KFUR:一个新型内核扩展安全模型

马 超[1] 尹 杰[1] 刘虎球[1] 李 浩[2]

[1](清华大学计算机科学与技术系 北京 100084)

[2](西安电子科技大学计算机科学与技术学院 西安 710126)

摘 要 保障内核扩展的安全性对操作系统具有重要意义.当前存在大量针对内核函数使用规则的攻击,内核扩展中也存在大量违反内核函数使用规则的错误,因此针对内核函数使用规则的安全性检测十分必要.虽然存在多种提高内核扩展安全性的方法,但很少有方法对内核函数的使用规则进行安全性检测.文中设计了KFUR(Kernel Function Usage Rule)内核扩展安全模型系统,用于在运行时检测内核扩展调用内核函数是否遵守内核函数使用规则.如果内核扩展调用内核函数满足模型安全运行条件,则允许对该内核函数进行调用,否则将错误报告给操作系统内核并终止该内核扩展的运行.文中所述研究在Linux操作系统上对KFUR安全模型系统进行实现,并将其运用于e1000网卡驱动、SATA硬盘驱动和HDA声卡驱动内核扩展.安全性评测表明安全模型系统能够对内核函数使用规则进行安全性检测,性能评测表明安全模型系统带来的开销很小.

关键词 操作系统;KFUR安全模型;内核扩展;内核函数使用规则

中图法分类号 TP316 **DOI号**:10.3724/SP.J.1016.2012.02091

KFUR: A New Kernel Extension Security Model

MA Chao[1] YIN Jie[1] LIU Hu-Qiu[1] LI Hao[2]

[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

[2](*School of Computer Science and Technology, Xidian University, Xi'an* 710126)

Abstract Security in kernel extensions is important for operating systems. There are lots of attacks aiming at kernel function using rules in kernel extensions and bugs about these rules which are caused by programmers' mistakes. Hence, it is significant to check it. Several methods have been proposed to improve the security of the kernel extension, but few approaches check whether kernel function calling in kernel extensions works in the right rules. We propose an innovative KFUR kernel extension security model system, which could check whether kernel extensions violate using rules at runtime. KFUR is short for the kernel function usage rule. If kernel function calling meets KFUR model safe operation condition, it is permitted, otherwise it is reported to the operating system kernel and the kernel extension is stopped. The KFUR security model system is implemented in the Linux operating system and used in device drivers of the e1000 network card, the SATA hard disk and the HDA sound card. Security evaluation shows that our system can check kernel function usage rules and performance evaluation shows that our mechanism only brings little overhead.

Keywords operating system; KFUR security model; kernel extension; kernel function usage rule

---

收稿日期:2012-06-30;最终修改稿收到日期:2012-08-14.本课题得到国家"八六三"高技术研究发展计划重大项目"以支撑公众与企业服务为主的网络操作系统研制"(2011AA01A203))资助.**马 超**,男,1986年生,博士研究生,主要研究方向为操作系统、内核扩展可靠性与安全性、确定性多线程.E-mail:mc08@mails.tsinghua.edu.cn.**尹 杰**,男,1987年生,博士研究生,主要研究方向为操作系统.**刘虎球**,男,1989年生,硕士研究生,主要研究方向为操作系统、内核扩展可靠性与安全性.**李 浩**,男,1989年生,本科,主要研究方向为操作系统.

1 引 言

内核扩展是可以在运行时添加到操作系统内核中的功能模块.如图1所示,在通用操作系统中,内核扩展运行于内核态,可以访问几乎所有计算机系统的资源,具有很高的特权.因此,内核扩展的安全性直接影响操作系统的安全性.

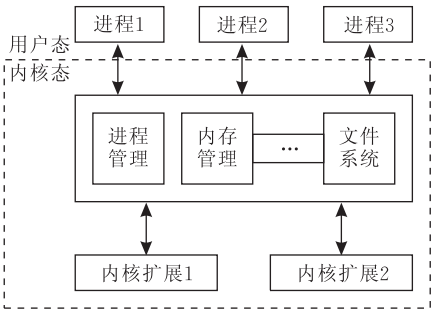


图1 操作系统结构图

由于内核扩展可以动态添加和删除,因此操作系统大量使用内核扩展实现各种功能.在Linux操作系统中,代码量占操作系统代码总量70%的设备驱动就是作为内核扩展运行的[1].但是内核扩展中存在很大的安全隐患,CVE(Common Vulnerabilities and Exposures)①中报告了大量针对内核扩展的攻击,并且针对内核扩展的攻击仍然在不断出现.因此,提高内核扩展的安全性具有重要意义.

CVE报告指出,大量的攻击和内核函数使用规则相关.内核函数使用规则是指使用一类相互依赖的内核函数需要遵守的规则.例如,Linux操作系统中的自旋锁可以用于保证对临界区的互斥访问,接口函数中有加锁和解锁两类操作,其中"自旋锁不能在未被加锁之前解锁"是一条关于自旋锁的内核函数使用规则.同样,对内存使用相关的函数来说,"被释放的内存不能再进行操作,除非重新被分配"是必须要遵守的一条规则.违反内核函数使用规则会造成整个操作系统的崩溃、机密信息被窃取等严重后果.操作系统的崩溃会带来很大的损失,如表1所示为不同应用1分钟宕机的代价②.而在Linux操作系统中,作为内核扩展运行的设备驱动失效的频率是其它部分的3～7倍[1],在Linux Kernel Mail List③中报告了在内核扩展中存在大量内核函数使用规则相关的错误.

