计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(8):1−9 [doi: 10.15888/j.cnki.csa.007018]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Android 框架层完整性度量方案①

周星锦[1], 秦　宇[1], 吴秋新[2], 习　伟[3], 赵世军[1]

[1](中国科学院软件研究所 可信计算与信息保障实验室, 北京 100190)
[2](北京信息科技大学 理学院, 北京 100196)
[3](南方电网科学研究院有限责任公司, 广州 510080)
通讯作者: 周星锦, E-mail: zhouxingjin@tca.iscas.ac.cn

摘　要: 长期以来 Android 系统一直是黑客攻击的主要目标之一, 自发布以来一直面临着 root、镜像篡改、恶意程序等安全风险, 框架层是在系统安全中容易被忽视但又能产生极高的安全风险. 本文分析了 Android 系统中框架层的表现形式和框架层的使用方式, 针对框架层特点提出了一种框架层完整性度量方法 (FIMM), 以此保障 Android 系统框架层代码完整性和运行时的完整性. 对于 Android 系统针对框架层组件完整性保护的缺失, 该方法能提供框架层组件在加载时的完整性度量和完整性校验. 而对于 Android 的系统服务, 我们考虑到其较长的运行周期的特征, 于是研究了系统服务的调用过程并为其提供了较为细粒度的动态度量, 在每次系统服务调用时确认系统服务进程代码段的完整性. 最后我们给出了基于 Android 模拟器的原型系统的实现, 并分析了 FIMM 的安全性和性能损耗, 认为 FIMM 能完全达到我们的安全预期, 并且只会造成少量的性能损耗.

关键词: Android 安全; 完整性度量; 系统服务; 动态度量

引用格式: 周星锦,秦宇,吴秋新,习伟,赵世军.Android 框架层完整性度量方案.计算机系统应用,2019,28(8):1–9. http://www.c-s-a.org.cn/1003-3254/7018.html

Integrity Measurement Method for Android Framework

ZHOU Xing-Jin[1], QIN Yu[1], WU Qiu-Xin[2], XI Wei[3], ZHAO Shi-Jun[1]

[1](Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](College of Science, Beijing Information Science and Technology University, Beijing 100196, China)
[3](China Southern Power GRID, Guangzhou 510080, China)

Abstract: Android system has been one of main targets of hacking. Since its realease, it has been facing security risks such as root, image tampering, and malicious programs. System security usually overlooks framework layer which may cause high security risks. This study analyzes the component in Android framework and how it works. We present an Integrity Measurement Method for Android Framework (FIMM) to ensure Android framework's code integrity and runtime integrity. We focus on the issue of lack of protection for Android framework. The method is able to measure the integrity of component of Android framework and verity it. We consider that Android system service has a long running period, then we analysis the calling process for Android system services and present a dynamic integrity measurement for process providing fine-grained measurement. It measures and verities system service process every time system service is called. Finally, the FIMM impelemention based on Android emulator is discussed and presented. We believe that FIMM achieves the security goals, nevertheless, it causes a little performance loss as well.

Key words: Android security; integrity measurement; system service; dynamic integrity measurement

① 基金项目: 国家自然科学基金 (61872343, 61602455, 61802375); 国家重点研发计划 (2018YFB0904900, 2018YFB0904903)
Foundation item: National Natural Science Foundation of China (61872343, 61602455, 61802375); National Key Research and Development Program of China (2018YFB0904900, 2018YFB0904903)
收稿时间: 2019-01-28; 修改时间: 2019-02-26; 采用时间: 2019-03-08; csa 在线出版时间: 2019-08-08

1 引言

移动互联网带来的移动无线通信方式使人们连通互联网的方式不再受到束缚, 随之兴起的移动设备更是彻底的改变了人们的生活方式. 手机的功能早已不限于电话和短信, 人们会用它来连接自己的电子邮件、社交网络、移动支付、照片分享等等. 智能手机储存了许多敏感的个人信息, 如登录凭证、照片、甚至银行账户等敏感信息. 这些数据对于用户来说是至关重要的, 因为可能会有不法分子会收集人们手机上的敏感信息作为数据供自己使用, 甚至会造成不可承受的经济损失和精神打击. Android 系统是 Google 公司 2007 年发布的基于 Linux 内核的开源移动操作系统, 主要应用于智能手机、平板、智能手表等智能终端. 截至 2015 年 Android 收集已发行 14.3 亿部, 占有智能手机市场 82% 的份额, 是目前最受欢迎的移动设备操作系统[1].

