

《软件安全》实验报告

姓名： 邢清画 学号： 2211999 班级： 1023

实验名称：

Shellcode 编写及编码

实验要求：

复现第五章实验三，并将产生的编码后的 shellcode 在示例 5-4 中进行验证，阐述 shellcode 编码的原理、shellcode 提取的思想。

实验过程：

1. 用 C 语言编写要执行的 shellcode

代码如下：

```
#include<stdio.h>
#include<windows.h>
void main(){
    MessageBox(NULL,NULL,NULL,0);
    return;
}
```

在 MessageBox (NULL, NULL, NULL, 0);处设置断点，进入反汇编模式，观察该函数在调用时的反汇编代码，共调用了四个参数，如下：

```
5:      MessageBox(NULL, NULL, NULL, 0);
00401028  mov     esi,esp
0040102A  push    0
0040102C  push    0
0040102E  push    0
00401030  push    0
00401032  call    dword ptr [__imp__MessageBoxA@16 (0042428c)]
00401038  cmp     esi,esp
0040103A  call    __chkesp (00401070)
6:      return;
7:      }
0040103F  pop     edi
00401040  pop     esi
00401041  pop     ebx
00401042  add     esp,40h
00401045  cmp     ebp,esp
00401047  call    __chkesp (00401070)
0040104C  mov     esp,ebp
0040104E  pop     ebp
0040104F  ret
```

在编译器中使用 _asm 编写 MessageBox 的反汇编语句：

```
#include<stdio.h>
#include<windows.h>
void main(){
    _asm{
        xor ebx,ebx
        push ebx
        push ebx
        push ebx
        push ebx
        mov eax,77d507eah
        call eax
    }
    return;
}
```

正确运行后，证明该代码的功能与上述 MessageBox 函数相同。

在 xor ebx, ebx 语句处加入断点，进入反汇编

```
5:      _asm{
6:      xor ebx,ebx
00401028  xor     ebx,ebx
7:      push ebx//push 0
0040102A  push    ebx
8:      push ebx
0040102B  push    ebx
9:      push ebx
0040102C  push    ebx
10:     push ebx
0040102D  push    ebx
11:     mov  eax,77d507eah
0040102E  mov     eax,77D507EAh
12:     call eax
00401033  call    eax
13:     }
14:
15:     return;
16:  }
00401035  pop     edi
```

地址: 0x00401028

00401028	33 DB 53 53 53 53 B8 EA 07 D5 77 FF D0 5F 5E 5B	3 堡 SSS 戈 . 谔 . 略 ^ [
00401038	83 C4 40 3B EC E8 2E 00 00 00 8B E5 5D C3 CC CC	施 @ ; 虐 ... 嬉 1 猛 .
00401048	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC	烫 烫 烫 烫 烫 烫 烫 烫
00401058	CC CC CC CC CC CC CC CC CC FF 25 8C 42 42 00 CC CC	烫 烫 烫 烫 . % 嬉 B . 烫

根据汇编语句的地址，获得该汇编语句对应的机器码，并在 shellcode 处替换进行调试

```
main.cpp *
#include<stdio.h>
#include<windows.h>
char ourshellcode[]="33\xD8\x53\x53\x53\x53\xB8\xEA\x07\xD5\xFF\xD0";
void main(){
    loadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    |
    return;
}
```

Press any key to continue_

2. 编写 shellcode:

实现可以输出“hello world”的运行框，代码如下：

```

main.cpp *
#include<stdio.h>
#include<windows.h>
void main(){
    loadLibrary("user32.dll");//加载user32.dll
    _asm{
        xor ebx,ebx
        push ebx//push 0
        push 20646C72h
        push 6F77206Fh
        push 6C6C6568h
        mov eax,esp
        push ebx//push 0
        push eax
        push eax
        push ebx
        mov eax,77d507eah//77d507eah是MessageBox函数在系统中的地址
        call eax
    }
    return;
}

```

提取 shellcode 代码:

\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0

3. shellcode 编码:

用异或编码, 输入得到的 shellcode 代码, 代码如下:

```

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
void encoder(char* input, unsigned char key)
{
    int i = 0, len = 0;
    FILE * fp;
    len = strlen(input);
    unsigned char * output = (unsigned char *)malloc(len + 1);
    for (i = 0; i < len; i++)
        output[i] = input[i] ^ key;
    fp = fopen("encode.txt", "w+");
    fprintf(fp, "\n");
    for (i = 0; i < len; i++)
    {
        fprintf(fp, "\\x%0.2x", output[i]);
        if ((i + 1) % 16 == 0)
            fprintf(fp, "\\n\\n");
    }
    fprintf(fp, "\n");
    fclose(fp);
    printf("dump the encoded shellcode to encode.txt OK!\n");
    free(output);
}

int main(){
    char shellcode[]="\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0";
    encoder(shellcode,0x44);
    getchar();
    return 0;
}

```

```

dump the encoded shellcode to encode.txt OK!
a

```

并输出异或操作之后的 shellcode 编码到 encode.txt 中, 获得编码后的 shellcode:

```
encode - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
"\x77\x9f\x17\x2c\x36\x28\x20\x64\x2c\x2b\x64\x33\x2b\x2c\x2c\x21"
"\x28\x28\xcf\x80\x17\x14\x14\x17\xfc\xae\x43\x91\x33\xbb\x94\xd4"
....
```

4. shellcode 解码操作:

解码代码如下:

```
void main()
{
    __asm
    {
        add eax, 0x14
        xor ecx, ecx
        decode_loop:
            mov bl, [eax + ecx]
            xor bl, 0x44
            mov [eax + ecx], bl
            inc ecx
            cmp bl, 0x90
            jne decode_loop
    }
}
```

解码器与编码后的 shellcode 共同执行, 以 eax 作为 shellcode 的起始地址, 在之后的每次操作中, 将代码和 0x44 做异或操作后覆盖掉之前的代码, 在最后放置 0x90 为结束符。

```
#include <iostream>
using namespace std;
int main(int argc, char const *argv[])
{
    unsigned int temp;
    __asm{
        call lable;
        lable:
            pop eax;
            mov temp,eax;
    }
    cout <<temp <<endl;
    return 0;
}
```

进入反汇编代码部分:

```
5:      unsigned int temp;
6:      __asm{
7:          call lable;
00401578 call     lable (0040157d)
8:          lable:
9:              pop eax;
0040157D pop     eax
10:         mov temp,eax;
0040157E mov     dword ptr [ebp-4],eax
11:     }
12:     cout <<temp <<endl;
00401581 push    offset @ILT+195(std::endl) (004010c8)
00401586 mov     eax,dword ptr [ebp-4]
00401589 push    eax
0040158A mov     ecx,offset std::cout (004767e0)
0040158F call    @ILT+250(std::basic_ostream<char,std::char_traits<char> >::operator<<) (004010ff)
00401594 mov     ecx,eax
00401596 call    @ILT+475(std::basic_ostream<char,std::char_traits<char> >::operator<<) (004011e0)
13:     return 0;
0040159B xor     eax,eax
```

观察到在执行完 call lable 语句之后, 将 eip 存储的下一条指令地址的值 (0040157d) 压入栈, eax 是当前的指令地址。

编写并执行下面的程序代码：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
int main()
{
    __asm
    {
        call labl;
        labl: pop eax;
        add eax, 0x15 ;越过 decoder 记录 shellcode 起始地址
        xor ecx, ecx
        decode_loop:
        mov bl, [eax + ecx]
        xor bl, 0x44 ;用 0x44 作为 key
        mov [eax + ecx], bl
        inc ecx
        cmp bl, 0x90 ;末尾放一个 0x90 作为结束符
        jne decode_loop
    }
    return 0;
}
```

产生含有解码程序的 shellcode，进入反汇编部分：

```
8:          call labl;
00401028  call     labl (0040102d)
9:          labl: pop eax;
0040102D  pop     eax
10:         add eax, 0x15 ;越过 decoder 记录 shellcode 起始地址
0040102E  add     eax, 15h
11:         xor ecx, ecx
00401031  xor     ecx, ecx
12:         decode_loop:
13:         mov bl, [eax + ecx]
00401033  mov     bl, byte ptr [eax+ecx]
14:         xor bl, 0x44 ;用 0x44 作为 key
00401036  xor     bl, 44h
15:         mov [eax + ecx], bl
00401039  mov     byte ptr [eax+ecx], bl
16:         inc ecx
0040103C  inc     ecx
17:         cmp bl, 0x90 ;末尾放一个 0x90 作为结束符
0040103D  cmp     bl, 90h
18:         jne decode_loop
00401040  jne     decode_loop (00401033)
19:         }
20:         return 0;
00401042  xor     eax, eax
21:         }
00401044  pop     edi
```

地址:	0x00401028
00401028	E8 00 00 00 58 83 C0 15 33 C9 8A 1C 08 80 F3X就.3落..
00401038	44 88 1C 08 41 80 FB 90 75 F1 33 C0 5F 5E 5B 83 D...A.麗u.3缩^
00401048	C4 40 3B EC E8 1F 00 00 8B E5 5D C3 CC CC CC 膾;虞...嬌]錘
00401058	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC 烫烫烫烫烫烫烫

根据汇编语句的地址（00401028），获得该汇编语句对应的机器码。从开始地址不断提取机器码。00401042 处为结束地址，在提取到 00401041 时，与 encode.txt 中的 shellcode 编码合并得到完整的 shellcode 编码：

E8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1\x77\x9F\x17\x2C\x36\x28\x20\x64\x2C\x2B\x64\x33\x2B\x2C\x2C\x21\x28\x28\xCF\x80\x17\x14\x14\x17\xFC\xAE\x43\x91\x33\xBB\x94\xD4

在示例 5-4 中代替 ourshellcode 具体值：

```
#include<stdio.h>
#include<windows.h>
char ourshellcode[]="E8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1\x77\x9F\x17\x2C\x36\x28\x20\x64\x2C\x2B\x64\x33\x2B\x2C\x2C\x21\x28\x28\xCF\x80\x17\x14\x14\x17\xFC\xAE\x43\x91\x33\xBB\x94\xD4";
void main(){
    LoadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    return;
}
```

编译运行：



心得体会：

通过根据实验案例，编写代码实现了 shellcode 的编码和解码过程，并了解了内在原理。Shell 是命令解释器，用于解释输入的命令，植入 shell 的代码是 shellcode，用来表示广义上的植入进程的代码，它是指一段与操作系统交互的机器码，通常编写用于利用软件漏洞或执行恶意代码的目的，是用来利用计算机系统弱点的攻击手段之一。通过了解攻击过程，更好的防止恶意代码攻击。