Linux Shell 安全审计机制的扩展*

汪立东, 方滨兴

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)
E-mail: wld@mail.cnnisc.gov.cn; bxfang@mail.cnnisc.gov.cn
http://www.hit.edu.cn

摘要: Unix Shell 生成的命令历史记录是系统审计信息的重要来源,但它未能包含检测入侵所需的足够信息,且容易被用户本人篡改.利用可装入内核模块和系统调用劫持技术实现了对 Linux Shell 安全审计机制的扩展,并给出了用其进行安全监测的例子.

关 键 词: 安全;审计;Linux 可装入内核模块
中图法分类号: TP316 文献标识码: A

随着 Internet 应用的发展,计算机和网络已成为各类犯罪分子的目标.安全监测和入侵检测可以及时发现入侵动机和行为,促使管理人员及时修正系统漏洞,从而减少被入侵的可能性和可能造成的损失[1,2].安全监测和入侵检测的准确性在很大程度上取决于可用的审计信息.Unix Shell 记录的用户历史命令反映了登录用户在系统上的活动,是发现用户异常行为的最直观、最有用的审计数据.但其审计机制存在以下缺陷:①信息不足:仅记录用户命令,未记录时间和用户信息等;②完整性难以保证:Unix 的文件访问控制机制是自主访问控制,命令历史文件对用户本人是可写的,用户可通过对其进行修改而掩盖自己执行的非法操作;③易受欺骗:只反映命令名,不能反映命令的实质,若用户将非法命令用 system()或 exec()调用在 C 程序中执行,shell 审计机制不能捕获真正的非法命令;④时间滞后性:必须等用户退出后才记录其用过的命令,不利于实时监测.

我们曾修改 Linux 上的 bash 来增加其记录的审计信息,但由于 shell 程序种类繁多,这种方法不具通用性,而其完整性仍不能保证.本文提出了一种扩展 Linux Shell 的安全审计机制的方法,利用系统调用劫持和 Linux 的动态可装入内核模块(loadable kernel module)技术来实现,不仅可以记录 shell 命令行,也可以记录所有经 exec 系统调用执行的命令.记录下来的信息可以方便地用于安全监测和入侵检测.

1 LKM 相关技术

1.1 LKM与设备驱动程序

Linux 支持的设备很多,将所有设备驱动程序预先链入内核是不现实的.Linux 2.0 以后的内核提供了一种将设备驱动程序动态链入正在运行的内核的机制,称为 LKM 支持.LKM 使 Linux 可用较小的初始内核,而在需要时再链入设备支持.装入一个 LKM,从功能上等同于传统的编译内核.每个 LKM 最少有两个基本函数: init_module()和 cleanup_module().前者在模块装入内核时执行,后者在模块从内核卸出时执行.将一个模块装入内核包括 4 个任务:(a) 在用户空间准备模块:读出目标文件并解析未定义的符号,运行时由模块装入器 insmod 完成链接过程;(b) 在内核地址空间分配内存;(c) 将模块代码复制到新分配的空间并向内核提供必要信息,以维护此模块;(d) 执行模块初始化例程 init_module().

LKM 装入后即成为内核的一部分.在 Linux 操作系统中,通常用 LKM 扩充内核(特别是硬件驱动程序),但也

* 收稿日期: 2000-04-12; 修改日期: 2000-07-31

作者简介: 汪立东(1970－),男,安徽黄山人,博士生,助教,主要研究领域为网络安全;方滨兴(1960－),黑龙江哈尔滨人,博士,教授,博士生导师,主要研究领域为网络安全,计算机系统结构,并行计算.

可以用它来截获系统调用.利用 LKM 挂接系统调用已被黑客用来作为一种新的入侵手段[3,4],但这种技术也为系统管理员提供了监测系统活动的新方法.

1.2 系统调用劫持

shell 作为 Unix 系统的外壳,是用户交互使用 Unix 的一个接口,负责解释和执行用户的命令和程序.shell 的基本动作是对用户命令进行语法分析,然后 fork()一个子进程并用 exec()来执行此命令.如果我们能劫持 exec 系统调用(exec 是一族系统调用,它们最终都通过 execve()来实现,所以只需劫持 execve()),则可以获得关于此命令的重要信息.