表1 1分钟宕机代价

应用	代价/$
ERP系统	13000
供应链管理系统	11000
电子商务系统	10000
网银系统	7000
电话服务系统	6000

目前已经提出很多方法提高内核扩展的安全性.Mondrix[2]和Loki[3]等基于特殊硬件对内核扩展进行隔离;Minix 3[4]和SUD[5]等提出将设备驱动放到用户态执行;也有一些研究者将内核扩展放到隔离的虚拟机中运行[6-7];SafeDrive[8]和LXFI[9]使用隔离机制来提高在内核态运行的内核扩展的安全性.SafeDrive主要解决内核扩展违反类型安全的问题,LXFI主要针对内核函数完整性进行检测.内核函数完整性检测是针对具体的内核函数接口调用进行的检测,但并没有对内核函数使用规则进行检测.Ball等人[10]在SDV(The Static Driver Verifier tool)中提出了大量需要被检测的内核函数使用规则.

大量提高内核扩展安全性的方法没有被主流操作系统使用的主要原因之一是性能问题.虽然这些方法能够提高内核扩展的安全性,但同时会带来很大的开销.

本文设计了一种新型安全模型系统——KFUR内核扩展安全模型系统,能以很小的开销对内核函数使用规则进行检测,防止针对内核函数使用规则的攻击和程序员编码错误导致的违反内核函数使用规则对操作系统造成的破坏.首先,本文提出了内核函数使用规则的KFUR安全模型,用于判断内核扩展调用内核函数是否遵守内核函数使用规则.基于内核函数使用规则的KFUR安全模型,在Linux操作系统上实现了KFUR内核扩展安全模型系统.该安全模型系统由三部分组成:内核函数使用规则库、检测标记和动态检测器.内核函数使用规则库根据KFUR安全模型实现了需要被检查的内核函数使用规则.在内核扩展代码中插入少量检测标记,在运行时,插入的标记将触发动态检测器的运行.动态检测器根据内核函数使用规则库判断内核扩展对内核函数的调用是否安全.为了提高KFUR内核扩展安全模型系统的性能,设计了快速Hash算法用于存储数据.

本文将KFUR内核扩展安全模型系统运用于e1000网卡驱动、SATA硬盘驱动和HDA声卡驱

① Common Vulnerabilities and Exposures[EB/OL]. http://cve.mitre.org
② 数据来源于清华大学高等计算机体系结构课程讲义
③ Linux Kernel Mail List[EB/OL]. http://www.lkml.org

动内核扩展并采用错误注入的方法进行安全性评测，评测结果表明 KFUR 内核扩展安全模型系统能够有效地进行安全性检测. 分别使用 netperf① 基准测试程序、dd 命令测试和 postmark 基准测试程序、real player 声音播放和 sound recorder 声音录制对 e1000 网卡驱动、SATA 硬盘驱动和 HDA 声卡驱动进行性能评测，评测结果表明 KFUR 内核扩展安全模型系统仅带来很小的开销.

本文第 2 节介绍近年来在提高内核扩展安全性方面的工作；第 3 节对 KFUR 安全模型进行描述；第 4 节对 KFUR 内核扩展安全模型系统的主要组成部分进行详细的描述；第 5 节给出安全性评测结果；第 6 节给出性能评测结果；第 7 节对全文进行简要总结.

2　相关工作

目前已经有很多提高内核扩展安全性的方法，下面将分类进行介绍. 一些对内核扩展进行隔离的方法需要依赖特殊的硬件. Mondrix[2] 基于 Mondriaan 内存保护机制实现了对内核扩展的隔离. Mondriaan 内存保护机制采用对中央处理器的流水线进行修改，对 load，store 指令进行权限检测等方法实现了对多个受保护的内存域共享同一线性地址空间的细粒度内存保护方法. 但是，Mondriaan 内存保护机制所依赖的硬件目前还不存在. Loki[3] 基于标签内存机制实现隔离. 通过使用标签内存机制，Loki 将安全策略和物理内存予以关联，简化了安全机制的实现. 但 Loki 依赖于 HiStar 操作系统的特殊结构，不能被应用于主流实用的操作系统中.

一些研究者采用将内核扩展移到用户态运行的方法来达到提高安全性的目的. Minix 3[4] 操作系统的设备驱动在用户态运行，但其性能与运行在内核态的设备驱动相比较差. Leslie[11] 提出的将设备驱动移到用户态运行的方法可以保证良好的性能，但需要对设备驱动代码进行重写. SUD[5] 采用在用户态模拟 Linux 操作系统内核环境的方法将 Linux 操作系统的设备驱动在不进行修改的情况下移到用户态运行.

有些研究者使用虚拟机提高内核扩展的安全性. Fraser 等人[6] 把内核扩展放到隔离的虚拟机中运行，在保证了安全性的同时可以实现内核扩展被多个操作系统共享. LeVasseur 等人[7] 通过将内核扩展和相应的操作系统同时运行于隔离的虚拟机中来提高安全性并实现内核扩展的复用. 但采用虚拟机技术对内核扩展进行隔离会带来很大的性能开销. TwinDrivers[12] 提出将内核扩展中影响性能的部分放到虚拟机虚拟层的方法提高内核扩展的性能.

有很多使用软件方法实现的隔离机制用于提高内核扩展的安全性. SFI（Software-based Fault Isolation）[13] 隔离机制实现了一种单一地址空间的隔离机制，它将内核扩展的代码和数据加载到逻辑隔离的内存域. SFI 还对内核扩展的代码进行修改，使内核扩展不能写或者跳转到其所在内存域以外的地址. XFI[14] 隔离机制能够提供灵活的访问控制和完整性保证. SafeDrive[8] 通过在编写内核扩展时添加标记，对内核扩展违反类型安全的问题进行检测和恢复. LXFI[9] 通过在内核扩展代码中添加对象访问权限的授予、回收和检测标记，采用介于操作系统和内核扩展之间的隔离机制实现对内核扩展使用内核函数的完整性检测. LXFI 能够对同一内核扩展的不同实例实现不同的隔离.

目前很少有能够对内核函数使用规则进行检测的方法，另外，大多数提高内核扩展安全性的方法都会带来较大的开销. 事实上，在使用内核扩展时，需要对内核函数的访问予以约束，该约束完全可以通过定义规则的形式实现对内核扩展中的函数使用进行检测和约束. 鉴于此，本文设计和实现了 KFUR 内核扩展安全模型系统，用于对内核函数使用规则进行检测. 在设计安全模型系统时充分考虑了性能问题.

