

基于Linux系统调用的内核级 Rootkit技术研究

石晶翔¹, 陈蜀宇², 黄晗辉¹

(1. 重庆大学 计算机学院, 重庆 400030;

2. 重庆大学 软件工程学院, 重庆 400030)

摘 要:系统调用是用户程序和操作系统进行交互的接口。劫持系统调用是内核级Rootkit入侵系统后保留后门常用的一项的技术。研究Linux系统调用机制及系统调用劫持在内核级Rootkit中的应用可以更好地检测和防范内核级Rootkit,使Linux系统更加安全。文中在分析Linux系统调用机制的基础上,研究了内核级Rootkit劫持系统Linux系统调用的5种不同方法的原理及实现,最后针对该类内核级Rootkit给出了3种有效的检测方法。在检测过程中综合利用文中提出的几种检测方法,能提高Linux系统的安全性。

关键词:Linux;系统调用;Rootkit

中图分类号:TP393.08

文献标识码:A

文章编号:1673-629X(2010)04-0175-04

Research on Kernel Level Rootkit Technology Based on Linux System Call

SHI Jing-xiang¹, CHEN Shu-yu², HUANG Han-hui¹

(1. College of Computer Science, Chongqing University, Chongqing 400030, China;

2. College of Software Engineering, Chongqing University, Chongqing 400030, China)

Abstract: System call is the interface between operating system and user's application. System call hijacking is a common technology used by kernel level Rootkit to attack operating system and keep backdoors. Research on the application of Linux system call mechanism and Linux system call hijacking in kernel level Rootkit can help to detect and protect Linux system from Rootkit. This paper analyzes the mechanism of Linux system call, and discusses the principle and implementation of how the kernel level Rootkit to hijacking Linux system call. Finally, three effective methods of detecting kernel level Rootkit are proposed. Based on these methods in the detecting process, it can improve the security of Linux system.

Key words: Linux; system call; Rootkit

0 引 言

Rootkit最早出现于20世纪90年代初,是攻击者在攻击Unix/Linux时用来隐藏自己的踪迹和保留root访问权限的工具。Rootkit的目的不是为了突破系统获得进入系统的权限,而是通过其他手段获得系统权限后对所得到的权限进行保存,同时隐藏攻击活动的痕迹,防止暴露。

正是因为这样,如今Rootkit与木马等恶意软件有

相互结合的趋势,随着影响范围的不断扩大,对网络信息安全可能带来的危害也逐渐增强。

Rootkit分为应用级Rootkit和内核级Rootkit两种类型。应用级Rootkit在操作系统应用层修改系统文件,比较容易预防和检测。内核级Rootkit攻击操作系统内核,与应用级Rootkit相比功能更为强大,更难检测。因此内核级Rootkit以其强大的功能和较为完善的隐蔽性更加受到瞩目。

1 Linux系统调用机制

系统调用是内核提供的用户程序与内核的一种接口,它位于用户态与核心态之间的边界。用户程序通过系统调用向核请求服务,当内核完成服务后将结果

收稿日期:2009-07-28;修回日期:2009-10-11

基金项目:科技部科技型中小企业技术创新基金(05C26215111378)

作者简介:石晶翔(1985-),男,福建龙岩人,硕士研究生,研究方向为嵌入式系统;陈蜀宇,教授,博士生导师,研究方向为嵌入式系统等。

返回给用户程序^[1]。

在 Linux 系统中,系统调用是用异常类型实现的,用户态的程序通过门陷入到系统内核中去,执行一些具有特权的函数。例如,用户程序在调用系统调用 fork() 时,它通过执行相应的软中断指令陷入操作系统内核空间,系统进行中断权限检查后根据中断描述符表查找到系统调用的中断服务例程 system_call。中断服务例程根据 fork() 对应的系统调用号在系统调用表 sys_call_table 中查找到系统调用函数 system_fork() 后执行,将结果返回给用户程序。系统调用完成后,系统通过执行返回指令从内核态返回到用户态,将控制权返回给用户程序。执行 fork() 的执行过程如图 1 所示。

