Linux用户行为记录器的一种内核级实现方法

阮 越
(安徽工业大学 计算机学院,安徽 马鞍山 243002)

摘 要:用户登录后敲击键盘的行为记录是判断用户是否有恶意行为以及安全审计的重要信息来源。基于Linux,通过使用内核函数劫持技术劫持并修改键盘输入的相关内核函数,同时利用可装载内核模块技术将修改后的内核函数作为可装载模块插入内核,实现了一个内核级的行为记录器。这种实现方法可以记录本地用户以及远程用户所有敲击键盘的行为,包括ctrl,alt,shift等特殊键码,利用键盘映射码表转换成标准的ASCII码格式输出到日志文件中。

关键词:内核函数劫持;可装载内核模块;行为记录器
中图分类号:TP316 **文献标识码**:A **文章编号**:1673-629X(2008)02-0152-04

Implementation of Users' Behavior Recorder in Linux Kernel

RUAN Yue
(School of Computer Science, Anhui University of Technology, Ma'anshan 243002, China)

Abstract: The behavior records which are generated by users knocking on the keybord after users, login can be used to judge users' malicious act and is also one of the important sources of system auditing information. A Linux kernel-based behavior recorder is implemented in this paper, which has adopted kernel function hijacking technique to hijack and modify kernel functions and also utilized loadable kernel module technique to insert modified functions into kernel. All the keyboard records which are generated by local users and remote users, including some special key codes, such as ctrl, alt, shift, can be noted down by this method. And then the records which are converted to ASCII codes by key map output into a log file.

Key words: kernel function hijacking; loadable kernel module; behavior recorder

0 引 言

随着Internet的发展,网络安全已经成为人们越来越关注的目标。如何及时发现用户的恶意行为是安全防范的第一步。用户登录后敲击键盘的动作真实记录了用户的所作所为,但原始的Linux日志文件中的记录并不全面,日志文件和历史文件也很容易被有经验的hacker修改或者删除。针对这种情况,文中利用内核函数劫持技术和可装载内核模块技术(LKM, loadable kernel module)实现了一个内核级的用户行为记录器,全面记录了用户登录主机后的行为。

1 LKM相关技术

1.1 LKM

Linux内核是作为单内核体系结构(Monolithic architecture)而实现的,为了获得微内核体系结构(Microkernel architecture)带来的可扩展性和可维护性,Linux引入了可装载内核模块机制(LKM, Loadable Kernel Module),藉此来保证内核的紧凑性和单一体系结构的优点——上下文切换速度快。

在Linux中,用户(通常需要root权限)通过modutils软件包中提供的工具,动态地将模块插入、移出内核来扩展和删除内核功能,这个过程不需要编译内核也不需要关机和重启。因为模块运行的环境是内核,因而它具有内核特权,模块编程也就是内核编程,Linux中绝大多数的设备驱动程序都是通过LKM加载进内核的。

实现一个LKM最少需要两个基本函数: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)在用户空间编写/编译模块代码,解析未定义的符号,由模块装入器insmod完成链接装入过程;

(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

收稿日期:2007-05-30
基金项目:安徽省自然科学研究项目(2006KJ063B);安徽省高等学校青年教师科研资助项目(2007jq1028)
作者简介:阮 越(1972-),男,湖北红安人,硕士,讲师,研究方向为系统安全和嵌入式系统。

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,它具有内核程序所有的权限(核心态运行),可以访问所有的内核资源。利用LKM挂接系统调用已经成为hacker入侵的一种常用手段[1,2],但这种技术同时也为系统管理员监测系统活动提供了一种方法。

1.2 系统调用与内核函数劫持

系统调用与内核函数劫持是Linux平台上hacker的一种常用技术。在内核2.4.18以前,系统调用劫持相对简单,源文件include/asm/unistd.h中为每个系统调用定义了唯一的编号—系统调用号,它给出了系统调用对应的内核函数在系统调用表中的相对偏移量(源文件arch/i386/kernel/entry.S)。也就是说,只需在include/sys/syscall.h中找到需劫持的系统调用,然后保存sys_call_table[x]的旧入口指针(x代表所想要截获的系统调用的索引),然后将自定义的新的函数指针存入sys_call_table[x]即可。