3　内核函数使用规则 KFUR 安全模型

这一节首先对 KFUR 安全模型进行描述，然后基于该模型对内核函数使用规则进行建模.

3.1　KFUR 安全模型

KFUR 安全模型由访问主体、系统状态、访问客体、初始状态、不安全状态、转移条件和转移函数 7 个部分组成. 下面将分别对这 7 个部分进行描述.

访问主体：是指提出访问请求的实体，使用符号 s 表示，访问主体的集合使用符号 S 表示.

系统状态：是指 KFUR 模型在某一时刻的快照，使用符号 q 表示，系统状态的集合使用符号 Q 表示，系统初始状态使用符号 q_0 表示，不安全状态

① Netperf：A network performance benchmark，version 2.50 [EB/OL]. http://www.netperf.org 2011，5，12

使用符号 q_e 表示.

访问客体:是指接受访问主体访问的实体,使用符号 o 表示,访问客体的集合使用符号 O 表示.访问客体中包括一个或多个访问属性,访问属性为在某一系统状态下访问客体允许被访问的方式,使用符号 p 表示.访问方式用符号 m 表示.访问客体的访问属性可以表示为

$$p(o)=\{(q_0,m_0),(q_1,m_1)\cdots(q_n,m_n)\}.$$

转移条件:是指发生状态转移的条件,使用符号 a 表示,转移条件的集合使用符号 A 表示.

转移函数:是指发生状态转移时模型发生的变化,使用符号 f 表示,转移函数的集合使用符号 F 表示.转移函数的输入和输出为访问客体和系统状态,可以描述为

$$(q_n,o_1,\cdots,o_n)\times f\rightarrow(q_{n+1},o_1',\cdots,o_n'),$$

其中 q_n 为模型当前处于的状态,当某个转移条件发生时,触发相应的转移函数对模型进行改变:一是对模型进行状态转移,即由 q_n 转化为 q_{n+1},二是对访问客体的访问属性进行变更,即由 o_1 转化为 o_1' 等.

转移函数分为两类:满足安全要求的转移函数和不满足安全要求的转移函数.满足安全要求的转移函数可以使模型转移到系统的安全状态,不满足安全要求的转移函数将使模型转移到系统的不安全状态.

这样,KFUR 模型可以表示为七元组:

$$(S,Q,O,q_0,q_e,A,F).$$

KFUR 模型安全运行的充分必要条件为:不存在状态转移使模型进入不安全状态.如果状态转移始终发生在安全状态之间,则该模型安全运行,如果模型状态转移到了不安全状态,则表示有违反安全性的行为发生.

3.2 内核函数使用规则的 KFUR 安全模型

内核函数使用规则 KFUR 安全模型的访问主体为所有调用内核函数的内核扩展;访问客体为内核函数使用规则中涉及的内核函数、变量和内存区域,访问方式主要有 3 种:只读、可写和可调用;系统状态为内核函数使用规则中的状态;转移条件为对内核函数使用规则中涉及的内核函数进行调用;转移函数即为内核函数使用规则中的规则,即判断当前状态能否进行相应的操作,如果允许,则转移到另一个安全状态;如果规则不允许,则转移到不安全状态.

下面以自旋锁为例对内核函数使用规则的 KFUR 安全模型进行描述.自旋锁的使用规则有:(1)自旋锁在使用前必须进行初始化;(2)自旋锁不能在未被加锁之前解锁.如图 2 所示为 *e1000_read_eeprom* 函数的源代码,代码中使用了 *e1000_eeprom_lock* 自旋锁,在调用函数 *e1000_do_read_eeprom* 之前调用 *spin_lock* 加锁函数进行加锁,在 *e1000_do_read_eeprom* 函数执行完毕后调用 *spin_unlock* 解锁函数解锁.

```
1. static DEFINE_SPINLOCK(e1000_eeprom_lock);
2. s32 e1000_read_eeprom(struct e1000_hw *hw, u16 offset,
                         u16 words, u16 *data)
3. {
4.     s32 ret;
5.     spin_lock(&e1000_eeprom_lock);
6.     ret=e1000_do_read_eeprom(hw, offset, words, data);
7.     spin_unlock(&e1000_eeprom_lock);
8.     return ret;
9. }
```

图 2 *e1000_read_eeprom* 函数

在第 5 行代码执行完毕时自旋锁 *e1000_eeprom_lock* 处于加锁状态,自旋锁内核函数使用规则的 KFUR 安全模型如下,其中只读访问方式使用符号 r 表示,可写访问方式使用符号 w 表示,可调用访问方式使用符号 c 表示.

$S=\{$e1000 网卡驱动$\}$;

$O=\{$DEFINE_SPINLOCK, *spin_lock*, *spin_unlock*, *e1000_eeprom_lock*$\}$;

$Q=\{0,1,2,3,4\}$;

$p($DEFINE_SPINLOCK$)=\{(0,c)\}$;

$p(spin\_lock)=\{(1,c),(2,c),(3,c)\}$;

$p(spin\_unlock)=\{(2,c)\}$;

$p(e1000\_eeprom\_lock)=\{(2,w)\}$;

$q_0=\{0\}$;

$q_e=\{4\}$;

$A=\{$call DEFINE_SPINLOCK; call *spin_lock*; call *spin_unlock*$\}$;

$F=\{(0,$DEFINE_SPINLOCK, $e1000\_eeprom\_lock\{0,w\})\times f\rightarrow$
$(1,e1000\_eeprom\_lock\{1,w\})$,
$(1,spin\_lock,e1000\_eeprom\_lock\{1,w\})\times f\rightarrow$
$(2,e1000\_eeprom\_lock\{2,w\})$,
$(2,spin\_lock,e1000\_eeprom\_lock\{2,w\})\times f\rightarrow$
$(2,e1000\_eeprom\_lock\{2,w\})$,
$(2,spin\_unlock,e1000\_eeprom\_lock\{2,w\})\times f\rightarrow$
$(3,e1000\_eeprom\_lock\{3,w\})$,
$(3,spin\_lock,e1000\_eeprom\_lock\{3,w\})\times f\rightarrow$
$(2,e1000\_eeprom\_lock\{2,w\})$,
$(0,spin\_lock,e1000\_eeprom\_lock)\times f\rightarrow$

