Bootkit 技术演变趋势及研究分析(上)

---by nEINEI

2011.10

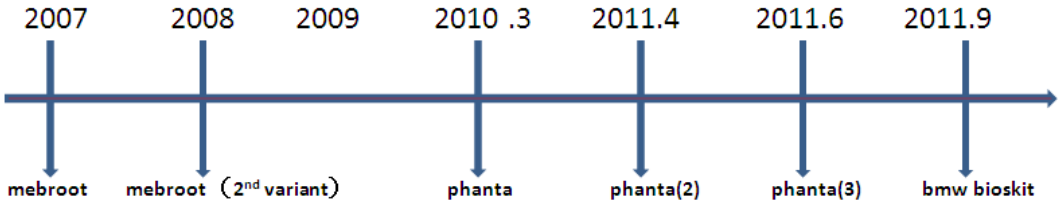
Bootkit 演化及分析共 2 篇，本文是上篇，着重讨论截至 2011 底，中国地区主要 Bootkit 病毒的技术演化过程及新技术的对抗，下篇着重讨论 TDL4 变种的最新技术变化。

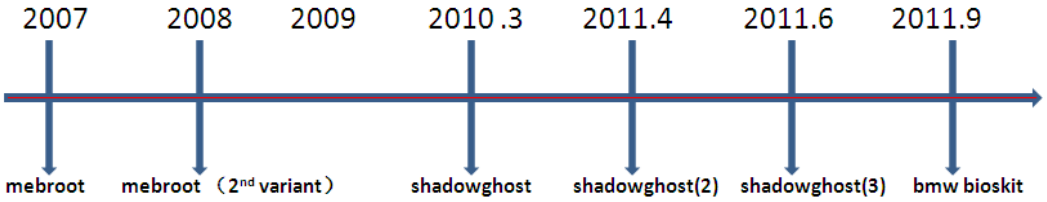
一 简介

在持续的跟踪 bootkit 病毒发展的过程中我们发现仅 2011 年 4 月~ 9 月,在中国区分别出现了，鬼影 2 代，鬼影 3 代，ZeroAccess(aka:Max++不是严格的 BK，利用驱动感染的方式加载，TDL3 的另一个分支变种)，Tdss-4，bmw bios 等复杂的攻击 windows 引导启动阶段的方法。

在深入的代码分析中发现，流行的 bootkit 病毒特别是中国地区的流行的病毒已经从最初的借鉴 eeye bootroot，mebroot 技术，发展为更复杂的隐蔽攻击方式。在撰写本文的过程,我们获了更底层的感染 award bios 的病毒。但这只是开始，我们确信，黑客已经熟练了的多种 bootkit 技术。这里将是未来的新战场。

下面是发现典型 bookit 病毒在中国区的时间表.





这些病毒技术都和 mebroot 有非常大的联系，为了更好的说明后续技术的变化，下面将首先介绍 mebroot 是如何做到引导系统启动并加载恶意驱动程序的。

二　从哪里来 -- mebroot 深入剖析

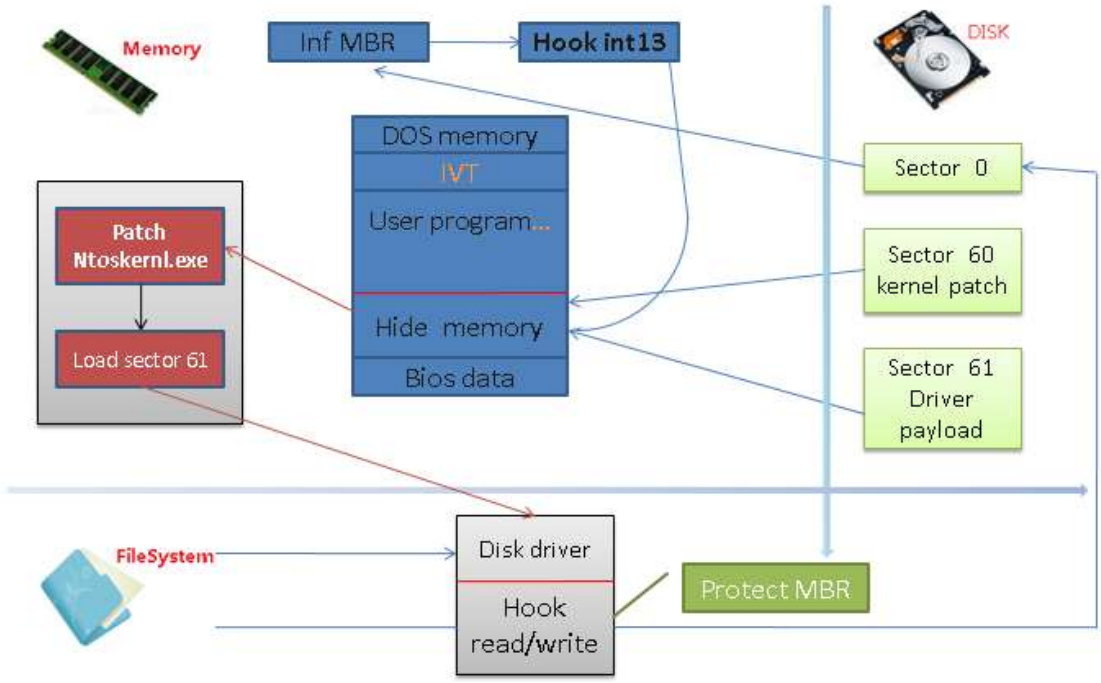
mebroot 感染过程分三个阶段：

1) 修改 MBR 数据，隐藏自身到实模式系统内存空间（0~1MB）的尾部，hook int 13h 中断。

2) patch ntldr 在 0x422a6f 位置，搜索获得 osloader 的 BlLoaderBlock 结构，利用 BlLoaderBlock 定位 LDR_DATA_TABLE_ENTRY 的 DllBase 成员，该指针指向 ntoskrnl 模块基址，拷贝自身数据到 ntoskrnl 模块尾部，同时 hook IoInitSystem 函数

3) 系统调用 IoInitSystem 时进入病毒代码空间，搜索导出表，获得 ntOpenFile，ntReadFile，ntClose, ExAllocatepool ，利用以上函数读取磁盘上的病毒驱动程序到内存中，解析驱动程序 PE 结构，模拟 PEloader 加载过程，把病毒驱动加载到另外一块非分页内存中执行。病毒驱动程序 hook disk read/write dispatch functions ，保护自身 MBR 数据不被恢复。

Mebroot 感染系统后的流程图如下：



Mebroot 感染 windows 流程图

(I) MBR

1 通过 mov esi,413h // 实模式下 bios 中记录的系统内存信息
Sub word ptr[esi],2 减小内存信息，使自身隐藏在系统内存的末端

2 拷贝自身代码到系统内存后 2k 的位置

3 读取磁盘 60，61 扇区的病毒代码到系统内存后 1.5k 的位置

3 Hook int 13 中断，扫描读入的内存是否包含 （8B F0 85 F6 74 21/22 80 3D）特征，有则修改为0xff15（call）跳向病毒在60扇区代码

5 加载原 windows MBR（62 扇区） ，重新跳向 0x7c00 正常引导系统启动

4 在 xp 系统下 ntldr 被加载运行，当执行到 call BlLoadBootDrivers 的下一句时，转向了病毒第二阶段的病毒代码中

```
00422a69: (                    ): push eax                      ; 50
00422a6a: (                    ): call .+4802                   ; e8c2120000
00422a6f: (                    ): call dword ptr ds:0x9f5fc ; ff15fcf50900
00422a75: (                    ): cmp byte ptr ds:0x43aef8, 0x00 ; 803df8ae430000
00422a7c: (                    ): jz .+7                        ; 7407
00422a7e: (                    ): xor esi, esi                  ; 33f6
00422a80: (                    ): jmp .+597                     ; e955020000
00422a85: (                    ): lea eax, dword ptr ss:[ebp-388] ; 8d857cfefffff
00422a8b: (                    ): push eax                      ; 50
00422a8c: (                    ): call .-4742                   ; e87aedffff
```

virus hook

(II) hook ntldr

a) 在系统堆栈中取出系统当前执行的指令地址 a， a = a &0xfff00000;（确定 osloader 32bit 部分的基址）从此时位置开始扫描特征。寻址 BlLoaderBlock 结构

```
.text:00415912 75 F0                                    jnz     short loc_415904
.text:00415914 C7 46 34 00 40 00 00                     mov     dword ptr [esi+34h], 4000h
.text:0041591B 66 C7 46 38 01 00                        mov     word ptr [esi+38h], 1
.text:00415921 A1 C4 82 46 00                           mov     eax, _BlLoaderBlock
.text:00415926 8D 48 04                                 lea     ecx, [eax+4]
.text:00415929 8B 11                                    mov     edx, [ecx]
.text:0041592B 89 06                                    mov     [esi], eax
```

