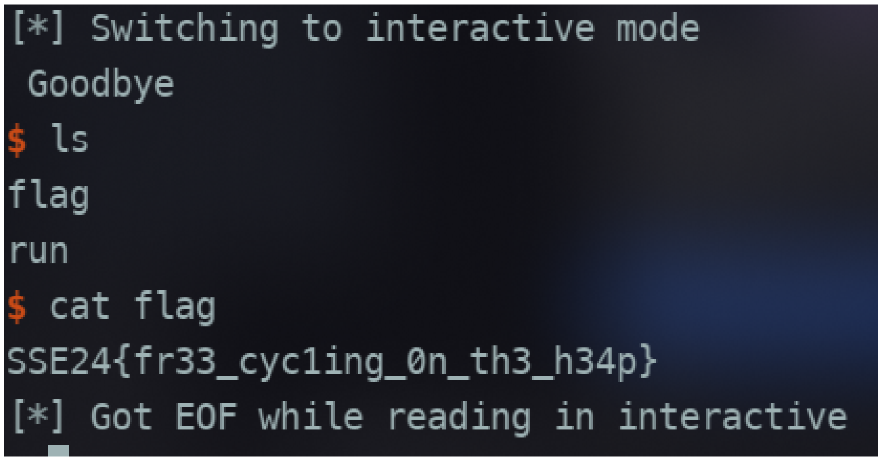
SSE 2024 Assignment 4

Yuvraj Talukdar (CS23D009)

April 16, 2024

Question 1.

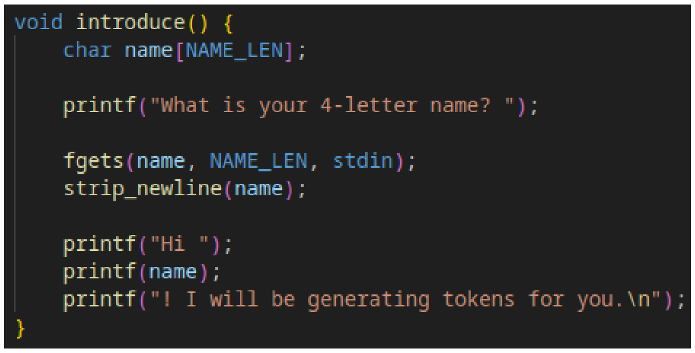
Challenge 1



```
[*] Switching to interactive mode
 Goodbye
$ ls
flag
run
$ cat flag
SSE24{fr33_cyc1ing_0n_th3_h34p}
[*] Got EOF while reading in interactive
```

Figure 1: Challenge 1 Exploit working proof.

In challange 1 sectok.c code we have a function void binsh() which calls the shell, but this function is not alled form any user callable fuction, so we need to hijack the execution to this function. The solution is to overwrite the stack address where the return address of main function is stored with the address of the binsh() function. Now in there server **ASLR** may be enabled, which will lead to the stack address where return address of main function is stored will be different for every launch. We need a way to lead a stack address so that we can calculate the offset to the stack location where main function return address is stored. Fourtunately We Also see a **format string** vulnerability in the introduce() function. If we enter %10$*llp* after "What is yout 4-letter name?" we return the address which is exactly 1 word before our required address. So to get the required stack address we add 8 to the leaked address.



```
void introduce() {
    char name[NAME_LEN];

    printf("What is your 4-letter name? ");

    fgets(name, NAME_LEN, stdin);
    strip_newline(name);

    printf("Hi ");
    printf(name);
    printf("! I will be generating tokens for you.\n");
}
```

Figure 2: Format String vulnerability in introduce function.

