

移动智能终端中操作系统安全监控研究及实现

曹 波, 杨 杉, 梁新建

(国网湖北省电力公司信息通信公司, 湖北 武汉 430077)

摘 要: 提出了一种基于 mini-PCI 卡和语义不变式的移动智能终端操作系统内核安全监控方法: 在仅仅初始安装了操作系统的机器上, 通过学习获取操作系统运行过程中必须保持的语义不变式集合, 将语义不变式集合作为内核运行的安全规范. 然后, 在 FPGA 快速开发原型实验板上, 利用 mini-PCI 卡监控针对操作系统内核的每一次写操作. 一旦发现有内核写操作, 则将本次写操作(通过变换之后)与语义不变式集合作对比. 如果与语义不变式集合中的某一个条目相符, 则可以确认本次写操作是可信的, 系统不做任何处理; 如果不能与语义不变式集合中的任意一条项目相符, 则可以判定本次写操作是不可信的, 系统报警. 实验证明了有效性.

关键词: 移动智能终端; 操作系统安全; 语义不变式

中图分类号: TP309

文献标识码: A

文章编号: 1000-7180(2014)10-0113-04

Design and Implementation of Operating System Security Monitoring of Mobile Intelligence Terminal

CAO Bo, YANG Shan, LIANG Xin-jian

(Information & Communication Branch of Hubei Electric Power Company, Wuhan 430077, China)

Abstract: A mini-PCI card and semantic invariants-based method for monitoring operating system security of mobile intelligence terminal is introduced; Firstly, inferring the semantic invariants of the operating system in the training period, and all of the semantic invariants will be used as security specifications of operating system during runtime; secondly, in the fast FPGA prototyping experiment board, a mini-PCI card is designed to monitoring modifications to kernel, and any violation against the inferred semantic invariants will be treated as an attack.

Key words: mobile intelligence; operating system security; semantic invariants

1 引言

目前, 移动智能终端已经深入到人们生活的方方面面. 由于是智能终端就离不开操作系统的支持. 操作系统作为移动智能终端最底层的系统软件, 且对上层虚拟机、应用程序、进程间通信等提供各类服务, 保证操作系统的安全性是至关重要的. 因此, 对移动智能终端的操作系统安全进行研究, 具有重要的意义. 2014 年国家“八六三”计划指南明确指出, 要研究智能终端系统的内核安全增强技术.

当前移动智能终端采用最有代表性的两类操作系统是 Android 和 iOS, 其中 Android 占据了市场

的主导地位, 本文选择对 Android 进行研究.

在操作系统在运行过程中, 其内核会被不断地被改写, 而操作系统被破坏, 本质上正是由于恶意代码对内核数据的控制流或者数据流非法修改造成的. 与此同时, rootkit 之类的恶意代码也会对内核进行改写, 以达到攻击目的^[1-2]. 因此, 要保证操作系统内核的可信性, 就必须从各类形式复杂的、随时动态生成的写操作中, 判定哪些写操作是系统正常运行所产生的, 哪些是恶意代码恶意攻击所发出的, 并根据不同的情形允许或者拒绝写操作的进行, 从而达到保护目的.

针对上述问题, 本文提出了一种基于内核不变

收稿日期: 2013-12-03; 修回日期: 2014-01-28

基金项目: 智能电网移动设备信息安全督查技术研究(521520130217)

式的操作系统内核安全监控方法.

2 恶意软件触发行为分析

2.1 基本思想

由于 Linux 操作系统已经位于移动智能终端的最底层,因此若要对其进行保护,就必须从操作系统的更低一层入手,即:采用额外硬件的方式^[3],或采用虚拟机的方式^[4],通过“独立”于移动智能终端之外的软件或者硬件来对其内核进行安全监控和管理.本文当中,采用额外硬件的方式.我们基于 FPGA 实现相对简单的快速原型开发,设计并实现了一个 PCI 卡(mini-PCI 卡,以下简称 PCI 卡),利用 PCI 卡来对移动智能终端的操作系统内核进行监控.图 1 给出了基本架构.