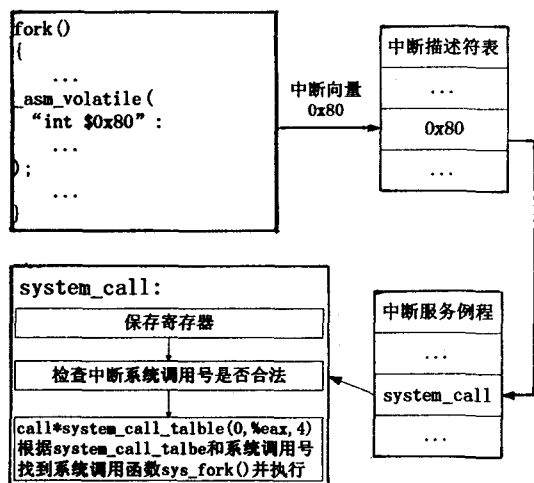


图 1 fork() 系统调用执行过程

目前有以下两种方式实现系统调用:

(1) 执行 int \$0x80 指令进入内核态,执行 iret 指令返回,在 Linux2.4 内核及以前这是从用户态到内核态转换的唯一方式;

(2) 执行 sysenter 指令进入内核态,执行 sysexit 指令返回,Linux2.6 内核开始支持这种指令方式,同时也支持(1)方式。

Linux2.6.24 的内核源代码中,设置系统调用中断服务例程的代码如下:

```
# define SYSCALL_VECTOR 0x80
set_system_gate(SYSCALL_VECTOR, &system_call);
```

系统调用中断服务例程 system_call 实现如下:

```
ENTRY(system_call) /* arch/i386/kernel/entry.S */
pushl %eax
SAVE_ALL /* 保存寄存器 */
GET_THREAD_INFO(%ebp)
testw $(-TIF_SYSCALL_EMU | -TIF_SYSCALL_TRACE |
-TIF_SECCOMP | -TIF_SYSCALL_AUDIT)
TI_flags(%ebp)
```

```
jnz syscall_trace_entry /* 检查是否处在被跟踪状态 */
cmpl $(NR_syscalls), %eax
jae syscall_bad_sys /* 检查系统调用号是否合法 */
syscall_call:
call *sys_call_table(0, %eax, 4) /* 调用系统调用函数 */
movl %eax, EAX(%esp) /* 保存返回值 */
syscall_exit:
.....
```

系统调用号定义为宏,从 0 开始编号,如 exit() 系统调用的宏定义是 #define NR_exit 1。系统调用表 sys_call_table 依次保存着所有系统调用函数的地址,以 _NR_name 为下标,找出系统调用表 sys_call_table 中对应表项的内容,就是其对应的系统调用函数 sys_name 的地址。

```
ENTRY(sys_call_table) /* arch/386/kernel/syscall_table.S */
.long sys_restart_syscall
.long sys_exit
.....
```

2 基于系统调用的内核级 Rootkit 技术

目前的内核级 Rootkit 大多数利用 Linux 系统中的 LKM(可装载内核模块)机制,将其模块挂在系统的模块链上,成为内核代码运行于内核态,享有内核的所有特权^[2]。

Rootkit 通过拦截系统调用,可以实现隐藏文件、进程和网络连接、重定向可执行文件、端口复用等功能^[3,4]。

2.1 修改系统调用函数

Linux 系统中,每个系统调用在内核中都对应着一个系统调用函数,这些函数通常都以 sys_ 开头,并且这些系统调用函数符号都被内核输出。运行在内核空间的 Rootkit 可以获得这些系统调用函数符号地址并修改这些系统调用函数的指令代码,使得在其中可以执行 Rootkit 提供的恶意代码。

2.2 修改系统调用表

这种类型的 Rootkit 修改系统调用表中的某些系统调用函数地址,把一些系统调用函数重定向到包含恶意代码的系统调用函数,当用户在调用该系统调用时便触发替换后的系统调用函数。

通常,为了保持 Rootkit 的隐蔽性,替换的系统调用函数的实现中也会调用替换前的系统调用函数,这样用户就不容易察觉。

使用这种攻击方法的一个典型内核级 Rootkit 实现是 KNARK Rootkit^[5]。KNARK Rootkit 使用直接输出的系统调用表符号来修改系统调用表,但是在 Linux2.6 内核中,sys_call_table 不再被内核输出^[6],所以

KNARK 使用的方法在最新的 Linux2.6 内核上不能使用。

2.3 重定向系统调用表

这种类型的 Rootkit 把系统调用中断服务例程 system_call 中使用的系统调用表 sys_call_table 重定向到内核空间中一个新的系统调用表。新的系统调用表必然包含 Rootkit 的定义的系统调用函数的地址,以及一些未被修改的 sys_call_table 中系统调用函数地址(如图 2 所示)。

