基于 PBD 技术的 Anti-Bootkit 优先启动①

Anti-Bootkit Boot Ahead Based on PBD

李月锋　周学海　(中国科学技术大学 计算机科学与技术系 安徽 合肥 230027;
中国科学技术大学 苏州研究院 江苏 苏州 215000)

摘　要： Bootkit 是继承自 Rootkit 内核权限获取和自我痕迹擦除技术的 Rootkit 高级发展形式，对系统启动和内核准入安全提出了最新挑战。在 Bootkit 入侵和 Anti-Bootkit 的检测的攻防对抗中，能做到优先启动往往是制胜的关键；而目前主流的 Anti-Bootkit/Rootkit 软件由于固守传统的启动方式，很容易被 Bootkit 利用固有优势绕过、劫控、篡改、移除而形同虚设。通过反汇编并跟踪调试 Windows 引导执行流程，反利用 Bootkit 入侵思想，提出了新的利用 PBD(pre-bootable driver)技术来做到优先启动的方案设计。利用该方案可以有效做到 Anti-Bootkit 优先启动，为其实施布控提供了可信的内核执行环境，从而为防范 Bootkit 提供了新的思路和技术参考。

关键词： PBD(pre-bootable driver) Bootkit 反汇编 优先启动 劫控

1　引言

Rootkit 作为程序的集合，用于实现自身及系统中特定资源和活动的隐藏，破坏可信任计算机的完整性[1]。Bootkit 是继承自 Rootkit 内核权限获取和自我痕迹擦除技术的 Rootkit 高级发展形式。Bootkit 通过感染可引导的外设固件和关键系统文件，驻留在整个系统的启动过程，获取系统控制底层权限并且擦除自我存在痕迹，从而为网络恶意代码的进一步攻击行为提供隐蔽、可靠、持久的执行环境。

Bootkit 这个概念自 2005 年 eEye Digital 安全公司的研究人员研究如何在系统启动时利用 BIOS 接入 windows 内核的"BootRoot"项目中被第一次提出以后，也越来越为黑客所钟爱，最近更被全球知名的杀毒厂商卡巴斯基在其年度报告中列为 2008 年度互联网安全的重大挑战[2]。

在 Bootkit 感染入侵和 Anti-Bootkit 检测恢复的攻防策略中，能做到优先启动往往是制胜对方的关键；而目前主流的 Anti-Bootkit/Rootkit 软件由于固守传统的启动方式，很容易被 Bootkit 利用固有优势劫控、篡改、移除而形同虚设。

本文在第2节中分析了传统的Anti-Bootkit启动方式和时机的弱点和不足，反利用 Bootkit 入侵思想，提出了新的利用 PBD(pre-bootable driver)技术来做到优先启动的方案设计。在第 3 节中分析了面临的技术问题，并给出可行解决方案，在第 4 节描述了设计实现以及实例验证。

2　传统的Anti-Bootkit启动方式和时机

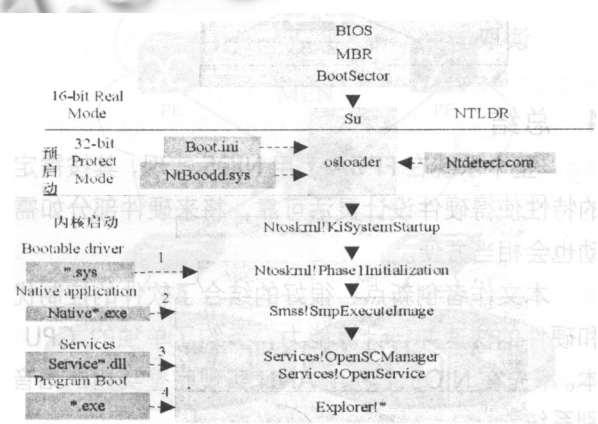

图 1　Windows 引导过程执行流程

① 基金项目:电子信息产业发展基金(财建[2008]329,工信部运[2008]97)
收稿时间:2009-03-26

作为单纯的学术研究，我们通过反汇编并跟踪调试 Windows 引导执行流程，根据启动顺序还原出正常的 windows 引导过程如上图灰色框，以及执行流程中加载可执行模块的方式和时机如上图绿色框。

