**电子科技大学信息与软件工程学院**

**标 准 实 验 报 告**

**（实验）课程名称 网络安全技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：陈奎 学 号：2017221303023 指导教师：赵洋**

**实验地点：信软楼西304 实验时间：19.11.19**

**一、实验室名称：信软楼西304**

**二、实验项目名称：缓冲区溢出实验**

**三、实验学时： 4 学时**

**（一）实验目的**

了解缓冲区溢出的相关概念；明确缓冲区溢出的危害；理解栈溢出、堆溢出、整型溢出、格式化字符串溢出及文件流溢出的原因；掌握安全编程技术。C/C++之类的高级语言通过指针直接对内存地址中所蕴藏的数据进行修改，因此可以利用C语言修改堆栈内容而修改程序的执行方式。本实验通过对缓冲区溢出利用的实践，理解缓冲区溢出的利用方法和带来的危害后果。

**（二）实验内容**

1. 初步认识栈溢出实验
2. 利用栈溢出漏洞实验
3. 溢出编程实验

**四、实验原理**

缓冲区溢出是目前最常见的一种安全问题，操作系统以及应用程序大都存在缓冲区溢出漏洞。缓冲区是一段连续内存空间，具有固定的长度。缓冲区溢出是由编程错误引起的，当程序向缓冲区内写入的数据超过了缓冲区的容量，就发生了缓冲区溢出，缓冲区之外的内存单元被程序“非法”修改。

一般情况下，缓冲区溢出导致应用程序的错误或者运行中止，但是，攻击者利用程序中的漏洞，精心设计出一段入侵程序代码，覆盖缓冲区之外的内存单元，这些程序代码就可以被CPU所执行，从而获取系统的控制权。

目前，操作系统（Windows、Linux、Unix）、数据库以及应用软件主要采用C/C++语言开发，但C/C++语言缺乏数组边界条件检查、程序执行不受控制等特点，因此，这些软件不可避免地存在缓冲区溢出漏洞，成为安全隐患。

**五、实验器材（设备、元器件）**

1. 实验人数50～80人，每人1台计算机；独立完成本实验。
2. 拓扑：（A、B范围中的主机分别简称为A主机和B主机）



**A**

**B**

1. 设备：以太网交换机2～4台；计算机50～80台
2. 软件：Visual Studio软件

**六、实验步骤**

1. **栈溢出的跟踪与解析实验**
2. 使用VS打开stack.dsw文件。
3. 在push语句前设置断点，按F5键进入调试界面。
4. 打开Watch、Variables、Registers、Memory、Disassembly五个窗口。
5. 记录EAX、ESP、EBP、EIP的值，并计算栈的大小为多少字节？
6. 按F11键执行push语句，观察栈、 ESP和EIP的值是否变化？为什么？
7. 继续执行pop语句，观察EAX、ESP、EIP的值是否变化？为什么？
8. 继续执行下面的语句，观察寄存器值的改变，最后按shift+F5结束调试。
9. 使用VS打开hanshu.dsw文件。
10. 在ourfunction语句前设置断点，按F5键进入调试界面。
11. 记录ESP、EBP、EIP的值，按F11键执行ourfunction语句，观察ESP、EIP的值是否变化，以及现在栈顶存放的4个字节是什么地址？
12. 继续执行，直到“int our=0”语句，观察每一步寄存器值的改变。
13. 执行“int our=0”语句，观察0值被存放在栈中的哪个位置？为什么？
14. 继续执行，直到“ret”语句，记录ESP，EBP的值。
15. “ret”语句执行后，什么值弹出到EIP，这时程序跳转到什么位置？ESP和EBP的值是否和步骤10时相同？
16. 按shift+F5结束调试。
17. 使用VS打开overflow.dsw文件。
18. 在“char longbuf[100]=……”语句前设置断点，F5调试，F11执行。
19. 记录EAX、EBP、ESP的初始值，查看longbuf的存放地址，计算此地址与EBP相差多少字节？
20. 继续执行“overflow(longbuf)”直到“push eax”语句后，EBP、ESP、EAX的值是否变化？栈顶现在存放的是什么数据？

lea是将源操作数的地址传到目的操作数中，那么“lea eax,[ebp-64h]”是将什么地址赋给了EAX寄存器？

1. 继续执行到“strcpy (des,buf)”指令后，mov是将数据从源操作数传到目的操作数中，那么“mov edx, dword ptr[ebp+8]”是将什么数据传到EDX中？

