Name:______________________________________Student ID:_______________

You will need:
RedHat Linux 7.3 (VM)
Windows XP Pro (VM)
Windows 95 (VM)
A computer with internet access

COS80013 Internet Security

Demo 3 (week 3)

In this lab you will experiment with Buffer overflows in the C language.

1. Using Virtual Machine Launcher, start up the COS80013 / **RedHat Linux with local network** VM image.

   Alternatively zipped copies are on Cloudstor here:
   Virtual Machines - OneDrive

Log in as *student* (user)
student (password)

Let's write a C program:

Start the editor thus:

```
pico memtest1.c
```

Type in the following code:

```
/* memtest1.c*/
#include <stdio.h>
#define SIZE 8
void test(int*, char*);
int main(){
      int i = 0;
      char buf[SIZE];
      printf("Type in 5-20 chars into the text buffer\n");
      printf("Watch the value of i \n");
      printf("it will be corrupted when you exceed %i chars\n",SIZE);
      printf("Type \"q\" to exit:\n");
      printf("\t| i posn\t| buf start\t| buf end\t| i value\n");
      do {
            test(&i, buf);
            i++;
      }while(buf[0] != 'q');
      return 0;
}
void test(int *j, char* buf){
      scanf("%s",buf);
      printf("OK.\t| %u\t| %u\t| %u\t| %d\n",j, buf, &buf[SIZE],*j);
      return;
}
```

Use
Control+O to write to file (followed by ENTER)
Control+X to exit

Name:____________________________________Student ID:______________

Compile thus:

```
gcc -o memtest1 memtest1.c     (creates executable)
```

When the function is called, ***i*** is passed by reference. The ***test*** function can access this variable.
buf, being an array, is passed by reference, so the memory location is shared between the main function and ***test***.

```
chmod +x memtest1              (adds execute permissions)
./memtest1                     (runs the program)
```

Where in memory (the address) is *i*? What is its value?

	Location (posn)	**Comments**
i		*1*
buffer start		*Start of user input*
buffer end		*should be \0*
buffer size		
Bytes between buffer start and i		*e.g. 3221223812 – 3221223800 = 12* *input of more than 12 bytes should start to corrupt memory (i.e. value of i)*

Try to find the largest number of chars before a buffer overflow or segfault.
Try to make the value of ***i*** jump.

When a buffer fills up, it writes forwards or backwards depending on the CPU and operating system.

Type in a few strings smaller than 10 characters – note that the integer ***i*** is counting correctly.

Play with the program (type stuff into it) and record your observations here. Progressively input more characters into the program and note changes on behaviour.

Test No (i)	**Text typed into buf**	**Number of characters**	**Behaviour**
0	*12345*	*5*	*Increments i*
1			
2			

Name:____**Student ID:**_______________

3			
4			
5			*i jumps to …?*
6			
7			
8			

What is the distance (in bytes) between ***buffer start*** and ***i***?

***n* = (&*i* - buffer start)** =

Try inputting that many characters (***n***) to overwrite the variable ***i***.
Try it. Does *i* change?

If you type 12 characters and <Enter>, the computer inserts a NULL character ('\0') in the 13th position.

If it crashes, just start it up again and keep typing longer strings. Watch the value of i.

Try inputting n + 1 characters to overwrite the variable ***i***.
Try it. Does *i* change?

If a string is really long, you will crash the program (Linux will report a Segmentation Fault).
Code can be injected into the window between buffer overflow and segmentation fault, overwritingotherpartsoftheprogramsuch as the ***return address*** andthe ***EBP***

More background here:
http://www.tenouk.com/Bufferoverflowc/Bufferoverflow2a.html

2. Try this program:

```
/* memtest2.c*/
#include <stdio.h>
int main()
{
     char first[12];
     char last[12];

     printf("Type in your first name: ");
     gets(first);
```

Name:______________________________________Student ID:_______________

```
      printf("Type in your last name: ");
      gets(last);

      printf("Hello %s %s\n", first, last);

      return 0;
}
```

