MBR 病毒分析

Date：2010.10.29
Author：Cryin’
Blog：http://hi.baidu.com/justear

一、基础知识

1、 Windows 启动过程

系统引导过程主要由以下几个步骤组成(以硬盘启动为例)

1、 开机;

2、 BIOS 加电自检(POST---Power On Self Test),内存地址为 0fff:0000;

3、 将硬盘第一个扇区(0 头 0 道 1 扇区,也就是 Boot Sector)读入内存地址 0000:7c00 处;

4、 检查(WORD)0000:7dfe 是否等于 0xaa55.若不等于则转去尝试其他介质;如果没有其他启动介质,则显示 ”No ROM BASIC” ,然后死机;

5、 跳转到 0000:7c00 处执行 MBR 中的程序;

6、 MBR 先将自己复制到 0000:0600 处,然后继续执行;

7、在主分区表中搜索标志为活动的分区.如果发现没有活动分区或者不止一个活动分区,则停止;

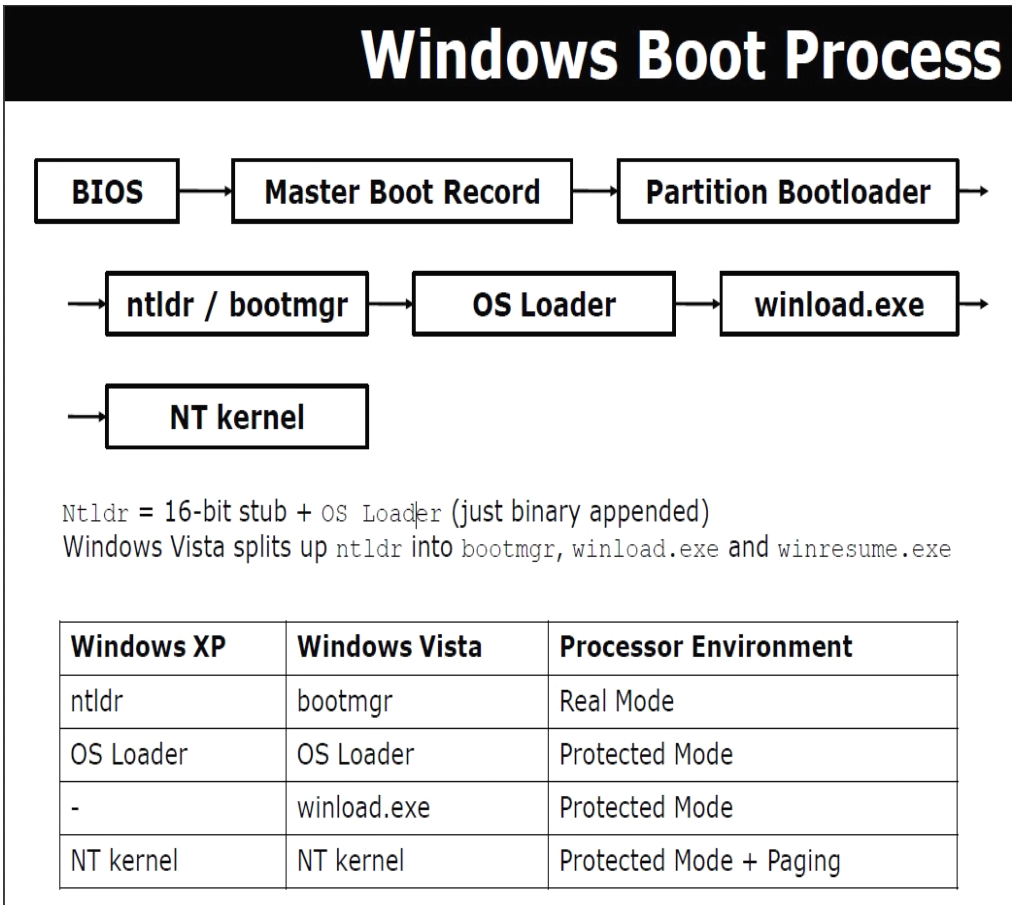
8、 将活动分区的第一个扇区读入内存地址 0000:7c00 处;

9、检查(WORD)0000:7dfe 是否等于 0xaa55,若不等于则显示 “Missing Operating System”,然后停止,或尝试软盘启动;

10、 跳转到 0000:7c00 处继续执行特定系统的启动程序;

11、 启动系统. 以上步骤中(2),(3),(4),(5)步由 BIOS 的引导程序完成;(6),(7),(8),(9),(10)步由 MBR 中的引导程序完成.

