Linux 目标代码内核补丁的机理和应用

胡勇其[1,3]　匡先锋[1,3]　侯紫峰[2]
[1](中国科学院计算技术研究所,北京 100080)
[2](联想研究院,北京 100085)
[3](中国科学院研究生院,北京 100083)
E-mail:huyqa@lenovo.com

摘　要　该文介绍了基于目标代码(二进制编译文件)为 Linux 运行时系统内核内存映像打补丁以修改内核的原理和方法,并给出了一个示例场景和程序以说明其应用。掌握该方法对于内核开发、系统调试和系统安全都具有重要的实用价值。

关键词　内核补丁　Linux　目标代码

文章编号 1002-8331-(2004)32-0121-03　**文献标识码** A　**中图分类号** TP311

Introduction of Mechanism and Application of Linux Kernel Patching with Object Code

Hu Yongqi[1,3]　Kuang Xianfeng[1,3]　Hou Zifeng[2]
[1](Institute of Computer Technology, the Chinese Academy of Sciences, Beijing 100080)
[2](Lenovo Corporate R&D, Beijing 100085)
[3](Graduate School of the Chinese Academy of Sciences, Beijing 100083)

Abstract: The mechanism and method of kernel patching on linux kernel image running in memory with object code is introduced.The source code as an example is presented to demonstrate the usage of this method.It is important to know this method for kernel development and debugging and system security.

Keywords: kernel patching, Linux, object code

1　引言

Linux 内核源代码开放为基于内核的系统开发提供了便利和基础。对内核功能的扩展和对现有功能、机制的改进主要通过两种方式来实现:可加载内核模块和修改内核核心代码。前者既可以编译到内核映像中[1],在内核启动时加载,也可以在内核启动后通过内核模块工具如 insmod 等加载,并在不需要时通过内核工具从内核卸载,以节约系统资源,一般认为这是内核开发的一种很好的方式,因为它只调用内核 API,与内核其他部分的耦合度低,无需太关心内核其他部分的细节,同时便于调试和开发;而通过修改内核核心代码为内核加补丁的方式涉及对内核核心部分的基本机制、结构和功能的修改,涉及内核 API 内部实现甚至 API 的改动,无法通过可加载内核模块来实现,必须将改动编译到内核映像中作为内核不可或缺的一部分,在系统启动时加载。

通过修改内核核心基本代码,可以实现符合特定功能要求的系统,因此深层次的开发常需要给内核打补丁,重新编译代码,然后构造一个新的系统。这种方式对于已经开发调试完毕的系统而言无疑是很好的选择,但是整个开发过程、特别是调试过程将会很麻烦费时,需要不断地修改代码、编译内核、重新启动系统,即使是通过虚拟机能够缩短一部分调试时间,大部分工作仍然需要在真实的环境中进行,编译、重启系统、运行,这一冗长而重复的过程依然不可避免。

该文介绍基于目标代码(二进制编译文件)的运行时内核补丁,这种方式利用 Linux 现有的功能和机制,无需重新启动系统就可以将目标代码加入到运行中的内核内存映像中,方便了内核开发过程特别是系统调试过程,甚至最终发行系统也可以采用这种方式,系统无须停止运行就可以修改内核映像,这对于一些宕机时间要求很严格的系统尤其是个很好的更新方式。同时,它还是一种黑客攻击方式,了解这种方式的运用,对于增强系统管理员的防范意识和提高系统安全性无疑也很有帮助。

该文从内核映像的编译、链接和执行出发,介绍了通过目标代码给内核映像加补丁的原理和机制,以及其实现的途径和方式,并给出了在实际内核开发中应用的一个例子来直观说明其机理和实际应用。

2　Linux 直接内核内存访问

系统启动后,Linux 内核映像被加载到内存的内核区域,并且锁定在该区域中,所有的系统调用和新加载的内核模块都要调用内核映像中的函数和数据结构,即内核 API。Linux 系统实现了可以直接访问内核内存区域的字符设备,包括/dev/mem和/dev/kmem,前者为物理内存访问,是线性内存(物理内存)

作者简介:胡勇其(1975-),男,硕士研究生,研究方向:嵌入式系统,网络存储,无线网络。匡先锋(1979-),男,硕士研究生,研究方向:视频编码。侯紫峰(1955-),男,博士生导师,博士,联想研究院首席研究员,主要研究方向为无线局域网、移动通信、数字信号处理、并行计算。