“lea eax,[ebp-8]”又是将什么数据传到EAX中？

1. 继续执行到“ret”语句，查看将弹出什么值到EIP中？
2. 利用栈溢出漏洞实验
3. 使用VS打开liyong.dsw文件。
4. 在“char longbuf[100]=……”语句前设置断点，F5调试，执行完赋值语句后查看longbuf的存放起始地址和结束地址？记录EBP、ESP的值。
5. 继续执行，“lea eax,[ebp-5fh]；push ecx”是将什么地址压栈？
6. “call @ILT+1175(overflow)”语句是调用overflow函数，在这之前要将EIP入栈，这时记录压入堆栈的EIP值是多少，存放在哪个地址处？
7. 调用overflow函数后，新分配给它的栈顶和栈底地址分别是多少？栈的大小是多少？
8. “char des[5]=””语句执行完后，是将那段地址分配给des变量，大小是多少字节？
9. 继续执行”strcpy(des,buf)”语句，mov语句是将什么值传递给edx? Lea语句是将什么值传递给eax?
10. Strcpy完成后，查看des变量空间是否有改变？
11. Ret语句执行完后，查看EBP，ESP，EIP的值分别是多少？程序将跳到哪里执行？
12. **溢出编程实验**

1、试自己编写一个缓冲区溢出的程序，通过溢出覆盖返回地址，从而跳转到一个指定的程序。运行结果截图。

参考代码：

#include <stdio.h>

#include <stdlib.h>

void why\_here(void)

{

printf("why u here !n\n");

printf("you are traped here\n");

system("pause");

\_exit(0);

}

int main(int argc,char \* argv[])

{

int buff[1];

buff[2] = (int)why\_here;

system("pause");

return 0;

