Week0-Refresher Write-ups

Name: **Xinsheng Zhu**
UnivID: **N10273832**
NetID: **xz4344**

!!! 455/300 pts solved !!!

Vault 0 (50 pts)

```
root@17b95fe8a8e6:~/wk0/vault0# nc offsec-chalbroker.osiris.cyber.nyu.edu 1230
......
Can you tell me the address of the secret vault?

>
......
```

In this challenge, we need to find the address of the secret vault.

First, we inspect the ELF headers of the binary `vault0` using `readelf` with `-h` flag to check if Position Independent Executable (PIE) is on.

```
root@17b95fe8a8e6:~/wk0/vault0# readelf -h vault0
ELF Header:
......
  Type:                              EXEC (Executable file)
......
```

The `Type` field in the ELF header shows `EXEC`, meaning PIE is off for the binary `vault0`.

Then, we use `readelf` with `-Ws` flag to show the symbol table in the binary `vault0`. Because PIE is off, the binary has hardcoded addresses for its symbols.

```
root@17b95fe8a8e6:~/wk0/vault0# readelf -Ws vault0
......
Symbol table '.symtab' contains 48 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
......
    38: 0000000000401236    21 FUNC    GLOBAL DEFAULT   15 secret_vault
......
```

We can see that addresses are in the `0x400000` range, which is absolute, so the address of the secret vault in hexadecimal is `0x401236`.

Finally, we write a script using `pwntools` to send the address to the server in decimal.

Part of the script is as follows. It should be noted that interaction with running processes in `pwntools` always deals with bytes, not strings, so make sure that we send and receive bytes and not strings.

```
from pwn import *
......
p = remote(URL, PORT)
......
print(p.recvuntil(b"> ").decode())
addr = 0x401236
p.sendline(str(addr).encode())
log.info(f"Sending address in base 10: {str(addr).encode()}")

p.interactive()
```

The captured flag is `flag{Th3_g00d_0ld_d4ys_0f_N0_PIE!_99ef8f305b88cb52}`.

Vault 1 (50 pts)

```
root@17b95fe8a8e6:~/wk0/vault1# nc offsec-chalbroker.osiris.cyber.nyu.edu 1231
......
Can you still find the address of the secret vault?
I was told this time it's protected by some 'PIE' 🥧
But I found this base address 0x55cde88ef000 on a post-it note!

>
......
```

In this challenge, with the base address in hexadecimal, we need to find the address of the secret vault. But this time, PIE is on for the binary `vault1`.

First, we verify the `Type` field in the ELF header using `readelf` with `-h` flag, showing `DYN`.

```
root@17b95fe8a8e6:~/wk0/vault1# readelf -h vault1
ELF Header:
......
  Type:                              DYN (Position-Independent Executable file)
......
```

Then, we display the symbol table of the binary `vault1` using `readelf` with `-Ws` flag to access offsets from the base address.

```
root@17b95fe8a8e6:~/wk0/vault1# readelf -Ws vault1
......
Symbol table '.symtab' contains 50 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
......
    38: 0000000000001249    26 FUNC    GLOBAL DEFAULT   16 secret_vault
......
```

We can see that the offset of the secret vault in hexadecimal is `0x1249`.

Finally, we write a script using `pwntools` to match the base address in hexadecimal with regex, calculate the address by `address = base address + offset`, and send it to the server in hexadecimal.

Part of the script is as follows.

```python
from pwn import *
import re
......
p = remote(URL, PORT)
......
data = p.recvuntil(b"> ").decode()
print(data)
pattern = re.compile(r"0x[0-9a-fA-F]+")
base_addr = int(pattern.findall(data).pop(), 16)
offset = 0x1249
addr = base_addr + offset
p.sendline(hex(addr).encode())
log.info(f"Sending address in base 16: {hex(addr)}")

p.interactive()
```

The captured flag is `flag{n0t_s00_PIE_1f_w3_g3t_th3_BASE!_a304d898a1771efb}`.