Android 系统由于受众广泛、发行量大, 是移动设备中黑客攻击的主要目标. 好在 Android 硬件层面、内核层和应用层现在都有着比较有效的解决方案能防止常见的攻击手段, 但是框架层往往被认为是提供给应用层使用的 JAVA API, 对于框架层的安全性各安全方案一般只在意其逻辑上的正确性, 即是否存在漏洞等问题, 而没有着重于保护它的静态和动态完整性. 考虑到 Android 系统的体量以及设备被攻破后对用户造成的损失, 我们认为对框架层完整性保护的研究是有必要的. 本文从可信计算的角度提出了一种针对 Android 框架层以及系统服务的完整性度量方法 (Framework Integrity Measurement Method, 以下简称 FIMM), 将框架层的信任转移至内核层.

2 背景

Android 系统体系复杂, 国内外学者针对 Android 不同层次的安全性进行了非常多的研究. KNOX[2]是较为典型的 Android 信任链方案, 它的安全启动过程会逐步验证 Bootloader、内核以及系统软件的签名, 确保系统加载的组件都是经过授权的, 以此保证 Android 系统从加电到正式运行时整个过程的完整性. Google 在 Android4.4 版本后移植了类 SELinux 的强制访问控制 (MAC) 机制—SEAndroid[3], 严格来说 SEAndroid 实现在框架层, 保护的对象是系统中的资源 (文件、进程等). 虽然 SEAndroid 的引入很大限度的阻止了恶意应用对系统的攻击, 也有效地减小了内核层出现漏洞时产生的威胁, 但是 SEAndroid 依赖于可信的内核, 无法防御敌手对内核的直接攻击[4]. Android 系统内核安全的威胁承自 Linux 内核安全威胁, 国内外对于 arm 平台架构的设备的内和完整性有多种高效的解决方案, 比较主流的是利用 Trustzone 管理以及校验内核指令、页表或者内存的方法, 例如 SKEE[5]和 tz-Rkp[6]以及为不可信的内核提供可执行文件、运行时代码和控制流完整性的方案[7]. 对于应用层 Android 系统拥有自己的一套比较成熟的隔离和证书机制, 通过沙箱机制的隔离确保 APP 间互不干涉, 防止一个恶意程序侵害其他 APP. 证书机制可以管理 APP 之间的调用权限、APP 的版本控制, 同时在设备的网络通信过程上保证其安全性.

2.1 Android 信任链

自可信计算组织 (TCG) 成立以来, 可信计算取得了飞速的发展. 信任链的建立与传递是可信计算的基本问题, 其中涉及到三点: 信任根、信任传递和可信度量. 信任根是系统可信的基石, 也是信任传递的起点. 信任传递指的是一层一层向上提供完全可信的功能, 系统的每一层的实现都是基于下一层的信任, 通过可信传递可以实现系统可信范围的延伸[8–10]. 可信度量指的是对文件及其相关配置信息进行完整性验证, 防止其被篡改. 通过这三点构建的信任链自上而下赋予信任, 将大规模系统的信任管理缩小至信任根. 在一个完整的信任链体系中只要能保障信任根的可信即可保障整个系统的可信.

Android 系统发布早期, 三星的研究员就已经对移动高效完整性度量和认证机制进行过研究[11,12]. 移动设备在硬件层面难以扩展, 在将移植 Linux 上的完整性度量架构移植到 Android 上时选择使用软件 TPM 进行扩展. 该方案以厂商的内核证书作为信任根来进行安全引导, 确保平台能够引导到安全状态.

三星 KNOX[2]的安全启动是一种防止未经授权的操作系统和软件在启动过程中加载的过程. 固件镜像是由已知的、受信任的权威机构进行加密签名的, 被认为是授权的固件. 安全启动时硬件验证 bootloader、内核、系统软件的签名. 安全启动过程中用到的 X.509 证书和公钥嵌入在 bootloader 中. 默认情况下, 设备的信任根是三星颁发的证书.

2.2 SEAndroid

Android 是一个分层的体系结构, 允许应用程序利用底层 Linux 内核提供的服务. 然而, Android 并不阻止应用程序通过系统调用直接触发内核功能. Linux 系统包括早期 Android 系统下的自主访问控制 (DAC) 通过访问控制列表限制主体对设备数据和文件的操作. 在 DAC 策略下, 主体拥有自己的 UID 和 GID, 客体拥有允许访问的访问控制列表, 以此来判断主体是否与访问的文件权限匹配, 来确定主体能否访问文件. 这种机制的缺点非常明显, 当主体获得更高的权限甚至获得设备的 root 权限, 就可以随意操作任何敏感文件.