因为在 BlLoaderBlock 是 LOADER_PARAMETER_BLOCK 结构，该结构第一个成员是 LoadOrderListHead，该指针执行当前加载的模块 LDR_DATA_TABLE_ENTRY 结构。
BlLoaderBlock 链表是由 BlAllocateDataTableEntry 影响的，而在 BlOsLoader 运行时，第一个调用 BlAllocateDataTableEntry 的位置是 ntoskrnl.exe

```
.text:004227E9 68 C8 44 43 00                           push    offset aNtoskrnl_exe ; "ntoskrnl.exe"
.text:004227EE C7 05 98 62 43 00 00 00 02 00            mov     _BlUsableLimit, 20000h
.text:004227F8 E8 55 30 FF FF                           call    _BlAllocateDataTableEntry@16 ; BlAllocateDataTableEntry(x,x,x,x)
.text:004227FD 8B F0                                    mov     esi, eax
.text:004227FF 85 F6                                    test    esi, esi
.text:00422801 74 0F                                    jz      short loc_422812
```

```
kd> dt nt!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY
   +0x008 InMemoryOrderLinks : _LIST_ENTRY
   +0x010 InInitializationOrderLinks : _LIST_ENTRY
   +0x018 DllBase          : Ptr32 Void
   +0x01c EntryPoint       : Ptr32 Void
   +0x020 SizeOfImage      : Uint4B
   +0x024 FullDllName      : _UNICODE_STRING
   +0x02c BaseDllName      : _UNICODE_STRING
   +0x034 Flags            : Uint4B
   +0x038 LoadCount        : Uint2B
   +0x03a TlsIndex         : Uint2B
   +0x03c HashLinks        : _LIST_ENTRY
   +0x03c SectionPointer   : Ptr32 Void
   +0x040 CheckSum         : Uint4B
   +0x044 TimeDateStamp    : Uint4B
   +0x044 LoadedImports    : Ptr32 Void
   +0x048 EntryPointActivationContext : Ptr32 Void
   +0x04c PatchInformation : Ptr32 Void
```

此时，取指针偏移 0x18 的位置就可以获得 ntoskrnl.exe 的加载基址了。

b) 按照基址的位置开始搜索 InbvSetProgressBarSubset，这主要为了定位 IoInitSytem

```
INIT:005DAFFE 6A 4B                                     push    4Bh
INIT:005DB000 6A 19                                     push    19h
INIT:005DB002 E8 3C B5 E5 FF                            call    _InbvSetProgressBarSubset@8 ; InbvSetProgressBarSubset(x,x)
INIT:005DB007 FF B5 80 FB FF FF                         push    [ebp+var_480]
INIT:005DB00D E8 6F E9 FF FF                            call    _IoInitSystem@4 ; IoInitSystem(x)
INIT:005DB012 84 C0                                     test    al, al
INIT:005DB014 0F 84 FA B0 01 00                         jz      loc_5F6114
INIT:005DB01A 6A 64                                     push    64h
INIT:005DB01C 53                                        push    ebx
```

将该偏移值保留在当前 0x98000+4 的位置处。

```
seg000:00000000                         seg000      segment byte public "CODE" use32
seg000:00000000                                     assume cs:seg000
seg000:00000000                                     assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000 8B 14 24                            mov     edx, [esp]
seg000:00000003 68 78 56 34 12                      push    12345678h       ; push IoInitSystem
seg000:00000008 8B 0C 24                            mov     ecx, [esp]
seg000:0000000B 68 78 56 34 12                      push    12345678h       ; push _imp__VidInitialize
```

c) 计算 ntoskrnl.exe 的 SizeOfImage 值，使其减小 512 字节，然后将 0x98000 偏移的数据拷贝 512 个字节到 ntosknl.exe 的尾部。完成 hook call IoInitSystem 指令，使其指向 ntosknl.exe 末尾 512 偏移处。

(III) load drivers

a) 此时系统已经进入windows的欢迎界面，并设置启动进度。此时的堆栈数据如下， hook 的地方在 Phase1Initialization内部

f8ac1dac 805c5a28 80087000 00000000 00000000 nt!Phase1Initialization+0x9b5 (FPO: [Non-Fpo])
f8ac1ddc 80541fa2 80684528 80087000 00000000 nt!PspSystemThreadStartup+0x34 (FPO: [Non-Fpo])
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16

```
kd> u Phase1Initialization+0x9a5
nt!Phase1Initialization+0x9a5:
80684ecd e81295e6ff      call    nt!InbvSetProgressBarSubset (804ee3e4)
80684ed2 ffb590fbffff    push    dword ptr [ebp-470h]
80684ed8 e8239f0400      call    806cee00
80684edd 84c0            test    al,al
80684edf 7507            jne     nt!Phase1Initialization+0x9c0 (80684ee8)
80684ee1 6a69            push    69h
80684ee3 e90a070000      jmp     nt!Phase1Initialization+0x10ca (806855f2)
80684ee8 6a64            push    64h
```

在调用 IoInitSytem 时进入病毒代码空间，IoinitSytem 是系统引导过程中较为复杂的一部分。病毒驱动在此时加载可以完成很多自己保护的工作。
从 ntoskrnl.exe 的导出表中获得 ExAllocatepool 函数，比较时的函数名称是用 CRC32 代替的。

```
seg000:00000013 50                      push    eax
seg000:00000014 25 FF FF FE FF          and     eax, 0FFFEFFFFh
seg000:00000019 0F 22 C0                mov     cr0, eax
seg000:0000001C 2B CA                   sub     ecx, edx        ; IoInitSytem - hook next instrctions
seg000:0000001E 58                      pop     eax
seg000:0000001F 0F 22 C0                mov     cr0, eax
seg000:00000022 FF 34 24                push    dword ptr [esp] ; push ntoskrnl base address
seg000:00000025 68 62 E0 07 37          push    3707E062h       ; Get ExAllocatePool Address
seg000:0000002A E8 3B 00 00 00          call    Get_ke_api
```

1) 分配一段 0x1ab 大小的内存，用来运行新的代码。从偏移 0x6a 到结束处的代码拷贝到上面分配的内存当中，跳向该位置运行。

2) 调用原 windows IoInitSystem，这部分会对系统的 I/O 子系统的状态变量进行初始化、驱动程序对象类型和设备对象类型的创建、加载，“引导-启动“类的驱动程序，加载“系

统-启动”类型的驱动程序，执行 IoInitSystem 后，开始加载自身磁盘上的驱动程序。下面是病毒调用系统的 IoInitSystem

```
821a500f ff542408        call    dword ptr [esp+8]
821a5013 59              pop     ecx
821a5014 5a              pop     edx
821a5015 60              pushad
821a5016 87cd            xchg    ecx,ebp
821a5018 e852000000      call    821a506f
821a501d 60              pushad
821a501e 8b6c2428        mov     ebp,dword ptr [esp+28h]
kd> r
eax=821a5008 ebx=00000000 ecx=000001b8 edx=656e6f4e esi=00000000 edi=00043000
eip=821a500f esp=f8ac1834 ebp=f8ac1dac iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
821a500f ff542408        call    dword ptr [esp+8]    ss:0010:f8ac183c={nt!IoInitSystem (80686336)}
```