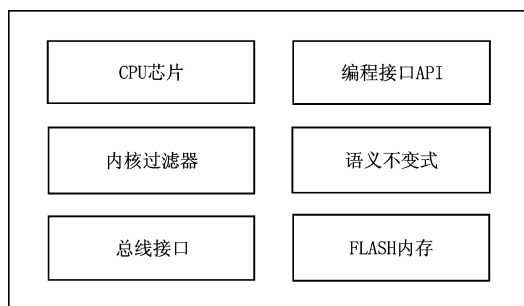


图 1 mini-PCI 卡架构

上述 PCI 卡本质上是一个片上系统 SoC(System on Chip). 其中,总线接口用来接收系统总线发过来的针对内核写操作的“原始数据拷贝”:在我们的设计中,FPGA 上的特殊电路使得在任何针对内核的写操作发生时,从系统总线发送一份写操作拷贝给 PCI 卡.得到写操作拷贝之后,PCI 卡上的 CPU 芯片对其进行预处理变换,然后将预处理后的数据传送给内核过滤器和语义不变式模块进行判定:如果写操作符合内核过滤器或者语义不变式中的任何一个条目,则该写操作是合法的;否则是不合法的.图 1 中的编程接口 API 用来从外部对 PCI 卡进行配置和管理,以增强系统的易用性.FASH 用来作为运算载体,并存储必要的信息(如配置信息、核符号表、内核数据结构信息等).

需要指出的是,要监控并管理操作系统安全只有两种方式:采用平台硬件^[3]或者虚拟机^[4].显然,采用虚拟机会不可避免地会影响到移动智能终端的实时运行效率,在移动智能终端的 CPU、内存、功耗和电源受限的情况下,这种影响更加明显.同时,若采用虚拟机,底层硬件也必须做相应改动以适应虚拟机.因此,本文选择直接采用额外硬件的方式进行

处理.实际上,结合硬件进行系统安全增强不仅可行而且很常见^[5].

在 PCI 卡的基础上,整个系统的运行流程如图 2 所示.

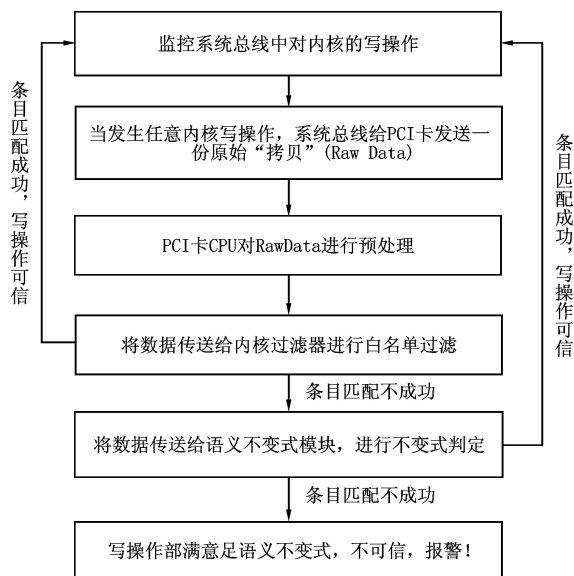


图 2 系统工作流程

(1)监控系统总线对内核的写操作,并发送原始数据“拷贝”Raw Data.

当系统启动之后,FPGA 电路不断监控系统总线上对内核的写操作.一旦有写操作发生,FPGA 通过特殊的电路将写操作发送一份“拷贝”给 PCI 卡.注意这份拷贝是来自系统总线的原始写操作信息,因此又称为 Raw Data. PCI 卡在收到 Raw Data 之后,将其变换为内核数据结构的形式.与 Raw Data 相对应,变换后的数据称为 Refined Data.

(2)基于语义不变式判定写操作是否可信.

得到 Refined Data 之后,CPU 首先将其发送给内核过滤器进行白名单过滤.内核过滤器类似于一个白名单系统,其中的每一个条目类似于 $\langle \text{addr}, \text{object_name}, \text{value} \rangle$. 其含义是:在地址 addr,如果对象 object_name 写入的是 value 值(addr、object 和 value 也可以是集合,包含空集),那么这个写入一定是正常的.这样,如果内核过滤器中的某一个条目被匹配,则可以立即断定本次写操作是可信的,PCI 卡不做任何处理;如果不能与白名单中的任何一个条目匹配,则内核过滤器进一步将 Refined Data 转发给语义不变式模块.

语义不变式形式中的条目更加复杂,其类似于“run-list $\subseteq$ all-list”或“length(formats) = 5”等形式,因此,CPU 需要根据语义不变式中的每一个条

目对 Refined Data 进行变换计算,之后与条目进行比对。同样地,如果语义不变式模块中的某一个条目匹配能够成功,则可以断定本次写操作是可信;否则如果不能与任何一个条目匹配成功,则可以断定本次写操作不可信,系统报警。

内核过滤器本质上也是一种特殊的语义不变式,其对应于内核运行过程中总是保持为“常量不变”的部分;而语义不变式则更具有普遍意义,其对应于内核运行过程中“动态变化,但可抽象为不变式”的部分(如当前正在运行的进程链表 run-list 是动态变化的,但 run-list 当中的进程一定位于 all-list 链表当中,即无论内核如何运行,其都必须满足不变式:run-list $\subseteq$ all-list)。通过内核过滤器和语义不变式模块的结合,能够在进行复杂的语义不变式匹配之前先进行简单过滤,从而提高了运行效率。

