SECURE SYSTEMS ENGINEERING

Lab Report on Heap Exploitation

Soham Tripathy (CS20B073), Arunesh J B (CS20B009)

31-02-2023

CS6570 - LAB 4 Report

Introduction

In this assignment we will be taking advatage of certain vulnerabilities using certain heap exploitation techniques.

Lab_1

Aim

Given source code (**users.c**) and executable (**users**). We have to change the SECRET to our team name by using heap exploitation techniques.

Our Approach

- By default, each thread has 64 singly-linked tcache bins. Each bin contains a maximum of 7 same-size chunks ranging from 24 to 1032 bytes on 64-bit systems.
- So when a chunk is freed the heap manager sees if the chunk fits in the tcache(correspomnding to the chunk size), if it does put it in the tcache, else obtain the arena lock and check if the chunk fits into the fastbin and put it there.
- So we are going to first allocate 9 chunks(perform 9 mallocs) by calling the adduser() procedure.
- Then we free 7 chunks by calling the removeuser() procedure, these go into the tcache.
- Now we perform a double free.Thus the fastbin state is as follows:- head -> chunk1 -> chunk2 -> chunk1 -> null
- Now we allocate 7 chunks (from the tcache).
- Next do a malloc with the username = address of the secret value. Hence the FD (chunk1) = address of SECRET. fastbin state : head -> chunk2 -> chunk1 -> addr of SECRET
- First malloc : head -> chunk1 -> addr of SECRET Second malloc : head -> addr of SECRET
- The next malloc we do is basically going to be a write operation into the SECRET variable. Hence we do a malloc with username =

"4rch41c" (teamname) and thus successfully changing the secret value to our team name.
- Now exit

Expoit String

We have provided a gen.py file whivh generates the exploit string

```
python gen.py
```

Result

Lab_2

Aim

We are given remote executable, its source code **users_2.c** and used glibc (**libc.so.6**). We have to use heap exploits to leak the flag corresponding to our team name from the flag_.txt file in the remote server.

Our Approach

- We connect to the remote server and start interacting with the executable using pwnlib.tubes.remote and extract the base address of libc.

```
a = p.recvline()
libc_addr = int(a.split()[4].decode(), 16)
print(libc_addr)
```

- __free_hook is a variable in libc which is pointing to a function which is always called by free. We get the address of this variable as follows:

```
elf = ELF("libc.so.6")
print(elf.sym.__free_hook)
```

- The sendlineafter(delim,data) procedure from pwnlib.tubes which recives data until the delim is encountered after which send the data. We use thisto interact with the executable of the remote server.

- By default, each thread has 64 singly-linked tcache bins. Each bin contains a maximum of 7 same-size chunks ranging from 24 to 1032 bytes on 64-bit systems.
- So when a chunk is freed the heap manager sees if the chunk fits in the tcache(correspomnding to the chunk size), if it does put it in the tcache, else obtain the arena lock and check if the chunk fits into the fastbin and put it there.
- So we are going to first allocate 9 chunks(perform 9 mallocs) by calling the adduser() procedure.
- Then we free 7 chunks by calling the removeuser() procedure, these go into the tcache.
- Now we perform a double free.Thus the fastbin state is as follows:- head -> chunk1 -> chunk2 -> chunk1 -> null
- Now we allocate 7 chunks (from the tcache).
- Next do a malloc with the username = address of __free_hook . Hence the FD (chunk1) = address of __free_hook (base address of libc + offset of __free_hook).

```
packed_no_byte_string = p64(libc_addr + elf.sym.__free_hook)
```

fastbin state : head -> chunk2 -> chunk1 -> address of __free_hook - First malloc : head -> chunk1 -> address of __free_hook Second malloc : head -> address of __free_hook - The next malloc we do is basically going to change the value which __free_hook points to the base_adress of libc + address of **\bin\sh**. Hence we do a malloc with username = p64(base_address_libc + offset(**\bin\sh**)) and hence when we do the next free we spawn a bash shell through which we open the flags file and retrive our flag.

Exploit String

We have provided a gen2.py file which uses pwn tools to interact with the remote user given and spawn a shell through which we retrive our flag.

```
python gen2.py
```

Result