Windows 内核级 ROOTKIT 技术的分析与对抗

nEINEI/2007-08-03

摘要：反病毒类产品最终要检测的对象是“数据”，当我们要检测的对象通过一定的手段隐藏起自身逃过我们的检测时，安全隐患就已然产生，更为重要的是在技术的对抗当中我们处于了劣势状态。Rootkit 技术正是这种用于隐藏数据自身，对抗检测技术的一种手段。

一 ROOTKIT 简介

Rootkit 源自于 Unix 操作系统，是指入侵者在已入侵成功的机器系统中设置后门并隐藏起踪迹的工具。Windows Rootkit 出现较 Unix 和 Linux 晚。公开的资料较少，难于掌握。近年，随着对 Windows 系统研究的深入，应用 Rootkit 技术的病毒，木马，流氓软件日益增多。

二 ROOTKIT 分类

按 Windows 下程序运行模式的不同，Windows Rootkit 可分为两类，1 内核模式 Rootkit, 2 用户模式 Rootkit。以下所谈到的 Rootkit 技术均指 Windows 下的 Rootkit。

（1）用户模式下的 Rootkit，以 exe,dll 文件为主。Windows NT 系统中，每一个运行的进程拥有独立的虚拟的 4GB 的地址空间，低 2GB 为私有地址空间。用户模式下的 Rootkit 一般是将自身的 dll 文件注入到系统中每一个运行的进程低 2GB 地址空间中，同时挂钩（hook）被注入进程的创建进程函数，这样系统中运行中的进程的相应函数均被挂钩。该操作完成后，Rootkit 截获被挂钩进程的 API 调用，以此来达到保护自身 exe 文件，隐藏进程目的。常见的技术有：SetWindowsHookEx，注册服务，远程线程注入，hook IAT,BHO 插入,SPI 拦截等。因 Ring3（用户模式）下该 Rootkit 技术已经泛化，本文将不做介绍。

（2）内核模式下的 Rootkit,以 sys 文件为主，同时会配合 exe dll 一起使用，用于 Ring0 与 Ring3 之间的通信，完成特定功能。内核级 Rootkit 拥有系统最高权限，所以内核级 Rootkit 可谓依附于内核之上，操纵于内核之中，破坏于用户之下。Rootkit 在内核中主要完成 hook 内核模式函数，修改内核数据结构等手段来达到隐藏自身的目的。因 Windows NT 系统中内核空间（相当于每个进程中高 2GB 地址空间）对整个系统是共享的，所以内核 Rootkit 实质是控制了整个系统的信息，当然这里指它所感兴趣的那部分。

Rootkit 隐藏方式包括：隐藏进程，隐藏内核模块，隐藏服务，隐藏注册表，隐藏文件，隐藏端口等。

三 ROOTKIT 的技术手段

目前 Rootkit 的所使用的技术手段主要有四种，1 代码注入（以用户模式为主），2 通过修改系统内核的数据结构完成隐藏某些对象，3 修改内核执行路径，4 过滤驱动信息，下面就这几方面做简要说明。

(1) 内核数据修改： 操作系统是根据内核数据来管理内核对象的，Rootkit 利用修改内核数据的方法来隐藏指定的进程、模块或驱动程序等。典型的有 fu_rootkit,它通过摘除内核中的 PsActiveProcessList 链上的进程对象来隐藏进程，摘除 PsLoadModeList 链上的驱动对象来隐藏驱动设备。

(2) 修改内核执行路径：一般有三种方式,（a）API 函数挂钩，(b)修改系统服务表，（c）修改中断服务表。

（a）：例如要隐藏文件可分两个方面对 API 进行挂钩。

用户模式下：Kernel32.dll 的导出函数 FindFileFirst 和 Ntdll.dl 的 NtQueryDirectoryFile。

内核模式下：Ntoskrnl.exe 的导出函数 NtQueryDirectoryFile。

（b）：因大多数用户模式的 API 函数调用都是通过系统服务转到内核执行，所以 Rootkit 只需替换系统服务表中的相应函数指针就可达到隐藏信息的目的。例如隐藏文件，只需将系统服务表中 ZwDeviceIoControlFile()函数替换为 MyDeviceIoControlFile()并对所得到的数据进行过滤即可。