的文件映射,其主从设备号为 1(1),后者为内核虚拟内存访问,是所有虚拟内存的文件映射,即线性地址和交换空间,其主从设备号是 1(2)。系统运行期间,可以通过这两个设备访问系统内存,只要具有操作权限,按照操作正常文件的方式打开该设备就可以使用,要访问指定的内存地址可以通过 seek 调用指定到所要的位置即可,该设备是可读可写的。

3 目标代码内核补丁机理分析

3.1 内核符号表[2]

这里所讨论的 Linux 内核符号是指内核中的全局性函数、变量和数据结构的名称和相应的内存地址。编译工具通过解析和重定向这些符号来实现程序的编译、链接和动态加载。按照对于外界目标文件(即可加载内核模块)的可见度,Linux 内核符号分为导出符号和未导出符号,导出符号不仅可以在已经编译到内核映像中的其他内核函数中调用、访问,而且可以在动态加载的内核模块中使用,导出内核符号及其在内核中的地址可以通过/proc/ksyms 查看,proc 文件系统读取内核中的符号链表并返回给用户;也可以通过系统调用 get_kernel_syms 获得内核和模块的导出符号表,模块工具就是通过此系统调用来获得符号的内存地址并解析、重定向和加载内核模块的。未导出符号相当于目标文件中的静态全局变量,对于目标文件本身的所有函数都是可见的,但是对于目标文件外部的其他模块是不可见的,Linux 中的未导出符号仅仅在编译到内核映像中的内核函数和模块是可见的,但是对于动态加载到内核中的模块是不可见的,因而无法解析和重定向模块中对这些符号的调用,在可加载内核模块无法使用。

Linux 在编译内核时也同时保留了内核中的非导出内核符号的地址记录,存放在内核源代码根目录中的名为 System.map 的文件中,这也是内核调试和 Oops 信息处理解析内核地址所必须使用的文件,一般需要复制到 boot 目录下,并设置其路径,以便在内核出错时如 OOPS 等工具可以解析错误所在的内核地址,提供用户可以理解的调试信息。

Linux 内核符号表是实现可加载内核模块的基础,可加载内核模块作为目标文件,包含未解析的内核变量和函数,加载到内核内存空间之前必须通过模块工具解析、重定向对内核符号的调用,错误的或无法解析的内核符号都会导致内核加载失败或者系统崩溃。

实现内核目标代码补丁不仅仅需要内核和模块的导出符号,更需要内核中的未导出符号,而后者是该文的基础,导出符号表是通过系统调用可获得的,而非导出符号对可加载内核模块是不可见的,必须通过别的途径来解析这些符号。这是目标代码内核补丁和普通内核源代码补丁的本质区别,普通内核源代码补丁通过修改源代码加入内核,成为内核源代码的基本组成部分,内核中的所有全局符号对于该部分代码都是可见的,而目标代码内核补丁是在内存中的执行内核映像上进行的,已经无法通过正常的途径(系统调用、proc 文件系统)见到这些符号了,必须采用其他的方式获得符号表。

3.2 目标代码内核补丁的机制

实现目标代码内核补丁的关键是获得内核中的未导出符号的地址,然后通过把代码加入到内核并替换或者更换内核符号,从而实现非源代码的内核映像更改,实现所期望的功能。以上这两者需要通过上述的内核符号和内核内存直接访问来实现。

3.2.1 获取非导出内核符号表

有多种方式可以获取未导出内核符号的地址,对于已经在文件 System.map 中有记录的未导出内核符号,其内存地址为已知的,通过/dev/kmem 访问相应的内存位置就可以获得该符号的值;还可以通过所获得的变量、结构的值,间接访问其他变量,比如通过指针可以访问其他值,通过找到数组的首地址可以访问数组中其他的值。这部分内核符号表已经可以视同导出符合了,差别仅仅是获取的途径不同。

对于文件 System.map 中没有记录的内核符号,可以通过查看内核代码,查找与其相邻的符号、通过指针访问他的其他变量、通过已知其值的情况下搜索(比如某一字符串)也可以找到其地址。

当通过以上方式都没有办法找到所要的地址时,还可以通过查看内核代码,编译所要查找部分的代码,然后通过目标代码模式匹配来查找其地址,当然,这种方式查到的代码不一定唯一,效率低并需要其他的辅助条件来协助判断。

总之,可以通过多种方式获取非导出内核符号表,只要是可以找到符号而又不破坏内核映像的方式都是可以接受和使用的。

3.2.2 加载模块和更改内核映像

找到所需要的符号的地址后,需要替换、修改或者使用其值来为内核映像打补丁,这需要把用以替换内核映像中已有代码的目标文件加载到内核地址空间。有两种方式可以实现这一目的:

通过可加载内核模块[2,3]:将用于替换内核映像的目标代码(函数、变量、结构)作为一个可加载内核模块,通过模块工具链接到内核,通过参数提供要替换的内核映像部分的目标地址,然后在内核模块的初始化函数中执行替换。这种方式的一个优点是充分利用了内核所提供的机制,并且可以在不需要的时候通过工具卸载并恢复系统的原有状态,当然,在加载和卸载时要注意保持内核原有的地址,并在卸载时不要留下垃圾、造成内核污染乃至系统崩溃(比如,在为符号指定名字时就需要注意在卸载时恢复原来的字符串,否则就留下无效的地址,当内核访问该地址时就会引发错误乃至内核崩溃)。

通过直接修改内核映像,将模块直接通过/dev/kmem 加载到内核未使用或者保留的地址空间来实现代码、数据的加载和内核更改、替换。这种方式不利用内核所提供的动态加载机制,即使在没有打开该功能的内核映像上也可以实现。内核预留了一定的内存空间用于将来分配,找出这部分空间并修改其使用记录,将目标代码写到该空间就可以实现代码加载。但是这种方式需要开发者解析对所有内核符号的引用(包括导出符号)并且需要重定向代码,实现起来比较复杂,而且卸载也比较复杂,不推荐使用。

4 目标代码内核补丁的应用

4.1 应用范围

基于目标代码的内核补丁主要可以有两种应用:一是用于内核开发,既可以方便开发调试过程,又可以在系统不停机的情况下修改内核映像;二是黑客攻击手段,通过在内核映像中加载代码、修改内核映像来实现对于目标系统的攻击,并通过修改内核映像来隐藏其自身不为用户所观察到。由于大部分的

运行系统的内核映像都是相同或者基本相同的(对于基于发行系统的系统尤其如此),实现这种方式的攻击还是不太困难的。当然,能够访问内核内存需要获得访问权限,但是这并不是特别困难,而且一旦侵入系统,这种方式无疑会给系统更大、更持久的控制和损害,这是系统管理员应该注意的。

4.2 应用示例

以下通过例子来说明基于目标代码的内核补丁的应用,由于直接将代码加载到内核未用内存涉及到目标文件格式、符号解析和内核内存管理等内容,描述起来比较复杂,所以该文仅描述通过可加载内核模块将代码加载到内核内存空间,同时,所有的内核符号以本机所采用的 RedHat Linux 7.3(内核版本 2.4.18-3)为例,因此在其他的系统上也许会有所不同。

应用场景如下:拟实现一个替换现有的基于 X86 的系统的键盘中断处理程序。键盘中断序号为 1,在 Linux 中键盘中断处理程序是通过非共享方式加载的,因此无法把新的中断处理函数加到原来的函数上。普通的调试和替换方式为:在模块初始化函数中通过 free_irq(1,NULL)将原有的中断处理函数卸载,然后通过 int request_irq(unsigned int irq,void(*handler)(int,void*,struct pt_regs*),unsigned long irqflags,const char *devname,void*dev_id)设置中断 1 的处理函数为将要调试或者替换的函数。这种方式可以实现所要达到的目的,但是有以下缺陷:无法恢复原有的中断,因为没有办法得到原有的中断处理函数的地址,因此模块无法卸载,否则没有了中断处理函数,键盘将无法输入任何字符,只能通过重新启动系统来调试。为了将新的中断处理函数稳定地加入现有的系统内核,需要修改内核源代码、编译内核、重新启动系统。

通过基于目标代码的内核补丁可以克服上述缺点。步骤和相应的代码如下:

(1)查找原中断处理程序的地址

内核中断处理例程只可以通过 API 操作,内核没有导出中断处理函数的地址,需要通过/dev/kmem 获得。所有的中断处理函数的地址放在 irq_desc [NR_IRQS]数组中,内核中所定义的数据结构如下:

```
#define NR_IRQS 224
struct irqaction {
    void(*handler)(int,void *,struct pt_regs *);
    unsigned long flags;
    unsigned long mask;
    const char *name;
    void *dev_id;
    struct irqaction *next;
};
typedef struct {
    unsigned int status;        /* IRQ status */
    hw_irq_controller *handler;
    struct irqaction *action;   /* IRQ action list */
    unsigned int depth; /* nested irq disables */
    spinlock_t lock;
} ____cacheline_aligned irq_desc_t;
extern irq_desc_t irq_desc [NR_IRQS];
```

