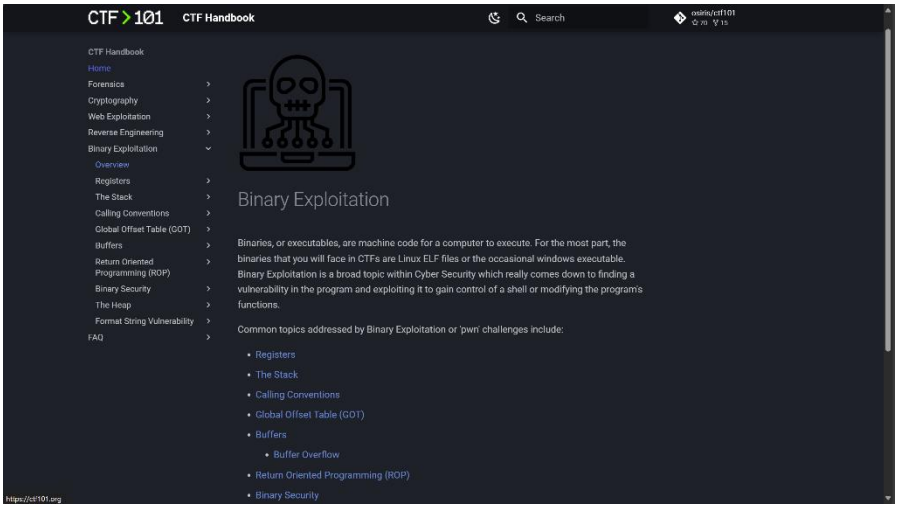
Capture The Flag (CTF)

By Sebastian Medina

1. **CTF 101:**

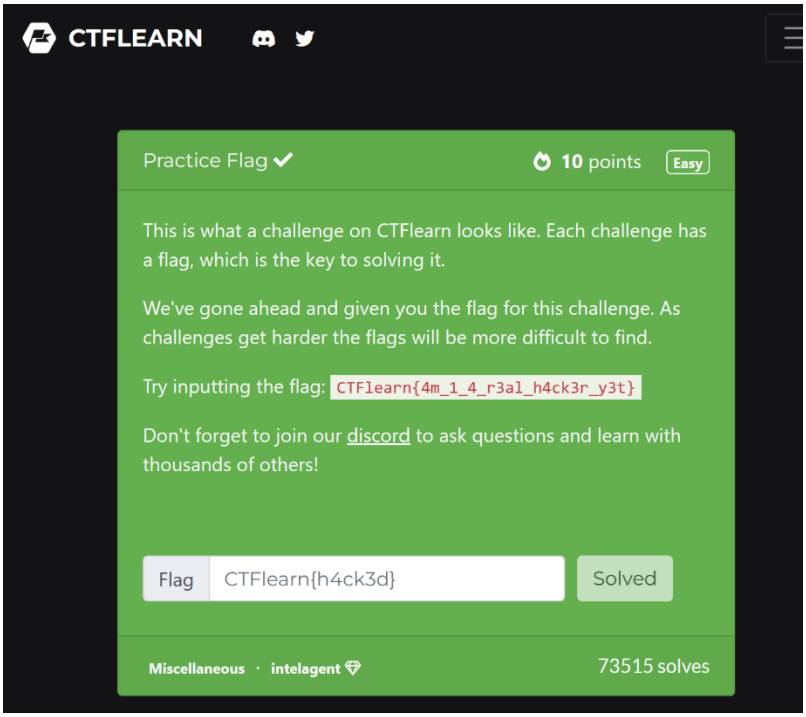
   For this first task I clicked on the link "https://ctf101.org/binary-exploitation/overview/" where it took me to a page called the CTF Handbook. On this page it talked about Binary Exploitation. This talks about how in CTF most of the binaries you will encounter are Linux ELF or windows executable. Also, whenever you are dealing with binaries in CTF you are looking for a vulnerability within the binary and exploiting it to gain control of a root shell or to have the program complete a certain task.