$(4,e1000\_eeprom\_lock)$,
$(0,spin\_unlock,e1000\_eeprom\_lock)\times f\rightarrow$
$(4,e1000\_eeprom\_lock)$,
$(1,spin\_unlock,e1000\_eeprom\_lock)\times f\rightarrow$
$(4,e1000\_eeprom\_lock)$,
$(3,spin\_unlock,e1000\_eeprom\_lock)\times f\rightarrow$
$(4,e1000\_eeprom\_lock)\}$.

对于如上模型的转移函数，仅标出了访问客体发生变化的访问方式. 该模型安全运行的条件是不存在某次状态转移使安全模型转移到状态 4.

4　KFUR 内核扩展安全模型系统

如图 3 所示，KFUR 内核扩展安全模型系统由三部分组成：内核函数使用规则库、检测标记和动态检测器. 内核函数使用规则库中包含被检查内核函数使用规则的 KFUR 安全模型描述，作为动态检测器进行检测的依据. 检测标记为插入到内核扩展代码中的标记，在运行时，检测标记触发动态检测器进行检测. 如果发现异常，则通知操作系统内核终止内核扩展的运行.

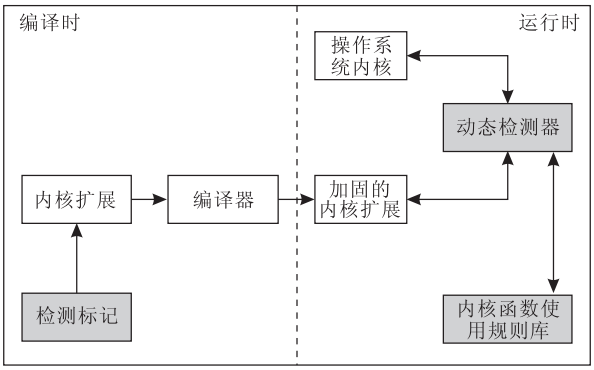


图 3　KFUR 内核扩展安全模型系统结构

本文在 Linux 操作系统上实现了 KFUR 内核扩展安全模型系统. 4.1 节对内核函数使用规则库进行介绍，4.2 节对检测标记进行介绍，4.3 节对动态检测器进行介绍.

4.1　内核函数使用规则库

内核函数使用规则库中的内核函数使用规则基于 KFUR 模型实现. 为了提升访问内核函数使用规则 KFUR 模型中访问方式相关数据的性能，本文设计了快速 Hash 算法，用来存储访问方式相关数据. 访问客体分为两类：一类是变量和函数，另一类是内存区域. 如果访问客体为变量或函数，则采用一种简单的 Hash 算法存储. 假定变量或函数的地址为 A，Hash 表大小为 N，Hash 值为 K，则该 Hash 算法的 Hash 函数如下：

$$K=A\%N.$$

Hash 值即为变量地址或函数地址在哈希表中的位置. 为了处理发生碰撞即该位置已经存有其它变量地址或函数地址的情况，哈希算法为每个位置开辟了一个小的碰撞处理表，变量地址或函数地址存放在碰撞处理表中的位置采用和如上哈希函数类似的计算方法，即用变量或函数的地址和碰撞处理表的大小做模运算. 如果该位置仍然存在冲突，从该位置起依次查询是否有空位，直到找到空位为止. 如果在某一位置的碰撞处理表中没有空位，则到哈希表下一位置的碰撞处理表中寻找空位. 该算法能够实现对变量或函数地址的快速添加、查询和删除.

在实际应用中，存在大量查询内存区域子区间的访问方式的情况，比如需要对某个结构体中的某一项是否允许被访问进行查询. 例如在表 2 中存放了一些内存区域的信息，内存区域的信息由内存起始地址和内存大小两部分构成. 在实际应用中存在如表 3 所示的查询. 例如表 3 中第 1 项为对内存起始地址为 2155，内存大小为 3 的内存区域的访问方式进行查询，可以看到表 2 的第 1 项为内存起始地址为 2140，内存大小为 33 的内存区域，因此表 3 中第 1 项查询的内存区域应当允许被访问. 针对这种特定的场景，本文设计了一种新型的快速哈希算法进行存储. 该哈希算法的哈希函数为：首先得到内存区域的大小为几位数，假设为 n 位数，内存起始地址为 A_s，然后按照如下公式计算哈希值：

$$K=(A_s/10^n)\%N.$$

表 2　Hash 表中存放的内存区域信息

	内存起始地址	内存大小
1	2140	33
2	3458	45
3	5467	20
…	…	…

表 3　内存区域查询信息

	内存起始地址	内存大小
1	2155	3
2	3469	5
3	5477	8
…	…	…

哈希值即为内存区域在哈希表中的位置. 如果该位置已经存储了其它内存区域，采用如算法 1 所示方法进行碰撞处理.

算法 1. 内存区域存储碰撞处理算法.

输入:内存区域起始地址,内存区域大小

1. 初始化循环变量 $i \leftarrow 1$;

2. 计算 $A_s/10^{(n+i)}$ 的值,并判断该值是否为 0,如果不为 0,执行下一步;如果为 0,跳转到步 4;

3. 内存区域存储的位置为:$(A_s/10^{(n+i)})\%N$,如果 Hash 表中该位置为空,则将内存区域存储到该位置,算法执行结束;否则跳转到步 5;

4. 从位置 $(A_s/10^{(n+i-1)}+1)$ 开始顺序查找 Hash 表,如果 Hash 表中该位置为空,则将内存区域存储到该位置,算法执行结束,如果找不到空位置说明 Hash 表已满报错;

5. $i \leftarrow i+1$.