符号 irq_desc 未导出,但是通过查看 System.map 发现该符号的地址为 c0322900,通过该地址可以从内核内存中找到该数组,其第二个变量所存放的就是原来的键盘中断处理的 irq_desc_t 结构,通过该结构可以找到原来的中断处理函数的地址和名字字段如下(其中的标识字段是笔者在程序中加的):

```
IRQ Number=1
Status 0
irq.Handler address 0xc02c2580
Action address 0xc3fcf4e0
Action.name address 0xc0221819
Action.name :keyboard
Action.Handleraddress 0xc0183c60
IRQ dev_id address :(nil)
```

(2)中断处理程序的替换和恢复

该例子是一个简单的按键扫描码输出程序,模块初始化时用用户定义的中断处理程序替换系统原有的键盘中断处理程序,按下键发生中断,输出相应的扫描码和按键状态,直到按下键 Q 为止,然后恢复原来的中断处理函数。在内核模块初始化函数、中断处理函数和模块清理函数中通过如下代码来加载和恢复中断处理函数:

```
static int OldHandler;
MODULE_PARM(OldHandler,"i");
static int stop=0;
char *mydev_id="MyOwn";
static void got_char(void *scancode)
{
printk("Scan Code %x %s.\n.
(int)*((char *)scancode)& 0x7F,
*((char *)scancode)& 0x80 ? "Released" :"Pressed");
if(((int)*((char *)scancode)& 0x7F)==0x10){
      stop=1;
      printk("The handler will be stopped.\n");
 }
}
void(*handler)(int,void *,struct pt_regs *);
static struct tq_struct task;
static unsigned char scancode;
/* 中断处理函数 */
void irq_handler(int irq,void *dev_id,struct pt_regs *regs)
{
   unsigned char status;
   if(stop==1){ /* 以共享方式恢复原来的中断 */
request_irq(1,OldHandler,SA_SHIRQ,0xc0221819,NULL);
          stop=2 ;return;
 }
 if(stop==2)return;
 /* 读键盘状态 */
 status=inb(0x64);
 scancode=inb(0x60);
 /* Scheduale bottom half to run */
 queue_task(&task,&tq_immediate);
 mark_bh(IMMEDIATE_BH);
}
int init_module()
{
      if(OldHandler==0)OldHandler=0xc0183c60;
      INIT_TQUEUE(&task,got_char,&scancode);
      free_irq(1,NULL);
      return request_irq(1,irq_handler,/* our handler */
```

(下转 180 页)

询请求，察看数据仓库元数据，例如各种数据库表和列所包含的业务逻辑意义，包括各种数据库内容的业务逻辑意义，市场环境背景说明等，决定数据挖掘的商业逻辑和目标方向；察看数据挖掘服务器的各种算法功能和用法描述并指定采用的算法；根据从元数据中了解到的业务信息指定挖掘算法的操作对象和范围（如数据库的列、表，数据立方体的层次和维度等）；定义或选择数据挖掘元规则，数据挖掘约束规则，提交挖掘请求。挖掘结果用各种形象的图表展现或者自动发送到用户指定邮箱。

正如前文所述，前端展现服务响应采用 WEB 服务技术实现。前端展现的客户端平台是没有限制的，可以是基于浏览器的，也可以是客户端应用程序，只要客户端发送一个符合预定格式的消息请求，前端展现接口对消息进行解读、传送，服务端做出响应服务。这样前端展现的形式具有灵活多样性，甚至可以实现用户用 EMAIL 方式发送规定格式的信件标题消息或者用手机发送一条短信息到指定接收端，前端展现接口解读出消息内容，传送到服务端处理，最后把数据挖掘结果发送到用户邮箱或者以短信息形式发送到用户手机。

3.4.2 数据仓库

CWM 规范已成为业界公认的主流统一标准，越来越多的数据库厂家支持 CWM 规范。目前，声称支持 CWM 标准的厂商有 ORACLE，IBM，Unisys，Hyperion，SAS，Meta Integration 等，能够利用 CWM 在不同的数据仓库工具上成功地实现互联和互通，主要是通过在元数据模型导入导出方面提供对符合 OMG CWM 规范的 XML 文件的解析读入支持以及导出 XML 格式文件的元模型描述功能。例如 Oracle9i Warehouse Builder（OWB）中附带有 CWM DTD，OWB 提供的数据导入/导出工具都带有 CWM 格式的选项，其源/目标对象是描述 CWM 元模型的 XML 文件。