SEAndroid 提供的强制访问控制 (MAC) 针对 DAC 的缺陷, 每个访问文件的操作都会通过 MAC 的安全策略进行判断, 违反安全策略的操作会被阻止, 所有不合法的操作都会被记录在日志里.

虽然 SEAndroid 依赖于可信的内核, 无法防御敌手对内核的直接攻击, 但是 SEAndroid 的引入很大限度的阻止了恶意应用对系统的攻击, 也有效地减小了内核层出现漏洞时产生的威胁, 针对移动设备来说在保证内核可信的层面上能保障系统不被破坏.

2.3 Trustzone

TrustZone 是一组用于 ARM 的硬件安全扩展, 主要针对高性能计算平台上的大量应用, 包括安全支付、数字版权管理、企业服务和基于 Web 的服务. TrustZone 空间控制器可以将 DRAM 划分为不同的内存区域, 并将内存区域指定为安全或正常. TrustZone 内存适配器为 OCM 提供了类似的功能. 可信的 DMA 控制器防止将一个外围设备分配给正常的世界, 从而执行 DMA 传输到安全的世界内存. TrustZone 保护控制器可以使外围设备安全或正常. 这些隔离机制将所有 SoC 的硬件资源划分为两个世界: 安全的世界和正常的世界. 处理器执行的世界由一个 NS 位表示, 它通过系统总线传播. 可信的总线结构确保了正常的世界组件无法访问任何安全的世界资源[13]. Trustzone 可以为系统上运行在正常世界但由具备一定安全需求, 如移动支付等程序提供隔离与正常世界的安全服务. 由于 Trustzone 与正常世界的操作系统隔离的特点, Trustzone 在系统安全领域内多用于保护运行时的内核[14].

2.4 现有方案存在的问题

近年对于 Android 安全的研究, 各国的研究人员都提出了可行或具备实验意义的优化方案, 从 Android 系统的各个角度去减少 Android 面对的风险. 但是对于 Android 框架层的完整性的解决方案则相对比较少见, 框架层作为 Android 体系内非常重要的一层, 保证其完整性是非常重要的. 虽然部分框架层的功能是作为底层向应用层提供的接口, 但是如果这部分被恶意篡改则很容易造成设备异常甚至信息泄露等危害. 而且框架层包括多种系统服务, 移动设备的正常运行依赖于这些系统服务, 如果系统服务被修改为恶意行为的程序, 对设备的以及个人数据也是很大的威胁. 所以对 Android 系统框架层完整性的保护也是不可或缺的.

3 相关工作

3.1 Android 框架层

Android 框架层作为应用层和底层代码的中间层, 它封装了规范化的模块向应用层提供 JAVA API, 同时也包含 JNI 方法调用底层库函数以提供部分系统服务, 例如 CameraService 和 MediaPlayerService 等. 其中系统服务与用户的隐私数据密切相关.

在 Android 系统的/system/framework 目录下主要存在三类文件: jar 包、odex 文件、boot.art 和 boot.oat. Jar 包为 Android 部分功能提供框架层的各类库的支持, 例如在执行 am 命令时会加载 am.jar 文件. 从 Android 4.4 版本起, Google 向 Android 里移植了 ART 虚拟机, 在 5.0 版本后 ART 虚拟机彻底代替了原先的 Dalvik 虚拟机, 运行 ART 需要该目录下的 boot.art 和 boot.oat 两个文件. 在 Android 源码编译时, 会将部分常用公共类打包到 boot.oat 中; boot.art 则中则包含了 boot.oat 中的方法代码的指针, 是 ART 虚拟机的启动镜像. 在/system/framework/oat/arm 目录下的 odex 文件则是编译源码时将部分 jar 包优化生成的结果, 例如在创建系统服务时会加载 services.odex.

我们的目的是度量完整的 Android 框架层, 所以/system/framework 目录下的所有文件都是我们度量的目标.

3.2 System_server 与 service_manager

在 Android 系统中框架层一部分功能以系统服务的形式提供给应用层使用, 包括窗口管理、电源相关操作等功能. 所有系统服务作为子线程运行在 system_server 进程中, 然后再注册到 service_manager 进程中. 当应用层想要获取某项系统服务时需要通过 service_

manager 得到相关系统服务的 IBinder, 然后通过 IBinder 与 system_server 中的系统服务通信.

Android 系统内核启动之后有 init 进程创建 zygote 进程, 这是 Android 上所有 java 程序的祖先, 之后 zygote 会创建 system_server 进程. System_server 会加载系统服务相关的代码并分别将各系统服务注册到 service_manager 中, 该进程在系统启动时会一直运行直至系统关闭. Android 系统的初始化过程如图 1 所示.