Windows 启动过程主要由以下几个步骤组成，其中 vista 和 win7 可一概而论；



Windows XP	Windows Vista	Processor Environment
ntldr	bootmgr	Real Mode
OS Loader	OS Loader	Protected Mode
-	winload.exe	Protected Mode
NT kernel	NT kernel	Protected Mode + Paging

2、 硬盘主引导区结构

硬盘的主引导区在 0 柱面 0 磁道 1 扇区，包括硬盘主引导记录 MBR（Main Boot Record）、四个分区表 DPT（Disk Partition Table）信息和主引导记录有效标志字三部分，如表所示：

区域	信息
0000H—008AH	主引导记录启动程序
008BH-00D9H	主引导记录启动字符串
00DAH—01BDH	空闲区
01BEH—01CDH	分区1结构信息
01CEH—01DDH	分区2结构信息
01DEH—01EDH	分区3结构信息
01EEH—01FDH	分区4结构信息
01FEH—01FFH	55□AAH主引导记录有效标志

主引导记录 MBR 从 0000H 开始到 00D9H 结束，共 218 个字节。MBR 的作用就是检查分区表是否正确以及确定哪个分区为引导分区，并在程序结束时把该分区的启动程序（也就是操作系统引导扇区）调入内存加以执行。MBR 是由分区程序（例如 DOS 的 Fdisk.exe）产生的，在不同的操作系统平台下，这个扇区的内容可能不完全相同。主引导记录比较容易编写，例如，我们自己也可以编写一个这样的程序，只要能完成前述的任务就可以了（参见网上资料“主引导扇区代码（MBR）”）。正是因为主引导记录容易编写，所以才出现了很多的引导区病毒（eeye bootroot、 Vbootkit 、Stoned bootkit、Sinowal 等）。

我们都知道，任何硬盘最多只能有四个分区。分区表自偏移 01BEH 处开始，共 64 个字

节，表中可填入四个分区信息，每 16 个字节为一个分区说明项，这 16 个字节含义如分区表结构信息表（偏移量）所示。

图：分区表结构信息表（偏移量）

偏移	长度	含义
00H	1	活动分区指示符，可能取值为80H或00H。其中80H表示为可自举分区(仅有一个)，00H表示其他分区
01H	1	分区起始磁头号
02H	1	低6位是分区开始的扇区，高2位是分区开始的柱面的头两位
03H	1	分区开始的起始柱面号的低8位
04H	1	系统标志，可能取值有01H、04H、05H、06H。其中01H表示采用12位FAT格式的DOS分区；04H表示采用16位FAT格式的DOS分区；05H表示为扩展DOS分区；06H表示为DOS系统
05H	1	分区终止头号
06H	1	低6位为分区结束的扇区号，头2位为结束柱面号的前2位
07H	1	分区结束柱面号的低8位。
08H	4	本分区前的扇区数，低位字节在前
0CH	4	本分区总的扇区数，低位字节在前

分区表每一项结构简介

BYTE State:分区状态,0=未激活,0x80=激活(注意此项);

BYTE StartHead:分区起始磁头号;

WORD StartSC:分区起始扇区和柱面号,底字节的底 6 位为扇区号,高 2 位为柱面号的第 9,10 位,高字节为柱面号的低 8 位;

BYTE Type:分区类型,如 0x0B=FAT32,0x83=Linux 等,00 表示此项未用;

BYTE EndHead:分区结束磁头号;

WORD EndSC:分区结束扇区和柱面号,定义同前;

DWORD Relative:在线性寻址方式下的分区相对扇区地址(对于基本分区即为绝对地址);

DWORD Sectors:分区大小(总扇区数).

3、 MBR 病毒感染基本思想

MBR 病毒感染的基本思想，首先是读取主引导记录和把分区表从主引导记录中复制出来。然后，MBR 病毒把自己的包含恶意二进制数据的主引导记录放到主引导扇区，并复制新的分区表到它。但是，并非只有分区表需要保留，还有微软公司原来的主引导记录也需要保存下来。为此，MBR 病毒复制原始主引导记录到其它 64 个未用到的扇区。到 MBR 病毒执行完自己的操作后在读取原始的主引导记录并跳到 0x7c00 处执行来引导开机。

二、样本分析

1、 常驻内存

最早设计 DOS 操作系统时，PC 机的硬件系统只支持 1M 字节的寻址空间，所以 DOS 只能直接管理最多 1M 字节的连续内存空间。在这 1M 内存中，仅有 640K 被留给应用程序使用，它们被称为常规内存或基本内存(Base Memory)。

在内存 0040h:0013h 开始处的两个字节描述了内存中未用内存的高端位置，它的值是 KB 的倍数。病毒程序首先将自身复制到内存的高端，修改内存容量标志单元 0:413H，在原有值的基础上减去病毒长度，使得病毒的 int 13h 能够常驻内存，并将原 int 13h 磁盘中断服务程序的中断向量保存，然后将该中断向量置成新的 int 13h 中断程序入口，即加上一段病

毒感染程序，每次调用 int13 时病毒程序就会被执行。

```
mov   ax, ds:[0413h]
and   al, NOT 3
sub   ax, 4
mov   ds:[0413h], ax
shl   ax, (10-4)
mov   es, ax
```

以上代码实现为病毒程序预备 4KB 的常规内存以实现代码常驻内存。

2、 HOOK INT 13

BIOS Int 13H 调用是 BIOS 提供的磁盘基本输入输出中断调用，它可以完成磁盘(包括硬盘和软盘)的复位，读写，校验，定位，诊断，格式化等功能。 所以 MBR 病毒一般都是通过 HOOK int 13h 执行恶意代码。

```
mov   eax, ds:[13h*4]
mov   es:[INT13HANDLER - @SYSRQCODE16_START], eax
mov   word ptr ds:[13h*4], (@Int13Hook)
mov   ds:[(13h*4) + 2], es
```

