00001414      noreturn
```

- Since the secret was not accessible, and the key space was limited to 256 values, I decided that brute-forcing the key was the most efficient approach. The program's logic ensured that this range was small enough for brute force.
- I wrote a Python script using **pwntools** to automate the brute-force attack. The script connected to the server, sent my NetID, and tested each possible key from 0 to 255. If the response contained "flag", the script stopped and returned the flag.

Open | secrets.py ~/Downloads/offsec

```python
from pwn import *

context.log_level = 'error'

for key in range(256):
    with remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1254) as conn:
        conn.sendlines([b'vc2499', str(key).encode()])
        if b'flag' in (response := conn.recvall()):
            print(f"Correct key: {key}\nFlag: {response.decode()}")
            break
    print(f"Key {key} was incorrect.")
```

- After testing the keys, the correct key was found to be **233**. I received the flag: **flag{4_0n3_byt3_k3y_g1v3s_4_v3ry_sm4ll_k3y_sp4c3!_9991cd62cf9b0b80}**

```
Key 210 was incorrect.
Key 211 was incorrect.
Key 212 was incorrect.
Key 213 was incorrect.
Key 214 was incorrect.
Key 215 was incorrect.
Key 216 was incorrect.
Key 217 was incorrect.
Key 218 was incorrect.
Key 219 was incorrect.
Key 220 was incorrect.
Key 221 was incorrect.
Key 222 was incorrect.
Key 223 was incorrect.
Key 224 was incorrect.
Key 225 was incorrect.
Key 226 was incorrect.
Key 227 was incorrect.
Key 228 was incorrect.
Key 229 was incorrect.
Key 230 was incorrect.
Key 231 was incorrect.
Key 232 was incorrect.
Correct key found: 233
Flag:
Good job! that's the right key!


        Here's your flag, friend: flag{4_0n3_byt3_k3y_g1v3s_4_v3ry_sm4ll_k3y_sp4c3!_9991cd62cf9b0b80}
```

Challenge – StrS

The challenge involved connecting to a remote server and providing the correct answer to retrieve a flag. The correct answer was hidden within a binary file.

Solution Steps:

- I opened the provided binary file in **Binary Ninja** to inspect the logic behind the answer verification process. In the main() function, I saw that the user input was being compared to a hardcoded value using strcmp().
- The disassembly of the **main()** function showed that it called **strcmp()** to compare the input (stored in var1) with a predefined string stored in var2. The program outputted whether the comparison was successful or not, and if the strings matched, it called a print_flag() function.

```
00001249  int32_t main(int32_t argc, char** argv, char** envp)

00001255      int32_t argc_1 = argc
00001258      char** argv_1 = argv
00001261      init()
00001270      puts(str: "\n\tGive me the right answer and…  ")
00001284      printf(format: &data_2040)
000012a2      fgets(buf: &var1, n: 0x32, fp: stdin)
000012c7      *(strcspn(&var1, &data_2044) + &var1) = 0
000012c7
000012e6      if (strcmp(&var1, &var2) != 0)
00001312          puts(str: "\n\tThat's not the correct answe…  ")
00001317          return 1
00001317
000012f2      puts(str: "\n\tYep, that's the right answer…  ")
000012fc      print_flag()
00001301      return 0
```

- I traced var2 in the init() function. However, the full string **wasn't fully visible**, so I decided to use **gdb** to further analyze the exact value being compared.

```
0000131e  int64_t init()

0000133f      setvbuf(fp: stdout, buf: nullptr, mode: 2, size: 0)
0000135d      int64_t result = setvbuf(fp: stdin, buf: nullptr, mode: 2, size: 0)
00001376      __builtin_strncpy(dest: &var2, src: " SMSS J052915.80", n: 0x10)
0000138e      data_40b0 = 0x35313533349288e2
00001395      __builtin_strncpy(dest: &data_40b8, src: "2.0 ", n: 5)
000013a8      return result
```

- Used GDB to Inspect Memory:
    - I disassembled the main to find the strcmp() location.

```
        Give me the right answer and I'll give you the flag!

        >

        That's not the correct answer! Try again friend!