}

2、试自己编写一个缓冲区溢出的程序，通过溢出覆盖返回地址和植入Shellcode，从而跳转到Shellcode执行，给出源码和运行结果截图。（附加题）

**七、实验数据及结果分析：**

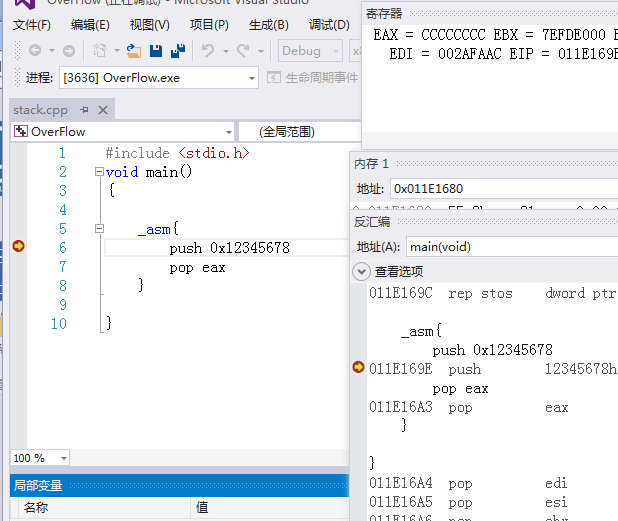
**（一）栈溢出的跟踪与解析实验**

1. 新建项目，并将stack.cpp拷贝进去

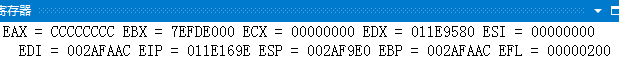


2．在push前设置断点，进入调试界面

3. 打开内存、反汇编、寄存器、局部变量窗口

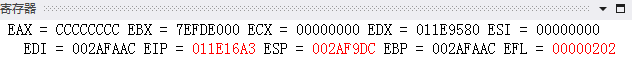


4. 记录EAX、ESP、EBP、EIP的值，并计算栈的大小为多少字节



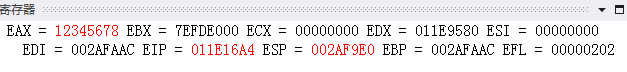
EAX=0xCCCCCCCC ESP=0x002AF9E0 EBP = 0x002AFAAC EIP = 0x011E169E 栈大小：EBP-ESP=0xCC字节

5. 按F11键执行push语句，观察栈、 ESP和EIP的值是否变化



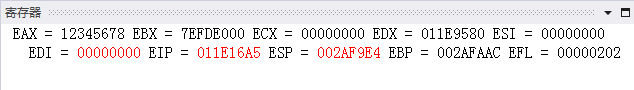
EIP和ESP发生变化，执行push时ESP减4，因为是压栈并且压入的占4字节所以减4；EIP+5；栈+4因为ESP减小了4

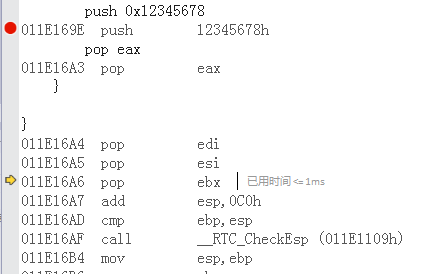
6. 继续执行pop语句，观察EAX、ESP、EIP的值是否变化



EAX=0x12345678，因为将0x12345678压栈再出栈到EAX中；ESP+4，因为执行了pop指令；EIP+1，本次指令1字节

7. 继续执行下面的语句，观察寄存器值的改变，最后按shift+F5结束调试





EDI置为0，EIP+1，ESP-4



EIP+1,ESP+4



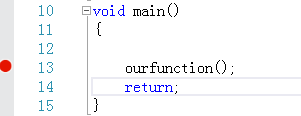
EIP+1,ESP+4



EIP+6,ESP+0xC0,EFL+4

8. 使用Visual Studio建立新项目，并将hanshu.cpp文件拷贝到项目中;

9. 在ourfunction语句前设置断点，按F5键进入调试界面



10. 记录ESP、EBP、EIP的值，按F11键执行ourfunction语句，观察ESP、EIP的值是否变化，以及现在栈顶存放的4个字节是什么地址

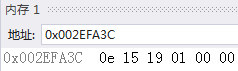
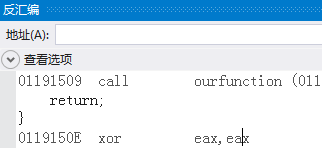


ESP=0x002EFA40 EBP=0x002EFA8C EIP=0x01191509

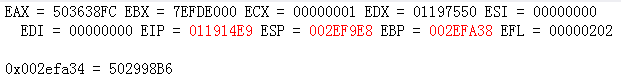
执行ourfunction语句后



EIP变为0x011914E0,ESP+4，栈顶存放的4字节为”xor eax,eax”返回地址

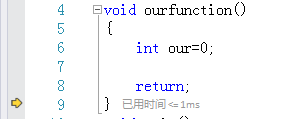
 

11. 继续执行，直到“int our=0”语句



12.执行“int our=0”语句，观察0值被存放在栈顶

13.继续执行，直到“ret”语句，记录ESP，EBP的值





14.Ret指令的地址被弹出到EIP，函数ESP、EBP与第10步不同

15.Ret指令的地址被弹出到EIP，函数ESP、EBP与第10步相同

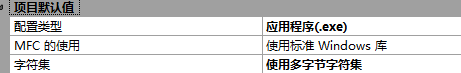


16.按shift+F5结束调试

17. 使用Visual Studio建立新项目，并将overflow.cpp文件拷贝到项目中；



此处存在问题："const char \*" 类型的实参与 "LPCWSTR" 类型的形参不兼容，需要将字符集换为使用多字符集



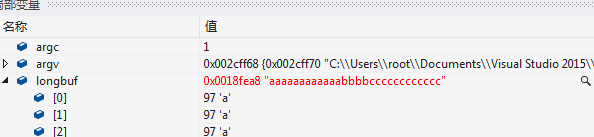
18. 在“char longbuf[100]=……”语句前设置断点，F5调试，F11执行；