传统的 Anti-Bootkit 启动时机可总结为如下四类[3]:

a.可启动驱动

Osloader 是预启动过程中最后一个获取引导控制权的可执行模块，由它调用使用内置的文件系统操作加载内核镜像(ntoskrnl)并切入内核(KiSystem Startup),传入的加载参数为 LOADER_PARAMETER_BLOCK 复杂数据结构。其中的 BootDriverListHead 为保存了所有可以启动的驱动路径的双向链表；在内核启动的 phase1 阶段初期 Phase1Initialization 函数会依次调用其所指示的可启动驱动的入口函数 DriverEntry，而获取执行机会。

b.Native Application 启动方式:

Windows 内核启动后 phase1 阶段的初期，调用 RtlCreateUserProcess 创建会话管理器进程(smss)。该进程的进程体函数最终会调用 SmpExecuteImage 加载 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager 下 BootExecute 指向的 Native Application 磁盘镜像和传入参数,切入其入口函数 NtProcessStartup 执行操作。由于此时可以使用 NTDLL.DLL 提供的原始服务(Native API)，具有和内核同等的最高执行权限；而且内核尚未初始化完毕，此时的执行环境较为干净，往往是主流杀毒厂商例如卡巴斯基等青睐的启动点。

c.服务方式

Windows 服务是在用户登陆前为拥有用户授权级进行管理的后台程序。内核启动后期创建的 Services 进程会调用 OpenService 例程搜索 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services 下的所有服务名，加载 Parameters 子键中的 ServiceDll 指向的动态链接库，调用已注册的服务入口函数 ServiceMain 执行服务任务。主流杀毒厂商往往将自己的 Ring3 级的 Anti-Bootkit 功能封装为服务。

d.注册表相关启动项和系统配置文件

用户登陆后，由 explorer 进程读取注册表相关启动项和系统配置文件指示的路径执行例程。

四种传统的启动时机分别对应图 1 中的数字 1、2、3、4 所指示的函数调用点，可见传统的启动方式都是在内核已经启动之后的某个阶段被触发的，相比于在 BIOS 自检、MBR、ntldr 等预启动阶段已经感染劫控了内核镜像的 Bootkit 来说，已经失去了守方优势。Bootkit 很容易通过更改内核入口点、IAT/EAT hook、inline hook、SSDT hook、DKOM 等组合的复杂方式[4,5]，擦除感染痕迹、安植 Ring0 后门、绕过或者直接 disable Anti-Bootkit 系统而存活下来并伺机窃密。

为此，本文提出了一种基于 PBD(预启动磁盘驱动)的新的优先启动方案。该方案劫控预启动时的内核级别可执行模块，可以做到最早获取内核级启动权，提供完备且不受限的 Anti-Bootkit 执行环境，从而为防范 Bootkit 提供了新的思路和技术参考。

3 PBD优先启动面临的问题以及解决方案

通过反汇编二进制镜像和动态跟踪调试分析 Windows 预启动过程执行流程，本文分析了选择预启动驱动作为启动方式的原因和可行性，以及实现 PBD 优先启动方案面临的问题和解决方案。

3.1 为何选择预启动 Osloader 阶段以及 PBD 方式

由于 Windows 预启动阶段的初期---BIOS、MBR、BootSector、Su(ntldr 的前半部分)的启动执行环境是处于 16-bit 实模式下，实用的底层功能模块(磁盘操作驱动等)还未能载入内存,因此这些启动位置难以作为劫控对象；

而预启动阶段的后期到内核启动之前，第一次载入并且运行在 32bit 保护模式执行环境里的启动模块就是 Osloader,更重要的是 Osloader 会先于内核以及所有基于内核引导的启动点之前加载预启动驱动。

Windows 通过将所有的驱动镜像分配在 GDT 索引为 8 的 DPL=0 的段描述符所指示的逻辑段中，使得驱动拥有全部的操作系统底层权限和编程资源，预示着可以作为劫控对象为 Anti-Bootkit 提供绝佳的布控、检查、还原的内核执行环境。