Vault 2 (50 pts)

```
root@17b95fe8a8e6:~/wk0/vault2# nc offsec-chalbroker.osiris.cyber.nyu.edu 1232
......
Can you still find the address of the secret vault?
I found this fake vault at 0x5568d7a5a029, but it doesn't appear to be the right one!

>
......
```

In this challenge, with the leaked fake address in hexadecimal, we need to find the address of the secret vault.

First, we similarly learn that PIE is on for the binary `vault2`.

Then, we display the symbol table of the binary `vault2` using `readelf` with `-Ws` flag to access offsets from the base address.

```
root@17b95fe8a8e6:~/wk0/vault2# readelf -Ws vault2
......
Symbol table '.symtab' contains 52 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
......
    12: 0000000000004029     1 OBJECT  LOCAL  DEFAULT   26 fake_vault
......
    40: 0000000000001269    26 FUNC    GLOBAL DEFAULT   16 secret_vault
......
```

We can see that the offset of the fake vault and the secret vault in hexadecimal are `0x4029` and `0x1269`.

Finally, we write a script using `pwntools` to match the leaked fake address in hexadecimal with regex, calculate the address by `address = fake address - fake offset + offset`, and send it to the server in hexadecimal.

Part of the script is as follows.

```python
from pwn import *
import re
......
p = remote(URL, PORT)
......
data = p.recvuntil(b"> ").decode()
print(data)
pattern = re.compile(r"0x[0-9a-fA-F]+")
fake_addr = int(pattern.findall(data).pop(), 16)
fake_offset = 0x4029
offset = 0x1269
addr = fake_addr - fake_offset + offset
p.sendline(hex(addr).encode())
log.info(f"Sending address in base 16: {hex(addr)}")

p.interactive()
```

The captured flag is `flag{wh0_n33ds_th3_BASE_1f_w3_h4v3_4_lEaK!_ead46bf902ab7a8c}`.

Vault 3 (50 pts)

```
root@17b95fe8a8e6:~/wk0/vault3# nc offsec-chalbroker.osiris.cyber.nyu.edu 1233
......
Can you still find the address of the secret vault?

I found this base address written on a post-it note: 0j0rU
Agh! But this time the address is in raw bytes!

>
......
```

In this challenge, with the base address in raw bytes, we need to find the address of the secret vault.

First, we similarly learn that PIE is on for the binary `vault3`.

Then, we display the symbol table of the binary `vault3` using `readelf` with `-Ws` flag to access offsets from the base address.

```
root@17b95fe8a8e6:~/wk0/vault3# readelf -Ws vault3
......
Symbol table '.symtab' contains 51 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
    39: 0000000000001269    26 FUNC    GLOBAL DEFAULT   16 secret_vault
......
```

We can see that the offset of the secret vault in hexadecimal is `0x1269`.

Finally, we write a script using `pwntools` to obtain the base address in raw bytes, convert it to an unsigned integer, calculate the address by `address = base address + offset`, and send it to the server in hexadecimal.

Part of the script is as follows. It should be noted that the function `pwntools.u64()` is used to convert 8 bytes (64 bits) of binary data into a 64-bit unsigned integer, which can also be replaced by a standard Python function `from_bytes(8, byteorder='little')`.

```python
from pwn import *
......
p = remote(URL, PORT)
......
print(p.recvuntil(b": ").decode())
base_addr = p.recv(8)
log.info(f"Receiving address in raw bytes: {base_addr}")
print(p.recvuntil(b"> ").decode())
base_addr = u64(base_addr)
offset = 0x1269
addr = base_addr + offset
p.sendline(hex(addr).encode())
log.info(f"Sending address in base 16: {hex(addr)}")

p.interactive()
```

The captured flag is `flag{th3_l34st_s1gn1f1c4nt_byt3_c0m3s_f1rst!_1511c380d483ffb6}`.