   To really understand CTF and binary exploitation the website goes over registers, stacks, and Global Offset Table also know as GOT. However, this is not it they also go over Buffers and how a Buffer Overflow works as well as what is Return Oriented Programming aka ROP. Within this website they also go over many Binary Securities. Some of the examples they listed for Binary Security are No eXecute (NX), Address Space Layout Randomization (ASLR), Stack Canaries, and Relocation Read-only (RELRO). Most of these were explained in the tryhackme and are ways that can stop threats from attacking your system. Knowing what each of these topics are a crucial step completing CTF challenges on the binaries.



2. Practice Flag

For this task I created an account at ctflearn.com. Then I went to the challenge practice flag. This challenge was super simple, all the challenge was doing is explain how a flag is going to look when you find it in the other challenges. Also, it goes over how to enter them into the answer box.

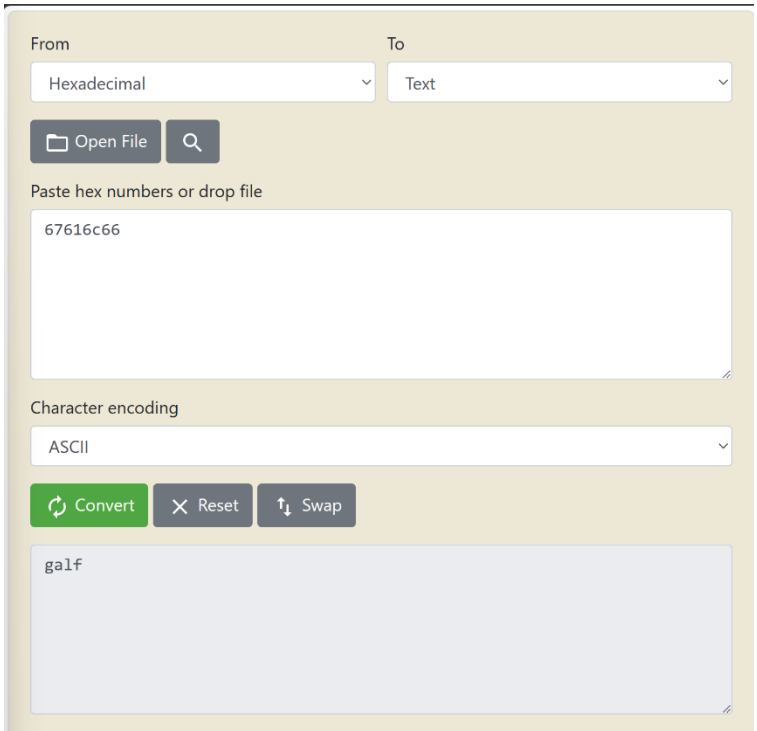


3. Simple bof

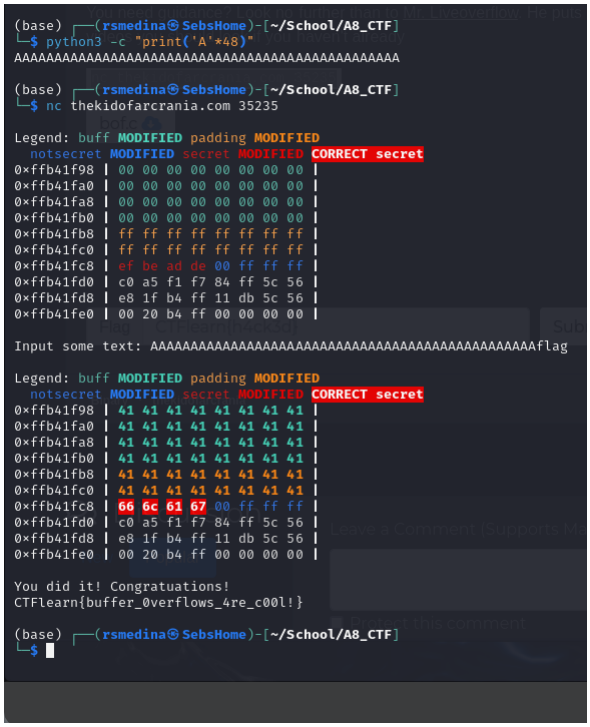
For this task we had to complete a simple bof challenge and find the flag within the challenge. From the start of the challenge, I downloaded a file called bof.c. Right away the first thing I did was view the content that the file was containing with “vi”. When I got into the file, I was reviewing the code and saw that an if statement was checking if secret equals a certain hex value.