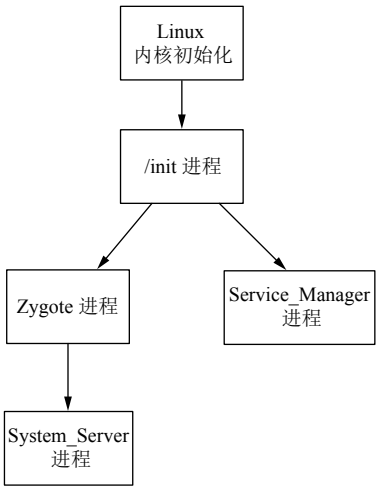

图 1 Android 系统初始化

对于 system_server 这样的守护进程, 生命周期等同于系统开启的时间, 容易遭受具有针对性的进程攻击, 例如敌手利用进程代码中的漏洞使进程重定向到指定的程序, 如果敌手使用已经妥协的系统服务代码替换正常的系统服务, 将会对系统造成极大的影响.

3.3 国密算法

目前 Android 系统的密码库仅支持传统密码算法, 即 RSA、MD5、ECC 等. 为了推广国产加密系列算法, 我们向 Android 内核移植了运行在内核态下的国产密码算法库.

国密是国家密码局认定的国产密码算法, 即商用密码. 在本文中, 我们主要用到了 SM2 公钥加密算法和 SM3 哈希算法.

SM2 是使用的是 ECC 椭圆曲线密码机制的公钥密码算法, 公钥长度为 64 字节, 私钥 32 字节. SM2 算法在签名、密钥交换方面与 ECDSA、ECDH 等国际标准不同, 它采取了更为安全的机制. SM2 算法有推荐的 256 位曲线作为标准曲线. SM2 标准包括总则, 数字签名算法, 密钥交换协议, 公钥加密算法四个部分. 我们的实验中主要使用 SM2 签名算法.

SM3 算法是哈希算法, 算法输入为长度小于 2 的 64 次方的比特消息, 填充和迭代压缩之后生成长度为 256 比特的摘要. 其中使用了异或, 模, 模加, 移位, 与, 或, 非运算等, 并由填充、迭代、消息扩展和压缩函数所构成. 由于 SHA-1 和 MD5 哈希算法在早些年已经被证实存在碰撞攻击的问题, 这两种算法不再是安全的哈希算法. SM3 算法与 SHA256 具有相似的压缩结构, 但是 SM3 设计的更为复杂, 所以相比之下 SM3 哈希算法拥有更高的安全性.

3.4 进程动态度量方法

与加载时直接对文件度量不同, 动态度量的对象是进程, 度量进程的目的是确保进程运行时执行的代码不被修改, 所以需要访问进程空间存储代码段的地址[14].

Linux 内核会通过进程描述符 mm_struct 来管理运行在系统上的进程, 并且以 start_code 和 end_code 字段标识进程空间中存放运行的代码段的地址. 内核需要首先获取被度量进程的 task_struct, 然后获取 mm_struct 描述符, 通过描述符中的 start_code 和 end_code 确定存放代码段的线性地址, 将这一部分的页面复制到内核空间. 使用 SM3 哈希算法对这一部分的页面数据进行计算可以得到进程代码段的散列值.

对于基于 Linux 内核的系统, 可以实现一个内核模块, 使内核模块在/dev 下部署一个文件设备以获取度量请求. 系统启动时批量加载模块, 模块加载后首先获取内核空间中全局的 task_struct 链表头指针, 然后遍历链表通过 task_struct 的 comm 成员存储的进程运行的可执行文件名找到目标进程的 task_struct 的指针. 设置文件的写入函数功能为复制代码段页面到内核模块然后计算其 SM3 哈希值与上一次计算的值作比较.

3.5 DM-verity

Android4.4 引入了 VerityBoot 新特性, DM-verity 在其中用于安全启动时校验系统分区. DM-verity 是基于 Linux 内核中提供的一种从逻辑设备到物理设备的映射框架 device mapper 为系统分区创建哈希树实现的. 它将系统镜像划分成 4 k 大小的块, 并为每个块计算 SHA256 哈希值, 以这些块作为叶子结点, 然后将这些哈希值组合成以 4 k 大小的为单位的块形成了第一层, 然后再将这一层的块进行哈希填充到 4 k

大小的块中形成第二层, 然后重复上述操作知道所有哈希值能填充到一个块内, 形成一个哈希树. 这样一个叶子结点存储的数据变化就可以通过它的父节点到根节点的哈希值来描述.

