

# Lab 8.1: Introduction to Buffer Overflows

## IN618 Security

April 5, 2016

### Introduction

The vulnerabilities we have explored so far in this paper are fairly “modern” ones, but in today’s lab we will examine a “classic” vulnerability, the buffer overflow. Many of the most critical security holes in computer systems have come from buffer overflow vulnerabilities. In fact, the first major Internet worm, the 1988 *Morris Worm*, exploited a buffer overflow.

Another difference between a buffer overflow vulnerability and the vulnerabilities we have looked at so far this semester is that a buffer overflow requires a higher degree of technical sophistication to exploit it.

### 1 A vulnerable example

Using a text editor, write the C program below. Save your file with the name `overflow.c`.

*file: overflow.c*<sup>1</sup>

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

int main(void)
{
    char buff[15];
    int pass = 0;
    printf("\nEnter the password : ");
    gets(buff);
    if(strcmp(buff, "in618"))
    {
        printf ("\nWrong Password \n");
    }
    else
    {
        printf ("\nCorrect Password \n");
        pass = 1;
    }

    if(pass)
    {
        /* Now Give root or admin rights to user*/
    }
}
```

---

<sup>1</sup>Source: <http://www.thegeekstuff.com/2013/06/buffer-overflow/>

```

        printf ("\nRoot privileges given to the user \n");
    }

    return 0;
}

```

## 2 Compile and run the program

We will compile and run our program on the `sec-student.foo.org.nz` server. Use WinSCP to upload your source code file to your home directory on the server. Then, use PuTTY to get a shell session on your server. Compile your program with the command

```
gcc -fno-stack-protector -o overflow overflow.c
```

You will get a warning message, but your program should compile correctly provided that you typed it in without errors.

This will produce an executable file named `overflow`. Run the program with the command

```
./overflow
```

Try it first by entering the correct password and see what happens. Next try an incorrect password, using five `a` characters and see what happens.

Now let's try something more interesting. Run the program and use 32 `a` characters. What happens this time? Do you know why?

What happens when you try 60 `a`s?

## 3 A little debugging

Whether or not the program gives us "Root privileges" depends on the variable `pass`. Let's see what is happening to it. In your source code file, right after the line that reads `gets(buff)`, add the line

```
printf("pass: %d\n", pass);
```

Then recompile your program and re-run your tests. What is happening with `pass`? Do you know why?