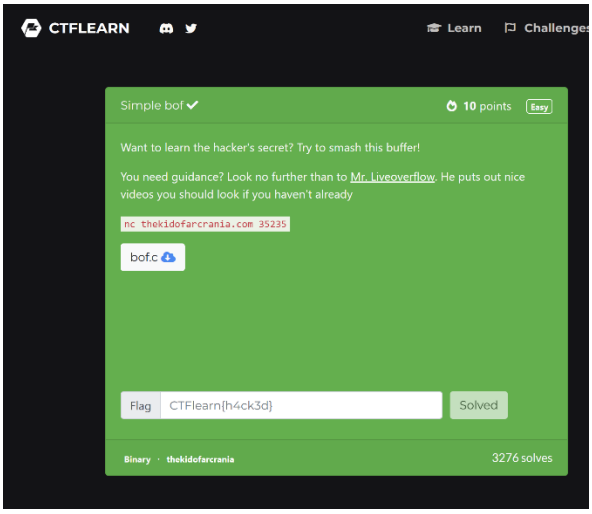
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

// Defined in a separate source file for simplicity.
void init_visualize(char* buff);
void visualize(char* buff);
void safeguard();

void print_flag();

void vuln() {
  char padding[16];
  char buff[32];
  int notsecret = 0xffffff00;
  int secret = 0xdeadbeef;

  memset(buff, 0, sizeof(buff)); // Zero-out the buffer.
  memset(padding, 0xFF, sizeof(padding)); // Zero-out the padding.

  // Initializes the stack visualization. Don't worry about it!
  init_visualize(buff);

  // Prints out the stack before modification
  visualize(buff);

  printf("Input some text: ");
  gets(buff); // This is a vulnerable call!

  // Prints out the stack after modification
  visualize(buff);

  // Check if secret has changed.
  if (secret == 0x67616c66) {
    puts("You did it! Congratuations!");
    print_flag(); // Print out the flag. You deserve it.
    return;
  } else if (notsecret != 0xffffff00) {
    puts("Uhmm... maybe you overflowed too much. Try deleting a few characters.");
  } else if (secret != 0xdeadbeef) {
    puts("Wow you overflowed the secret value! Now try controlling the value of it!");
  } else {
    puts("Maybe you haven't overflowed enough characters? Try again?");
  }