3.2 侦测预启动阶段 PBD 加载并执行的条件与时机

以 windows xp sp2 系统的启动流程为例，分析 Osloader 的执行流程得出的执行步骤用伪代码描述如下：

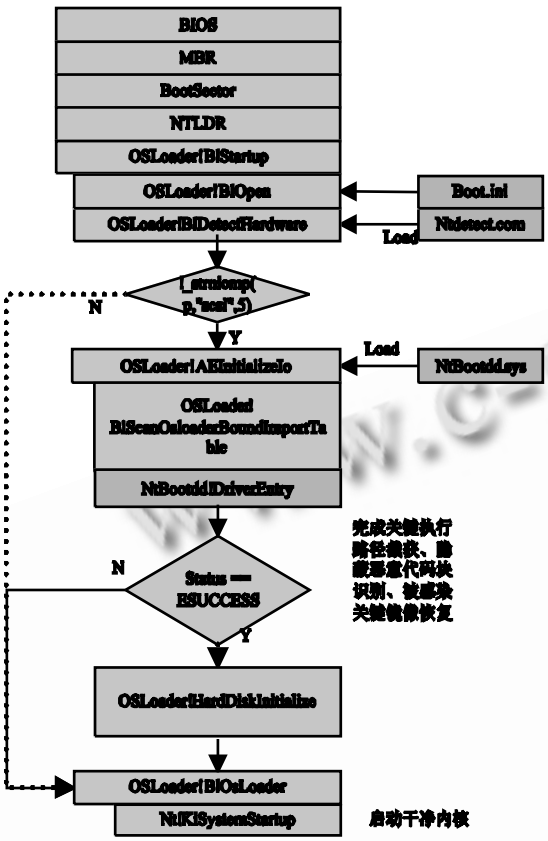

图 2　PBD 劫控原理

```
NtProcessStartup()
/*NtProcessStartup 是 Osloader 的入口点函数，在初始化内置内存/文件管理系统后,把控制权交给 BlStartup 函数　完成主体任务。*/
{
    BlMemoryInitialize();//初始化内存
    BlIoInitialize();//内置文件系统 IO 初始化
BlStartup(BootPartitionName);
                //*从可启动的分区启动内核*/
}
BlStartup(IN PCHAR PartitionName)
{          //预启动阶段的设备检测和加载内核
    Status=BlOpen( DriveId,"\\Boot.ini",Arc
OpenReadOnly,&BootFileId );
    /*BlStartup 调用内置文件系统的打开函数读入 Boot.ini 到内存缓存*/
    BlDetectHardware(DriveId,LoadOptions);
    /*此处 Windows 会在预启动阶段第一次加载并执行可执行文件，但是不适合作为劫控点*/
    if(!_strnicmp(pItem,"scsi(",5))
/*在这里 windows 启动全局流程中会第一次加载预启动驱动 Ntbootdd.sys 用于 scsi 型号磁盘 IO 操作，也是本文选择的劫控时机。开启条件是当前启动选项的前五个字节是 scsi*/
    {AEInitializeIo(DriveId); };
                /*加载并且初始化 SCSI 驱动*/
    BlOsloader(9,Argv,NULL);
/*初始化内核加载所需环境，加载并执行内核*/
};
AEInitializeIo(IN ULONG DriveId)
{/*从启动分区加载 SCSI 启动驱动 NtBootdd.sys，并初始化*/
    BlLoadImage(DriveId,MemoryFirmwarePermanent, "\\NTBOOTDD.SYS",TARGET_IMAGE,&ImageBase);
                /*加载 PBD 磁盘镜像到内存*/
    BlScanOsloaderBoundImportTable(
                DriverDataTableEntry);
/*扫描 NtBootdd 文件头中的导入表初始化 Osloader 的导入表使之用于Osloader 的磁盘读写*/
Entry = (PDRIVER_ENTRY)
    DriverDataTableEntry->EntryPoint;
                /*获取 PBD 的入口点函数*/
Status=(*Entry)(NULL,NULL);
            /*调用 PBD 入口函数，获取执行机会*/
if (Status == ESUCCESS)
{
HardDiskInitialize();
/*如果 PBD 初始化成功那么初始化硬盘*/
}
```

```
return(Status);                    //返回操作状态
}
```

经过 Windows 启动流程的全局分析，我们发现 NtBootdd.sys 是 Windows 内核加载前的预启动阶段最早加载并且获取到 Ring0 执行权限的驱动文件，可以作为 Anti-Bootkit 优先启动的理想 Ring0 模块载体。

3.3 获取最小内存/磁盘操作功能集

由于我们劫控的PBD获得执行权限时内核并没有加载到内存，因此无法使用其提供的原始服务，只有当前供 Osloader 使用的内置文件/内存管理操作的函数集可以调用。找到这些函数集主要函数的入口地址成为实现 Anti-Bootkit 的技术焦点。