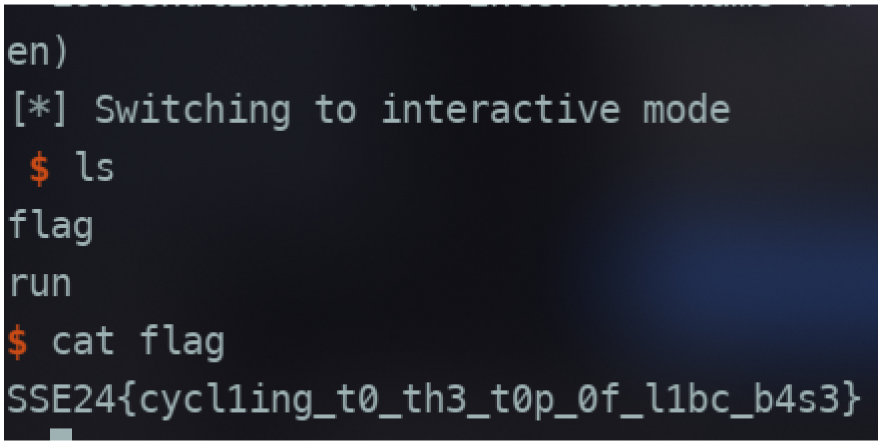
Once the required stack address to be overwrited is calculated we start with out heap attack using double free.

1. In glibc 2.27 double free attack is directly not allowed in tcache as there are check in the library, so we fill the 7 tcache bins and 2 fastbins using gentok.
2. Than we discard the 7 tokens from index 0 to 6 to add the filled tcache bins to free list and reach the fast bins.
3. We can now start the double free attack. Discard 7 than 8 and than again 8. As a result of double free token with index 7 is present twice in the free fastbins list.
4. Next we again fill the 7 tcache bins to reach the fast bins.
5. On reaching the fast bins we add the stack address we calculated where return address of main function is stored using the gentok function.
6. Next we add 2 dummy tokens.
7. And finally we add the address of binsh() function, this leads to the binsh function getting writen on the stack address where the return affress of the main function was stored.
8. Now on exiting the sectok app using x the binsh function gets called and we have a shell.

What just happened was after double free we had fastbin 0 fastbin 1 and fastbin 0 again on the free list. When we added the target stack address as a token the address got writen on fast bin 0's forward pointer. Next 2 token was to unlink the fastbin 1 and again fastbin 0, and since fastbin 0's forward pointer is pointing to the stack address the next token addition will write the token value in this case the address of binsh function on the stack address.

Question 2.

Challenge 2 Challenge 2 do not have any binsh function but it displays the base address of libc



```
en)
[*] Switching to interactive mode
 $ ls
flag
run
$ cat flag
SSE24{cycl1ing_t0_th3_t0p_0f_l1bc_b4s3}
```

Figure 3: Challenge 2 Exploit working proof.

which is dynamically loaded when the program is started. Since binsh function in not present we need to call the system() function and pass parameter "/bin/sh" to get a shell.

We target **__free_hook** function's address which is called inside the free function from libc. In pwntools we load the provided glibc v2.27 and set the base glibc address which the program provides during execution into ELF.address. Now when we call libc.symbols['__free_hook'] or libc.symbols['system'] we get the address as glibc base address + the particular function's offset.

1. Perform step 1 to 4 from challenge 1. With this double free operation will be complete.
2. Now add the address obtained from ;ibc.symbols['__free_hook'] as the new token.
3. Next add a dummy token.
4. After than add **"/bin/sh"** as token indexed 9.
5. And finally add the address obtained from libc.symbols['system'] as the new token , this will overwrite the address of __free_hook with the address of system function.
6. Now to launch the shell we discard the token with index 9. The value of the token is **"/bin/sh"** as as the distok function is called and "/bin/sh" is passed to the free function, inside the free function in place of __free_hook system function gets called and parameter as "/bin/sh", this finally launches the shell.

Extra Tools Used

1. pwntools- for stitching the payload.
2. GDB-PEDA https://github.com/longld/peda- for searching for the address of "/bin/sh" using command find "/bin/sh".
3. pwndbg- Another debugger.