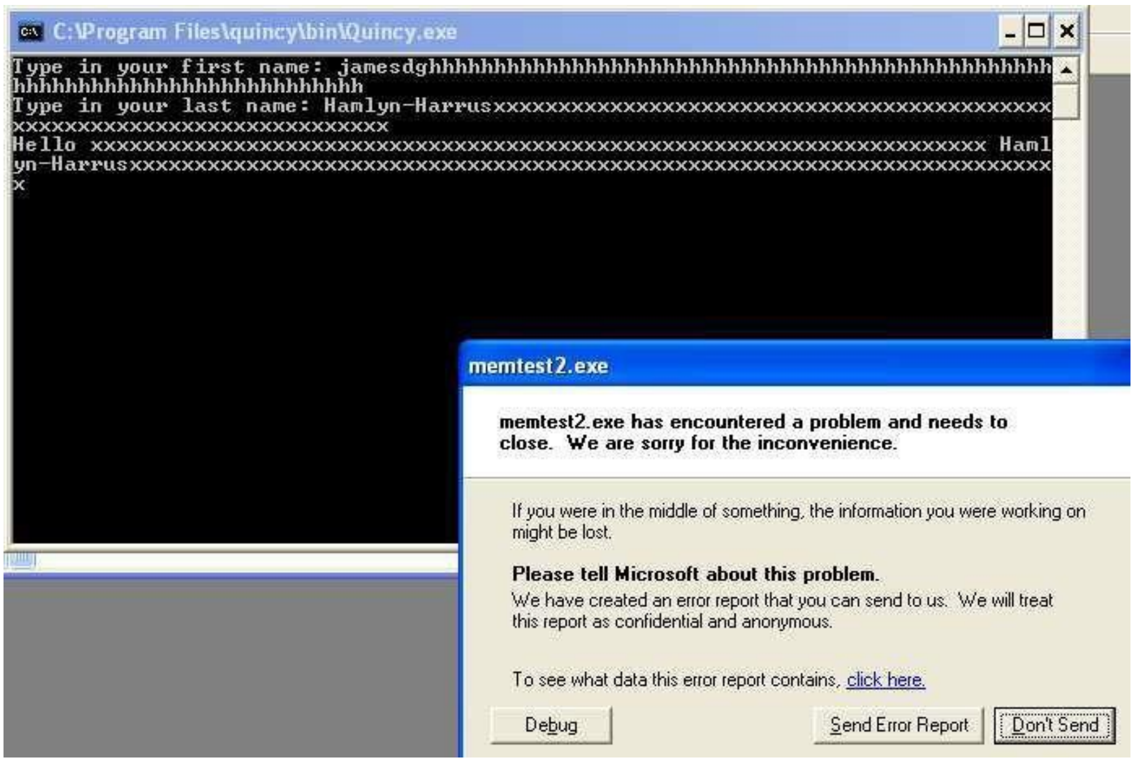
The compiler warning about **gets** will not stop compilation. You can ignore it.

Unlike ***scanf, gets*** lets you type in spaces and other non-printing characters, so is allows an attacker to enter executable code.

If you put in long strings, you can overflow one string with the contents of the other. If they are big enough, you will get a segmentation fault. If the size is just right, you will overwrite a function somewhere else in the program.

How many characters must you input into the last name to overwrite the first name (try it)?

3. (optional) You can also try these programs in Windows using the Quincy IDE (in the **Windows 95** VM). If a 'segmentation error' occurs, Windows will pop up this message:



C:\Program Files\quincy\bin\Quincy.exe

```
Type in your first name: jamesdghhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhh
Type in your last name: Hamlyn-Harrusxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Hello xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Haml
yn-Harrusxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
x
```

