Buffer Overflow Code Explanation

BIMAL VARGHESE
FISAT

February 18, 2015

Sample Code

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main(inr argc,char* argv[])
{
        char buf[400]

        if (argc <2)
        {
                printf("Please enter command line arguments\n");
                exit(0);
        }

        strcpy(buf,argv[1]);
        return 0;
}
```

Explanation

The *strcpy()* function does not check whether or not the number of bytes being copied to the buffer is less than the size of the buffer.The *strcpy()* function copies the command line argument passed to the program directly, without checking the number of bytes. As a result,if the user give an input string of length greater than 400(a little more than 400), the buffer over flows and the memory address adjacent to buffer, **ebp** and **eip**, gets overwritten.

```
gcc example.c -o example
./example 'python -c 'print "A" *412'
```

When the above program executed with given command line arguments, the **ebp** register get overflowed with 0x41, which is the hex value of '**A**'.

Overflow Attack

To perform overflow attack, we need to over flow the **eip** register, because **eip** points to the next instruction. **eip** lies next to the **ebp** register, and hence is just 4 bytes away. So if we run

```
gcc example.c -o example
./example ’python -c ’print "A" *416’
```

the **eip** pointer gets over written.

Now instead of giving a single string value, if we insert a code which can spawn a shell then we will have a shell where commands can be executed. For that, we have to point the eip to a assembly code that will spawn a shell (*/bin/sh*).

```
gcc example.c -o example
./example ’python -c ’print "A" *412 + <shell code>’
```

On executing the above program, a shell is spawned with the same privileges as the program. If the program is having a high privilege then attacker will have elevates access rights. If the root user have created the binary, then spawned shell give complete access to the attacker.