由于数据挖掘工具采用与 CWM 元数据模型标准相结合的技术，使得对不同的数据仓库平台，只要其支持 CWM 规范就同样适用。数据仓库导出对应的元数据模型，数据挖掘工具就可以解析并构成元数据模型实例。把各种元数据信息反馈到前端展现界面供用户查询浏览以辅助决策支持过程。

对于尚未提供支持 CWM 标准的数据仓库平台，可以通过加入元数据转换层来实现对 CWM 元模型的解析读取。

结合前文所述，数据仓库接口实现了对不同数据仓库平台环境的判断和调用相应的 API 访问接口，实现了底层数据仓库平台访问的无限制，因此底层数据仓库是平台无限制的，具有灵活性和通用性。

4 小结

由于企业的发展和应用环境的复杂化，对各种平台无关的技术和应用要求越来越迫切。平台无关，强调通用性、灵活性和可重用性已成为软件产品的主流方向。而目前各种数据仓库平台所带的数据挖掘工具受到各种限制，缺乏通用性、灵活性和可重用性。迎合应用的需要，该文提出了一个平台无限制的数据挖掘服务中心的设计方案，利用 WEB 服务技术响应任何平台多种客户端形式的服务请求，利用公共数据仓库元数据模型 CWM 可在所有支持 CWM 的数据仓库平台中工作。由于组件间都通过接口通信，企业数据仓库和客户端的平台和结构就变得非常灵活且易于扩展升级。当企业应用需求发生变化，现有环境不满足要求时，可更改底层和外层环境而不需改动数据挖掘服务中心。而且，把此数据挖掘服务中心体系推广到别的企业上实施也不需要太多的改动，由此实现了目前数据挖掘工具所缺乏的通用性、灵活性和可重用性问题。

（收稿日期：2004 年 6 月）

参考文献

1.Anca Vaduva，Jürg-Uwe Kietz，Regina Zücker.M4：a metamodel for data preprocessing[C].In：Proceedings of the fourth ACM international workshop on Data warehousing and OLAP，Atlanta，Georgia，USA，2001-09：85~92

2.Object Management Group（OMG）.Common Warehouse Meta Model（CWM）Specification（V1.1）.http：//www.cwmforum.org/，2003-03

3.Rawn Shah.Start Here to learn about Web services.http：//www-900.ibm.com/developerWorks/cn/webservices/ws-starthere/index.shtml，2003-07

4.Object Management Group（OMG）.Meta Object Facility（MOF）Specification（V1.4）.http：//www.cwmforum.org/，2002-04

5.Object Management Group（OMG）.Common Warehouse Meta Model（XMI）Specification（V1.1）.http：//www.cwmforum.org/，2003-03

6.A Vaduva，K R Dittrich.Metadata management for data warehousing：Between vision and reality[C].In：International Database Engineering & Applications Symposium，IDEAS'01，Grenoble，France，2001-07：129~135

7.Colin Atkinson，Thomas Kühne.The Role of Metamodeling in MDA[C].In：International Workshop in Software Model Engineering（in conjunction with UML'02），Dresden，Germany，2002-10

8.王强，刘东波，王建新.数据仓库元数据标准研究[J].计算机工程，2002，28（12）：123~125

9.吴建芹，吴建林，张雷.基于 CWM 的企业数据仓库体系结构设计[J].计算机工程与应用，2002，38（21）：202~204

（上接 123 页）

```
        SA_SHIRQ,"t_irq_handler",(void *)mydev_id);
    }
    void cleanup_module()
    {
    free_irq(1,(void*)mydev_id);/* 删除自定义的中断 */
    }
```

5 小结

该文介绍了通过目标代码（二进制编译文件）内存中的 Linux 运行内核映像加补丁的机制和方法，并给出了一个例子加以说明其应用。认识和掌握这种方式对于内核开发和系统安全都是有价值的。当然，在应用时也应该进一步考虑内核锁和其他互斥机制，防止污染或者破坏内核。（收稿日期：2004 年 3 月）

参考文献

1.Peter Jay Salzman，Oripomerantz.Linux kernel programming guide.v2.4.0，2003-04-04

2.Alessandro rubini，Jonathan corbet.Linux Device Driver[M].2nd Edition，O'Reilly& Associates，Inc

3.Daniel P Bovet，Marco Cesati.Understanding the Linux Kernel[M].2nd Edition，O'Reilly，2002-12