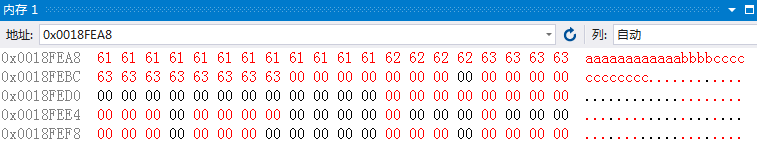


按F11执行后

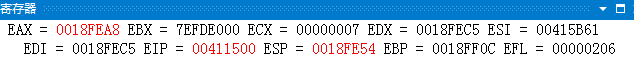


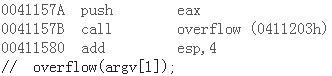
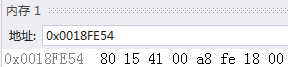
longbuf的存放地址为0x0018FEA8，此地址与EBP相差64字节





19. 继续执行“overflow(longbuf)”直到“push eax”语句后



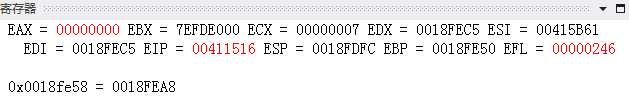


EBP未发生变化，ESP减8，EAX变为了longbuf的存放地址，栈顶存放的是下一条指令的地址

20. lea是将源操作数的地址传到目的操作数中，那么“lea eax,[ebp-64h]”是将什么地址赋给了EAX寄存器？

longbuf的存放地址

22. 继续执行到“strcpy (des,buf)”指令后，mov是将数据从源操作数传到目的操作数中，那么“mov edx, dword ptr[ebp+8]”是将什么数据传到EDX中？ “lea eax,[ebp-8]”又是将什么数据传到EAX中？

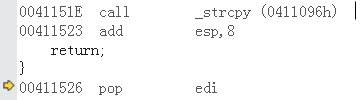


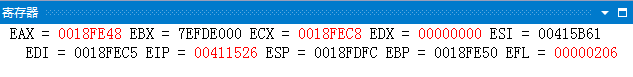


“mov edx, dword ptr[ebp+8]”是将longbuf数组的地址传到EDX

“lea eax,[ebp-8]”又是将longbuf数组的值传到EDX中

23. 继续执行到“ret”语句，查看将弹出什么值到EIP中





将 pop edi 的地址弹出到EIP中

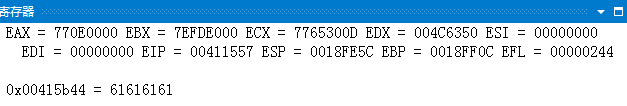
**（二）利用栈溢出漏洞实验**

1. 使用Visual Studio建立新项目，并将liyong.cpp文件拷贝到项目中

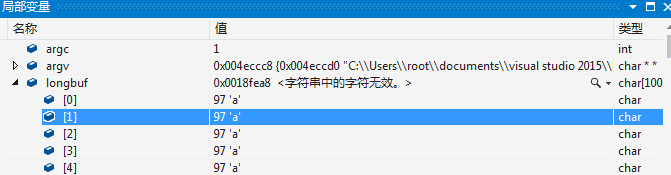


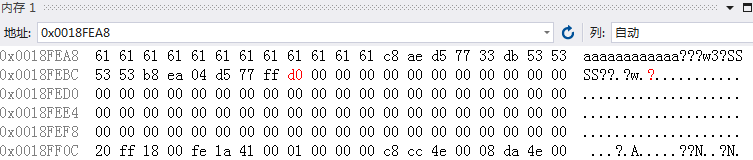
2. 在“char longbuf[100]=……”语句前设置断点，F5调试，执行完赋值语句后查看longbuf的存放起始地址和结束地址？记录EBP、ESP的值。