3) 获得一些列内核函数，ntOpenFile , ntReadFile , ntClose, ExAllocatepool ,打开磁盘上固定偏移的病毒驱动文件。

```
kd> dt  _OBJECT_ATTRIBUTES  f8ac17ec
ntdll!_OBJECT_ATTRIBUTES
   +0x000 Length           : 0x18
   +0x004 RootDirectory    : (null)
   +0x008 ObjectName       : 0xf8ac1804 _UNICODE_STRING "\??\PhysicalDrive0"
   +0x00c Attributes       : 0x40
   +0x010 SecurityDescriptor : (null)
   +0x014 SecurityQualityOfService : (null)
```

分配 0x3BC00 大小的内存空间，用作 ntReadFile 的 buffer。

```
821a5106 ff11            call    dword ptr [ecx]
kd> r
eax=f8ac1810 ebx=821a501d ecx=f8ac1810 edx=00000000 esi=81e67000 edi=80533708
eip=821a5106 esp=f8ac17b0 ebp=804d8000 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
821a5106 ff11            call    dword ptr [ecx]      ds:0023:f8ac1810={nt!NtReadFile (80571618)}
kd> dd esp
f8ac17b0  000005c8 00000000 00000000 00000000
f8ac17c0  f8ac1810 81e67000 0003bc00 f8ac17d4
f8ac17d0  00000000 7f404000 00000000 84fcd516
f8ac17e0  804d8000 3707e062 804d8000 00000018
f8ac17f0  00000000 f8ac1804 00000040 00000000
f8ac1800  00000000 00260024 821a5086 000005c8
f8ac1810  80571618 00000001 852974b8 804d8000
f8ac1820  00043000 00000000 f8ac1dac f8ac1840
```

4) 解析读取的驱动文件，获得文件的 SizeofImage 大小，再次分配 SizeofImage 大小的内存空间，这是为了模拟 PEloader 的功能，展开 PE 文件。

```
eax=81e67258 ebx=821a501d ecx=01c10031 edx=0003bbf8 esi=81e67000 edi=80533708
eip=821a5123 esp=f8ac17bc ebp=804d8000 iopl=0         nv up ei ng nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000282
821a5123 ffd7            call    edi {nt!ExAllocatePool (80533708)}
kd> dd esp
f8ac17bc  00000000 0003bbf8 0003bbf8 81e67258
f8ac17cc  dcd44c5f 804d8000 7f404000 00000000
f8ac17dc  84fcd516 804d8000 3707e062 804d8000
f8ac17ec  00000018 00000000 f8ac1804 00000040
f8ac17fc  00000000 00000000 00260024 821a5086
f8ac180c  000005c8 00000000 0003bc00 852974b8
f8ac181c  804d8000 00043000 00000000 f8ac1dac
f8ac182c  f8ac1840 00000000 80686336 804d8000
```

拷贝 PE 头到新的内存空间，按照 NumberofSectons 大小，拷贝展开的数据。

```
821a512f 8b4854          mov     ecx,dword ptr [eax+54h]
821a5132 f3a4            rep movs byte ptr es:[edi],byte ptr [esi]      copy PEheader
821a5134 61              popad
821a5135 2bc6            sub     eax,esi
821a5137 03c7            add     eax,edi
821a5139 0fb74806        movzx   ecx,word ptr [eax+6]
821a513d 8d90f8000000    lea     edx,[eax+0F8h]
821a5143 60              pushad
821a5144 037214          add     esi,dword ptr [edx+14h]      PointerToRawdata
821a5147 037a0c          add     edi,dword ptr [edx+0Ch]      VirtualAddress
821a514a 8b4a10          mov     ecx,dword ptr [edx+10h]      SizeofRawData
821a514d e302            jecxz   821a5151
821a514f f3a4            rep movs byte ptr es:[edi],byte ptr [esi]
```

然后获得驱动程序的 EOP,跳向该位置运行

```
821a5165 681f9d489d      push    9D489D1Fh        Get ExFreePool
821a516a ffd3            call    ebx
821a516c 95              xchg    eax,ebp
821a516d 56              push    esi
821a516e ffd5            call    ebp                          PEHeader
821a5170 8b742408        mov     esi,dword ptr [esp+8]
821a5174 ffb42484000000  push    dword ptr [esp+84h]
821a517b 57              push    edi                                  Get driver EOP
821a517c 8b4628          mov     eax,dword ptr [esi+28h]
821a517f 03c7            add     eax,edi
821a5181 ffd0            call    eax              Execute virus driver
821a5183 0bc0            or      eax,eax
821a5185 7d0e            jge     821a5195
821a5187 8b4e50          mov     ecx,dword ptr [esi+50h]
821a518a e309            jecxz   821a5195
kd> r
eax=81af0946 ebx=821a501d ecx=00000000 edx=00002200 esi=81ae7258 edi=81ae7000
eip=821a5181 esp=f8ac17b8 ebp=80545068 iopl=0         nv up ei ng nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000282
821a5181 ffd0            call    eax {81af0946}
```

此时病毒会清除掉当前的内存代码，在系统启动后不留下载入磁盘数据的痕迹。

```
Dispatch routines:
[00] IRP_MJ_CREATE                      f86e0c30      +0xf86e0c30
[01] IRP_MJ_CREATE_NAMED_PIPE           804f420e      nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       f86e0c30      +0xf86e0c30
[03] IRP_MJ_READ                        81a74410      +0x81a74410
[04] IRP_MJ_WRITE                       81a74410      +0x81a74410
[05] IRP_MJ_QUERY_INFORMATION           804f420e      nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             804f420e      nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    804f420e      nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      804f420e      nt!IopInvalidDeviceRequest
```

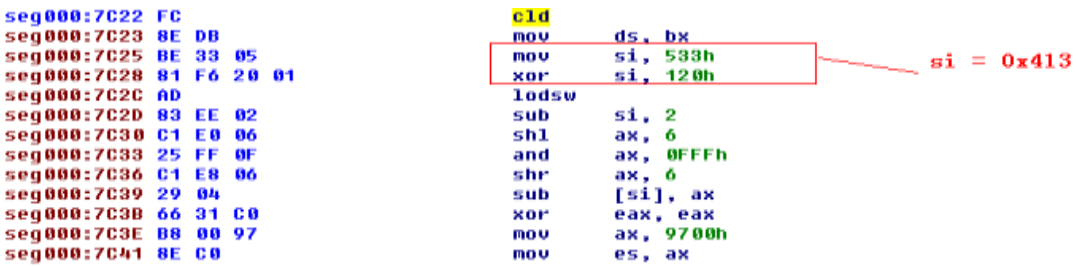
通过 hook 磁盘的 Read /write 以此来保护 MBR 数据不被修复。