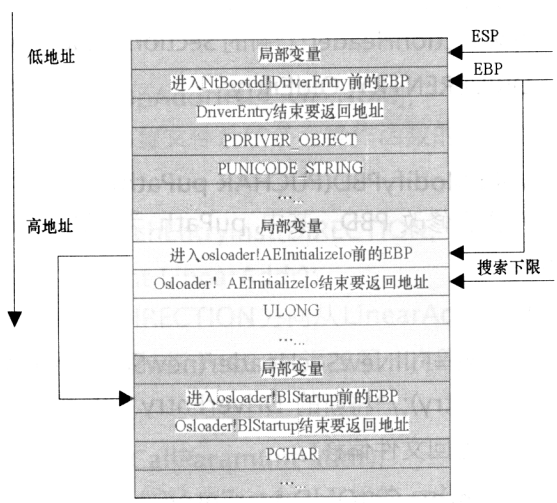

图 3　劫控点函数调用堆栈

经过反汇编 ntldr 分析发现，虽然内置文件/内存管理操作的函数集随着 Osloader 被加载到内存并一直提供低级 IO 操作和内存管理服务直到内核被加载并启动，但是由于 Osloader 并没有导出相应的操作函数供其他模块调用，因此不能直接在 DriverEntry 中使用。

但是同时发现内置文件/内存管理操作的函数集在 Osloader PE 文件的代码节中顺次存放，且入口指令统一，可以作为理想的二进制代码搜索的特征；只要定位到某一个函数的实际线性地址很容易根据入口指令特征寻找到其他函数的实际地址。例如内置文件操作的函数集存放次序依次为 BlIoInitilaze，BlGetFsInfo，BlClose，_BlOpen，BlOpen，BlRead，BlSeek，BlWrite，BlGetFileInformation，BlRename，BlReadAtOffset 等，入口指令都是 mov edi，edi，而该指令在代码节出现的位置恰恰都是内置文件操作函数的入口。

经分析 BlClose 函数的实际线性地址最容易根据劫控点 DriverEntry 所在的函数调用堆栈信息得到；反汇编还原出的调用堆栈如图 3 所示。因为在 DriverEntry 的二级调用者函数 BlStartup 调用点位置(也就是 AEInitializeIO 在 BlStartup 函数中的返回地址所指指令的上一条)向上第一次出现的内置文件/内存管理操作函数就是 BlClose。搜索特征可以定义为

```
push    [ebpBl- 0x10];
call BlClose;
```

这里的 ebpBl 指向 BlStartup 栈帧，搜索起始位置是 AEInitializeIO 栈帧中存储的返回地址。

由于在函数调用堆栈栈帧中 EBP 总是指向调用者所在栈帧 EBP 域，当前 EBP 寄存器(指向 DriverEntry 栈帧) 所指的 DWORD 值就是 ebpAE，再取 ebpAE 所指的 DWORD 值就是 ebpBl 了。至于待搜索代码块的开始位置，根据栈帧的内部结构定义，取 ebpAE+izeof(DWORD)作为地址，所指向的 DWORD 值就是开始位置。

3.4 控并回归 windows 正常的启动流程

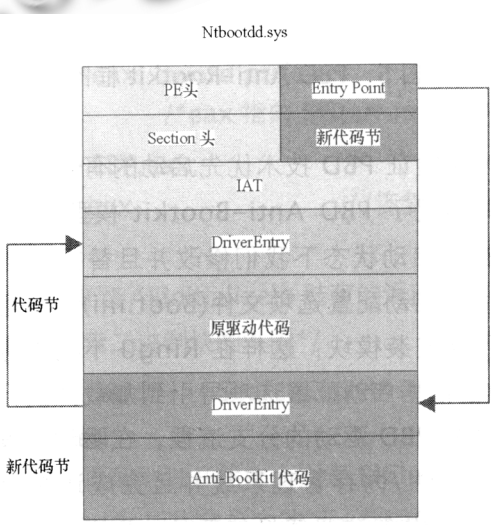

图 4　修改 PBD 磁盘镜像

为了在 PBD 加载运行时获取执行机会，在完成主体任务后回归到 windows 正常的启动流程，我们必须对 PBD 镜像做适当修改。如图 4 所示，绿色部分是我们修改和添加之处。

由于 BlScanOsloaderBoundImportTable 会扫描 NtBootdd 文件头中的导入表(IAT)初始化 Osloader 的导入表使之用于 Osloader 的磁盘读写，因此必须保持原文件中的 IAT 部分和代码节不变。

