《漏洞利用及渗透测试基础》实验报告

张明昆 2211585

1 实验名称

IDE 反汇编实验

2 实验目标

根据第二章示例 2-1，在 XP 环境下进行 VC6 反汇编调试，熟悉函数调用、栈帧切换、CALL 和 RET 指令等汇编语言实现，将 call 语句执行过程中的 EIP 变化、ESP、EBP 变化等状态进行记录，解释变化的主要原因。

3 实验过程

3.1 进入 VC 反汇编

在 VMware 中下载并安装好 windows xp 系统，将 VC6 中的文件转移到虚拟机中，按照实验视频的步骤创建一个 C++ 文件，并输入以下代码。

```
#include <iostream>
int add(int x, int y){
    int z = 0;
    z = x + y;
    return z;
}
void main() {
    int n = 0;
    n = add(1, 3);
    printf("%d\n", n);
```

```
}
```

在 n = add(1,3); 处设置断点。按 F5 进入测试状态，在断点处右键选择 Go to disassembly 进入反汇编模式，并打开寄存器窗口，内存窗口。

3.2 观察 add 函数调用前反汇编语句、寄存器的状态

通过如下语句，初始化 n=0，此时 n 的地址为 [ebp-4], 即 EBP 指针抬高 4 字后的位置。并将 add 函数所需要的两个参数从右向左依次压入栈中。

```
8:         int n = 0;
00401088   mov          dword ptr [ebp-4],0
9:         n = add(1, 3);
0040108F   push         3
00401091   push         1
```

3.3 观察 add 函数调用前后语句

通过 call 指令调用 add 函数，此时 call 指令隐含做了两件事情，将 EIP 中的下一条指令的地址入栈，然后跳转到 add 函数所在的代码块中。

```
00401093   call         @ILT+0(add) (00401005)
```

在 add 函数调用后，进行栈平衡操作，将 ESP 恢复到调用 add 函数之前的状态，然后将函数返回值从 EAX 寄存器中转移到局部变量 n 所在的地址处。

```
00401098   add          esp,8
0040109B   mov          dword ptr [ebp-4],eax
```

3.4 add 函数内部栈帧切换

在 call 指令执行时，发生了两件事情，将 EIP 中的返回地址入栈，ESP-4，设置 EIP 的值，实现从 main 函数到 add 函数的跳转。

```
EAX = CCCCCCCC EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF80
```

```
EIP = 00401030 ESP = 0012FF24
EBP = 0012FF80 EFL = 00000216
```

然后将 EBP 栈底指针上提到 ESP 处，将 ESP-44h，开辟 add 函数的栈帧。

```
EAX = CCCCCCCC EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF80
EIP = 00401036 ESP = 0012FEDC
EBP = 0012FF20 EFL = 00000212
```

再将原始的寄存器信息入栈保存，将 EBX，ESI，EDI 入栈。再对局部变量进行初始化。

```
EAX = CCCCCCCC EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF20
EIP = 0040104F ESP = 0012FED0
EBP = 0012FF20 EFL = 00000212
```

在 add 函数执行完成之后，return z；这时将函数的返回值从局部变量中转移到寄存器 EAX 中。

```
EAX = 00000004 EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF20
EIP = 0040105B ESP = 0012FED0
EBP = 0012FF20 EFL = 00000202
```

随后要清理函数开辟的栈帧，将原始的寄存器信息出栈，恢复状态。通过 mov esp,ebp 清除函数开辟的栈帧。

```
EAX = 00000004 EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF80
EIP = 00401060 ESP = 0012FF20
EBP = 0012FF20 EFL = 00000202
```

然后将原始栈帧的栈底 EBP 的值出栈，恢复了原始的 EBP 状态。

```
EAX = 00000004 EBX = 7FFD6000
ECX = 00000000 EDX = 003C0DD8
ESI = 00000000 EDI = 0012FF80
EIP = 00401061 ESP = 0012FF24
EBP = 0012FF80 EFL = 00000202
```

ret 指令完成了两件事情，将当前的栈顶 ESP 中保存原始 EIP 的值出栈到 EIP 中，然后跳回到调用函数的代码块中。

4 心得体会

通过本次实验，我更加直观清晰地了解了函数调用时栈帧、地址、EIP、ESP、EBP 等指针存放状态的变化，熟悉了反汇编、断点、单步执行等操作。我理解了栈帧调整的具体步骤，即：返回地址入栈，参数从右向左入栈，开辟函数空间，之后再相对应地释放函数空间，读取返回地址，最终返回到主函数。这为日后的汇编学习与软件安全的学习打下基础。