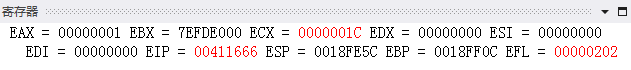
初始状态



执行完赋值语气







ESP =0x 0018FE5C EBP = 0x0018FF0C

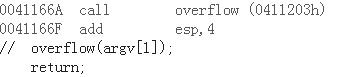
longbuf的存放起始地址为0x0018FEA8和结束地址为0x0018FF0C

3. 继续执行，lea ecx,[ebp-64h]；push ecx是将什么地址压栈



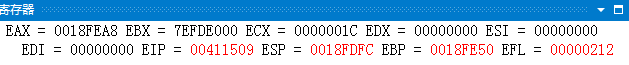
push ecx是将参数个数压栈

4. call @ILT+0(overflow) 语句是调用overflow函数，在这之前要将EIP入栈，这时记录压入堆栈的EIP值是多少，存放在哪个地址处



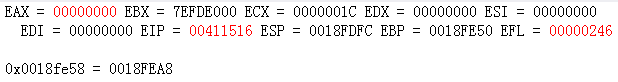
EIP=0x0041166F

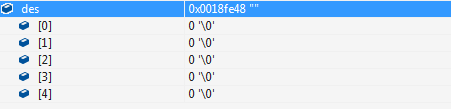
5. 调用overflow函数后，新分配给它的栈顶和栈底地址分别是多少？栈的大小是多少？



ESP = 0x0018FDFC EBP = 0x0018FE50,栈的大小为54字节

6. “char des[5]=””语句执行完后，是将那段地址分配给des变量，大小是多少字节



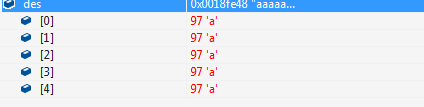


大小为5字节

7. 继续执行”strcpy(des,buf)”语句，mov语句是将什么值传递给edx? Lea语句是将什么值传递给eax?

Mov语句将des地址传给eax，lea语句是将des的地址传给ecx

8. Strcpy完成后，查看des变量空间是否有改变



des变量空间的前5个字符变为a

9. Ret语句执行完后，查看EBP，ESP，EIP的值分别是多少？程序将跳到哪里执行？



EIP = 0x00411526 ESP =0x0018FDFC EBP = 0x0018FE50



程序跳到pop edi 执行

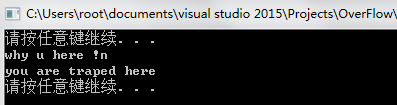
**（三）溢出编程实验**

1．试自己编写一个缓冲区溢出的程序，通过溢出覆盖返回地址，从而跳转到一个指定的程序。

代码

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  void why\_here(void)  {  printf("why u here !n\n");  printf("you are traped here\n");  system("pause");  \_exit(0);  }  int main(int argc, char \* argv[])  {  int buff[1];  buff[2] = (int)why\_here;  system("pause");  return 0;  } |

运行结果



使用why\_here函数的内存地址覆盖了返回地址的值，让函数溢出到why\_here执行，实现了缓冲区溢出

2.试自己编写一个缓冲区溢出的程序，通过溢出覆盖返回地址和植入Shellcode，从而跳转到Shellcode执行，给出源码和运行结果截图。(附加)

**八、实验结论、心得体会和改进建议：**

通过本次实验，加深了对缓冲区溢出的理解，了解了溢出时内存中数据的变化以及寄存器值的变化。

**九、对本实验过程及方法、手段的改进建议**

无

**报告评分：**

**指导教师签字：**