Vault 4 (100 pts)

```
root@17b95fe8a8e6:~/wk0/vault4# nc offsec-chalbroker.osiris.cyber.nyu.edu 1234
......
Can you still find the address of the secret vault?

I found this fake vault at: 0@@@FV
But it doesn't appear to be the right one.
Agh! and the vault coordinates are in raw bytes!

>
......
```

In this challenge, with the leaked fake address in raw bytes, we need to find the address of the secret vault.

First, we similarly learn that PIE is on for the binary `vault4`.

Then, we display the symbol table of the binary `vault4` using `readelf` with `-Ws` flag to access offsets from the base address.

```
root@17b95fe8a8e6:~/wk0/vault4# readelf -Ws vault4
......
Symbol table '.symtab' contains 49 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
......
    12: 0000000000004030     1 OBJECT  LOCAL  DEFAULT   26 fake_vault
    13: 0000000000004038     8 OBJECT  LOCAL  DEFAULT   26 secret_vault
......
```

We can see that the offset of the fake vault and the secret vault in hexadecimal are `0x4030` and `0x4038`.

Finally, we write a script using `pwntools` to obtain the leaked fake address in raw bytes, convert it to an unsigned integer, calculate the address by `address = fake address - fake offset + offset`, and send it to the server in raw bytes.

Part of the script is as follows. It should be noted that the function `pwntools.p64()` is used to pack a 64-bit unsigned integer into an 8-byte binary format, typically in little-endian order, which can also be replaced by a standard Python function `to_bytes(8, byteorder='little')`.

```python
from pwn import *
......
p = remote(URL, PORT)
......
print(p.recvuntil(b": ").decode())
fake_addr = p.recv(8)
log.info(f"Receiving address in raw bytes: {fake_addr}")
print(p.recvuntil(b"> ").decode())
fake_addr = u64(fake_addr)
fake_offset = 0x4030
offset = 0x4038
addr = fake_addr - fake_offset + offset
p.sendline(p64(addr))
log.info(f"Sending address in raw bytes: {p64(addr)}")

p.interactive()
```

The captured flag is `flag{b4ckw4rds_byt3_0rd3r_1s_n0t_s0_b4d!_8325aa4a8396b2db}`.

Baby glibc (50 pts)

```
root@17b95fe8a8e6:~/wk0/baby_glibc# nc offsec-chalbroker.osiris.cyber.nyu.edu 1235
......
Enough of PIEs 🥧 for today! What about some practice with ASLR and GLIBC?
I found glibc's `printf` function address written on a post-it note: ??
Agh! raw bytes again!

Can you tell me the address of the sleep() function?
>
......
```

In this challenge, with glibc's `printf()` function address in raw bytes, we need to find the address of glibc's `sleep()` function.

Firstly, for shared libraries glibc, all linked objects in the runtime end up with a randomized base address from ASLR. Therefore, `libc.so.6` 's `readelf` output values are all offsets from the library's base address.

Then, we use `readelf` with `-Ws` flag and `grep` to get the offsets of the `printf()` function and the `sleep()` function from the base address.

```
root@17b95fe8a8e6:~/wk0/baby_glibc# readelf -Ws libc.so.6 | grep -w 'printf'
  2922: 0000000000606f0   204 FUNC    GLOBAL DEFAULT   15 printf@@GLIBC_2.2.5
root@17b95fe8a8e6:~/wk0/baby_glibc# readelf -Ws libc.so.6 | grep -w 'sleep'
  1726: 00000000000ea570   106 FUNC    WEAK   DEFAULT   15 sleep@@GLIBC_2.2.5
```

We can see that the offset of the `printf()` function and the `sleep()` function in hexadecimal are `0x606f0` and `0xea570` .

Finally, we write a script using `pwntools` to obtain the `printf()` function address in raw bytes, convert it to an unsigned integer, calculate the `sleep()` function address by `sleep_addr = printf_addr - printf_offset + sleep_offset` , and send it to the server in hexadecimal.