memtest2.exe

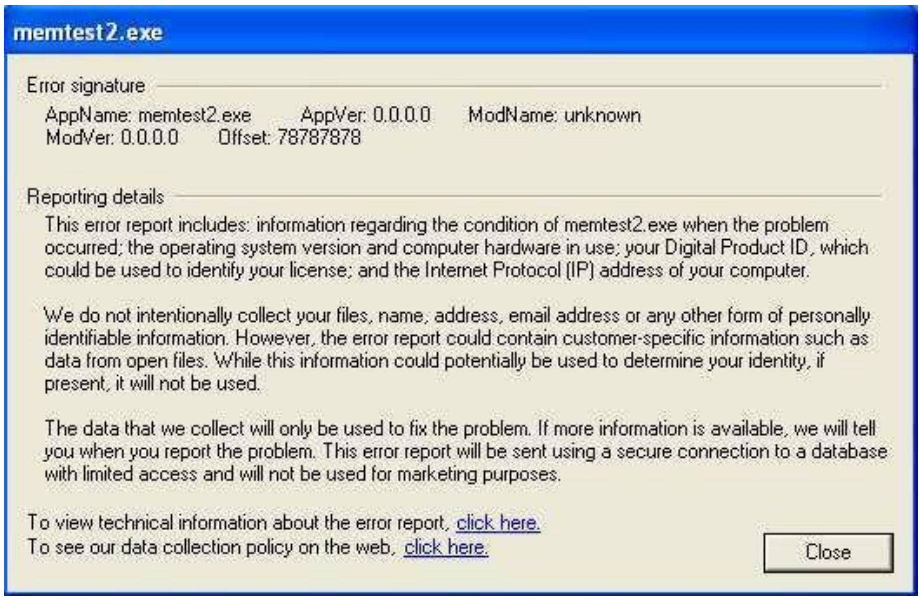
memtest2.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

If you click on click here, you will see the contents of the stack pointer, which contains the hex value of some of the string you typed in.


memtest2.exe

Error signature
AppName: memtest2.exe AppVer: 0.0.0.0 ModName: unknown
ModVer: 0.0.0.0 Offset: 78787878

Reporting details
This error report includes: information regarding the condition of memtest2.exe when the problem occurred; the operating system version and computer hardware in use; your Digital Product ID, which could be used to identify your license; and the Internet Protocol (IP) address of your computer.

We do not intentionally collect your files, name, address, email address or any other form of personally identifiable information. However, the error report could contain customer-specific information such as data from open files. While this information could potentially be used to determine your identity, if present, it will not be used.

The data that we collect will only be used to fix the problem. If more information is available, we will tell you when you report the problem. This error report will be sent using a secure connection to a database with limited access and will not be used for marketing purposes.

To view technical information about the error report, click here.
To see our data collection policy on the web, click here.

Note: 78787878 is hex for xxxx.

Hackers can use this feedback message to figure out which bytes in their exploit string will be copied into the stack pointer. Changing these critical bytes to the location of some (of their) executable code allows the exploit to run their code. With Linux, it's a bit harder, but not much.

4. Re-write ***memtest1.c*** or ***memtest2.c*** to prevent excess characters being pasted into the char arrays:

HINT:

replace ***scanf("%s")*** with ***scanf("%10s")***

replace ***gets*** with ***fgets***

```
usage: char * fgets ( char * str, int num, FILE * stream );
```
e.g. ***fgets(first, 12, stdin)***

Unlike gets, fgets leaves characters in the keyboard buffer.
In windows you can clean up the input stream with

```
        fflush(stdin);
```

In Linux you can do the same with

```
        char ch;  /*declare once only*/
        while ((ch = getchar( )) != '\n' && ch != EOF);
```

example code in ***safegets.c***
on Blackboard.

5. Using Google, look up the various "safe" string manipulating functions available in C. (e.g. strncpy, strncat)
What does the *n* do?

How is value of *n* determined?

6. Start up the COS80013 / ***Windows XP Pro with local network*** VM image
Surf to `http://www.server.com/remote`

Read the file: **readmeb.txt**
Follow the instructions – you will need to log in to the Linux machine to start the server, and then access it from the XP machine using the **passwordclient**.

7. On the host PC, surf to http://www.vividmachines.com/shellcode/shellcode.html and read up on injecting shellcode into Linux and Windows processes.

What does . *GetProcAddress()* do?

8. Using Google, look for tools and programs for checking a C program for potential buffer overflow vulnerabilities and memory leak testers. **What are the names of some?**

9. Shut down all VMWare images and browsers and log off.

End of Lab