4.2 检测标记

在内核扩展中需要添加的检测标记主要有两类:一类是注册和初始化标记,即在待检测变量定义时,添加的向动态检测器注册和初始化该待检测变量的标记;另一类是检测标记,即在内核扩展调用内核函数前,添加的触发动态检测器进行检测的标记.第一类标记需要将待检测变量、待检测变量所要遵守的内核函数使用规则作为参数传递给动态检测器;第二类标记需要将待检测变量、待调用函数、所要遵守的内核函数使用规则作为参数传递给动态检测器.检测标记的添加可以由编译器来完成.

仍然以 e1000 网卡驱动 *e1000_read_eeprom* 函数中使用的 *e1000_eeprom_lock* 自旋锁为例进行说明.从图 2 中可以看到,自旋锁 *e1000_eeprom_lock* 作为全局变量被定义和初始化.向动态检测器注册和初始化待检测变量 *e1000_eeprom_lock* 自旋锁的标记被添加到 e1000 网卡驱动的初始化函数 *e1000_probe* 中,自旋锁 *e1000_eeprom_lock*、自旋锁应当遵守的内核函数使用规则作为函数参数传递给动态检测器.在 *e1000_read_eeprom* 函数中调用 *spin_lock* 加锁函数之前插入检测标记,将自旋锁 *e1000_eeprom_lock*、函数 *spin_lock* 和自旋锁应当遵守的内核函数使用规则作为函数参数进行传递,同样也在调用 *spin_unlock* 函数之前插入检测标记.

4.3 动态检测器

动态检测器在运行时对内核扩展进行检测.注册和初始化标记触发动态检测器完成对需要被检测的变量进行注册和初始化,动态检测器将该变量作为访问客体进行存储.例如,在对待检测的自旋锁进行注册和初始化后,该自旋锁作为访问客体具有在状态 1 可写的访问属性.检测标记根据内核函数使用规则对当前函数调用是否正确进行检测.如果对内核函数调用正确,则允许该内核函数运行,同时完成 KFUR 模型的状态转移.例如,被初始化的自旋锁在调用加锁函数 *spin_lock* 后,自旋锁处于 KFUR 模型的状态 2,在调用解锁函数 *spin_unlock* 后,自旋锁处于内核函数使用规则 KFUR 模型的状态 3.如果在某状态调用不允许调用的函数或访问不允许访问的内存时,动态检测器会向操作系统内核报错,操作系统内核终止该内核扩展的运行.例如,如果在状态 3 时调用 *spin_unlock* 函数,自旋锁将会转移到 KFUR 模型的状态 4,即到达了不安全状态,因为这一调用违反了"自旋锁不能在未被加锁之前解锁"的自旋锁使用规则.

5 安全性评测

本节对 KFUR 内核扩展安全模型系统进行安全性评测.5.1 节采用错误注入测试的方法对系统进行评测,5.2 节将 KFUR 内核扩展安全模型系统和 SDV 进行比较.

5.1 错误注入测试

为了对 KFUR 内核扩展安全模型系统的检测能力进行评测,本文采用错误注入测试方法.其中,Linux 内核版本为 2.6.36.1,GCC 编译器的版本是 4.3,选择 e1000 网卡驱动、SATA 硬盘驱动和 HDA 声卡驱动内核扩展分别进行测试.

对于 e1000 网卡驱动,以自旋锁、内存和网络数据包处理三类内核函数使用规则进行测试.首先选择自旋锁进行测试.如上文所述,函数 *e1000_read_eeprom* 中配对使用了自旋锁加锁函数 *spin_lock* 和自旋锁释放函数 *spin_unlock*.将函数调用 *spin_lock* 删除,这样就违反了自旋锁不能在未被加锁之前解锁的使用规则.如果 KFUR 内核扩展安全模型系统正常运行,应当能够向操作系统内核进行错误报告,并且 e1000 网卡驱动内核扩展的运行将被终止.动态检测器采用向 Linux 操作系统的日志记录文件输出错误信息的方式进行报错.运行 e1000 网卡驱动,在 Linux 操作系统日志记录文件中找到了动态检测器输出的错误信息,并且 e1000 网卡驱动内核扩展的运行被终止.

Linux 操作系统内核提供给内核扩展的内存函数接口有内存分配函数 *malloc*、在一段内存中填充某个给定值的函数 *memset*、内存区域拷贝函数 *memcpy*、内存释放函数 *free* 等.在使用内存函数接口时,被 *free* 函数释放的内存不能成为 *memset* 和 *memcpy* 函数的参数.

e1000 网卡驱动函数 *e1000_setup_tx_resources* 中配对使用了内存分配函数 *vmalloc* 和内存释放函数 *vfree*. 在 *vfree* 函数后添加 *memset* 函数填充被释放的内存. 运行 e1000 网卡驱动，同样可以在 Linux 操作系统的日志记录文件中找到动态检测器输出的错误信息.

Linux 操作系统的网络模块为内核扩展提供了很多接口函数，如图 4 所示. 其中，前两个函数用于分配 sk_buff 结构体，第 3 个到第 6 个函数用于释放 sk_buff 结构体，这 4 个释放函数在不同情况下被使用. 其它函数用于对 sk_buff 结构体进行不同的操作. 因此，这些函数不能在 sk_buff 结构体被释放后使用. 另外，函数 *skb_pull* 用于从数据包的头部删除数据. 因此，这个函数不应当出现在 e1000 网卡驱动内核扩展中.

```
1. struct sk_buff *alloc_skb(unsigned int len, int priority);
2. struct sk_buff *dev_alloc_skb(unsigned int len);
3. void kfree_skb(struct sk_buff *skb);
4. void dev_kfree_skb(struct sk_buff *skb);
5. void dev_kfree_skb_irq(struct sk_buff *skb);
6. void dev_kfree_skb_any(struct sk_buff *skb);
7. void *skb_put(struct sk_buff *skb, int len);
8. unsigned char *skb_push(struct sk_buff *skb, int len);
9. int skb_tailroom(struct sk_buff *skb);
10. int skb_headroom(struct sk_buff *skb);
11. void skb_reserve(struct sk_buff *skb, int len);
12. unsigned char *skb_pull(struct sk_buff *skb, int len);
13. int skb_is_nonlinear(struct sk_buff *skb);
14. int skb_headlen(struct sk_buff *skb);
```