DM-verity 能够在开机时验证系统分区的完整性, 但这种方案只能保证系统分区中框架层组件存储时的完整性, 即静态完整性. 而且由于目前很多厂商订制 Android 系统时会为了提高开机速度等原因关闭 DM-verity 功能所以我们需要其他的方案来保护框架层组建的完整性.

4 方案设计

4.1 敌手模型

在我们方案中认为框架层会受到如下两种较为具有严重破坏性的攻击:

1) 本文利用完整性度量方案为 Android 系统提供框架层的完整性保护, 因此假设敌手具备修改、替换 Android 框架层文件的能力. Android 系统的 DM-verity 机制能保证系统在启动时校验系统分区的完整性, 但是可以假设敌手具备在校验完成之后修改框架层的文件, 这样即使分区校验可以正常进行但是在系统创建 zygote 进程时依然具备调用恶意的框架层代码的可能性.

2) Android 系统运行时框架层的作用体现在系统服务, 在系统启动时 init 进程会在创建 zygote 之前先 fork 出 service_manager 进程用于管理系统服务, 之后再由 zygote 进程孵化出 system_server 进程, Android 的系统服务以多线程的形式则运行在 system_server 进程中. 在本文中, 我们假设敌手在系统正常引导启动之后, 在系统运行时有能力篡改 system_server 进程正在运行的程序或者向该进程注入恶意代码.

上述两种手段都能达到破坏 Android 系统框架层完整性的目的, 敌手可以使系统运行存在漏洞的系统服务然后通过已知的漏洞提权或者窃取用户信息等操作.

4.2 详细设计

4.2.1 整体架构

本框架层完整性保护方案的目标是从编译 Android 源码得到镜像到框架层组件加载以及整个运行过程三个过程去全方位的保障框架层的完整性. 总体架构框图如图 2 所示.

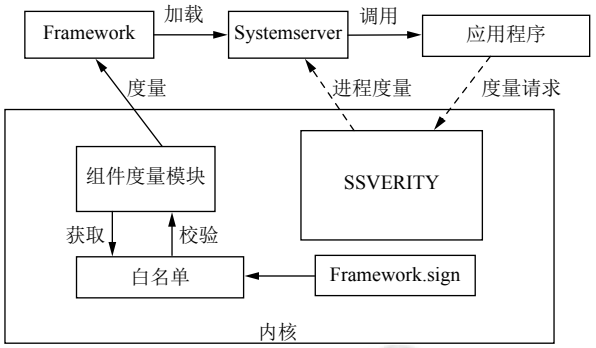

图 2 FIMM 整体架构

架构组件介绍:

Framework.sign: 系统镜像编译时, 记录对 framework 下的框架层组件使用 SM3 哈希算法计算得到的哈希值组成白名单并使用 SM2 算法签名过后的打包文件;

白名单: 由 framework.sign 得到的 framework 文件对应的哈希值列表, 由于是通过 SM3 哈希算法计算文件的摘要, 所以文件一旦被敌手修改那么再使用哈希算法计算出来的值一定不同, 完整性度量时以本文件中的哈希值为基准;

Framework: /system/framework 目录下的框架层文件, 包括 jar 包、odex 文件、boot.art 以及 boot.oat;

应用层度量代理: 运行在应用层, APP 在使用系统服务时向其产生一个度量请求, 之后向内核中的系统服务度量模块发起挑战, 等待接收内核模块处理完度量请求后得到度量结果并将校验结果返回到 APP;

SSVERITY: 在部署进内核之后会在/proc 虚拟文件系统下生成一个文件节点, 用于接收度量代理发起的挑战;

组件度量模块: 框架层组件在加载时, 由该模块计算加载的文件的哈希值并校验其完整性.

4.2.2 安全启动的改良方案

FIMM 源码在编译过程成中会对编译生成的框架层文件 (system/framework/), 我们在源码中添加脚本, 使用 SM3 哈希算法计算框架层文件的哈希值并生成白名单, 之后使用 SM2 私钥对白名单进行签名然后将白名单和签名文件一同打包至系统镜像中. 这个过程需要在 out/root/system/framework 目录中的所有文件都生成之后, 在 system_img 镜像文件生成之前执行; 或者在源码编译结束后编写脚本将 framework.sign 文件在 out/root/目录下生成然后再次编译 Android 源码即可.

FIMM 系统的启动过程中会在 DM-Verity 验证系

统分区完整性并挂载文件系统之后使用硬编码进内核代码中的 SM2 公钥验证签名文件, 如果验证成功则将框架层的白名单加载进内核空间中, 如果验证不通过则表示本系统已被敌手入侵不可正常启动. 由于签名文件中包括原白名单以及签名信息, 如果其中任意一部分被修改过那么都无法通过 SM2 公钥验签通过, 这样可保证系统在启动时加载的框架层完整性信息以及签名一定是未被篡改过的.