三 更多改进-- ShadowGhost 系列攻击手段

3.1 对抗静态检测

Mebroot 的 MBR 代码未做任何的保护，这是容易被检测的。

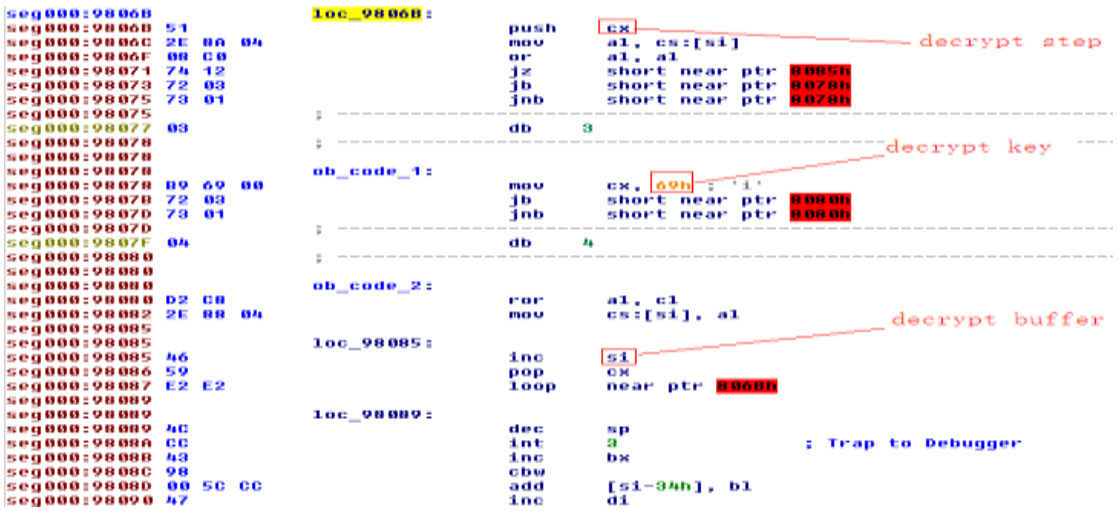
鬼影 1 代进行了微小的改动，不再出现直接操作 bios 0x413 的数据。



鬼影 2，3 则直接插入花指令干扰分析，同时前半部分是解密代码，MBR 后一部分代码及磁盘上的病毒数据都是以加密形式存在。

```
seg000:7C00                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:7C00                 jb      short near ptr loc_7C04+1
seg000:7C02                 jnb     short near ptr loc_7C04+1
seg000:7C04
seg000:7C04 loc_7C04:                               ; CODE XREF: seg000:7C00↑j
seg000:7C04                                         ; seg000:7C02↑j
seg000:7C04                 or      bh, dl
seg000:7C06                 mov     word ptr cs:600h, es
seg000:7C0B                 mov     cs:602h, sp
seg000:7C10                 mov     word ptr cs:604h, ss
seg000:7C15                 mov     dword ptr cs:7BFCh, 7C00h
seg000:7C1F                 lss     sp, cs:7BFCh
seg000:7C25                 pushad
seg000:7C27                 push    ds
seg000:7C28                 mov     bx, cs:413h
seg000:7C2D                 sub     bx, 1Eh
```

解密的算法类似，仅密钥不断变化。

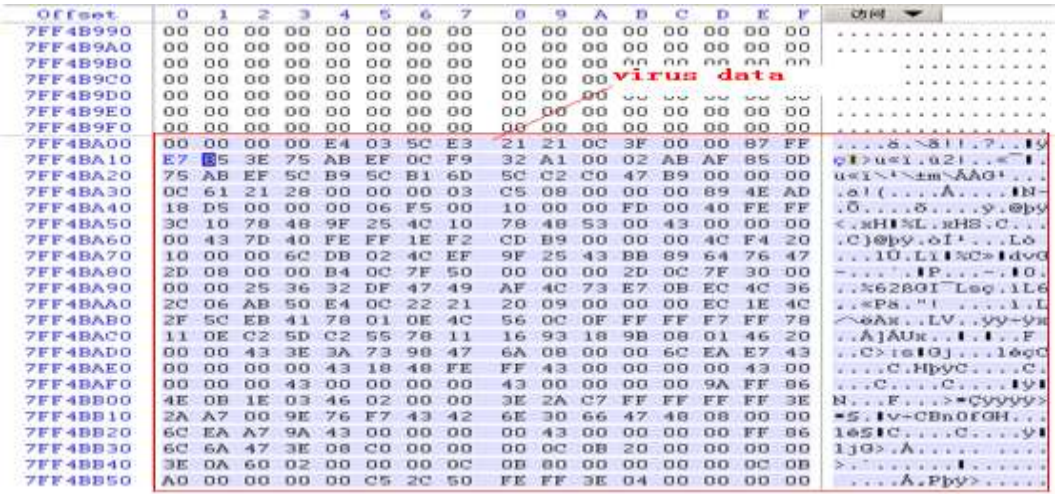


在鬼影 2 代中，加入了 anti-debug 手段，如果发现是调试状态将被病毒引入死循环当中使系统无法启动。

```
seg000:7C50 68 00 00          push    0
seg000:7C53 07                pop     es
seg000:7C54 0F 31             rdtsc
seg000:7C56 66 91             xchg    eax, ecx
seg000:7C58 0F 31             rdtsc
seg000:7C5A 66 29 C8          sub     eax, ecx
seg000:7C5D 66 3D 01 00+      cmp     eax, 1
seg000:7C63 7E 24             jle     short loc_7C89
seg000:7C65 BE 89 00          mov     si, 89h ; '?
seg000:7C68 B9 77 77          mov     cx, 7777h
```

3.2 加密磁盘数据

鬼影 2 代码中是将病毒数据加密写入磁盘 1 ~ 6 扇区当中。
鬼影 3 代在磁盘的 2~64 扇区内不再存放任何数据，而是将病毒数据放置计算过的磁盘尾部。使得反病毒软件对引导扇区的检测失效。



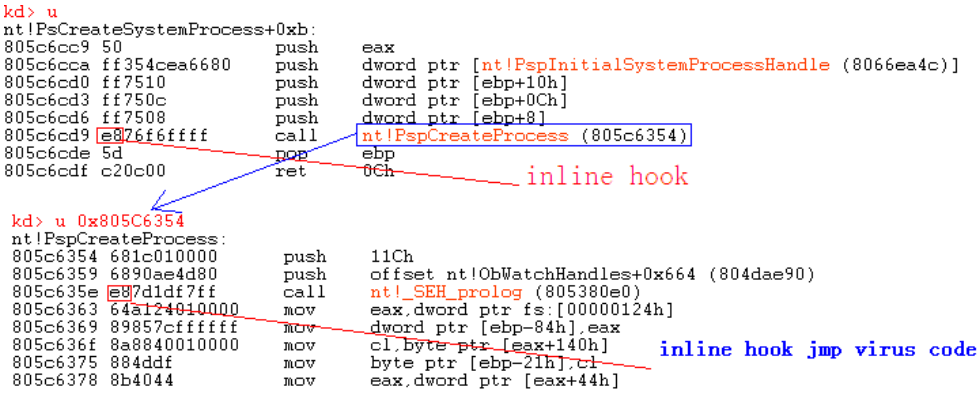
3.3 Kernel hook 方式

鬼影 1 改进定位 BlLoaderBlock 的方式，在进入病毒代码前，从堆栈中获得内核数据高 3 位地址，然后寻址系统调用 BlLoaderBlock 偏移位置，特征匹配时不同于 Mebroot 。

```
PHYSMEM:00422A6A call    near ptr unk_423D31
PHYSMEM:00422A6F call    off_97400          jmp virus code
PHYSMEM:00422A75 cmp     byte_43AEF8, 0
PHYSMEM:00422A7C jz      short loc_422A85
PHYSMEM:00422A7E xor     esi, esi
PHYSMEM:00422A80 jmp     loc_422CDA
```

而鬼影 2，3 则又有了进一步的改进，首先利用当前堆栈里面的数据，定位模块运行的地址高端地址 0x00400000，然后在使用 mebroot 的方式获得 ntoskrnl 基址。

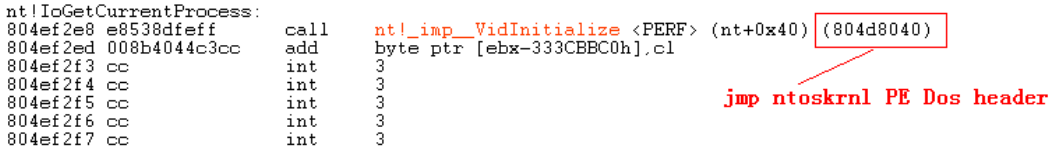
鬼影 1 对 ntoskrnl 感染的方式进行了改进，它会寻找节属性是 0x20000000 的节，然后搜索里面的空白区域，符合条件时，会拷贝 4 扇区的的病毒数据到 ntoskrnl 模块中。获得 PsCreateSytemProcess 地址，间接的进行两次寻址后， hook 调掉 PspCreateProcess 开头部分。



这样病毒的代码就将自身隐藏在内核 ntoskrnl 模块当中。

而鬼影 2，3 则使用另外一种方法来加载自身代码到内核模式。

首先从 ntoskrnl 模块中搜索 IoGetCurrentProcess 函数，然后拷贝 0x3c 代码到 ntoskrnl 模块的 Dos header 空隙处，也就是 0x40 的位置，hook IoGetCurrentProcess 函数开头，使其跳向 0x40 的位置。



很明显鬼影 2，3 的方式更加隐蔽。不容易被发现。

3.4 内存隐藏

不同于 mebroot 直接分配内核内存空间。鬼影 1 利用了 ntoskrnl 模块节当中的空隙，使得 ARK 工具不容易扫描到系统的改动。

鬼影 2，3 则使用改进的一种方式，该方式最早出现在 eeye 的 bootroot 项目中。利用内核态与用户态的共享内存 0FFDF0800h(SharedUserData),做病毒隐藏的驱动位置。