（c）：该方式以截获系统中断调用为主，例如在 Windows 2000 系统下，修改 IDT 表 2e 项的值，就可截获系统服务中断的调用。

(3) 过滤驱动信息：Rootkit 在系统驱动程序（如 TDI 驱动）的上层添加过滤驱动，这样系统发往底层驱动的信息就会被 Rootkit 截获，并对信息进行过滤，该方法可用于隐藏端口。

四 关于 ROOTKIT 的检测手段

目前检测 Rootkit 的技术手段较少，主要有：1 对比扫描法，2 内存扫描法，3 系统模块分析法，4 统计对比法。

(1) 对比扫描法：内核级 Rootit 以欺骗用户模式下检测技术为主，按此想法，只要检测程序以更加底层的方式获得准确的系统信息，就可跳出 Rootkit 的监控。所以对比扫描法以用户态和内核态两次扫描的结果进行对比分析，以此来获得隐藏信息。

(2) 内存扫描法：该方法主要针对修改执行路径的 Rootkit，例如，系统服务表中 NtDeviceIoControl() 地址不在 Ntoskrnl.exe 模块的地址空间，则表示服务表中该项几经被修改。

(3) 系统模块分析法：该方法主要针对直接修改内核对象的 Rooltkit，因不管 Rootkit 如何隐藏最终都要在系统内存中，所以可通过扫描 0x80000000 ~ 0XFFFFFFFF 地址空间数据来获得正在系统运行的进程和设备驱动信息。

(4) 统计对比法：该方法针对修改执行路径的 Rootkit，基于内核函数被 Rootkit 修改前后执行指令条数之差来判断该内核函数是否被修改。但 Windows 平台非常复杂，该方法并不容易实现。

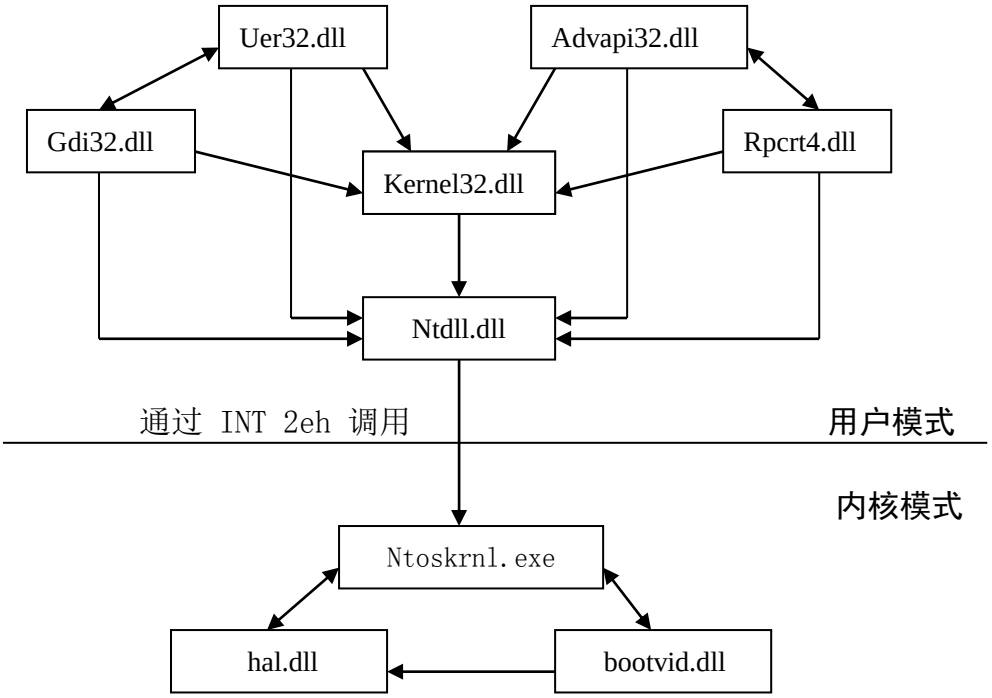
五 用户模式与内核模式的转换

Windows NT/2000 架构下都有包含一个名称为 kernel32.dll 的系统模块，但这却并非实际的操作系统的内核，它仅是 Win32 子系统的一个基本组件，我们平时所用到的绝大部分 API 函数都是由该模块导出的。这容易让我们联想到，既然该模块不是真正的内核，那么我们平时所用到的操作内核数据的 API 是如何生效的呢?

