Exploit1:

Approach: Buffer overflow (in the copy_file function)

Vulnerability:
The copy_file function uses a counter to track the file length and assign buffer value,
But there is no check on the counter to make sure it is less than the buffer limit. So it is possible to feed the buffer more than 4000 chars and overwrite the return address to execute the shellcode.

Implementation:
I create a buffer than is 4017 chars in length. And first fill with dot (same as the warm up video). Then I put in the shellcode and the eip location (found using gdb to check the stack frame). Then, put the char array into a txt file and upload. But I noticed that the check virus function will scan the /bin/sh and stop the file being copied by the copy_file function. Then after I noticed that the check for viruses function use a buffer with length of 1024. I decided to add dots in front of the shellcode to separate bin/sh into two scan loop to avoid this issue. After that, the file will be able to make the buffer in copy_file function overflow and overwrite the eip and point to the start position of shellcode (also found using gdb) and execute the shellcode when it returns.

Fix:
Limit the copy_file counter same as (smaller than) the buffer size to avoid overwriting the eip.
Or use function like snprintf.

Exploit 2:

Approach: Formatted string (in the print_version function)

Vulnerability: the print_version function will call printf (version_info). Since printf is not a safe way to write to output, we could pass a formatted string to the printf and use it to overwrite the return address.

Implementation:

Formatted String: `\x9c\xdb\xbf\xff\x9e\xdb\xbf\xff%56284x%24$hn%9147x%25$hn\xff\xff\xff`

First 8 are the address of eip,
The `%56284x%24$hn%9147x%25$hn` is used to represent the value equals to the shellcode address.
The padding \xff is used to align the start of shellcode.

I use the argv[0] to pass the formatted string and the string will be loaded into the version_info stack. Then the target is to replace the eip with the starting location of the shellcode. I first use gdb to find the address of eip from the stack frame when print_version is executed. This address is the one that need to be overwritten. Next step is to find the address of the starting point of shellcode. After I have these two values. I noticed that the address of shellcode is relatively large (0xffbfdc04). So I decided to split it into two parts (two overwrite step). And another issue is the

upper part is larger than lower part, which means I have to reverse the overwrite order. After serval trail, I did get the address overwritten.

Fix:

Use a fixed formatted string like “%s”, so the user input is used as pure data string instead of formatted string.
Do a safe check if you have to let user input treated as a formatted string. Filer out the potential symbol that could read/write memory.