利用 wbinvd 指令刷新 cache 缓存，然后将物理内存下的病毒驱动复制到 SharedUserData 区域中，目前的 ARK 工具同样忽略了这个内存区域的检测。

3.5 加载方式

鬼影1的加载方式也进行了改进，通过KeCapturePersistentThreadState 函数定位 PsLoadedModuleList 结构，利用 MmMapIoSpace 映射物理内存0x9800：0（不同的系统会有所变化） size = 1000h 数据到内核内存中，这里也就是病毒驱动程序，自己实现PEloader的展开功能，获得驱动程序的EOP，调用驱动DriverEntry， 这些都随着ntoskrnl的运行而执行。

鬼影2，3则开始利用，系统在boot阶段将磁盘数据写入文件系统“%systemroot%\xxx.exe”，同时为这个文件添加一个注册表的自启动项，做到随系统自启动。

病毒首先hook IoGetCurrentProcess，再获得PsCreateSystemThread 函数，调用它开启内核线程，在线程中获得zwOpenKey、zwSetValueKey、zwCreateFile 、MmMapIoSpace 函数。将病毒启动的注册路径写入系统。

下图是鬼影2的一个变种程序，写入mgr.exe到注册表启动项，在随系统启动后再删除自身文件。

```
FFDF0B59  6E 00 65 00 5C 00 53 00  4F 00 46 00 54 00 57 00  n.e.\.S.O.F.T.W.
FFDF0B69  41 00 52 00 45 00 5C 00  4D 00 69 00 63 00 72 00  A.R.E.\.M.i.c.r.
FFDF0B79  6F 00 73 00 6F 00 66 00  74 00 5C 00 57 00 69 00  o.s.o.f.t.\.W.i.
FFDF0B89  6E 00 64 00 6F 00 77 00  73 00 5C 00 43 00 75 00  n.d.o.w.s.\.C.u.
FFDF0B99  72 00 72 00 65 00 6E 00  74 00 56 00 65 00 72 00  r.r.e.n.t.V.e.r.
FFDF0BA9  73 00 69 00 6F 00 6E 00  5C 00 52 00 75 00 6E 00  s.i.o.n.\.R.u.n.
FFDF0BB9  00 00 06 00 08 00 C3 0B  DF FF 71 00 51 00 00 00  ......??q.Q...
FFDF0BC9  43 00 3A 00 5C 00 57 00  49 00 4E 00 44 00 4F 00  C.:.\.W.I.N.D.O.
FFDF0BD9  57 00 53 00 5C 00 4D 00  67 00 72 00 2E 00 65 00  W.S.\.M.g.r...e.
FFDF0BE9  78 00 65 00 00 00 26 00  28 00 00 00 00 00 5C 0C  x.e...&.(.....\.
FFDF0BF9  53 00 79 00 73 00 74 00  65 00 6D 00 52 00 6F 00  S.y.s.t.e.m.R.o.
FFDF0C09  6F 00 74 00 5C 00 4D 00  67 00 72 00 2E 00 65 00  o.t.\.M.g.r...e.
FFDF0C19  78 00 65 00 00 00 00 00  E8 03 00 EC 00 00 FF FF  x.e.....?.?.ÿÿ
FFDF0C29  00 00 00 9A CF 00 4D 5A  90 00 03 00 00 00 04 00  ...靴.MZ?.....
FFDF0C39  00 00 FF FF 00 00 B8 00  00 00 00 00 00 00 40 00  ..ÿÿ..?......@.
FFDF0C49  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
FFDF0C59  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

鬼影 3 则是直接打开系统的 beep.sys 文件，并将解密后的 hello_tt.sys 直接覆盖系统的 beep.sys 文件，这样不用修改注册表，就可以做到随系统一起加载启动了。

```
kd> db ffdf0a81
ffdf0a81  4a 00 4c 00 89 0a df ff-5c 00 53 00 79 00 73 00  J.L.....\.S.y.s.
ffdf0a91  74 00 65 00 6d 00 52 00-6f 00 6f 00 74 00 5c 00  t.e.m.R.o.o.t.\.
ffdf0aa1  73 00 79 00 73 00 74 00-65 00 6d 00 33 00 32 00  s.y.s.t.e.m.3.2.
ffdf0ab1  5c 00 64 00 72 00 69 00-76 00 65 00 72 00 73 00  \.d.r.i.v.e.r.s.
ffdf0ac1  5c 00 62 00 65 00 65 00-70 00 2e 00 73 00 79 00  \.b.e.e.p...s.y.
ffdf0ad1  73 00 00 00 4d 5a 90 00-03 00 00 00 04 00 00 00  s...MZ..........
ffdf0ae1  ff ff 00 00 b8 00 00 00-00 00 00 00 40 00 00 00  ............@...
ffdf0af1  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

3.6 自身保护

鬼影1代在驱动中获得PsLoadImageNotifyRoutine，PsCreateProcessNotifyRoutine，PsCreateThreadNotifyRoutine 函数，清除掉反病毒软件加入的过滤回调函数，同时遍历系

统进程，通过校验码，结束内置的杀毒软件进程。同时还会插入一个DPC调用，并在IDT中添加一个门调用，接受Ring3传来的参数，修改后续解密时的密钥，使得静态分析困难。

鬼影2代则 hook PsLoadImageNotifyRoutine,在内核模块加载时判断是否有符合条件的数字签名，是安全软件的模块，则patch模块入口点，使其返回失败。 阻止运行的安全公司产品包括360safe,BITDEFENDER,Trend,AVG,keniu,kingsoft,jingmin,rising, beike,ESET,kaspersky,Norton.

鬼影3代则通过hook磁盘读写保护自身MBR不被修复。一般Hips 都监控disk read/write，鬼影3则借鉴了tdss方式直接hook atapi/scsi DriverStartIo 来保护MBR，这比mebroot更加隐秘，即便清除掉病毒驱动程序，重启后还是会再次加载起来。

```
DriverEntry:   f98149f7
DriverStartIo: f9d95010 hello_tt          virus hook, protect MBR
DriverUnload:  f98103d6
AddDevice:     f980e47c

Dispatch routines:
[00] IRP_MJ_CREATE                      f98096f2        +0xf98096f2
[01] IRP_MJ_CREATE_NAMED_PIPE           804f454a        nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       f98096f2        +0xf98096f2
[03] IRP_MJ_READ                        804f454a        nt!IopInvalidDeviceRequest
[04] IRP_MJ_WRITE                       804f454a        nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION           804f454a        nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             804f454a        nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    804f454a        nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      804f454a        nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS               804f454a        nt!IopInvalidDeviceRequest
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION    804f454a        nt!IopInvalidDeviceRequest
[0b] IRP_MJ_SET_VOLUME_INFORMATION      804f454a        nt!IopInvalidDeviceRequest
[0c] IRP_MJ_DIRECTORY_CONTROL           804f454a        nt!IopInvalidDeviceRequest
[0d] IRP_MJ_FILE_SYSTEM_CONTROL         804f454a        nt!IopInvalidDeviceRequest
[0e] IRP_MJ_DEVICE_CONTROL              f9809712        +0xf9809712
[0f] IRP_MJ_INTERNAL_DEVICE_CONTROL     f9805852        +0xf9805852
[10] IRP_MJ_SHUTDOWN                    804f454a        nt!IopInvalidDeviceRequest
[11] IRP_MJ_LOCK_CONTROL                804f454a        nt!IopInvalidDeviceRequest
[12] IRP_MJ_CLEANUP                     804f454a        nt!IopInvalidDeviceRequest
[13] IRP_MJ_CREATE_MAILSLOT             804f454a        nt!IopInvalidDeviceRequest
[14] IRP_MJ_QUERY_SECURITY              804f454a        nt!IopInvalidDeviceRequest
[15] IRP_MJ_SET_SECURITY                804f454a        nt!IopInvalidDeviceRequest
[16] IRP_MJ_POWER                       f980973c        +0xf980973c
[17] IRP_MJ_SYSTEM_CONTROL              f9810336        +0xf9810336
[18] IRP_MJ_DEVICE_CHANGE               804f454a        nt!IopInvalidDeviceRequest
[19] IRP_MJ_QUERY_QUOTA                 804f454a        nt!IopInvalidDeviceRequest
[1a] IRP_MJ_SET_QUOTA                   804f454a        nt!IopInvalidDeviceRequest
[1b] IRP_MJ_PNP                         f9810302        +0xf9810302

kd> u f9d95010
hello_tt+0x1010:
f9d95010 8bff            mov     edi,edi
f9d95012 55              push    ebp
f9d95013 8bec            mov     ebp,esp
f9d95015 83ec30          sub     esp,30h
f9d95018 8b450c          mov     eax,dword ptr [ebp+0Ch]
f9d9501b 50              push    eax
f9d9501c e85f020000      call    hello_tt+0x1280 (f9d95280)
f9d95021 8945f0          mov     dword ptr [ebp-10h],eax
```