事实上，微软在这里保留了一个调用接口。我们平时在 Windows NT/2000 平台下的开发被简化为与 WIN32 API 交互的工作，实际上存在一个更为底层的调用接口：Native API，这并不需要我们写驱动程序才能访问它，普通的 WIN32 程序就可以调用该接口，微软并没有对该调用做技术上的限制，仅仅是不支持此种应用程序开发模式而已。

Native API 在用户模式下是用通过 ntdll.dll 来体现的。ntdll.dll 的功能就是为运行于用户模式下的程序提供一个内核函数的子集。真正的 Native API 接口是在 ntoskrnl.exe 中。ntdll.dll 通过 INT 2eh 中断将 CPU 的特权级别从用户模式切换到内核模式,以此来完成用户态的向内核态请求。用户模式与内核模式的关系如下图（一）所示。

当然实际的模块调用要比这复杂的多，我们只为了说明用户模式与内核模式关系，所以简化了该模型。其实进入内核的调用方式也是很简单的。以 kernel32.dll 导出的函数 CreateFile 为例，该函数最终会调用 ntdll.dll 的导出函数 NtCreateFile，反汇编该函数会有如下代码，如图（二）所示。



（图一）

```
00783DA8  $  B8 20000000    mov eax,20
00783DAD  .  8D5424 04      lea edx,dword ptr ss:[esp+4]
00783DB1  .  CD 2E          int 2E          ---> 由此处进入内核
00783DB3  .  C2 2C00        retn 2C
00783DB6  $  57             push edi
00783DB7  .  8B7C24 0C      mov edi,dword ptr ss:[esp+C]
00783DBB  .  8B5424 08      mov edx,dword ptr ss:[esp+8]
00783DBF  .  C702 00000000  mov dword ptr ds:[edx],0
00783DC5  .  897A 04        mov dword ptr ds:[edx+4],edi
00783DC8  .  0BFF           or edi,edi
00783DCA  .  74 11          je short ntdll_1.00783DDD
00783DCC  .  83C9 FF        or ecx,FFFFFFFF
00783DCF  .  33C0           xor eax,eax
00783DD1  .  F2:AE          repne scas byte ptr es:[edi]
00783DD3  .  F7D1           not ecx
00783DD5  .  66:894A 02     mov word ptr ds:[edx+2],cx
00783DD9  .  49             dec ecx
```

（图二）

图二为 WIN2K 系统下调用，INT 2E 指令在 WinXP 系统后被替换为 SYSENTER 指令。但功能仍然为激活系统服务调度程序，并由此转换到内核。该调用中 EAX 放的是一个分派（也可称之为标识）ID,这样利用 EAX 的值作为一个索引来查询一个特定的表，这个表就是系统服务表（System Service Table,SST）该值标识了请求系统服务的标识号。EDX 放的是指向堆栈中的调用参数。