图 4　网络数据包处理函数接口

数据包发送函数 *e1000_xmit_frame* 使用了 sk_buff 结构体. 在变量定义后，调用 *dev_kfree_skb_any* 释放 sk_buff 结构体，然后调用 *skb_transport_offset* 访问被释放的 sk_buff 结构体. 在运行时动态检测器成功检测出了异常并报告给了 linux 操作系统内核. 在 *e1000_xmit_frame* 函数中调用 *skb_pull* 函数，同样也被动态检测器成功检测.

除对 e1000 网卡驱动进行上述针对性注入测试外，还进行了其它一些针对性注入测试，测试结果如表 4 所示. 从表 4 可以看出，KFUR 内核扩展安全模型系统能够检测出所有被注入的错误. 另外，对于 e1000 网卡驱动还进行了随机测试. 每次测试随机对一些内核函数调用进行删除或随机添加一些内核函数调用，导致内核函数使用规则被违反. 从表 4 可以看出，KFUR 内核扩展安全模型系统每次都能成功检测.

表 4　错误注入测试结果

设备	测试方法	测试用例个数	成功检测个数
e1000 网卡	针对性错误注入	30	30
	随机错误注入	30	30
SATA 硬盘	针对性错误注入	20	20
	随机错误注入	20	20
HDA 声卡	针对性错误注入	25	25
	随机错误注入	25	25

SATA 硬盘驱动中存在大量自旋锁相关的内核函数调用，采用和 e1000 网卡驱动类似的方法进行了安全性评测. 对于 HDA 声卡驱动，对自旋锁、内存进行了安全性评测，评测结果如表 4 所示. 实验结果表明 KFUR 内核扩展安全模型系统均能成功检测错误.

5.2　相关技术比较

SDV 采用模型检测方法对 Windows 操作系统的设备驱动是否遵守内核函数使用规则进行检测，并找出了很多 Windows 操作系统设备驱动违反内核函数使用规则的问题. KFUR 内核扩展安全模型系统在如下两方面优于 SDV. 首先，SDV 是一种静态检测方法而 KFUR 内核扩展安全模型系统是动态检测方法. SDV 仅能检测出静态存在于设备驱动中违反内核函数使用规则的情况，而对由于系统运行时的安全攻击造成的违反内核函数使用规则的情况则无法检测. KFUR 内核扩展安全模型系统能够动态实时检测违反内核函数使用规则的情况并及时向操作系统内核报告，进行相应的处理，保证了操作系统运行时的安全. 其次，SDV 采用模型检测技术. 现有模型检测技术时间开销较大，对于代码量大的设备驱动无法在有限的时间内进行检测，而 KFUR 内核扩展安全模型系统仅需要很小的开销.

6　性能评测

同样采用 e1000 网卡驱动、SATA 硬盘驱动和 HDA 声卡驱动内核扩展进行性能评测. 性能评测用于评价 KFUR 内核扩展安全模型系统对内核扩展性能的影响.

本文采用 netperf 基准测试程序对 e1000 网卡驱动的性能进行评测. 实验环境包括一台客户端计算机和一台服务器计算机及千兆网络. 客户端计算机和服务器计算机的配置如表 5 所示，KFUR 内核扩展安全模型系统实现在客户端计算机中.

表 5 性能评测环境

	客户端	服务器端
CPU	Intel(R) Core(TM)2 Duo CPU E6750 2.66GHz	Intel(R) Xeon(R) CPU E5620 2.40GHz
内存	2GB DDR2	12GB DDR3
硬盘	320GB 7200r/min	1TB 7200r/min
网卡	Intel 82540EM Gigabit Ethernet Card	Intel 82540EM Gigabit Ethernet Card

选择 netperf 基准测试程序中的 TCP_STREAM、UDP_STREAM、TCP_RR 和 UDP_RR 四种测试分别对原始 Linux 操作系统和添加 KFUR 内核扩展安全模型系统后的 Linux 操作系统进行测试. TCP_STREAM 用于对 TCP 的吞吐量进行测试;UDP_STREAM 用于对 UDP 的吞吐量进行测试;TCP_RR 和 UDP_RR 用于测试 KFUR 内核扩展安全模型系统所带来的延迟. 测试时间均为 10s.

TCP_STREAM 的发送缓冲区大小为 16384 字节,接收缓冲区大小为 87380 字节. 每条消息的大小为 16384 字节. 在 UDP_STREAM 测试中,发送端 UDP 套接字的大小为 114688 字节,接收端 UDP 套接字的大小为 114688 字节,每条消息的大小为 1024 字节. 图 5 和图 6 分别为 TCP_STREAM、UDP_STREAM 吞吐量和 CPU 占用率的测试结果. 图中"Linux"表示原始 Linux 操作系统的测试结果,"KFUR"表示添加 KFUR 内核扩展安全模型系统后的 Linux 操作系统的测试结果.