Unix 系统调用是从用户空间到内核空间的转换,如果在用户空间打开文件,在内核空间用 sys_open 表示(参见 sys/syscall.h).每个系统调用有一个系统调用号.内核使用中断 0x80 来管理系统调用,系统调用号和参数被移入一些寄存器(如 eax 保存系统调用号).系统调用号是内核结构 sys_call_table[]的下标,若要劫持一个内核系统调用,需要修改 sys_call_table[]中相应的入口地址.Linux 上劫持系统调用的一般方法如下:

· 在 sys_call_table[]中找到所需系统调用的入口(参见 sys/syscall.h);

· 将旧的 sys_call_table[X]的内容保存在一个函数指针中(X 表欲截获的系统调用号,可以用内核的宏——NR_name 或 SYS_name 来表示名字为 name 的系统调用和调用号);

· 填入系统调用新处理函数的入口地址,即将 sys_call_table[X]设为所需的函数地址.

1.3 内核空间与用户空间的数据传送

内存分为内核空间和用户空间两部分.内核空间被映射到每个进程的地址空间,用户空间对每个进程则是局部的.一般情况下,用户进程不能写入内核空间,从内核空间也不能访问用户内存.比如,在 shell 中,用户命令是 execve()调用的一个参数,设我们劫持了此调用,但直接引用用户命令却会发生问题,因为 LKM 运行在内核空间,不能轻易读用户空间内存.Linux 提供了内核模式的函数(宏)来抽取用户空间内存字节,如 get_user()或 memcpy_fromfs()(在 asm/segment.h 中定义).同样,内核空间有时也需要向用户空间传送数据,可以采用 copy_to_user()系统调用将内核空间的数据复制到用户空间.但如果内核想在用户空间分配内存,则这种数据传送将变得更加复杂,可以参见文献[4].

2 实 现

我们用运行在内核空间的 LKM 来收集 shell 命令的信息,运行在用户空间的安全监测程序则通过与 LKM 的通信来获取这些信息.LKM 与用户空间的进程间的通信主要有两种方法:使用 proc 文件系统和使用设备文件.每个进程在/proc 目录下都有一个子目录,目录名是进程号,目录中的文件包含了进程的有关信息,但由于 LKM 和设备驱动程序之间的先天联系,使用设备文件与用户进程通信更为方便.

本方法的思想是用 mknod 命令创建一个字符设备/dev/cmdlog,指定其主设备号为 MAJOR_NO,用 LKM 来实现此设备的驱动程序,同时在 LKM 中劫持 execve()系统调用.新的 execve()与/dev/cmdlog 设备驱动程序共同维护一个内核 FIFO 队列:新的 execve()每次取得一条 shell 命令的信息,作为一个结点加入 FIFO 队列;而当有用户进程读/dev/cmdlog 设备时,/dev/cmdlog 设备驱动程序中的读操作处理函数从 FIFO 队列中取出一个结点并将其复制到用户空间供用户进程使用;若 FIFO 队列中没有数据可用,则阻塞用户进程.

整个模块包含 3 个主要部分:注册设备、设备驱动程序和系统调用劫持.由于在内核运行,LKM 编程时必须用内核代码,所以给出较完整的源代码及注释.

2.1 模块定义

init_module()和 cleanup_module()是 LKM 必需的两个函数.我们在 init_module()函数中首先调用 module_register_chrdev()注册一个字符设备(即将其设备驱动程序加入内核的字符设备驱动程序表),然后设置系统调用入口表中 execve()的入口地址,以截获对 execve()的调用.设备注册时需定义一个 file_operations 结构的 cmdlog_handler,用来指定对设备文件进行定位、读、写、释放等操作时对应的处理函数.当有进程对此设备文

件实施操作时,自动调用相应的动作函数.cleanup_module()取消注册此设备,并恢复 execve()系统调用原来的入口地址.

```
int init_module()
{
    register_chrdev(MAJOR_NO,“cmdlog”,&cmdlog_handler); /* 设备注册 */
    orig_execve=sys_call_table[SYS_execve]; /* SYS_execve 是 execve 调用号 */
    sys_call_table[SYS_execve]=new_execve; /* 用 new_execve 作为新的处理函数 */
    return(0);
}
int cleanup_module()
{
    sys_call_table[SYS_execve]=orig_execve; /* 恢复原 execve 系统调用 */
    unregister_chrdev(MAJOR_NO, “cmdlog”); /* 取消设备注册 */
    return(0);
}
```