内核的验签模块作为 FIMM 安全启动的一个环节, 它以一个内核模块的形式编译在内核中, 并生成一个文件节点用于启动其验签功能. 正常情况下该模块不提供任何服务, 当通过文件结点触发其功能后, 它会调用国密算法库提供的 SM2 算法验证/system/framework.sign 的签名, 如果验签通过那么将框架层组件的哈希值保存到内核空间中, 否则重启系统. Linux 内核正常引导之后, 由 init 进程挂载文件系统, 我们使 init 进程在挂载上系统分区之后运行通知程序以触发内核的验签模块. 通知程序只需要向验签模块部署的文件结点写入对应的字符指令即可. 可以通过修改 init.rc 等文件使 init 进程在正确的时刻运行通知程序.

4.2.3 框架层加载时度量

FIMM 的内核在挂载系统分区之后就会保存框架层文件的完整性度量值, 接下来需要在每次加载/system/framework 中的组件时对其度量.

我们通过修改 Linux 内核的 IMA[15]来实现这个功能. IMA 是一种开源可信计算组件. IMA 维护一个运行时度量列表, 如果将其保存到一个硬件可信平台模块 (TPM) 中, 则该列表上的聚合完整性值. 在 TPM 中保存聚合完整性值的好处是度量列表不会被任何软件攻击破坏, 并且不会被检测到. 因此, 在可信的引导系统上, 可以使用 IMA 度量来验证系统的运行时完整性. Android 平台大多没有 TPM 的支持, 我们只是使用 IMA 加载时度量的模块. IMA 现有的功能可以当系统上的文件在加载进内存时计算其散列值然后记录到日志中, 我们通过重构 IMA 部分代码, 使 IMA 使用 sm3 算法只度量/system/framework 目录下的文件, 得到哈希值后即可利用前一节获取的白名单进行对比, 如果对比出现错误则表示这个文件已经被敌手篡改.

在系统中有文件加载进内存时我们这个模块能获取文件的描述符, 我们首先通过文件路径和格式判断其是否为框架层组件, 如果不是则使其正常工作; 如果判断为框架层组件, 我们首先使用密码算法模块中的 SM3 哈希算法计算整个文件的哈希值, 然后使用已存入内核空间的白名单进行校验, 如果校验成功, 即文件的度量值与刚计算得到的哈希值相同则正常启动; 如果校验失败, 我们可以进行错误处理流程, 可以选择向应用层发起警告, 安全要求更高设备可以选择 kill 掉调用该组件的进程, 我们的实验中选择使用 netlink 机制向应用层广播设备被入侵的警告.

以 FIMM 系统从引导内核到提供系统服务的整个过程为例, 运行流程如图 3 所示.

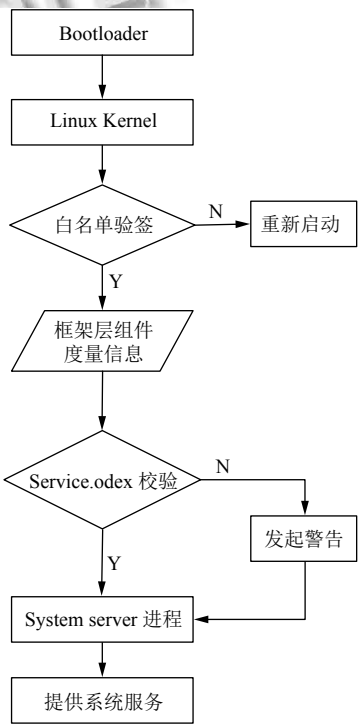

图 3 FIMM 启动过程

4.2.4 系统服务的实时度量

系统服务作为框架层比较特殊的一部分, 系统 zygote 进程创建出 system_server 进程时会加载系统服务, 然后将服务注册到 service_manager 进程中, 一般来说在系统启动之后框架层组件就会持续运行直到系统关闭. 上述方案可以保证在 Android 系统启动到加载框架层组件的过程中保证其完整性, 但是对于系统服务这种一次加载就运行至系统关闭的组件仅仅在加载时度量其完整性不能很好的保证系统不遭受敌手的攻击, 即容易遭受 TOU-TOC 攻击. 需要更细粒度的度量方案, 保证框架层在运行时的完整性.

Android 应用层通过 Context.getSystemService() 方法调用 system_server 进程中的系统服务. 在调用系统服务之前先通过 FIMM 中的度量代理对 system_server 进程进行动态度量, 确认其完整性. FIMM 一次系统服务调用过程如图 4 所示.