在内核2.4.18以后,由于sys_call_table不能直接导出,系统调用和内核函数的劫持要复杂得多。分析系统调用的实现机理,每一个系统调用都是通过int 0x80中断进入核心的,而中断描述符表给出了中断服务程序和中断向量的对应关系。只要获得中断描述符表的起始地址,就可以计算得到int 0x80中断描述符的位置。由于idtr寄存器指向中断描述符表的起始地址,用sidt[asm ("sidt %0":"=m" (idtr));]指令就可以得到中断描述符表起始地址,从这条指令中得到的指针可以获得int 0x80中断描述符所在位置,然后计算出system_call函数的地址。利用gdb装入Linux核心,反编译system_call函数:

```
$ gdb -q /usr/src/linux/vmlinux
(no debugging symbols found)…(gdb) disass system_call
Dump of assembler code for function system_call:
……
0xc0106bf2 <system_call+42>:jne 0xc0106c48 <tracesys>
0xc0106bf4 <system_call+44>:call *0xc01e0f18(,%eax,4)
0xc0106bfb <system_call+51>:mov %eax,0x18(%esp,1)
0xc0106bff <system_call+55>:nop
End of assembler dump.
(gdb) print &sys_call_table
$1=(<data variable,no debug info> *) 0xc01e0f18
(gdb) x/xw (system_call+44)
0xc0106bf4 <system_call+44>:0x188514ff <--得到机器指令(little endian)
(gdb)
```

可以看到在system_call函数内,是用call *0xc01e0f18指令来调用系统调用对应的内核函数的。因此,只要找到system_call里的call sys_call_table(,eax,4)指令的机器指令就可以了。这时读取内存映像(/dev/kmem),使用模式匹配的方法获得这条机器指令的地址[3]。

1.3 Linux键盘驱动的工作原理

Linux下每输入一个键盘值,键盘都将发送相应的扫描码(scancodes)给键盘驱动。一个独立的击键可以产生一个六个扫描码的队列。键盘驱动中的handle_scancode()函数解析scancodes流并通过kdb_translate()函数里的转换表(translation-table)将击键事件和键的释放事件(key release events)转换成连续的keycode。比如,'a'的keycode是30。击键'a'的时候便会产生keycode 30。释放a键的时候会产生keycode 158(128+30)。然后,这些keycode通过对keymap的查询被转换成相应key符号。这步操作之后,获得的字符被送入raw tty队列即tty_flip_buffer。receive_buf()函数周期性地从tty_flip_buffer中获得字符,然后把这些字符送入tty read队列。当用户进程需要得到用户的输入的时候,它会在进程的标准输入(stdin)调用read()函数。sys_read()函数调用定义在相应的tty设备(如/dev/tty0)的file_operations结构中指向tty_read的read()函数来读取字符并且返回给用户进程[4,5]。

键盘驱动器有4种工作模式:

(1)scancode(RAW模式):应用程序取得输入的scancode。这种模式通常用于应用程序实现自己的键盘驱动器,比如X11程序。

(2)keycode(MEDIUMRAW模式):应用程序取得key的击键和释放行为(通过keycode来鉴别这两种行为)信息。

(3)ASCII(XLATE模式):应用程序取得keymap定义的字符,该字符是8位编码的。

(4)Unicode(UNICODE模式):此模式唯一和ASCII模式不同之处就是UNICODE模式允许用户将自己的十进制值编写成UTF8的unicode字符。

2 实　现

2.1 思　路

通过对Linux键盘驱动原理的分析,要获取用户本地和远程会话的所有的键盘击键(包括ctrl、alt等特殊键),选择劫持工作在MEDIUM RAW模式下,直接读取键盘缓冲tty_flip_buffer的receive_buf函数。在内核中,tty_struct和tty_queue结构仅仅在tty设备打

开的时候被动态分配。因而,同样需要通过劫持 sys_open 系统调用来动态地 hooking 每次调用时 tty 或 pty 的 receive_buf()函数。

2.2 劫持 receive_buf()