2.2 设备驱动程序

本设备驱动程序仅需定义打开、读和关闭操作的处理函数.其中打开和关闭操作的处理比较简单,仅允许 root 用户在没有冲突时操作(在内核中用!suser()系统调用判断).而读操作处理函数(设其对应的函数名为 cmdlog_release())则要牵涉到数据从内核空间向用户空间的复制,可以用 memcpy_tofs()或 copy_to_user()内核系统调用来实现.当用户进程读/dev/cmdlog 设备时,自动调用 cmdlog_read()处理函数.

```
int cmdlog_read(struct file *f, char *buf, size_t buflen, loff_t *offset)
{
    char tmp[MAXNAME+128];
    if (!suser()) return(–1); /* 非 root 用户不可访问以保证日志完整性 */
    memset(tmp,0,MAXNAME+128);
    /* has 指向 FIFO 命令日志链表首，若无数据可读，则阻塞此进程 */
    if (!has) {interruptible_sleep_on(&wp);signal_pending(current);}
    if (!access_ok(VERIFY_WRITE,buf,buflen)) return(–1); /* 当前进程是否允许访问 buf */
    sprintf(tmp,“%d:%d:%d:%d:%s:”,has->start_time,has->uid,has->euid,has->gid,has->p_comm);
    strncat(tmp,has->cmd,MAXNAME–2);strcat(tmp,“\n”);
    copy_to_user(buf,tmp,buflen); /* 将日志信息复制到用户空间内存缓冲区 buf */
    … /* 释放已读过的日志结点的内核空间 */
}
```

2.3 中断劫持

new_execve 是 execve()系统调用新的处理函数.其主要工作是,从内核的任务控制结构中取出必要信息放在输出链表中,供读取此设备文件的进程使用,再真正执行 shell 命令.

```
int new_execve(struct pt_regs r) /* 用来劫持原 execve()系统调用 */
{
    char*cp,*filename=NULL;char ch;      int ret=0,i=0;j=0;
    lock_kernel();
    /* 为新的日志结点在内核分配空间,用 kmalloc() */
    struct lognode*ln=(struct lognode*)kmalloc((sizeof(struct lognode),GFP_HERNEL);
    ln->cmd=(char*)kmalloc(MAXNAME,GFP_KERNEL);
    /* 从用户空间取当前进程名,即 shell 命令字 */
```

```
filename=(char*)getname((char*)r.ebx);ln->next=NULL;
… /* 日志链表管理及读等待进程队列管理 */
/* 复制 shell 命令字到日志结点 */
memcpy(ln->cmd,filename,MAXNAME-1);i=strlen(ln->cmd);
/* 从用户空间复制命令行参数到内核空间,用内核系统调用 get_user() */
char**cmdargv=(char**)r.ecx;get_user(*cp,++cmdargv);
while ((cp!=NULL)&&(i<MAXNAME−2)) {
    ln->cmd[i++]=‘ ‘;
    while (1) { /* 每次循环从用户空间取一个参数 */
        get_user(ch, cp+j);
        if ((ch!=NULL)&&(i<MAXNAME−2)) {ln->cmd[i++]=ch;j++;}
        else break;
    }
    j=0;get_user(cp,++cmdargv); /* 取下一参数 */
}
/* 复制其他信息 */
memset(ln->p_comm,0,16);strcpy(ln->p_comm,current->p_opptr->comm,16);
ln->uid=current->uid;ln->euid=current->euid;
ln->current->gid=current->gid;…;
ln->start_time=CURRENT_TIME; /* 获得时间,参见 linux/timex.h */
ret=do_execve(filename,(char**)regs.ecx,(char**)regs.edx,&regs); / *执行 shell 命令 */
putname(filename);unlock_kernel(); return(ret);
}
```

current 是一个 task_struct 结构的全局变量,含有当前处理机上当前任务的信息,Linux2.0 内核在 linux/sched.h 中定义,而第 2.2 节内核定义在 asm/segment.h 中.

3 安全监测

3.1 审计能力比较

扩展 Shell 安全审计能力的 LKM 生成的审计记录格式为 todaytime:uid:euid:gid:parent:cmd.

各项分别为命令开始时间、用户号、有效用户号、组号、父进程名和 shell 命令.这些信息对于安全监测和入侵检测是必要的和方便的.我们用 vivie cron 入侵来比较扩展前后的审计能力(见表 1).