2.2 语义不变式生成

图 1 中的内核过滤器和语义不变式模块确定了内核写操作是否可信的判定标准,因而其是整个系统软件核心之一。前已说明,内核过滤器本质上来说也是一种特殊的语义不变式,因此,如何提炼语义不变式就成为一个关键问题。

本文中,语义不变式生成是一个训练的过程。注意从软件的多次执行当中提取可能的(likely)不变式是可行的,MIT 的 Daikon^[6]动态不变式生成工具就能够实现该功能;Daikon 通过多次运行程序并观察程序的运行迹,从中提炼出可能的不变式。

使用 Daikon 生成语义不变式要解决两个问题:第一, Daikon 需要对程序进行插装(instrument),以便指明生成不变式的入口点(entry point)和出口点(exit point)。然而我们无法对内核进行插装以指明入口点和出口点;第二,生成的语义不变式需要表达为内核数据结构的形式。如何将内核空间地址与内核数据结构对应起来? 本文的解决方案如下:

(1)对于程序的插装以指明入口点和出口点的问题。

注意到内核运行时必有特定的内存地址,因此,只要将整个内核地址空间记录下来——将内核起始地址作为入口点,将内核结束地址作为出口点即可。由于 Daikon 抽取可能的(likely)不变式时需要程序的多次执行迹(trace),因此,在训练的过程中,为了兼顾准确性和存储空间,我们设定每隔 2 秒钟抓取一次内核空间,从而“模拟程序多次执行的迹”(称为内核迹 Kernel Traces)。

(2)对于将内核空间地址与内核数据结构对应

的问题。

首先,从内核源代码获取内核中所有数据结构的定义,并获取内核符号表。分析内核源码获取内核数据结构可以采用编译工具;而内核符号表可以直接下载(如 Windows)或者直接可得(如 Linux 的 System.map 文件)。注意这两部分信息除了训练阶段使用外,也存储在图 1 的 Flash 当中以供动态监控时使用。

其次,利用递归地方法获取内核空间地址与内核数据结构间的对应关系。其基本思想是,以前一个步骤中得到的符号表(符号表中记录了是具有固定物理地址的符号)和内核数据结构定义作为基准点出发,对每一个内核迹利用工作表的方式进行分析,即:每遇到一个新的地址(指针域),就将其加入到工作表当中,这样递归下去,直到工作表为空。这样,就得到了每一个内核空间地址(指针域)与数据结构之间的对应关系。每一个 Kernel Traces,都会生成这样一个对应关系,供 Daikon 使用。

3 实验

从 <http://www.offensivecomputing.net/> 下载到本文实验样本。由于移动智能终端的操作系统如 Android 等本质上还是构建在 Linux 操作系统之上,因此本文选择 Linux 下常见的内核恶意代码(或称为 rootkit)进行试验。这些 rootkit 的恶意功能包括:进程隐藏^[7]、二进制格式增加、禁止防火墙^[8]、攻击随机数生成器等等。下面对上述 rootkit 的语义不变式生成做示例说明。

(1)进程隐藏

Linux 内核对于进程维护有多个链表。系统中所有处于可运行状态的进程都位于 run-list 链表当中,供内核调度执行;而系统中所有的进程都位于 all-tasks 链表当中。一个 rootkit 可以将自身(或者需要隐藏的进程)从 all-tasks 中删除从而从常见的进程列表工具,如 ps,中隐藏;但同时保留在 run-list 当中从而保持正常运行。

Daikon 通过对正常内核学习,能够给出语义不变式:run-list $\subseteq$ all-list。

(2)二进制格式增加

Linux 中程序并没有类似 Windows 的后缀名来区分不同的程序类型。如果某个程序要运行,在新创建一个进程后,search_binary_handler()函数将被调用。这个函数递归遍历存储不同文件格式的 formats 链表。当到达 formats 中的某个节点之后,

节点对应的 handler 就尝试加载二进制映像. 如果能够加载, 则 handler 返回 SUCCESS; 否则如果不能加载, 则 handler 返回 ENOEXEC, 并继续递归查找下一个节点.

攻击者通过修改 formats 链表恶意增加一种新的文件格式, 以便在链表头插入一个新的二进制格式, 同时让这个节点的 handler 包含恶意代码, 且总是返回 ENOEXEC. 这样, 每当一个新的进程被创建, 那么恶意代码就会被执行一次.