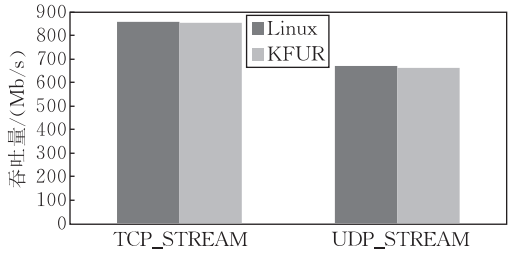


图 5 TCP_STREAM 和 UDP_STREAM 测试的吞吐量

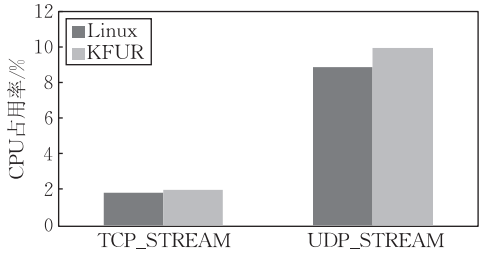


图 6 TCP_STREAM 和 UDP_STREAM 测试的 CPU 占用率

对于 TCP_STREAM 测试,采用 KFUR 内核扩展安全模型系统和不采用 KFUR 内核扩展安全模型系统的吞吐量基本相同,采用 KFUR 内核扩展安全模型系统的 CPU 占用率仅比不采用 KFUR 内核扩展安全模型系统高 0.15%.

UDP_STREAM 和 TCP_STREAM 测试的结果类似,采用 KFUR 内核扩展安全模型系统和不采用 KFUR 内核扩展安全模型系统的吞吐量基本相同,采用 KFUR 内核扩展安全模型系统的 CPU 占用率比不采用 KFUR 内核扩展安全模型系统的 CPU 占用率高 1.04%.

在 TCP_RR 测试和 UDP_RR 测试中,发送缓冲区的大小为 16384 字节,接收缓冲区的大小为 87380 字节. 采用 KFUR 内核扩展安全模型系统时吞吐量略低于不采用 KFUR 内核扩展安全模型系统,CPU 占用率略高于不采用 KFUR 内核扩展安全模型系统,测试结果分别如图 7 和图 8 所示.

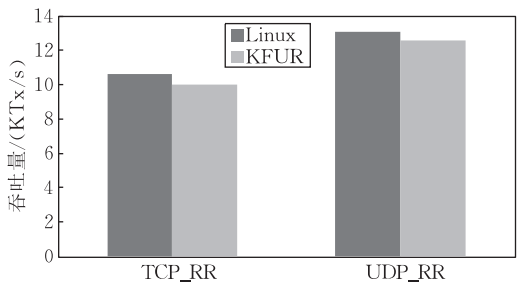


图 7 TCP_RR 和 UDP_RR 测试的吞吐量

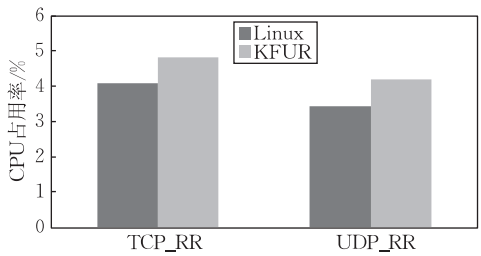


图 8 TCP_RR 和 UDP_RR 测试的 CPU 占用率

对添加了 KFUR 内核扩展安全模型系统的硬盘驱动和未修改的硬盘驱动的性能进行比较,首先使用 dd 命令对硬盘读速度、硬盘写速度和硬盘读写混合速度进行测试,测试结果如表 6 所示.

表 6 dd 命令测试结果

	Linux(MB/s)	KFUR(MB/s)
硬盘读速度	82.8	81.7
硬盘写速度	83.2	81.9
硬盘读写速度	32.8	31.9

采用 postmark 基准测试程序对多个文件并发访问进行测试. 设置文件大小下限为 10000 字节,文件大小上限为 20000 字节,事务数为 50000 次,

并发文件数从 1000 到 2000 次，文件读取速度和写入速度分别如图 9 和图 10 所示. 可以看出，加入了 KFUR 内核扩展安全模型系统的硬盘驱动和未修改的硬盘驱动相比性能相差不大.

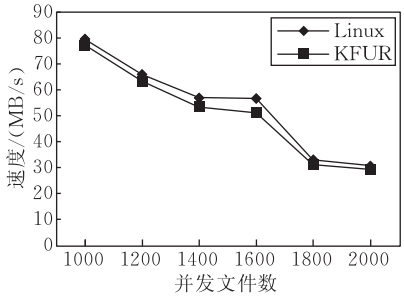


图 9　postmark 文件读取速度测试结果

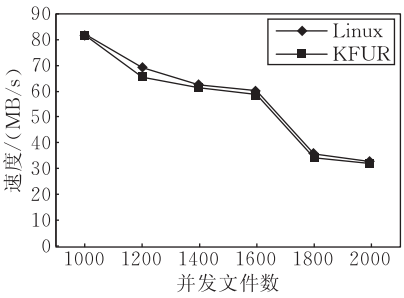


图 10　postmark 文件写入速度测试结果

对添加了 KFUR 内核扩展安全模型系统的声卡驱动和原始声卡驱动进行性能比较，采用 real player 和 sound recorder 分别进行声音的播放和录制，比较其 CPU 占用率，实验结果如图 11 所示.

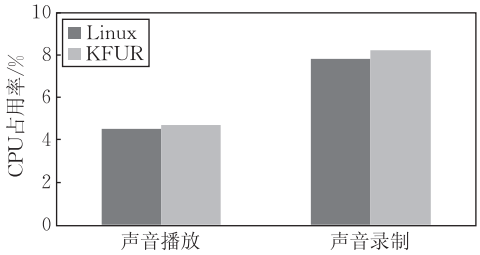


图 11　声音播放和声音录制的 CPU 占用率

从网卡驱动、硬盘驱动和声卡驱动的实验结果可以看出，KFUR 内核扩展安全模型系统对于内核扩展的性能影响不大. KFUR 内核扩展安全模型系统能在保持性能良好的情况下提高内核扩展的安全性.

7　总　结

内核扩展的安全性对于操作系统十分重要，不安全的内核扩展导致操作系统内核崩溃、机密数据被窃取，带来了巨大的损失. 本文提出 KFUR 内核扩展安全模型系统，该安全模型系统能以很小的开销对内核扩展调用内核函数时，是否遵守内核函数使用规则进行检测，从而提高内核扩展的安全性. 接下来将对内核扩展的其它安全性问题进行研究，提出实用的、高性能的提高内核扩展安全性的方法，进一步提高操作系统的安全性.