系统服务调度程序的执行过程是先对该调用程序的参数数目进行验证，然后将它们从

用户堆栈复制到内核堆栈，最后调用在 EAX 中的标识 ID 所指出的索引地址处的函数。这样就完成了一次由用户态到内核态的转换。

六　Rootkit 技术分析与检测手段

下面以隐藏进程为例，为来说明检测 Rootkit 的技术手段。

隐藏进程的方法有很多，但在 Ring0 下一般是修改 EPROCESS 结构，这就是我们前面提到的 Rootkit 所使用的技术手段之一，内核数据修改。Ring3 下可以通过两种方式枚举系统活动进程。1　利用 ToolHelp32 的 Process32First 和 Process32Next 函数。2 利用 PSAPI.dll 导出的 EnumProcesses 函数。但这两种方式最终都是通过调用 NTDLL.dll 中的 NtQuerySystemInfomation 函数来实现的。在 Win2000/XP/2003 中的 Taskmgr.exe 也是通过调用 ZwQuerySystemInfomation 函数来实现枚举进程的，其中 NT 开头的函数与 ZW 开头的函数都指向相同的代码，但反汇编 Ntoskrnl.exe 就会发现 Zw 集合的函数指向 INT 2eh 处,而 NT 集合函数则直接指向了代码,也就是说 Zw 集合函数经历了从用户模式进入内核模式的过程，而 Nt 集合函数则是模式转换后被执行。两个函数反汇编后如图下图所示：

```
PAGE:00492C62 ; NTSTATUS __stdcall NtQuerySystemInformation(SYSTEM_INFORMATION_CLASS SystemInformationClass,P
PAGE:00492C62                 public NtQuerySystemInformation
PAGE:00492C62 NtQuerySystemInformation proc near

PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:00490000 SIZE 00000027 BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C0921 SIZE 00000093 BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C88F6 SIZE 00000012 BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C891A SIZE 000000BF BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C89DE SIZE 000001D1 BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C8BB4 SIZE 0000017A BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C8D30 SIZE 0000012E BYTES
PAGE:00492C62 ; FUNCTION CHUNK AT PAGE:004C8E68 SIZE 00000051 BYTES
PAGE:00492C62
PAGE:00492C62                 push    ebp
PAGE:00492C63                 mov     ebp, esp
PAGE:00492C65                 push    0FFFFFFFFh
PAGE:00492C67                 push    offset unk_402A48
PAGE:00492C6C                 push    offset _except_handler3
PAGE:00492C71                 mov     eax, large fs:0
```

（图三）

```
.text:0040116A ; NTSTATUS __stdcall ZwQuerySystemInformation(SYSTEM_INFORMATION_CLASS SystemInformationClass,PVOID Syst
.text:0040116A                 public ZwQuerySystemInformation
.text:0040116A ZwQuerySystemInformation proc near      ; CODE XREF: RtlCreateHeap+11F↓p
.text:0040116A                                         ; sub_561C7E+13↓p
.text:0040116A
.text:0040116A SystemInformationClass= dword ptr  4
.text:0040116A SystemInformation= dword ptr  8
.text:0040116A SystemInformationLength= dword ptr  0Ch
.text:0040116A ReturnLength    = dword ptr  10h
.text:0040116A
.text:0040116A                 mov     eax, 97h
.text:0040116F                 lea     edx, [esp+SystemInformationClass]
.text:00401173                 int     2Eh             ; DOS 2+ internal - EXECUTE COMMAND
.text:00401173                                         ; DS:SI -> counted CR-terminated command string
.text:00401175                 retn    10h
.text:00401175 ZwQuerySystemInformation endp
```

（图四）

虽然 NT 集合函数与 Zw 集合函数功能一致，但在 Ntoskrnl.exe 中 Nt*/ Zw *并不一定成对导出，例如 NTCurrentTeb 函数就没有对应的 ZW 函数。

当 ZwQuerySystemInformation 所获得的内核对象被修改后，那么 Ring3 下所得到的进程信息也就不再具有了绝对的真实性。Ring0 下隐藏进程的一般步骤是:1 打开物理内存，2 找到活动进程链表，3 脱链（从双向链表中摘除自己）。

因双向链表在 EPROCESS 结构中，所以要隐藏进程就先要得到当前进程的 EPROCESS 结构。这里认为要隐藏的进程 PID 为 hide_id，因 PID 在 EPROCESS 结构中的偏移与操作系统版本有关，此处是 Windows Server 2003 系统，PID 相对偏移量是 0x84,则查找 EPROCESS 地址的函数可简单表示为如下代码：