在本系统中, 实验初始时有 5 种文件格式. 因此, Daikon 学习得到: formats 链表的长度为 5, 在语义不变式中表达为: $\text{length}(\text{formats}) = 5$.

(3) 非法禁止防火墙

在 Linux 中, 防火墙是由 iptables 来设置的. 而 iptables 通过 netfilter 框架来实现. netfilter 在 ip 栈的不同地方增加了钩子函数, 由用户通过钩子函数来注册回调函数, 从而实现用户特定的网络功能. 所有的 netfilter 钩子注册都保存在 nf_hooks 全局表中. 攻击者可以通过改写 netfilter 钩子, 将其指向攻击函数从而影响 iptables 而非法禁止防火墙.

TCP/IP 协议族钩子函数挂载点为 LOCAL_IN, 对应于 nf_hooks[2][1]. Daikon 学习到 nf_hooks[2][1] 是一个常数地址, 即: $\text{nf_hooks}[2][1] = \text{CONSTANT}$, 其中 CONSTANT 是一个静态的常数. 在不同机器上可能取值不同, 需根据实际情况配置.

注意到 nf_hooks[2][1] 在正常情况下是一个常数, 因此本条语义不变式添加到过滤器中.

实验过程中, 本文选取了 3 类典型的 rootkit, 共 13 个样本. 试验过程中均检测到了这些恶意代码对内核的完整性破坏, 其误报率约 10%. 发生误报的一个主要原因是: 在训练阶段, 内核迹的获取是异步的, 换句话说, 本文采用每隔 2 s 获取一次内核迹, 因而内核迹获取时存在时间延迟, 在这个过程中, 内核仍然在运行, 从而使得获取的不同内核迹之间可能存在不一致的情形. 在后续的研究当中, 我们拟研究采用实时增量式的方法来获取不同的内核迹, 即: 精确记录每一次内核中的改写行为, 仅仅将这些改写操作, 而不是整个内核地址空间, 记录下来. 至于实时内核安全性检测则是由 PCI 卡完成的, 检测时间定义为从 rootkit 安装到 rootkit 被监测出来的时间. 这种定义是合理的, 因为这 16 个样本当中, 并没有 rootkit 故意推迟其恶意功能的实施 (如: 包含特定触发条件才触发). 平均计算, rootkit

对内核完整性破坏的检测时间约为 40 s. 这个时间相对较长, 在将来, 我们拟采用更快的 ARM CPU, 并优化检测算法以提升检测速度.

4 结束语

本文选择世界上市场占有率最高的 Android 系统为研究对象, 对其安全性进行了初步研究. 由于 Android 本质上建立在裁减修改过的 Linux 操作系统之上, 因此, 作为第一步研究, 本文提出了一种基于 PCI 卡及语义不变式的 Linux 内核完整性监控和保护方法, 实验表明了方法的有效性. 在下一步研究中, 我们拟仍以 FPGA 平台为基础, 将本方法应用到包含上层的 Android 系统上.

参考文献:

- [1] 易再尧, 黄本雄, 孙建华. 一种基于虚拟化哈佛体系结构的 Rootkit 技术[J]. 微电子学与计算机, 2010, 24(4): 129-136.
- [2] 孙淑华, 马恒太, 张楠, 等. 内核级木马隐藏技术研究与实践[J]. 微电子学与计算机, 2004, 21(3): 76-80.
- [3] Petroni N L, Fraser T, Molina J, et al. Copilot - a coprocessor-based kernel runtime integrity monitor[C] // Proceedings of the USENIX Security Symposium, San Diego, CA, USA: IEEE, 2004.
- [4] 项国富, 金海, 邹德清, 等. 基于虚拟化的安全监控[J]. 软件学报, 2012, 23(8): 2173-2187.
- [5] 赵波, 张焕国, 李晶, 等. 可信 PDA 计算平台系统结构与安全机制[J]. 计算机学报, 2010, 33(1): 82-92.
- [6] The Daikon Invariant Detector[EB/OL]. [2013-12-01]. <http://groups.csail.mit.edu/pag/daikon/>
- [7] 祝为. Android 平台的内核级 Rootkit 攻击与检测技术[D]. 上海: 上海交通大学, 2013.
- [8] Baliga A, Kamat P, Ifode L. Lurking in the shadows: identifying systemic threats to kernel data[C] // Proceedings of IEEE Symposium on Security and Privacy, Oakland, USA: IEEE, 2007.

作者简介:

曹波男, (1963-), 高级工程师. 研究方向为信息通信系统管理与信息安全.

杨杉男, (1983-), 工程师. 研究方向为信息系统运维与安全管理.

梁新建男, (1963-), 高级工程师. 研究方向为信息通信系统管理与信息安全.