针对中国区的 bootkit 样本，可以看到加载方式，对抗检测手段都有了很大的改进。一旦计算机的底层的控制权被病毒获得将是很危险的，在 windows 启动阶段找到更多的隐藏加载驱动方式虽然困难，但总还是有突破的手段，tdl4 已经展现了这方面新的思路，因为在这一阶段操作系统是没有太多的安全措施防护恶意程序对内核的修改。

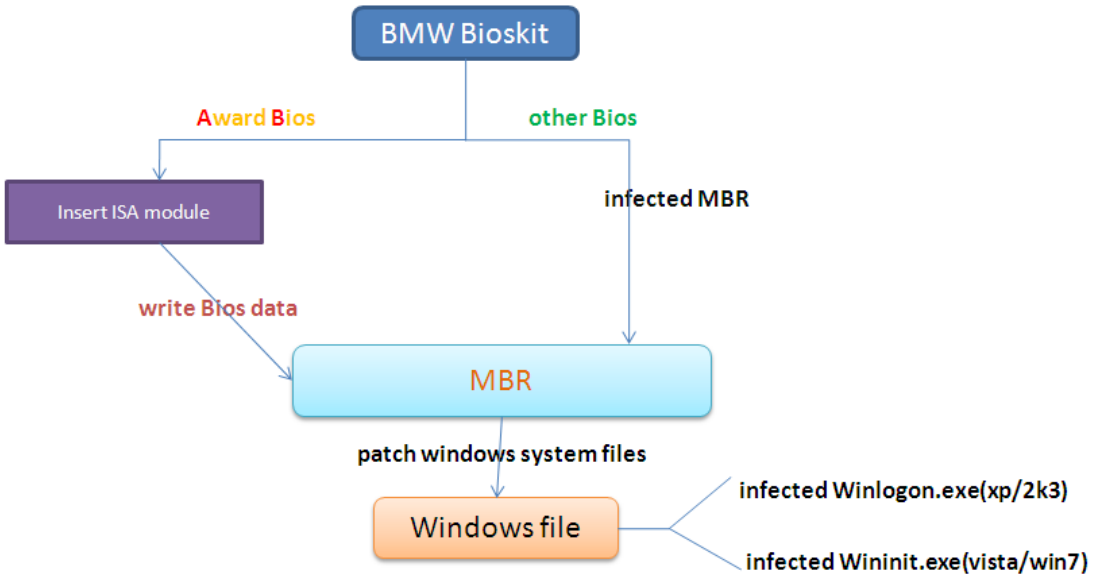
四 更高级的攻击手段 bioskit

2011 年 9 月 14 日，中国安全公司 360safe 发布预警，在半个月内 BMW bioskit 病毒感染约 5 台电脑，目前 bioskit 病毒仅攻击 Award BIOS 主板，对于非 AwardBIOS 的主板则感染 MBR。

经过代码分析我们认为 BMW bioskit 是一个过渡性质的病毒，是 bioskit 和 bootkit 的结合体，这取决于在 bios 感染方面的不成熟。因为感染 bios 目前还仍处于概念阶段，虽然

blackhat 07，CanSecWest 09 都有安全研究人员提出了攻击 bios 的方法，但通用的方法仍然困难，这使得 BMW bioskit 选择仅攻击 AwardBIOS。

下面是 BMW bioskit 的感染系统的流程图



插入 bios 中的病毒 ISA 模块，主要是保护 MBR 数据不被修复。

- 1　从 ISA 代码偏移 0x5d 开始，拷贝 15 个扇区大小的数据到引导区　。
-
- 2　判断当前 bios 是否支持 int 13 扩展读，不支持退出 isa 模块，交由 MBR 继续引导　。
-
- 3 硬编码 0x97f0 地址为 DAP 数据，从磁盘偏移 0 开始读 1 扇区数据到 0x8a00 ，如果失败退出 isa 模块　。
-
- 4 读取 0x8a92 dword 数据，比较是否是 0x31746e69 （可看作感染表示，“int1”），如果是，则设置 ax = 1，继续运行。否则 copy 0x80 代码到 0x7d80 ,该代码功能是向屏幕打印字符串，是病毒作者调试代码的输出提示。
-
- 5 如果已经感染，正常引导系统，退出 isa 模块。
-
- 6 如果没感染，则从当前内存 0x7c00 地址位置，写 15 个扇区大小的数据到磁盘 0 柱-0 面-1 扇区位置。

病毒在 MBR 中主要是通过的文件系统的解析感染系统文件。

1 伪造一个开头，anti-mbr check 。

2 copy 自身去 0x600 内存，判断是否 int 13 支持读。

3 从 0 柱 0 道 2 扇区开始读 6 个扇区的数据到 0x7c00.

4 读取 cmos 数据，利用 0x70 0x71 端口，记录日期，运行次数，当日期不符合 coms 中记录的日期则重写当前日期到自身内存 0x625(byte),0x626(byte)，以此来记录当天运行次数，再将当前 0x600 地址大小 512 字节数据写回磁盘 MBR。

5 读取 7 扇区（原 windowsMBR）的数据，到 0x4000 位置，获得磁盘活动分区，获得引导分区的起始位置 main_dbr,保存该值到 0x60c（byte）。

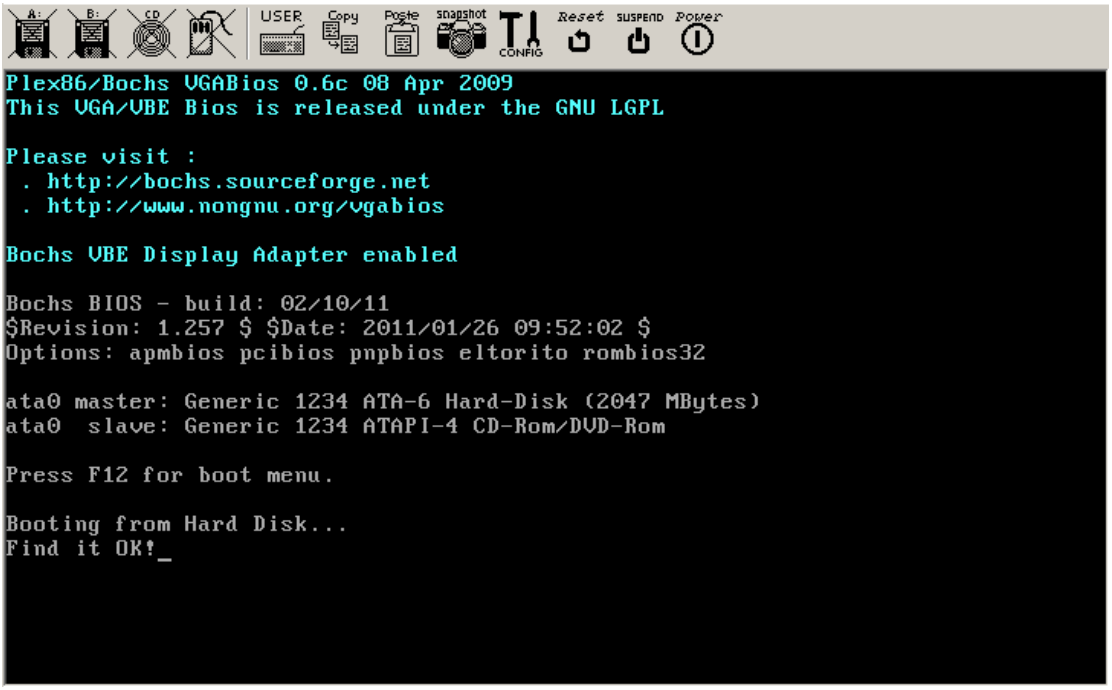
6 自己加载 DBR 到 0x7c00，判断引导分区的文件系统，NTFS or FAT32, 计算扇区字节数据 sec_size,每簇扇区数 cluster_by_seccunt,隐藏扇区数 hide_seccunt, MFT 簇号 MFT_NO, 每个 MFT 记录簇数 MFT_record，记录这些数据到当前 0x600 空白处。