  exit(0);
}

int main() {
  setbuf(stdout, NULL);
  setbuf(stdin, NULL);
  safeguard();
  vuln();
}
```

Now that I saw this, I copied the Hex value which was “67616c66” and put it though a translator to translate hex to English. When I did this, I got “galf” Witch is flag backwards. The reason why flag is backwards instead of forwards is because in a stack they are read last in first out meaning that in the stack galf will be read as flag.



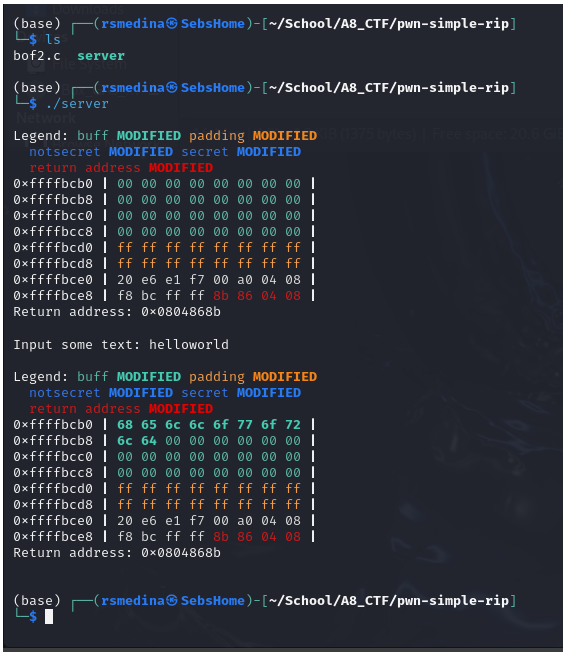
After I made that discovery, I then ran the command “nc thekidofarcrania.com 35235” that took me to a visualization of the stack and asked me to input characters. I then inspected the image of the stack and noticed that the secret/flag in the buffer is about 48 bytes. I then ran the command “python3 -c “print(‘A’*48)”” Witch gave me 48 A. I then ran the command “nc thekidofarcrania.com 35235” again and when it asked me to input characters, I took the 48 A and added “flag” to the end of it and it spit out the flag I needed completing this challenge.



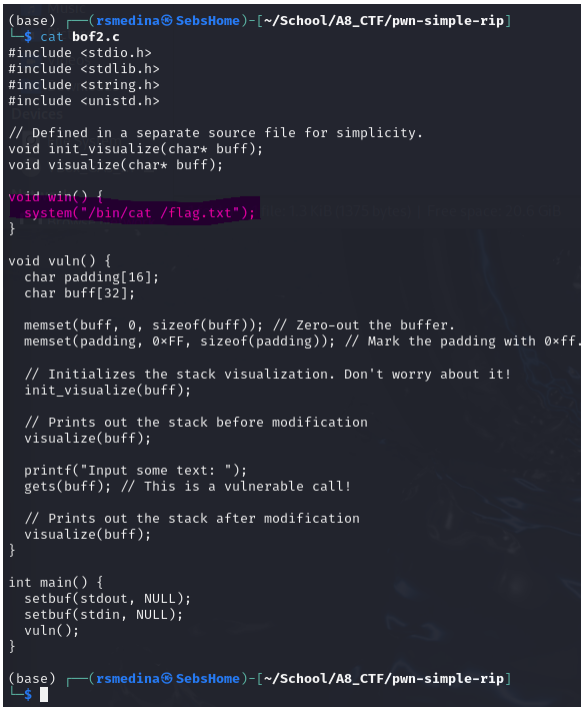


4. RIP my bof

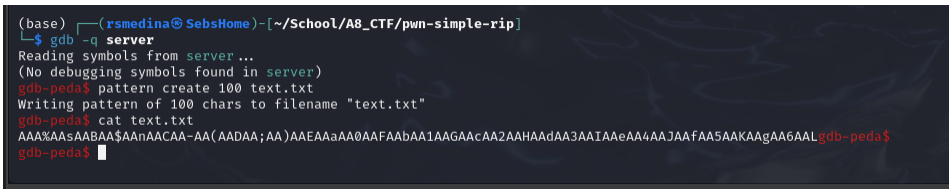
When I first got into this task I ran "./server" to see what it does. When I did this I saw this I noticed that it probably another Bufferoverflow exploitation that we are going to have to accomplish.



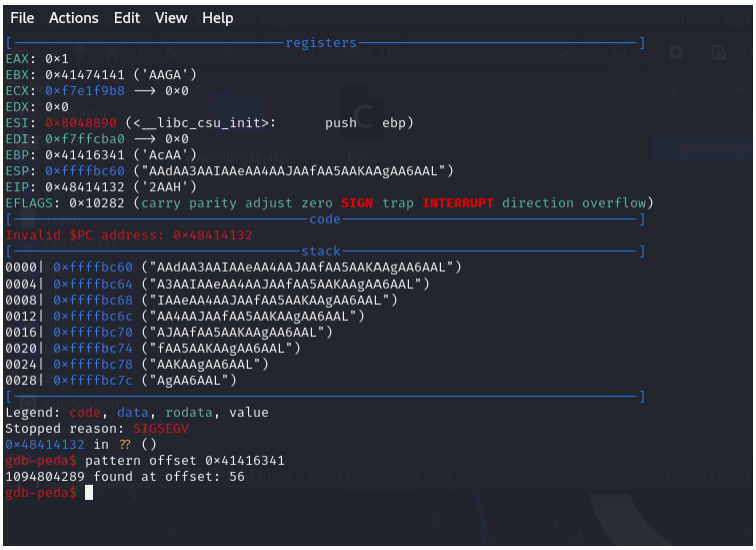
Next I check the content of bof2.c and notice that the win () function has a file named file.txt so I kept that in mind for later when I run a pattern though that function.



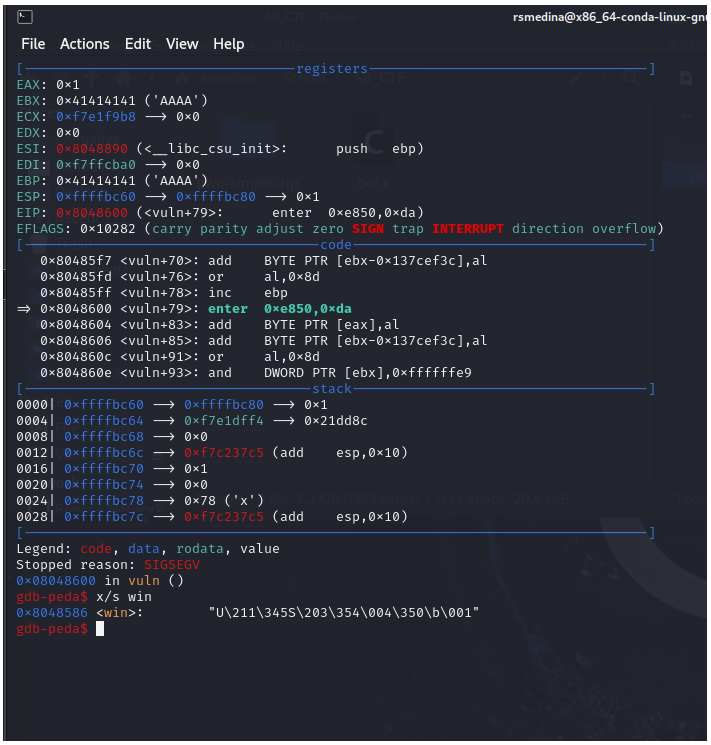