源文件"/usr/src/linux/drivers/char/n_tty.c"中给出了 n_tty_receive_buf 的函数定义,static void n_tty_receive_buf(struct! tty_struct * tty, const unsigned char * cp,char * fp,int count),其中参数 cp 是一个指向设备接收的输入字符的 buffer 的指针。参数 fp 是一个指向一个标记字节指针的指针。在源文件"/usr/include/linux/tty.h"中查看 tty 结构:

```
struct tty_struct {
    int magic;
    struct tty_driver driver;
    struct tty_ldisc ldisc;
    struct termios * termios, * termios_locked;
.....
}
```

在源文件"/usr/include/linux/tty_ldisc.h "中查看 tty_ldisc 结构:

```
struct tty_ldisc {
  int magic;
  char * name;
  .....
  void ( * receive_buf)(struct tty_struct * ,
  const unsigned char * cp, char * fp,int count);
  int ( * receive_room)(struct tty_struct * );
  void ( * write_wakeup)(struct tty_struct * );
};
```

要劫持这个函数,先保存原始的 tty receive_buf()函数,然后重置 ldisc.receive_buf 到新的 new_receive_buf()函数中记录用户的输入。假如需记录在 tty0 设备上的输入:

```
int fd=open("/dev/tty0",O_RDONLY,0);
struct file * file=fget(fd);
struct tty_struct * tty=file->private_data;
old_receive_buf = tty -> ldisc.receive_buf;//保存原始的 receive_buf()函数
tty->ldisc.receive_buf=new_receive_buf;//替换成新的 new_receive_buf 函数
//新的 new_receive_buf 函数
void new_receive_buf(struct tty_struct * tty,const unsigned char * cp,char * fp,int count)
{
logging(tty,cp,count);//记录用户击键
/* 调用回原来的 receive_buf */
( * old_receive_buf)(tty,cp,fp,count);
}
```

2.3 劫持 sys_open()

劫持 sys_open 调用需首先保留原先的 sys_open,然后将系统调用号对应到新的 new_sys_open 上。并在新实现中完成对内核函数 receive_buf()的劫持。

```
original_sys_open = sys_call_table[_NR_open];
sys_call_table[_NR_open]= new_sys_open;
//new_sys_open()
asmlinkage int new_sys_open(const char * filename,int flags,int mode)
{
.....
//调用 original_sys_open
ret=( * original_sys_open)(filename,flags,mode);
if (ret >= 0) {
    struct tty_struct * tty;
    .....
    file = fget(ret);
    tty = file->private_data;
    if (tty ! =NULL && tty->ldisc.receive_buf ! = new_receive_buf) {
      .....
      //保存原来的 receive_buf
      old_receive_buf=tty->ldisc.receive_buf;
        .....
    /*
    * 开始劫持该 tty 的 receive_buf 函数
    * tty->ldisc.receive_buf=new_receive_buf;
    */
    init_tty(tty,TTY_INDEX(tty));
    }
.....
}
//我们的新的 receive_buf()函数
void new_receive_buf(struct tty_struct * tty,const unsigned char * cp,char * fp,int count)
{
    if (! tty->real_raw && ! tty->raw)//忽略 raw 模式
    //调用我们的 logging 函数来记录用户击键
    vlogger_process(tty,cp,count);
    //调用回原来的 receive_buf
    ( * old_receive_buf)(tty,cp,fp,count);
}
```

2.4 模块定义

init_module()和 cleanup_module()是 LKM 必需的两个函数。在 init_module()函数中需要将实现注册进内核,重新指定系统调用对应的入口地址。cleanup_module()取消注册,并恢复系统调用和劫持的

内核函数原来的入口地址。

```
int init_module(void)
{
    original_sys_open = sys_call_table[__NR_open];
    sys_call_table[__NR_open] = new_sys_open;
    my_tty_open();
    MOD_INC_USE_COUNT;//模块使用计数
    return 0;
}
void cleanup_module(void)
{
    int i;
    sys_call_table[__NR_open] = original_sys_open;
    for (i=0; i<MAX_tty;i++) {
      if (ttys[i] != NULL) {
      ttys[i]->tty->ldisc.receive_buf = old_receive_buf;//
恢复系统原先的 receive_buf 函数
    }
    }
}
```