[Inferior 1 (process 62435) exited with code 01]
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000555555555249 <+0>:     endbr64
   0x000055555555524d <+4>:     push   %rbp
   0x000055555555524e <+5>:     mov    %rsp,%rbp
   0x0000555555555251 <+8>:     sub    $0x10,%rsp
   0x0000555555555255 <+12>:    mov    %edi,-0x4(%rbp)
   0x0000555555555258 <+15>:    mov    %rsi,-0x10(%rbp)
   0x000055555555525c <+19>:    mov    $0x0,%eax
   0x0000555555555261 <+24>:    call   0x55555555531e <init>
   0x0000555555555266 <+29>:    lea    0xd9b(%rip),%rax        # 0x555555556008
   0x000055555555526d <+36>:    mov    %rax,%rdi
   0x0000555555555270 <+39>:    call   0x5555555550d0 <puts@plt>
   0x0000555555555275 <+44>:    lea    0xdc4(%rip),%rax        # 0x555555556040
   0x000055555555527c <+51>:    mov    %rax,%rdi
   0x000055555555527f <+54>:    mov    $0x0,%eax
   0x0000555555555284 <+59>:    call   0x5555555550f0 <printf@plt>
   0x0000555555555289 <+64>:    mov    0x2da0(%rip),%rax        # 0x555555558030 <stdin@GLIBC_2.2.5>
   0x0000555555555290 <+71>:    mov    %rax,%rdx
   0x0000555555555293 <+74>:    mov    $0x32,%esi
   0x0000555555555298 <+79>:    lea    0x2dc1(%rip),%rax        # 0x555555558060 <var1>
   0x000055555555529f <+86>:    mov    %rax,%rdi
   0x00005555555552a2 <+89>:    call   0x555555555110 <fgets@plt>
   0x00005555555552a7 <+94>:    lea    0xd96(%rip),%rax        # 0x555555556044
   0x00005555555552ae <+101>:   mov    %rax,%rsi
   0x00005555555552b1 <+104>:   lea    0x2da8(%rip),%rax        # 0x555555558060 <var1>
   0x00005555555552b8 <+111>:   mov    %rax,%rdi
   0x00005555555552bb <+114>:   call   0x555555555100 <strcspn@plt>
   0x00005555555552c0 <+119>:   lea    0x2d99(%rip),%rdx        # 0x555555558060 <var1>
   0x00005555555552c7 <+126>:   movb   $0x0,(%rax,%rdx,1)
   0x00005555555552cb <+130>:   lea    0x2dce(%rip),%rax        # 0x5555555580a0 <var2>
   0x00005555555552d2 <+137>:   mov    %rax,%rsi
   0x00005555555552d5 <+140>:   lea    0x2d84(%rip),%rax        # 0x555555558060 <var1>
   0x00005555555552dc <+147>:   mov    %rax,%rdi
   0x00005555555552df <+150>:   call   0x555555555120 <strcmp@plt>
   0x00005555555552e4 <+155>:   test   %eax,%eax
   0x00005555555552e6 <+157>:   jne    0x555555555308 <main+191>
   0x00005555555552e8 <+159>:   lea    0xd59(%rip),%rax        # 0x555555556048
```

- I set a breakpoint at the strcmp() call in the main() function.

```
(gdb) b *0x5555555552df
Breakpoint 1 at 0x5555555552df
(gdb) run
Starting program: /home/kali/Downloads/strs
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

        Give me the right answer and I'll give you the flag!

        >  SMSS J052915.80

Breakpoint 1, 0x00005555555552df in main ()
```

- After hitting the breakpoint, I inspected the value of var2 (the correct answer). I found that the correct answer stored in var2 was **" SMSS J052915.80–435152.0"** (with a leading space and additional numbers at the end).

```
(gdb) x/s 0x5555555580a0
0x5555555580a0 <var2>:  " SMSS J052915.80-435152.0 "
```

- I reconnected to the server and submitted **" SMSS J052915.80–435152.0"** as the answer.
- After submitting I recieved the flag **flag{str1ng_c0mp4r1s0n_ch3cks_3v3ry_ch4r!_b9f39b5792fbef98}**

```
┌──(kali㉿kali)-[~/Downloads/offsec]
└─$ nc offsec-chalbroker.osiris.cyber.nyu.edu 1253
Please input your NetID (something like abc123): vc2499
hello, vc2499. Please wait a moment ...

        Give me the right answer and I'll give you the flag!

        >  SMSS J052915.80-435152.0

        Yep, that's the right answer!

        Here's your flag, friend: flag{str1ng_c0mp4r1s0n_ch3cks_3v3ry_ch4r!_b9f39b5792fbef98}
```

Challenge – Cosmic Distance

This challenge required submitting the correct distance from a quasar to Earth. The distance was hidden within the binary file, and it needed to be input in hexadecimal format.

Solution Steps:

- I opened the binary in **Binary Ninja** and identified the functions responsible for reading the input and comparing it to a predefined value. The main() function compared the user input with a variable distance, which was initialized in the init() function.

```
00001229  int32_t main(int32_t argc, char** argv, char** envp)

00001235      int32_t argc_1 = argc
00001238      char** argv_1 = argv
00001241      set_buffering_mode()
0000124b      init()
0000125a      puts(str: "\tCan you tell me the distance f…  ")
0000126e      printf(format: &data_2045)
0000126e
0000128c      if (read_input() != distance)
000012b8          puts(str: "\n\tThat's not the correct dista…  ")
000012bd          return 1
000012bd
00001298      puts(str: "\n\tYeah! You got the right dist…  ")
000012a2      read_flag()
000012a7      return 0
```

- In the init() function, the correct distance was set as 0x2cb417800.