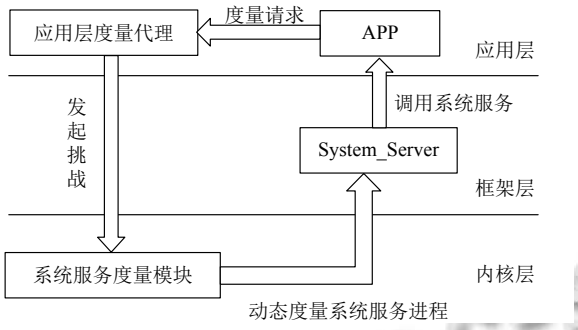

图 4 系统服务实时度量

应用层度量代理是运行在应用层, 应用层软件(如 APP、桌面显示等) 在使用系统服务时向其产生一个度量请求, 之后向内核中的系统服务度量模块发起挑战, 等待接收内核模块处理完度量请求后得到度量结果并将校验结果返回到 APP. 应用层度量代理实现为 Android 源码中的一个 java 类, 通过 jni 方式调用 C 程序实现的本地方法去向 SSVerity 控制的文件结点写入度量请求并提供三字节的缓存区存储 SSVerity 的完整性度量结果, 在发起挑战之后进入自旋检查缓存区, 得到完整性度量结果之后返回, 如果度量结果无误则正常执行 system.getSystemService() 的流程, 如果度量出现错误则向用户发出警告.

SSVerity 在部署进内核之后会在/proc 虚拟文件系统下生成一个名为 SSVerity 的文件节点, 用于接收度量代理发起的挑战. 内核模块会维护一个用来获取 system_server 进程信息的描述符, 在接受到挑战时首先检查该进程描述符是否为 system_server, 如果不是则需要在进程链表中寻找到 system_server 的描述符, 然后获取进程对应的代码段的线性地址, 通过线性地址将该进程的代码段页面拷贝到内核空间, 然后使用 SM3 哈希算法计算代码段的散列值, 使用内核模块维护的完整性值校验, 将校验结果返回到度量代理.

当度量代理向 SSVerity 部署的文件结点写入度量挑战时, SSVerity 首先检查模块当前定位的进程是否为 system_server, 如果不是则需要在内核的进程链表中查找出 system_server 的 task_struct 并将该描述符的指针保存在 SSVerity 模块中. 通过 system_server 的 task_struct 获取其 mm_struct 成员的指针, 再通过 mm_struct 的 start_code 和 end_code 得到 system_server 进程代码段的线性地址, 然后将这部分页面复制到 SSVerity 的内核空间中然后使用国密算法模块提供的 SM3 哈希算法计算代码段的 256 位哈希值, 与内核空间中保存的度量值对比. 如果内存中的度量值全为 0 则表示这是第一次度量系统服务, 那么将哈希值写入 SSVerity 维护的度量值中, 并判定为校验正确; 如果计算得到的哈希值与内核空间中的度量值相同那么判定为校验正确; 如果内核空间中的度量值不全为 0 且与哈希值不相同那么表示 system_server 进程空间中的代码段被修改, 判定为校验失败. 校验结束后 SSVerity 将对应的结果写入度量代理预留的 3 字节缓存中, 一次系统服务度量结束.

应用程序在调用 Context.getSystemService() 获取系统服务的时候先向度量代理发起系统服务度量请求, 再由度量代理向部署在内核的系统服务度量模块发起挑战, 此时该内核模块会动态度量 system_server 进程代码段的完整性并进行校验, 校验通过之后执行正常的 Android 系统服务调用流程.

由于 SEAndroid 会限制各进程对 SSVerity 文件的访问, 所以需要在编译 FIMM 源码之前修改 SEAndroid 的访问策略, 使所有进程都具备读写 SSVerity 的权限. 在本实验中 SEAndroid 不是我们考虑的重点, 所以我们修改了启动设置, 使系统启动时关闭了 SEAndroid.

5 实验结果评估

本文实现了 FIMM 系统原型, 实验环境是 Intel(R) Core(TM) i7-6700 CPU@3.40 GHz 处理器和 16 GB 内存的 PC 上的 Linux ubuntu. 在这个平台上, 本文首先搭建了 Android 6.0 arm 的运行环境, 在该版本的 Android 源码上搭建了 FIMM 的原型, 如图 5.