为此，我们在原来PBD磁盘文件的最后追加了新的代码节存储所有主体任务的代码。同时修改 Section 头维护所有节的数据结构，修改 PE 头的 EntryPoint 域为新代码节中 DriverEntry 的文件偏移，这样就可以作为 PBD 的入口点函数被调用了；最后在新的 DriverEntry 执行主体任务完毕后显示调用了原来的 DriverEntry，这样便可以归还给正常的 windows 启动流程。此外，我们在主体任务完成后利用内置的文件系统操作来还原 Boot.ini 的启动项，因为后续的内核启动过程有可能继续使用其中的启动项信息。

4　PBD Anti-Bookit的设计实现与验证

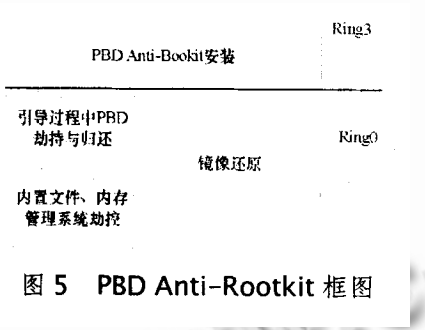

图 5　PBD Anti-Rootkit 框图

为了验证 PBD 技术优先启动的有效性，我们设计并实现了 PBD Anti-Bootkit 模型如图 5 所示。在非启动状态下我们修改并且替换了部分引导相关的启动配置选项文件(Boot.ini)和磁盘驱动文件作为安装模块；这样在 Ring0 下的引导过程中，会根据启动配置选项导引到加载并执行最早可启动的 PBD 驱动的分支流程，在驱动中具体劫控内置文件/内存管理系统并且完成还原内核、ntldr、MBR 等磁盘镜像的任务，之后导引回正常的启动执行流程。

4.1 实现

(1) 在非启动状态修改 Boot.ini 的启动条目选项为 SCSI 磁盘控制器型号;利用 3.4 节中阐述的方法修改 Ntbootdd.sys 磁盘文件。

我们设计的劫控并回归 windows 正常启动流程的伪代码如下。

```
IMAGE_NT_HEADERS
NtHeader=GetNtHeader(puPath);
        \*根据 PE 文件路径获取 NTHeader *\
IMAGE_SECTION_HEADER
  SectionHeader=GetSecHeader(puPath);
        \* 根据 PE 文件路径获取 Section Header *\
  IMAGE_SECTION_HEADER
  newSectionHeader;\*新的 Section Header*\
  DRIVERENTRY OldEntry;          \\入口点函数

  void ModifyPBD(PUCHAR puPath)
  /*负责修改 PBD，其中 puPath 为 PBD 的磁盘路径*/
  {
  OffSet=FillNewSecHeader(newSectionHeader,DriverEntry);/*将新的 DriverEntry 信息填充新增加的节，返回文件偏移*/
  OldEntry=GetOldDriverEntry(
  NtHeader.EntryPoint);/*得到原来镜像的入口函数*/
  NtHeader.EntryPoint=OffSet;
  /* 修改镜像的入口函数为我们的 DriverEntry*/
  AdjustSecHeader(SectionHeader,newSectionHeader);/*调整 Section 头*/
  WriteFile(puPath,newSectionHeader,NtHeader,
        SectionHeader);/*写入磁盘文件*/
  };
  NTSTATUS DriverEntry(
        IN PDRIVER_OBJECT DriverObject,
```

```
    IN PUNICODE_STRING RegistryPath )
{
        /*这里的代码完成主体任务*/
    OldEntry (DriverObject, RegistryPath);
/*调用了原来的 DriverEntry,归还给正常的启动流程*/
}
```

(2) 在 NtBootdd!DriverEntry 中,利用 3.3 节中提到的方法劫控预启动阶段特有的内置文件/内存管理系统,作为在可调用的 API platform,完成关键磁盘镜像的读写等主体任务,最后还原 Boot.ini 中所修改的选项字。