3 结　论

利用LKM和内核函数劫持技术实现的用户行为记录器,可以有效地对用户登录后的行为进行跟踪,具有以下特点和优点:

(1)可以记录本地和远程会话的所有击键(通过tty和pts);

(2)支持记录所有的特殊键如方向键(left,right,up,down),F1到F12的功能键,ctrl+F1到Shift+F1等组合键;

(3)因为劫持的内核函数是直接跟键盘缓冲区相连的receive_buf,所以比起其它一些实现技术,比如劫持sys_read这样使用频繁的内核函数,对系统效率的影响较小。

参考文献:

[1] Pragmatic. Complete Linux Loadable Kernel Modules[EB/OL]. 2004 - 12. http://www.thehackerschoice.com/papers/LKM_HACKING.html.

[2] Salzman P J, Burian M, Pomerantz O. The Linux Kernel Module Programming Guide[EB/OL]. 2005 - 05. http://www.tldp.org/guides.html.

[3] Cesare S. Kernel function hijacking[EB/OL]. 2003 - 02. http://www.big.net.au/silvio/kernel - hijack.txt, Feb.

[4] Brouwer A. The Linux keyboard driver[EB/OL]. 2002 - 01. http://www.linuxjournal.com/lj - issues/issue14/1080.html.

[5] Halflife. Linux TTY hijacking[J]. Phrack Magazine, 1997, 7(50):5 - 6.

(上接第138页)

最后可得到最大频繁项集为{123},{134},支持度分别为20%,26.7%。对上述实例用自顶向下算法提取关联规则,在找到最大频繁项集之前要产生候选集14个,分别是{1234},{1235},{1345},{2345},{123},{124},{134},{234},{125},{135},{235},{345},{145},{245},其中{123},{134},{135},{234},{235},{345}会重复产生两次。可以看出,此过程产生了大量的候选集,同时出现了大量的冗余,而且每次判断侯选集是否是频繁项集时要扫描整个数据库。而QTTB算法既不用产生候选项集,也不用扫描整个数据库,这充分说明QTTB算法能显著地减少扫描数据库的次数和避免产生大量的冗余,提高最大频繁集的提取效率。

4 结　论

QTTB算法把事物操作数据作为候选集,用新的存储结构存储数据并利用集合运算性质寻找最大频繁集。实验证明该算法避免了组合爆炸问题,利用了前面扫描所得的结果,缩减了扫描数据量,提高了数据挖掘效率,在长模式多的大型数据库中应用更显优势。

参考文献:

[1] Han J, Kamber M. 数据挖掘:概念与技术[M]. 北京:机械工业出版社,2001.

[2] Kantardzic M. 数据挖掘[M]. 北京:清华大学出版社,2003.

[3] 邵峰晶,于忠清. 数据挖掘原理与算法[M]. 北京:水利水电出版社,2003.

[4] 毛国君,段立娟,王　实,等. 数据挖掘原理与算法[M]. 北京:清华大学出版社,2005.

[5] 李清峰,杨路明,张晓峰,等. 数据挖掘中关联规则的一种高效Apriori算法[J]. 计算机应用与软件,2004,21(12):84 - 86.

[6] 宋　雨,赵建利,王保义. 关联规则挖掘中最大频繁集的双向查找算法[J]. 华北电力大学学报,2005,32(2):67 - 71.

[7] 章　艳,刘美玲,张师超,等. Apriori算法的三种优化方法[J]. 计算机工程与应用,2004,40(36):191 - 193.

[8] 欧阳军,马　稳,沈钧毅,等. 一种新的广义关联规则挖掘算法[J]. 小型微型计算机系统,2004,25(5):875 - 877.

[9] 刘桂庆,胡学钢,李　凯. CR一种逆向的关联规则挖掘算法[J]. 微电子学与计算机,2004,21(9):83 - 86.

[10] 王创新. 关联规则提取中对Aprior算法的一种改进[J]. 计算机工程与应用,2004,40(34):183 - 185.