Linux 中/dev/kmem 是一个字符设备文件,是计算机主存的映像,通过读取这个设备文件可以得到内存中的数据。Rootkit 通过/dev/kmem 准确定位系统调用表的地址,对其用新的系统调用表进行覆盖就可以重定向系统调用表^[7]。

这类 Rootkit 并不修改原始系统调用函数,原来的系统调用表不受任何影响,因此能够逃避当前检测内核内存空间系统调用表完整性的检测方法。

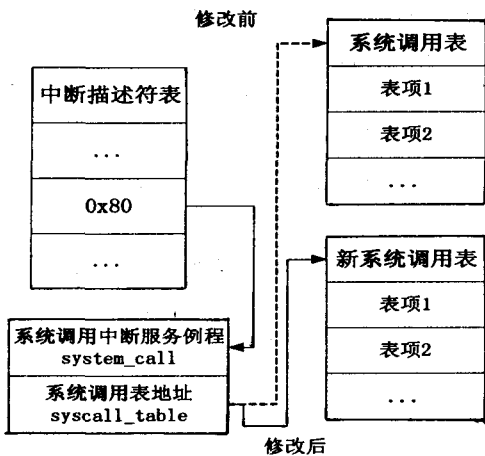


图 2 重定向系统调用表

2.4 修改系统调用入口函数

根据第 2 节的分析,用户执行的系统调用通过中断指令陷入内核态,操作系统接收到中断请求后找到中断服务例程 system_call,system_call 根据系统调用号在系统调用表中找到相应的系统调用函数后执行。因此,system_call 是系统调用的入口,我们可以通过修改 system_call 的指令代码来拦截系统调用。

在执行 call * sys_call_table 前,增加对系统调用号的判断,如果是需要拦截的目标调用号的话,就转向执行 Rootkit 指定的代码;如果不是,则继续执行原先剩余的代码。

修改系统调用入口函数的内核级 Rootkit 既可以修改系统调用表,也不用重定向它,这种攻击方法的典型实现有 Enyelkm。它是 Linux2.6 上 LKM 形式的内核级 Rootkit,其修改 system_call 的关键实现如下:

```
jnz syscall_trace_entry
idt_handler:
    cmpl $(nr_syscalls),%eax
    jae syscall_exit
    cmp $0x25,%eax /* 0x25 对应为 sys_kill */
    je hacked_kill
    .....
hacked_kill:
    call p_hacked_kill
    jmp after_call
    .....
syscall_call:
    call * sys_call_table(,%eax,4)
after_call:
    movl %eax,EAX(%esp)
syscall_exit:
    .....
2.5 修改中断描述符表
中断描述符表 IDT(Interrupt Descriptor Table)是一个有 256 个中断描述符的线性表,每个中断描述符对应一个中断号,长度为 8 字节。
中断描述符位移的高 16 位和低 16 位组成中断服务例程的地址,通过 idtr 寄存器获得中断描述符地址后,可以用自己编写中断服务例程 rootkit_system_call 替换原有的 system_call 函数,拦截 set_uid()系统调用。
关键代码如下:
asm_(
    ".globl rootkit_system_call\n"
    ".align 4,0x90\n"
    "rootkit_system_call:\n"
    .....
    /* 比较是否是要劫持的中断号 */
    "xor %%ebx, %%ebx\n"
    "movl %2, %%ebx\n"
    "cmpl %%eax, %%ebx\n"
    "jne finis\n"
    /* 是,则将参数压栈 */
    .....
    "call * %0\n" /* 调用 rootkit_system_call */
    "addl $8, %%esp\n"
    /* 恢复寄存器 */
    .....
    "finis:\n" /* 否,调用原来的中断服务例程 */
    "jmp * %\n"
    "::\"m\"(rootkit_code),\"m\"(old_systemcall),\"i\"(sys_number)
);
```

3 检测

检测是解决 Rootkit 的一个重要环节。内核级 Rootkit 是在内核级修改了内核结构和插入恶意代码,在内核层欺骗应用程序,它不修改用户应用程序本身,因此基于文件完整性校验的检测工具如 Tripwire 和 Aide 等对内核级 Rootkit 都不起作用。目前,针对攻击系统调用的内核级 Rootkit 主要有以下几种检测方法^[8]。