致　谢　本文作者得到了清华大学计算机科学与技术系操作系统实验室的老师和同学们的许多帮助和建议，在此表示感谢. 感谢审稿人对本文提出宝贵意见和建议！

参　考　文　献

[1] Chou Andy, Yang Junfeng, Chelf Benjamin, Hallem Seth, Engler Dawson. An empirical study of operating system errors//Proceedings of the 18th ACM Symposium on Operating Systems Principles. Chateau Lake Louise, Canada, 2001: 73-88

[2] Witchel E, Rhee J, Asanovic K. Mondrix: Memory isolation for Linux using Mondriaan memory protection//Proceedings of the 20th ACM Symposium on Operating Systems Principles. Brighton, UK, 2005: 1-14

[3] Zeldovich N, Kannan H, Dalton M, Kozyrakis C. Hardware enforcement of application security policies//Proceedings of the 8th Symposium on Operating Systems Design and Implementation. San Diego, USA, 2008: 225-240

[4] Herder J H, Bos H, Gras B, Homburg P, Tanenbaum A S. Failure resilience for device drivers//Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Edinburgh, UK, 2007: 41-50

[5] Boyd-Wickizer S, Zeldovich N. Tolerating malicious device drivers in Linux//Proceedings of the 2010 USENIX Annual Technical Conference. Boston, USA, 2010: 117-130

[6] Fraser K, Hand S, Neugebauer R, Pratt I, Warfield A, Williamson M. Safe hardware access with the Xen virtual machine monitor//Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure. Boston, USA, 2004: 1-10

[7] LeVasseur J, Uhlig V, Stoess J, Gotz S. Unmodified device driver reuse and improved system dependability via virtual machines//Proceedings of the 6th Symposium on Operating Systems Design and Implementation. Massachusetts, USA, 2004: 17-30

[8] Erlingsson U'lfar, Abadi M, Vrable M, Budiu M, Necula G C. SafeDrive: Safe and recoverable extensions using language-based techniques//Proceedings of the 7th USENIX Conference on Operating System Design and Implementation. Seattle, Washington, USA, 2006: 45-60

[9] Mao Yan-Dong, Chen Hao-Gang, Zhou Dong, Wang Xi, Zeldovich Nickolai, Kaashoek M Frans. Software fault isolation with API integrity and multi-principal modules//Proceedings of the ACM Symposium on Operating Systems Principles. Cascais, Portugal, 2011: 115-128

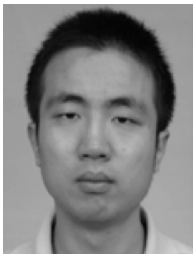
[10] Ball T, Bounimova E, Cook B, Levin V, Lichtenberg J et al. Through static analysis of device drivers//Proceedings of the 1st ACM SIGOPS/Eurosys European Conference on Computer Systems. Leuven, Belgium, 2006: 73-85

[11] Leslie B, Chubb P, Fitzroy-Dale N, Gotz S et al. User-level device drivers: Achieved performance. Journal of Computer Science and Technology, 2005, 20(5): 654-664

[12] Menon A, Schubert S, Zwaenepoel W. TwinDrivers: Semi-automatic derivation of fast and safe hypervisor network drivers from guest os drivers//Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2009: 301-312

[13] Wahbe R, Lucco S, Anderson T E, Graham S L. Efficient software-based fault isolation//Proceedings of the 14th ACM Symposium on Operating Systems Principles. Asheville, USA, 1993: 203-216

[14] Erlingsson U'lfar, Abadi M, Vrable M, Budiu M, Necula G C. XFI: Software guards for system address spaces//Proceedings of the 7th USENIX Conference on Operating System Design and Implementation. Seattle, USA, 2006: 75-88



MA Chao, born in 1986, Ph. D. candidate. His research interests include operating systems, reliability and security of kernel extensions and deterministic multithreading.

YIN Jie, born in 1987, Ph. D. candidate. His research interest is operating systems.

LIU Hu-Qiu, born in 1989, M. S. candidate. His research interests include operating systems and reliability and security of kernel extensions.

LI Hao, born in 1989, bachelor. His research interest is operating systems.

Background

This paper focuses on security of kernel extensions, which is an important problem of operating systems. Lots of researches have been done to solve this problem and published in OSDI, SOSP and Eurosys and so on, which are famous conferences on operating systems. By isolating kernel extensions with special hardware, moving kernel extensions out of operating system kernels, isolating kernel extensions with virtual machines and other software approaches, they achieve the goal of improving security. However, these existing methods often bring large cost. It is the reason why these approaches can not be used in commodity operating systems. In addition, few papers focus on kernel function usage rules checking, which is significant. We propose an innovative KFUR kernel extension security model system, which could check whether kernel extensions violate using rules at runtime. The KFUR kernel extension security model system is implemented in the Linux operating system and used in device drivers of the e1000 network card, the SATA hard disk and the HDA sound card. Evaluations show that this system does solve the problem with low cost. This research is part of the research on the network operating system for supporting public and enterprise service. This operating system can run on several hardware platforms and realize large scale computation, storage and network resources management and on-demanding scheduling. It supports multi-tenant service and large scale data accessing and computing with uniform security management framework and operator interfaces. It will be used in Tencent Incorporated and Shenzhen Huawei Technologies Co. Ltd. In the research area of reliability and security of kernel extensions, we have already published a paper in Science China, which is a SCI journal and a paper in the China National Computer Conference 2011. A paper has been submitted to IEEE transactions on computers. Besides that, four national patents have been submitted to State Intellectual Property Office of the People's Republic of China. Our research solves reliability and security problems of kernel extensions, which is the base of the network operating system reliability and security. This work is supported by the National High Technology Research and Development Program (863 Program) of China (No. 2011AA01A203).