Part of the script is as follows.

```python
from pwn import *
......
p = remote(URL, PORT)
......
print(p.recvuntil(b": ").decode())
printf_addr = p.recv(8)
log.info(f"Receiving address in raw bytes: {printf_addr}")
print(p.recvuntil(b"> ").decode())
printf_addr = u64(printf_addr)
printf_offset = 0x606f0
sleep_offset = 0xea570
sleep_addr = printf_addr - printf_offset + sleep_offset
p.sendline(hex(sleep_addr).encode())
log.info(f"Sending address in base 16: {hex(sleep_addr)}")

p.interactive()
```

The captured flag is `flag{y0ur_g0nna_g3t_re4lly_fam1li4r_w1th_Gl1bC!_7a6eeaaaf4dae15e}` .

Glibc (100 pts)

```
root@17b95fe8a8e6:~/wk0/glibc# nc offsec-chalbroker.osiris.cyber.nyu.edu 1236
......
Let's practice one more time finding GLIBC symbols! You know the drill 😎

I found glibc's `_IO_2_1_stdin_` address written on a post-it note: ﷺ
Can you tell me the address of glibc's `_IO_2_1_stdout_` variable?
>
......
```

In this challenge, with glibc's `_IO_2_1_stdin_` address in raw bytes, we need to find the address of glibc's `_IO_2_1_stdout_`.

Firstly and similarly, `libc.so.6`'s `readelf` output values are all offsets from the library's base address.

Then, we should notice that `_IO_2_1_stdin_` and `_IO_2_1_stdout_` are internal (private) symbols used by the C library and not listed in the standard (public) symbol table. Therefore, we should use `readelf` with `-Wa` flag and `grep` to get the offsets of `_IO_2_1_stdin_` and `_IO_2_1_stdout_` from the base address.

```
root@17b95fe8a8e6:~/wk0/glibc# readelf -Wa libc.so.6 | grep _IO_2_1_stdin_
......
  2298: 000000000021aaa0   224 OBJECT  GLOBAL DEFAULT   34 _IO_2_1_stdin_@@GLIBC_2.2.5
root@17b95fe8a8e6:~/wk0/glibc# readelf -Wa libc.so.6 | grep _IO_2_1_stdout_
......
   620: 000000000021b780   224 OBJECT  GLOBAL DEFAULT   34 _IO_2_1_stdout_@@GLIBC_2.2.5
```

We can see that the offsets of `_IO_2_1_stdin_` and `_IO_2_1_stdout_` in hexadecimal are `0x21aaa0` and `0x21b780`.

Finally, we write a script using `pwntools` to obtain the `_IO_2_1_stdin_` address in raw bytes, convert it to an unsigned integer, calculate the `_IO_2_1_stdout_` address by `stdout_addr = stdin_addr - stdin_offset + stdout_offset`, and send it to the server in raw bytes.

Part of the script is as follows.

```python
from pwn import *
......
p = remote(URL, PORT)
......
print(p.recvuntil(b": ").decode())
stdin_addr = p.recv(8)
log.info(f"Receiving address in raw bytes: {stdin_addr}")
print(p.recvuntil(b"> ").decode())
stdin_addr = u64(stdin_addr)
stdin_offset = 0x21aaa0
stdout_offset = 0x21b780
stdout_addr = stdin_addr - stdin_offset + stdout_offset
p.sendline(p64(stdout_addr))
log.info(f"Sending address in raw bytes: {p64(stdout_addr)}")

p.interactive()
```

The captured flag is `flag{3v3n_th3_st4nd4rd_1nput_and_0utput_4r3_d3f1n3d_1n_GLIBC!_1922020990dad5b4}`.

Are You Alive: Server Edition (5 pts)

```
root@17b95fe8a8e6:~/wk0# nc offsec-chalbroker.osiris.cyber.nyu.edu 1237
......
```

This challenge is a simple connection test to talk to the server. No action is required.

The captured flag is `flag{n0w_y0u_kn0w_wh4t_t0_d0_t0_w1n!_0eb983d281e65252}`.