```
000012c4  void init()

000012d6      distance = 0x2cb417800
```

- I entered the correct value: **0x2cb417800** and received the flag **flag{0nly_tw3lv3_b1ll10n_l1ght_y34rs_4w4y!_143f7e13a542445e}**

```
┌──(kali㉿kali)-[~/Downloads/offsec]
└─$ nc offsec-chalbroker.osiris.cyber.nyu.edu 1252
Please input your NetID (something like abc123): vc2499
hello, vc2499. Please wait a moment...
        Can you tell me the distance from this quasar to Earth?


        : 0x2cb417800

        Yeah! You got the right distance!

        Here's your flag, friend: flag{0nly_tw3lv3_b1ll10n_l1ght_y34rs_4w4y!_143f7e13a542445e}
```

Challenge – Favorite

This challenge involved connecting to a remote server and providing the correct address to retrieve the flag. The key was analyzing a binary file to figure out how to compute the correct address to send back.

Solution Steps:

- Using **Binary Ninja**, I examined the binary and found that the program first prints the address of the hint function.

```
000012a9  int64_t hint()

000012c2  |    return puts(str: "\noh PIE!\n")
```

- In the main() function, I saw that the user input was compared to a specific hardcoded value, 0xc01dc0ffee, which was stored at a memory address, data_43f8.

```
000012c3  int32_t main(int32_t argc, char** argv, char** envp)

000012cf      int32_t argc_1 = argc
000012d2      char** argv_1 = argv
000012d6      void* fsbase
000012d6      int64_t rax = *(fsbase + 0x28)
000012ea      set_buffering_mode()
000012f4      init()
00001308      printf(format: "\n\tFYI, this is the address of …  ")
00001314      int64_t (* buf)() = hint
00001329      write(fd: 1, &buf, nbytes: 8)
00001338      puts(str: &data_2048)
00001347      puts(str: "\n\tCan you tell me where is my …  ")
0000135b      printf(format: &data_2081)
00001385      int32_t result
00001385
00001385      if (**read_input() != 0xc01dc0ffee)
000013b1          puts(str: "\n\tNah, that's not my favorite …  ")
000013b6          result = 1
00001385      else
00001391          puts(str: "\n\tYou did it! Thanks for findi…  ")
0000139b          print_flag()
000013a0          result = 0
000013a0
000013bf      *(fsbase + 0x28)
000013bf
000013c8      if (rax == *(fsbase + 0x28))
000013d0          return result
000013d0
000013ca      __stack_chk_fail()
000013ca      noreturn
```

- The goal is to calculate the address of data_43f8 using the address of the hint function and send this to the server.
- I examined the binary to find the following information:
  - Offset of hint: 0x12a9
  - Offset of data_43f8: 0x43f8

```
000043f8  int64_t data_43f8 = 0x5e26fe8499a49815
```

```
000012a9  int64_t hint()
```

- The hint address was printed by the server. Using this address and subtracting the known hint offset (0x12a9)
- I wrote a **pwntools** script like the previous challenges to connect to the server, receive the hint address, calculate the base address by subtracting the hint offset, compute the data_43f8 address, and send it to the server
- By adding the data_43f8 offset (0x43f8) to the base address, I computed the address of data_43f8.

fav.py
~/Downloads/offsec

```python
from pwn import *

r = remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1251)

print(r.recvuntil(b'NetID (something like abc123): ').decode())
r.sendline(b'vc2499')

print(r.recvuntil(b'function `hint`: ').decode())
hint_addr = u64(r.recvn(8))

data_addr = hint_addr + (0x43f8 - 0x12a9)
print(f"data address: {hex(data_addr)}")

r.sendlineafter(b'> ', str(data_addr).encode())
r.interactive()
```

- After submitting the correct address to the server, I received the flag: **flag{l34ks_d0ubl3_p01nt3rs_4nd_0ffs3ts_t0_w1n!_cf466e9dcafd9ab0}**

```
┌──(kali㉿kali)-[~/Downloads/offsec]
└─$ python3 fav.py
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1251: Done
Please input your NetID (something like abc123):
hello, vc2499. Please wait a moment...

        FYI, this is the address of function `hint`:
data address: 0x55964f9c43f8
[*] Switching to interactive mode

        You did it! Thanks for finding my drink!

        Here's your flag, friend: flag{l34ks_d0ubl3_p01nt3rs_4nd_0ffs3ts_t0_w1n!_cf466e9dcafd9ab0}

[*] Got EOF while reading in interactive
$
```