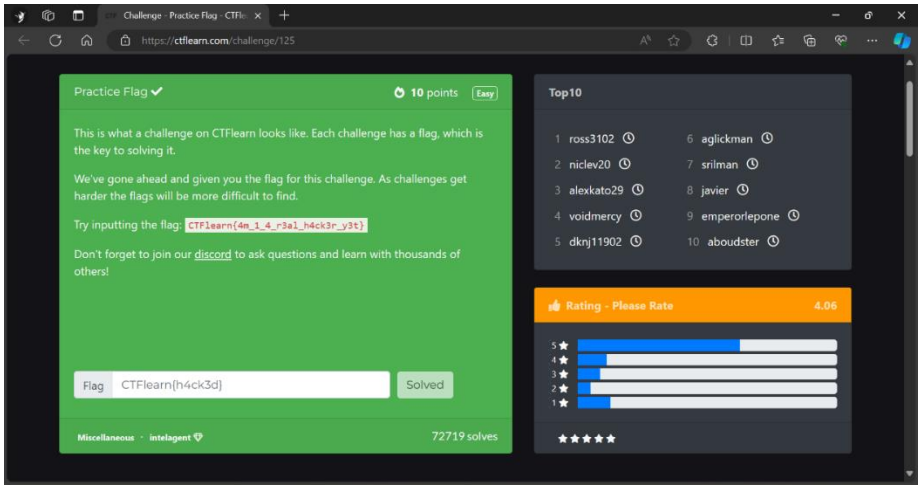
Capture The Flag (CTF)

1. Documentation

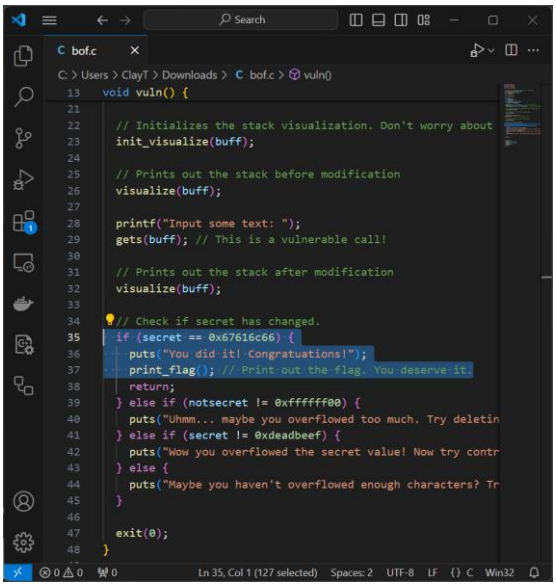
Reading through the ctf101 Binary Exploitation CTF handbook, I found detailed information regarding the typical layout of binaries (executables) and typical buffer overflow, heap, and format string exploit information. It also covered binary security (compilation flags and stack canaries).

2. Practice Flag

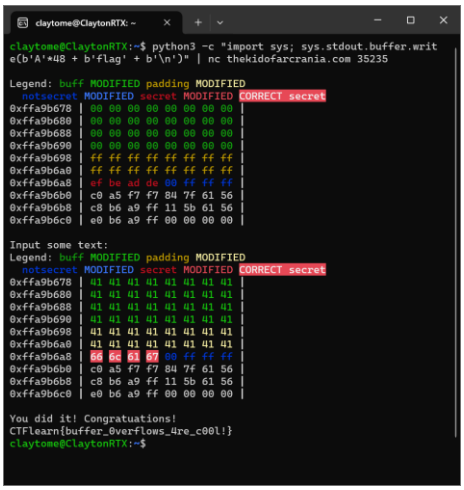


This challenge is an introduction to CTF, outlining the basic principles and layout of a typical challenge. All I had to do to complete this challenge was copy and paste the provided flag.

3. Simple bof

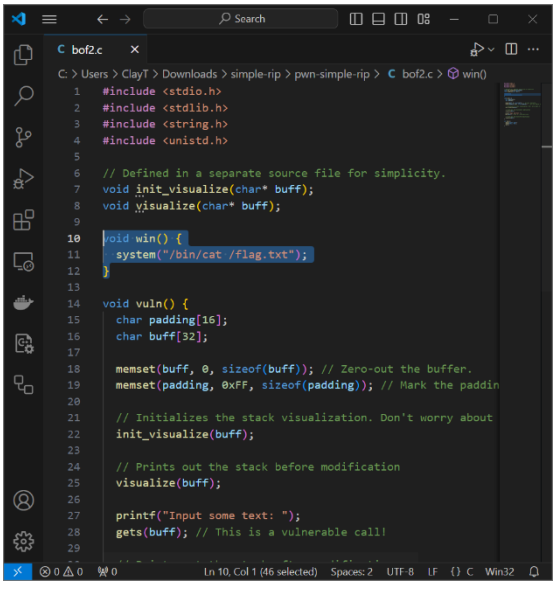


The first step in solving this challenge is a static analysis of the provided code. Doing so reveals an if check that will print the flag if the secret value is equal to 0x67616c66 or /x67/x61/x6c/x66. The value is ‘galf’ when converted into ASCII in this order, which is ‘flag’ backwards when popped off the stack in little-endian reverse order (/x66/x6c/x61/x67).