我们设计的通过搜素内置文件系统所有操作的函数地址来获取最小内存/磁盘操作功能集的伪代码如下。

```
int FileOpAddr[FILE_OP_NUM];
/*保存内置文件系统各个操作函数的实际线性地址*/
int SearchEntryInst(DIRECTION, char * Feature,    int LinearAddr);
/*按照DIRECTION方向从LinearAddr线性地址开始搜索代码特征 Feature,返回第一次匹配的线性地址*/
int FindCallParam(int addr);
/*返回从线性地址 addr 开始的 call 指令的操作数,即得到跳转到的地址*/
void SearchFileOpAddr(int iIndex, int nNum, int InitAddr)
/*以某    /*内置文件操作函数的地址为搜索的起始地址 InitAddr,顺次搜索其他的函数地址, nNum 指示操作函数集的大小, iIndex 指示待搜索函数在函数集里的次序*/
{
int LinearAddr=InitAddr;
for(int i=iIndex;i<nNum;i++)
{                    //搜索操作集合的后半部分
LinearAddr=SearchEntryInst(UP,"movedi,edi",LinearAddr);
/*搜索特征指令,每个内置文件操作函数的入口特征指令都是 Mov edi,edi*/
FileOpAddr[i]=LinearAddr;\*记录搜到函数的入口地址*\
};
for(i=iIndex-1;i--;i>=0)
{                    //搜索操作集合的前半部分
LineAddr=SearchEntryInst(DOWN,"mov edi,edi",LinearAddr);
FileOpAddr[i]=LinAddr;
}
}
void FillFileOp()
{/*得到所有的内置文件系统的操作函数的地址*/
InitFileOpAddr();//初始化 FileOpAddr 地址数组
int EBP;int iRet;//EBP 寄存器和返回地址变量
__asm
{
push eax;                    //保存 EAX
    mov eax ,dword ptr[ebp];
            /*eax 指向 AEInitializeIO 栈帧*/
    mov iRet, dword ptr[eax+4];
    /* 得到 AEInitializeIo 结束的返回地址,这个地址就是搜索二进制块的下限*/
    mov eax, dword ptr[eax];
            /*eax 指向 BlStartup 栈帧*/
    mov EBP,eax;
    pop eax                    //恢复 EAX
    };
    /* 得到 AEInitializeIo 结束的返回地址,这个地址就是搜索二进制块的下限*/
        int addr= SearchEntryInst(UP,"push [EBP-16]",iRet);
        /*从返回地址开始向上搜索 BlClose 函数调用点的特征,addr 所指向指令的下一条指令就是 call BlClose*/
```

```
addr = FindCallParam(addr);
/*从 Call BlClose 指令中提取出 BlClose 的函数地址*/
SearchFileOpAddr(2,11,addr);
/*从 BlClose 的地址开始顺次搜索其他的内置文件操作函数地址*/
};
```

4.2 验证

我们选取预启动关键引导位置 MBR、Boot Sector、Ntldr 以及 ntoskrnle 作为功能测试对象，结果表明可以正确检查磁盘镜像的完整性。

我们选取卡巴斯基、macfee 等主流杀毒软件为优先启动的对比对象，如表 1 所示，第一行是按照启动次序排列的启动点，第一列是对比对象，Y 代表对比对象选择的启动点。结果显示 BPD Anti-Bootkit 可以在它们启动之前优先启动。

经过实际的试验测试，BPD Anti-Bootkit 可以以预想的启动方式加载并且执行。

表 1　优先启动对比结果

启动 / 对比	PBD	启动驱动	NativeApp	服务	注册表、配置
卡巴			Y	Y	Y
Macfee				Y	Y
PBD	Y	Y	Y	Y	Y

5　总结与展望

我们通过反汇编并跟踪调试 Windows 引导执行流程，反利用 Bootkit 入侵思想，提出了新的利用 PBD (pre-bootable driver)技术来做到优先启动的可行方案设计。该方案相比于传统启动方式可以使 anti-bootkit 优先启动，为其实施布控提供了可信的内核执行环境，从而为防范 Bootkit 提供了新的思路和技术参考。作为下一步的工作，本文提出的优先启动方案还可以在普适性和功能拓展上进一步完善，通过综合防范、检测、恢复、自我保护等诸多功能和方式，在对抗中占据相对优势。

参考文献

1 Hoglund G., Butler J. The Definition of a Rootkit.1 [2006-1-2].http://www.Rootkit.com

2 Kaspersky Lab. Bootkit: The challenge of 2008. [2008-3-10].http://www.kaspersky.com

3 Russinovich M, Solomon D. Windows Internals. 4th ed, Redmond, Wash: Microsoft Press, December 8, 2004.251－288.

4 Ries C. Inside Windows Rootkits.[2006-3-10]. http://www.vigilant-minds.com/files/inside_windows_Rootkits.pdf

5 Hoglund G,Butler J.Rootkits:Suverting the Windows Kernel.Addison-Wesley Professional. July 22, 2005. 108－321.