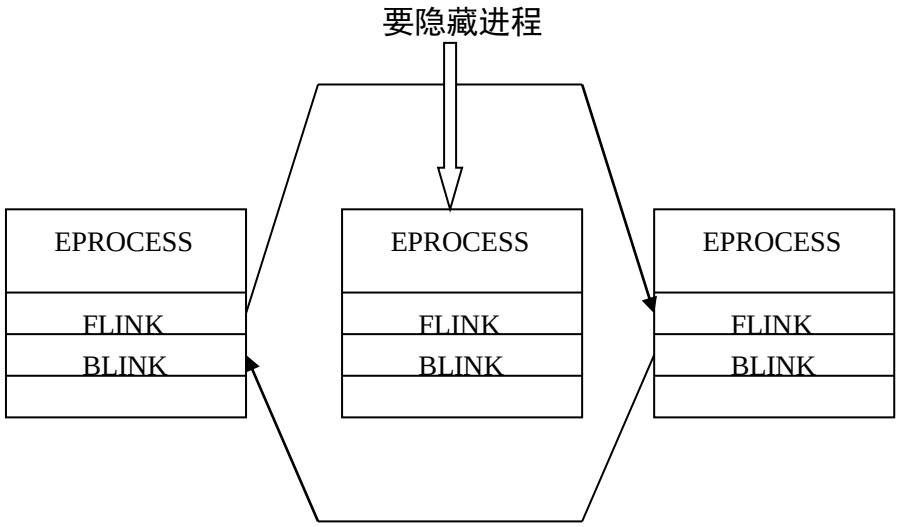
```
#define PID_OFFSET  0x84
#define FLINK       0X88
DWORD find_process_eproc(int hide_pid)
{
   int          i        = 0;
   int          first_pid, curr_pid =0;
   DWORD        eproc    = 0;
   PLIST_ENTRY  pactive_list;
   eproc = (DWORD)PsGetCurrentProcess();
   first_pid = curr_pid = *((int *)(eproc+PID_OFFSET));
   for(;;)
   {
        if (hide_pid == curr_pid)
           return eproc; // 找到了隐藏进程的 PID，返回该进程 EPROCESS 地址
        else if(i>0&&first_pid == curr_pid)
          return 0x00000000; // 遍历了整个链表也没找到，返回 0
        else
         {
          pactive_list = (PLIST_ENTRY)(eproc+FLINK);//继续找下一个链表
          eproc = (DWORD)pactive_list->Flink;  //定位 Flink 偏移处
          eproc -= FLINK; // 定位到 EPROCESS 数据块开头处
          curr_pid = *((int *)(eproc+PID_OFFSET));
           i++;
         }
    }
}
```

当获得到要隐藏进程的 EPROCESS 地址后，后面的工作将非常简单，仅需要把隐藏进程的 EPROCESS 结构从链表中摘除即可，限于篇幅此处不再给出代码，脱链具体过程如图（五）所示，此时 Ring3 下所获得的进程信息将不包含 hide_pid 进程。

对于隐藏进程的检测可以使用对比扫描法，在 Ring3 下枚举进程获得一个进程列表，我们称作 PTable1，同时我们利用安全的手段来获得进程列表 PTable2，PTable2 为真正的当前活动进程列表，这样通过二者之间的对比就可以发现隐藏的进程了。

其中 PTable2 的获取方法主要有，1 基于线程调度链表来获取进程列表、2 Hook SwapContext 函数。3 读取 Csrss.exe 的句柄表。方法 1 可以检测出 HOOK NTQuerySystemInfomation 和从 ActiveProcessLinks 摘除自身的 Rootkit。方法 2 可以检测出绝大部分隐藏进程的 Rootkit,但该方法稳定性较差。方法 3 是基于 Csrss.exe 句柄表中也包含了系统所有的进程信息，所以可以获得真实的当前活动进程信息。类似的隐藏服务，

隐藏注册表，隐藏文件等 Rootkit 技术都可通过我们的深入分析来找到其检测方法。



图五

结束语 本文仅是从归纳的角度讲述了 Rootkit 的分析与检测方法，因为在与 Rootkit 的对抗当中，是没有一成不变的终极解决办法。只有不断的归纳总结才能设计出安全稳定的 Anti Rootkit 工具。

参考文献：

【1】 www.rootkit.com/vault/fuzen_op/HideProcessHookMDL.zip
【2】 www.xfocus.net
【3】 《Undocumented Windows NT 》
【4】 sinister《隐藏任意进程，目录/文件，注册表，端口》
【5】 高级 WIN2K ROOTKIT 检测技术