以上代码实现对 INT 13H 的 HOOK，并将原始 int 13h 的入口 INT13HANDLER 保存并使其指向新的入口点@Int13Hook。

3、HOOK NTLDR

在 HOOK INT 13H 部分完成后需要对 NTLDR 进行 HOOK，所以在 HOOK INT 13H 中断操作中需要对 NTLDR 的特征码进行扫描。

```
  mov   cl, al
  mov   al, 8Bh
  shl   cx, 9                           ; (AL * 200h)
  mov   di, bx

@Int13Hook_scan_loop:
                ; 8B F0       MOV ESI, EAX
                ; 85 F6       TEST ESI, ESI
                ; 74 21       JZ $+23h
                ; 80 3D ...   CMP BYTE PTR [ofs32], imm8
                ; (the first 6 bytes of this signature exist
                ; in other modules!)
  repne scasb
  jne   short @Int13Hook_scan_done

  cmp   dword ptr es:[di], 74F685F0h
  jne   short @Int13Hook_scan_loop

  cmp   word ptr es:[di+4], 8021h
  jne   short @Int13Hook_scan_loop

  mov   word ptr es:[di-1], 15FFh       ; FFh/15h: CALL NEAR [ofs32]
```

以上代码实现对 NTLDR OSLoder 模块里的 6 字节进行扫描，并在 FFh/15h:使用 CALL NEAR [OFS32]指令进行 NTLDR HOOK，该指令寻址采用绝对地址，类似指令也可以。HOOK 之后置 CALL NEAR [OFS32]指令跳转到恶意代码的地址处执行。

3、 HOOK NTOSKRNL

当代码再次运行到 NTLDR OSLoder 被 HOOK 处时，就会执行恶意代码，此时代码运行在保护模式下。在这里通过扫描_BlLoaderData 模块基址来获取 NTOSKRNL 的镜像地址。通过 PE 搜索 NTOSKRNL.EXE 的导出函数，并对 NTOSKRNL 的导出函数 KeAddSystemServiceTable 进行 HOOK。因为进程 SMSS.EXE 会加载 WIN32K.SYS，当 WIN32K.SYS 初始化时，它必须调用 KeAddSystemServiceTable 来为用户以及 GDI 系统程序提供注册。所以通过对这个函数 HOOK 操作，可以更快更好的把它加载到执行的 WIN32K.SYS 中，同时我们可以使用免费的系统服务表(_W32pServiceTable)和相应的参数表 (_W32pArgumentTable)。

对 KeAddSystemServiceTable 的 HOOK 主要是通过 HOOK NtUserRegisterClassExWOW 这个系统调用函数来实现的，因为那个函数没有输出，因此需要一点更复杂的操作。针对这个问题，我们的答案是通过对在 W32pServiceTable 的代码进行逐个函数扫描，直到找到 NtUserRegisterHotkey。一旦发现这个函数，就替换掉 WIN32K 系统表的入口并用一个指针指到我们自己的钩子函数处，最终替换掉原先这个函数的指针，并从已恢复运行的 KeAddSystemServiceTable 中脱掉钩子。

该 NtUserRegisterClassExWOW 钩子函数并不复杂，它只用来比较这个类的名称(都 2 个参数与一个 unicode 字符串指针)和 SAS Windows class 来完成匹配，它用我们在用户映射的用户共享数据中的那个更换过的 SASWndProc 代码来替换掉‘lpfnWndProc 领域中的 WNDCLASSEXW 结构，替换完后再把在 PEB 特定位置的原始函数指针进行存储操作。对 NtUserRegisterClassExWOW 的 HOOK 操作可以截获所有的应用层程序窗口类，以便再次 HOOK 窗口类操作，此时代码就运行到 NT 的应用层模式下了。

经过在开源代码基础之上的试验，成功实现在 Winlogon 加载之前运行一个 CMD 实例。

参考文献：

NTFS 文件系统启动扇区代码
主引导扇区代码（MBR）
FAT 文件系统启动扇区代码
扩展 INT 13H 调用规范
驻留程序设计 TSR(Terminate but Stay Resident)
PE 文件格式
NTLDR 源代码及分析
Bochs 调试
Windbg 与 VMWare 的连接调试
80x86 汇编程序设计与保护模式概念

涉及工具：

Masm6.11
Nasm
IDA5.0
Bochs
Windbg
WinHex

参考开源项目：

eeye bootroot、Vbootkit、Stoned Bootkit、Sinowal Bootkit、ROMOS。

Date：2010-5-26

Author：Cryin
Link:　http://hi.baidu.com/justear/blog
注：文中多数内容均来自互联网，时间仓促，加上本人菜鸟初学，可能有遗漏、甚至错误的地方。还有在实际编程中对于保护模式下线性寻址的理解并不是很明确，所以未在本文中提及。对于后续的扩展的思考，bios 后门的开发只需要前面加一个 hook int 19 的模块即可实现，网上亦有同类文章可参考学习。