3.1 提前预防的方法

内核级 Rootkit 通常都是通过 LKM 来加载自身。利用这个特点,可以记录每一个模块的加载,把模块加载的信息记录在 syslog 或其他安全的地方,管理员定期进行比较和查对。同时还可以在加载模块的时候进行安全验证,加载模块是通过 sys_create_module() 系统调用函数来分配模块空间的,通过截获该系统调用函数,设置要求加载任何模块时必须密码验证,否则不能加载。

这种方法缺点是,如果系统中已经加载了 Rootkit 模块,则检测的安全性得不到保障。

3.2 kmem 与 System.map 差异比较法

/boot 目录下有一个 System.map 文件,这个文件在编译系统内核时生成,它包含了系统内核符号地址,这些符号地址就是检测过程中用来参照的正确地址。

/dev/kmem 是一个字符设备文件,它保存了系统中所有主存的一个映像,可以从 kmem 中获得系统当前实际使用的各符号地址值。读取 kmem 设备文件,查找系统调用表和与中断描述符表,可以得到当前系统中各个表项的值。

通过比较当前内存中内核表项的值与 System.map 中的相应表项的值,可以检测被劫持的系统调用。在 Linux2.6 内核之前有很多内核级 Rootkit 都通过/dev/kmem 文件修改内核,如 Suckit。

检测的具体步骤如下^[7]:

(1) 打开/dev/kmem,在/dev/kmem 中查找 sys_call_table 获取系统调用表地址,sidt %0%:(idtr)指令获取中断描述符表在内存中的首地址;

(2) 比较/dev/kmem 中的系统调用中断服务例程的地址与其在 System.map 中保存的地址值,如果相同则转(3),否则恢复中断描述符表;

(3) 比较/dev/kmem 中的系统调用表地址与其在 System.map 中保存的地址值,若相同则转(4),否则恢复系统调用表地址;

(4) 比较/dev/kmem 中的每一个系统调用函数地址与 System.map 中保存的值,如果相同则检测完毕,

没有出现异常;否则恢复系统调用表。

3.3 可执行路径分析(EPA)的方法

可执行路径分析的方法的原理^[8]是:如果修改了内核函数,内核的指令执行路径将改变,在系统调用过程中指令的数目将和原始的内核不同,恶意代码在返回到用户空间时肯定有一些附加行为。这意味着将有许多与未被感染系统不同的代码被执行,那么可以通过测试系统调用的可执行指令的数目变化来判断系统是否被侵入。

其实现过程分两步:

- (1) 跟踪进入内核来记录系统调用的指令数目;
- (2) 测试系统关键系统调用的指令数。

EPA 检测方法很好补充了以上两种方法的不足,在系统被入侵后也能正确检测,不过这类检测工具自身也容易受到攻击和欺骗。

4 结束语

Linux 在处理系统调用请求过程中查找中断描述符表和系统调用表以调用相应的处理函数,通过修改系统调用执行过程中所用的相关数据结构和函数可以实现系统调用劫持。因此,系统调用劫持在内核级 Rootkit 中被大量的应用。

目前,还没有一种通用的算法来检测和防御这类攻击;在将来的检测道路上,更趋向于混合利用几种检测方法,互相补充、互相弥补以期达到提高检测精度的目的。

参考文献:

- [1] 李善平,陈文智.边干边学——Linux 内核指导[M].杭州:浙江大学出版社,2002:133-16.
- [2] Salzman, Burian M, Pomerantz O. The Linux Kernel Module Programming Guide[EB/OL]. 2005. <http://www.tldp.org/guides.html>.
- [3] 陈华亭,吴邦欲.基于 LKM 的 Rootkit 技术[J].计算机工程与科学,2004,26(2):88-90.
- [4] 颜仁仲.实时检测 Rootkit 并自动修复系统的研究与实现[D].北京:中国科学院,2006:43-64.
- [5] Murillo T. Analysis of the KNARK Rootkit[EB/OL]. 2004. <http://www.ossec.net/rootkits/studies/knark.txt>.
- [6] 阮越. Linux 用户行为记录器的一种内核级实现方法[J].计算机技术与发展,2008,18(2):152-155.
- [7] 颜仁仲,钟锡昌,张倪.一种自动检测内核 Rootkit 并恢复系统的方法[J].计算机工程,2006,32(10):77-79.
- [8] 袁源,戴冠中. LKM 后门综述[J].计算机科学,2008,35(7):5-8.