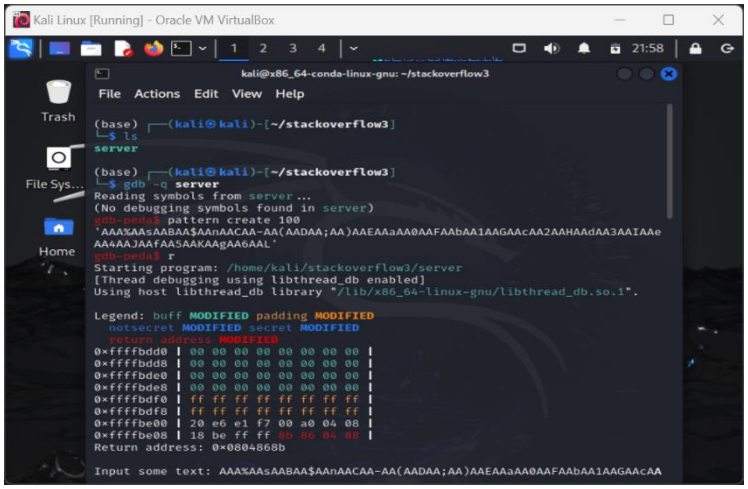


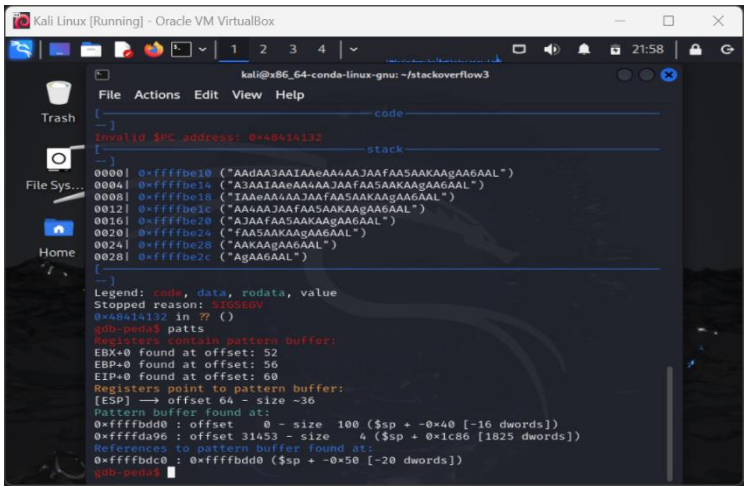
I typed and passed a string of 48 ‘A’s followed by the term ‘flag’ to overwrite the **secret** value, the flag is printed out because the if check is satisfied (python -c "import sys; sys.stdout.buffer.write(b'A'*48 + b'flag' + b'\n')" | nc thekidofarcrania.com 35235).

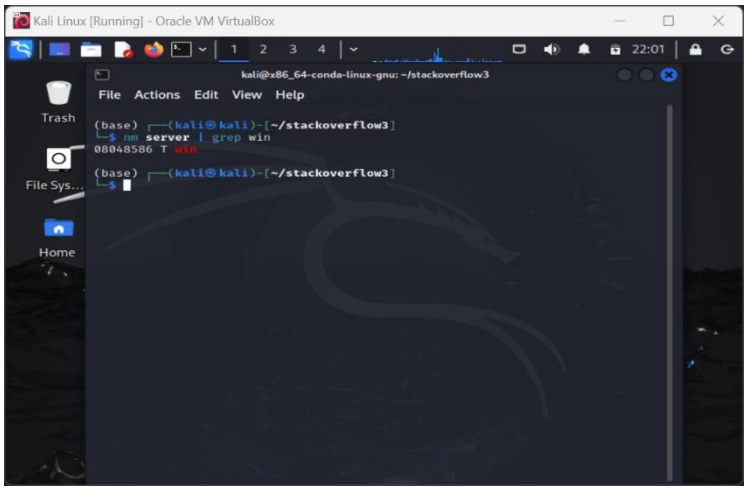
4. RIP my bof



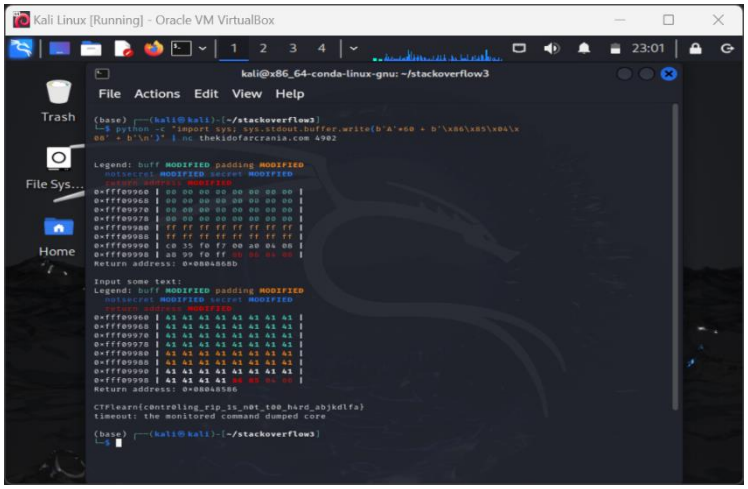
Starting again with a static analysis of the provided code, I discovered a **system** call function that will **cat** the contents of /bin/flag.txt. I must redirect the flow of the program to this function to print the flag.







I used **gdb** (**gdb -q server**, **pattern create 100**, **r**, and **patts**) to overflow the buffer and find the offset of EIP (the instruction pointer to the return address, 60). I was then able to use **nm server | grep win** to get the address of the **win** function.



With this information, I am able to pass ‘A’s to fill the offset up to the EIP followed by the return address of the win function and a newline input termination character (\n) to print the flag (python -c "import sys; sys.stdout.buffer.write(b'A'*60 + b'\x86\x85\x04\x08' + b'\n')" | nc thekidofarcrania.com 4902).