After gathering all that information, I ran “gdb server” so that I can debug server and figure out the information needed to complete the rest of this challenge. Also, when I was in gdb I created a pattern and stored it in a file named text.txt.



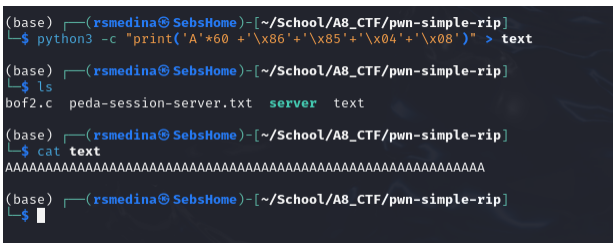
No that I created that pattern I ran server with the pattern that is in text.txt with the command “r < text.txt” and I got what is shown in the image below. After I ran that command I grabbed the EBP address and used it to find the pattern offset that is 56 this is also shown in the image below”



I then edit the text.txt file to only have A with the command “python -c “print(‘A’*60)” > text” this put 60 A into text.txt and changed the name to text. Now with this new file I ran “r < text” in gdb which can be seen below. What this did was run the text file though server. After I had done this, I then ran the command “x/s win” because I remembered that function from exploring the “bof2.c” file, this can be seen in the image below as well. After I ran that command, I gathered all the information needed to get the flag.



Now that I have all the information to get the flag I ran the command “python -c “print(‘A’*60 +‘\x86’+‘\x85’+‘\x04’+‘\x08’)” > text” witch loaded the text file with 60 A and is concatenated with the hexadecimal byte values.



Now that this is stored with in the file text I ran “nc thekidofarcrania.com 4902 < text”. When I ran this command, it spit out the flag “CTFlearn{c0ntr0ling_rlp_ls_n0t_t00_h4rd_abjkdlfa}” and when I put it into the answer box it was correct completing this challenge.