7 扫描 DBR 中 NDTLDR 字符串，相当于校验当前是否合法。

8　分 FAT32 or NTFS 方式解析，读取磁盘 2 扇区数据到 0x1200，检测是否是 winlogon.exe 或者 Wininit.exe，如果是，则解析 PE 头，定位 EOP，检测该 EOP 位置的指令是否已经被感染。

9　感染后的 exe 开头是自解密的代码，是则，已经感染，打印 Find it OK,加载 7 扇区数据到 0x7c00，控制权交原 MBR，否则通过写磁盘数据对 exe 进行感染，完成后打印 Find it ok，跳入原 MBR 执行。

感染后的系统启动如图



感染后的 winlogon 或者 wininit 先执行解密操作，最后跳向文件 EOP。这样的感染方式，使得 Award BIOS 的用户即便更换硬盘也不能消除病毒。

五　防御者的任务

bootkit 变化的手段是难以准确预测的，但防御的重点是直接的读写磁盘主引导扇区的操作。当一个未知的进程去执行这样的风险操作时，我们应提示警告，虚警是容易排除的，因为设置多系统引导的工具总是有限的几个。而未知的一个进程直接操作 MBR 是高度危险的。

5.1 Ring3 拦截

Hook CreateFile， 在打开\\\\.\\PhysicalDriver0 时，需要判断后续的写入操作，通过文件写入偏移来判断它的内容，甚至可用 16 位的仿真器来模拟分析写入 MBR 的数据行为。

5.2 Ring0 拦截

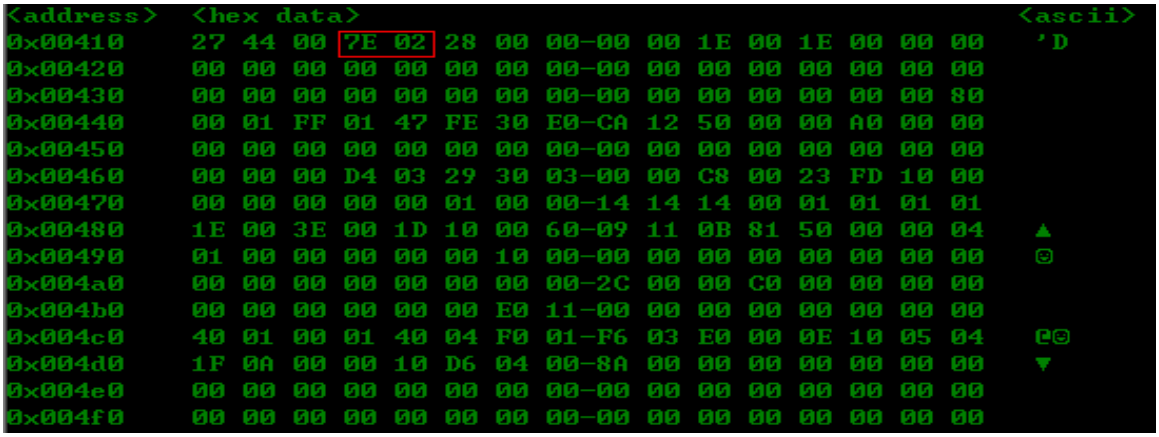
对磁盘的读写要加入判断,可以利用 RtlInitUnicodeString(&uniobjname,L"\\Driver\\Disk");调用 ObReferenceObjectByName 来获得磁盘的驱动对象，hook MajorFunction[IRP_MJ_WRITE] 写入操作，检测方案同上。

5.3 实模式内存搜索

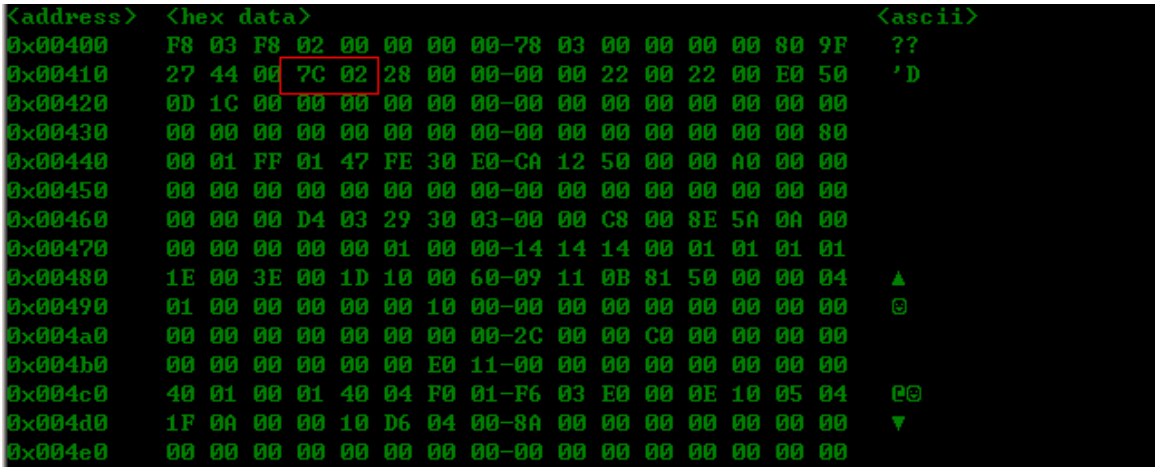
鬼影 2，3 都选择了在磁盘上存放已经加密的病毒数据包括原 windowsMBR 数据，但执行完毕后，自身解密的数据却在实模式内存中，恢复原 windows 系统的 MBR 可以从这个方向着手。

- 病毒在系统内存空间的驻留代码已经解密完毕，我们计算起止地址,然后搜索原系统的 MBR 特征即可。
- 读取 Bios 区数据，即 0x413 位置，WORD 大小数据,即实模式系统内存大小。
- int free_mem_addr = readphymem(0x413,2);
- 搜索范围(free_mem_addr ~ 0xa0000)

