**Lab 05. Function Call and Return**

作业提交要求：提交一个word文档，以 学号\_姓名.doc 命名

Practice1：decode part2 的key3,key4/解题思路说明/运行结果截图（截图中含你的姓名学号等信息）

Practice2：提供以表格方式填写的stackframe结构，当程序执行swap2中的temp=\*x;指令时

Practice3：提交输入字符串/解题思路说明/运行结果截图（截图中含你的姓名学号等信息）

**Practice 1: Decode Lab**

Continue on Decode Lab: Understanding a Secret Message, find key3 and key4 and get the right message. When you do so, you may follow the teacher's instruction [decode lab part2.doc](http://swjx.scu.edu.cn/moodle/file.php/49/systemlevelprogramming/week5/decode_lab_part2.doc).

**Practice 2: Stack Frame Drawing**

Disassemble the file swap.c in VC debugger and execute instructions one by one. Understand those instructions, observe changes in memory window and register files when execute each instructions.

-------------------------------------code: swap.c-----------------------------------------  
#include <stdio.h>  
  
void swap1(int x, int y) {

int temp;   
temp=x;   
x=y;   
y=temp;

}   
  
void swap2(int\* x, int\* y) {

int temp;   
temp=\*x;   
\*x=\*y;   
\*y=temp;

}   
int main() {

int a=1,b=2;   
swap1(a,b);   
printf("after swap1,a=%d, b=%d\n",a,b);   
swap2(&a,&b);   
printf("after swap1,a=%d, b=%d\n",a,b);

}   
------------------------------------code: swap.c----------------------------------------------

While execute each assembly instruction, get a series of snapshots to show the contents of Stack Frame and registers such as EIP, EBP, ESP etc. You need to explain each instruction behavior with details.

**Pactice 3: Bufbomb lab**

In this problem, you will mount a buffer overflow attack on your own program. As stated earlier, we do not condone using this or any other form of attack to gain unauthorized access to a system, but by doing this exercise, you will learn a lot about machine-level programming. Download the file [bufbomb.c](http://swjx.scu.edu.cn/moodle/file.php/49/systemlevelprogramming/week5/bufbomb.c) and compile it to create an executable program.   
  
In bufbomb.c, you will find the following functions:

1 int getbuf()  
2 {  
3 char buf[12];  
4 getxs(buf);  
5 return 1;  
6 }  
7  
8 void test()  
9 {  
10 int val;  
11 printf("Type Hex string:");  
12 val = getbuf();  
13 printf("getbuf returned 0x%x\n", val);  
14 }

The function getxs (also in bufbomb.c) is similar to the library gets, except that it reads characters encoded as pairs of hex digits. For example, to give it a string "0123", the user would type in the string "30 31 32 33". The function ignores blank characters. Recall that decimal digit x has ASCII representation 0x3x. A typical execution of the program is as follows:

> bufbomb  
Type Hex string: 30 31 32 33  
getbuf returned 0x1

Looking at the code for the getbuf function, it seems quite apparent that it will return value whenever it is called. It appears as if the call to getxs has no effect. Your task is to make getbuf return (0xdeadbeef) to test, simply by typing an appropriate hexadecimal string to the prompt.   
  
Here are some ideas that will help you solve the problem:

* Use VC 6.0 debugger to create a disassembled version of bufbomb. Study this closely to determine how the stack frame for getbuf is organized and how overflowing the buffer will alter the saved program state.
* Set a breakpoint within getbuf and run to this break point. Determine such parameters as the value of ebp and the saved value of any state that will be overwritten when you overflow the buffer.
* Determining the byte encoding of instruction sequences by hand is tedious and prone to errors. You can let tools do all of the work by writing an assembly code file containing the instructions and data you want to put on the stack. Assemble this file with disassembler.
* Keep in mind that your attack is very machine and compiler specific. You may need to alter your string when running on a different machine or with a different version of compiler.