Table 1 Comparison of capabilities between two logging systems

表 **1** 两种日志系统的能力比较

Executing process of Vivie.sh①	Extended log records②	Shell records③
[test1]**$**	**40214:503:503:100:bash:./vivie.sh**	
Normal user④	40214:503:503:100:vivie.sh:/bin/cat	
[test1]**$**./vivie.sh	40214:503:503:100:vivie.sh:/usr/bin/cc-o/tmp/sh/tmp/sh.c	
make shell	…(略)	
compile shell	40214:503:503:100:vivie.sh:/bin/chmod755/tmp/makesh	./vivie.sh
make execute	40214:503:503:100:vivie.sh:/bin/cp-f/etc/sendmail.cf/tmp/send mail.cf.tmp1	(failed to
hack sendmail.cf	…(略)	detect the
make cron file	40214:503:503:100:vivie.sh:/usr/bin/crontab/tmp/cronfile	attack if the
input cron file	40260:0:0:0:sendmail:/tmp/makesh–Y-a-d test1	program is
wait for 1 minute	**40260:0:0:0:makesh:/bin/chown root/tmp/sh ←可疑**	renamed⑤)
	40260:0:0:0:makesh:/bin/chgrp root/tmp/sh	
execute shell	**40260:0:0:0:makesh:/bin/chmod 4755/tmp/sh**	
	40266:503:503:100:vivie.sh:/tmp/sh	
bash#(becomes root)	**40266:0:0:0:vivie.sh:/bin/sh←检测规则:euid=0&cmd=/bin/sh**	

①Vivie.sh 执行过程,②扩展后的审计记录,③Shell 记录,④普通用户,⑤若换名,则无法检测.

cron 是 Unix 系统中定时执行预定命令的进程.RedHat Linux 4.2,5.0,5.1,6.0 系统上的 Vivie cron 在向用户发送确认邮件时是作为 root 运行的,这时向 sendmail 传送任意的参数会导致普通用户获得 root 权限.入侵程序 vivie.sh 从 http://www.rootshell.com/下载.它是一个 shell 程序,普通用户运行后即成为 root.

3.2 简单监测规则的描述

新生成的审计记录对描述安全监测和入侵检测规则非常方便,如:

① 监测所有在晚上 10:00 到早上 6:00 执行的命令,规则为

((6*60*60>todaytime)||(todaytime<22*60*60)).

② 监测所有以 root 权限执行 shell 程序以得到 root 的交互式 shell,其规则为

(euid==0)&&((cmd==/bin/bash)||(cmd==/bin/csh)||(cmd==/bin/sh)).

第 3.1 节中的 vivie cron 攻击即可用此规则检测出来.

③ 监测所有 suid root 程序的执行,以监测缓冲区溢出攻击,规则为

(uid!=0)&&(euid==0).

4 结 论

利用 LKM 扩展 Shell 安全审计功能具有如下优点:将 shell 的审计能力扩展到所有经 exec 调用执行的命令,可以获得充分的审计信息;由于在设备驱动程序中只允许 root 操作而保证了审计日志的完整性;这种审计功能对普通用户和应用是透明的;对入侵者保持了隐蔽性;审计记录的获取只需在装入此 LKM 以后,按正常的文件操作方式操作设备文件“/dev/cmdlog”即可.生成的审计记录可以方便地用于安全监测和入侵检测.

References:

[1] Liu, Mei-lan, Yao, Jing-song. Audit trail and intrusion detection. Computer Engineering and Applications, 1999,35(7):12~15.

[2] Durst, R., Champion, T., Witten, B., *et al*. Testing and evaluating computer intrusion detection systems. Communications of the ACM, 1999,42(7):53~61.

[3] Halflife. Linux TTY hijacking. Phrack Magazine, 1997,7(50):5~5.

[4] Plaguez. Weakening the Linux kernel. Phrack Magazine, 1998,8(52):18~18.

附中文参考文献:

[1] 刘美兰,姚京松.审计跟踪与入侵检测.计算机工程与应用,1999,35(7):12~15.

An Extension to Security Auditing Mechanism of Linux Shell*

WANG Li-dong, FANG Bin-xing

(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001, *China*)

E-mail: wld@mail.cnnisc.gov.cn; bxfang@mail.cnnisc.gov.cn

http://www.hit.edu.cn

Abstract: Command history records generated by Unix shell is one of the important sources of system auditing information. But command history does not include sufficient information for intrusion detection and the history records can be easily modified by user themselves. With Linux loadable kernel module technique and system call interception, an extension to security auditing mechanism of Linux shell is implemented in this paper, and then some examples are given for security monitoring with the new mechanism.

Key words: security; audit; Linux loadable kernel module

* Received April 12, 2000; accepted July 31, 2000