开始的 BIOS 中记录的系统内存信息,系统内存大小为 0x27e Kb



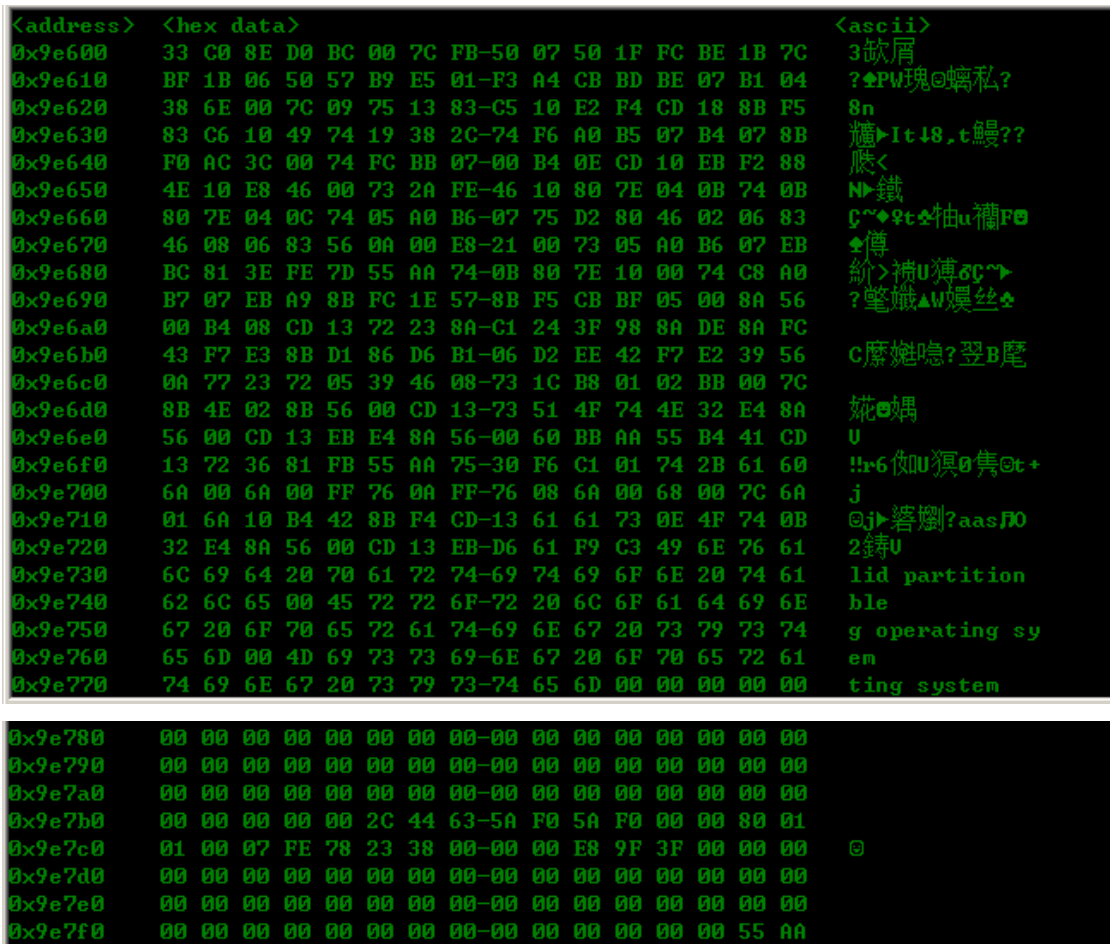
被病毒感染后，修改为 0x27cKB



```
<address>  <hex data>                                          <ascii>
0x00400    F8 03 F8 02 00 00 00 00-78 03 00 00 00 00 80 9F     ??
0x00410    27 44 00 7C 02 28 00 00-00 00 22 00 22 00 E0 50     'D
0x00420    0D 1C 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x00430    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 80
0x00440    00 01 FF 01 47 FE 30 E0-CA 12 50 00 00 A0 00 00
0x00450    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x00460    00 00 00 D4 03 29 30 03-00 00 C8 00 8E 5A 0A 00
0x00470    00 00 00 00 00 01 00 00-14 14 14 00 01 01 01 01
0x00480    1E 00 3E 00 1D 10 00 60-09 11 0B 81 50 00 00 04     ▲
0x00490    01 00 00 00 00 00 10 00-00 00 00 00 00 00 00 00     ☺
0x004a0    00 00 00 00 00 00 00 00-2C 00 00 C0 00 00 00 00
0x004b0    00 00 00 00 00 00 E0 11-00 00 00 00 00 00 00 00
0x004c0    40 01 00 01 40 04 F0 01-F6 03 E0 00 0E 10 05 04     @☺
0x004d0    1F 0A 00 00 10 D6 04 00-8A 00 00 00 00 00 00 00     ▼
0x004e0    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

所以搜索实模式内存的起始地址就是 0x27c * 2^10 /2 ^4 = 0x9f00,搜索的范围是 0x9f00 ~0xa0000。

下面是一个被鬼影 3 感染的系统上，实模式的物理内存中发现了解密后的原 Windows MBR 数据。修复内核 hook，直接将 0x9e600 的数据写入 MBR 即可恢复 bootkit 的破坏。



```
<address>  <hex data>                                          <ascii>
0x9e600    33 C0 8E D0 BC 00 7C FB-50 07 50 1F FC BE 1B 7C     3?|?P?P??????|
0x9e610    BF 1B 06 50 57 B9 E5 01-F3 A4 CB BD BE 07 B1 04     ?♠PW瑰☺螭私?
0x9e620    38 6E 00 7C 09 75 13 83-C5 10 E2 F4 CD 18 8B F5     8n
0x9e630    83 C6 10 49 74 19 38 2C-74 F6 A0 B5 07 B4 07 8B     爐▶It↓8,t鳗??
0x9e640    F0 AC 3C 00 74 FC BB 07-00 B4 0E CD 10 EB F2 88     瞧<
0x9e650    4E 10 E8 46 00 73 2A FE-46 10 80 7E 04 0B 74 0B     N▶鐵
0x9e660    80 7E 04 0C 74 05 A0 B6-07 75 D2 80 46 02 06 83     Ç~◆♀t♣怞u襹F☻
0x9e670    46 08 06 83 56 0A 00 E8-21 00 73 05 A0 B6 07 EB     ♠僔
0x9e680    BC 81 3E FE 7D 55 AA 74-0B 80 7E 10 00 74 C8 A0     紩>祌U獏ðÇ~▶
0x9e690    B7 07 EB A9 8B FC 1E 57-8B F5 CB BF 05 00 8A 56     ?氅孅▲W嬑丝♣
0x9e6a0    00 B4 08 CD 13 72 23 8A-C1 24 3F 98 8A DE 8A FC
0x9e6b0    43 F7 E3 8B D1 86 D6 B1-06 D2 EE 42 F7 E2 39 56     C縻嬔噫?翌B氅
0x9e6c0    0A 77 23 72 05 39 46 08-73 1C B8 01 02 BB 00 7C
0x9e6d0    8B 4E 02 8B 56 00 CD 13-73 51 4F 74 4E 32 E4 8A     嫉☻媧
0x9e6e0    56 00 CD 13 EB E4 8A 56-00 60 BB AA 55 B4 41 CD     V
0x9e6f0    13 72 36 81 FB 55 AA 75-30 F6 C1 01 74 2B 61 60     !!r6伋U猥0隻☺t+
0x9e700    6A 00 6A 00 FF 76 0A FF-76 08 6A 00 68 00 7C 6A     j
0x9e710    01 6A 10 B4 42 8B F4 CD-13 61 61 73 0E 4F 74 0B     ☺j▶簚嬋?aasJO
0x9e720    32 E4 8A 56 00 CD 13 EB-D6 61 F9 C3 49 6E 76 61     2鋳V
0x9e730    6C 69 64 20 70 61 72 74-69 74 69 6F 6E 20 74 61     lid partition
0x9e740    62 6C 65 00 45 72 72 6F-72 20 6C 6F 61 64 69 6E     ble
0x9e750    67 20 6F 70 65 72 61 74-69 6E 67 20 73 79 73 74     g operating sy
0x9e760    65 6D 00 4D 69 73 73 69-6E 67 20 6F 70 65 72 61     em
0x9e770    74 69 6E 67 20 73 79 73-74 65 6D 00 00 00 00 00     ting system
```

```
0x9e780    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x9e790    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x9e7a0    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x9e7b0    00 00 00 00 00 2C 44 63-5A F0 5A F0 00 00 80 01
0x9e7c0    01 00 07 FE 78 23 38 00-00 00 E8 9F 3F 00 00 00     ☺
0x9e7d0    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x9e7e0    00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0x9e7f0    00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA
```

5. 4 Hook DBR

DBR（Dos Boot Record）这是目前 malware 还没有十分关注的 windows 启动位置，因为无论 Bioskit 还是 Bootkit 很关键的一点是要通过 hook 磁盘 int13 中断，加载自身数据。能顺利的进入 BIOS 当然是最好的，否则修改 MBR 也是为了率先获得 int13 的执行权。

当病毒修改完 IVT 后，会加载原 windowsMBR，释放控制权，直到 ntldr 启动过程中不再参与启动过程。而在这一个过程中 DBR 会被系统载入到 0x7c00 位置加载执行，同时引导 ntfs 或者 fat32 系统进入内核初始化阶段。如果在这过程中我们能获得执行，检测中断情况，内核文件的完整性及已知 bootkit 病毒 patch 内核文件的位置。就可以断掉 bootkit 劫持系统启动的这一过程。

当然，目前已知 tophet 使用 NtBootdd.sys 来加载自身，避免了 MBR 修改，int13 修改，但这仅是少数研究性质攻击方式。

btw：分析使用工具包括，
bochs,
windbg,
ida,
txm