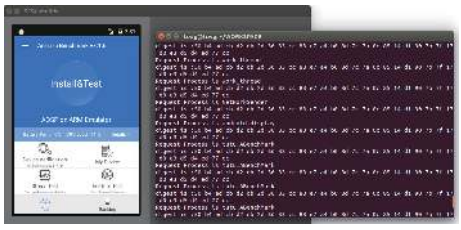

图 5 FIMM 原型

5.1 安全性评估

对于 4.1 节提到的敌手模型 1, 假设敌手 root 了设备并在系统启动之后用某个有漏洞的框架层组件替换了设备上正确的组件, 想使应用程序使用这个带有漏洞的组件, 然后利用这个漏洞攻击应用程序. 系统在需要使用该组件时会调用 mmap() 系统调用将该文件映射到内存, 但是在这之前 FIMM 会计算这个文件的 SM3 哈希值, 而且由于 FIMM 系统正常启动, 所以现在内核中保存有这个文件对应的哈希值, 之后 FIMM 会对比计算出来的哈希值是否与保存的值相同, 由于敌手使用其他版本的组件替换了正常的组件, 那么现在计算出来的哈希值必然与内核保存的哈希值不同, 按照 FIMM 的策略不允许其继续执行, 抛出调用错误并发起警报.

而对于敌手模型 2, 假设敌手为了修改 system_server 进程正在运行的程序, 首先使用恶意进程伪装成目标进程欺骗操作系统, 以此获得修改 system_server 进程地址空间内容修改的权限, 然后修改进程空间存放代码段的页面属性, 最后修改代码段页面的页面内容使进程运行恶意的程序. 在 FIMM 中, 每次系统服务的调用都会出发内核的系统服务进程检验机制, 它会计算 system_server 进程空间代码段的 SM3 哈希值并且和之前得到的结果对比, 所以当敌手实施攻击之后的下一次系统服务调用一定会发现系统服务进程被敌手攻击.

综上所述, FIMM 完全能够预防我们在 4.1 节中提出的 Android 框架层可能受到的攻击.

5.2 效率评估

对于动态度量系统服务的方案, 其性能影响之处在于每次通过 Context.getSystemService() 方法获取系统调用的时候都需要对 system_server 进程的代码段进行哈希运算, 所以对于应用程序的影响主要取决于程序调用系统服务的次数. 本文中我们对 Android 6.0.0 系统的五个系统 APP 的开机时间进行了测试, 一定程度反映了本度量方案对操作系统的影响, 开机时间采用多次启动取平均值的方法, 对比结果如表 1 所示.

表 1 实验结果对比

单位 (ms)	Browser	Music	Message	Phone	Camera
原生 Android	3070	1606	1436	1783	3564
FIMM	3426	1850	1754	2073	3935

由于本文实验是在 ubuntu 上搭建的 Android 模拟器上进行, 不能精确的得到性能损耗的具体值. 通过表我们可以看出使用 FIMM 和未使用 FIMM 会造成大约 15% 的性能损耗, 这个结果在预期范围内的, 对于系统运行时只是在系统调用过程中增加了进程空间代码段的页面复制以及一次 SM3 哈希运算.

6 总结与下一步工作

在本文中, 我们提出一种针对 Android 框架层的完整性度量方案. 我们的方案使用源码编译时生成的白名单为基准对一般的框架层文件采用加载时度量, 对运行周期长的系统服务采用细粒度的动态度量, 力求在系统应用层与框架层交互时能保证其完整性. 通过安全评估可以了解到 FIMM 完全可以检验系统框架层是否受到破坏. 实验表明, 本方案主要对系统服务请求次数频繁的 APP 造成的一定的性能影响, 但是这种细粒度的度量方式可以更加有效的保护系统服务. 但 FIMM 仍然存在缺陷, FIMM 是完全基于内核的保护方案, 即通过内核层来保护 Android 框架层, 不对内核层进行安全加固, 如果内核被攻破本方案的功能就不具备意义. 考虑到 Android 内核面临的安全风险, 因此下一阶段可以采用内核与 TrustZone 结合的方法为框架层提供完整性保护, 或者使用基于 TrustZone 的内核保护方案为 FIMM 提供内核的完整性保护.

参考文献

1 许艳萍, 马兆丰, 王中华, 等. Android 智能终端安全综述. 通信学报, 2016, 37(6): 169–184. [doi: 10.11959/j.issn.1000-436x.2016127]

2 Samsung Electronics Co. KNOX white paper. 2013.

3 Smalley S, Craig R. Security enhanced (SE) Android: Bringing flexible MAC to Android. Proceedings of the 20th Annual Network and Distributed System Security Symposium. San Diego, CA, USA. 2013. 20–38.

4 Merlo A, Costa G, Verderame L, *et al*. Android vs. SEAndroid: An empirical assessment. Pervasive and Mobile Computing, 2016, 30: 113–131. [doi: 10.1016/j.pmcj.2016.01.006]

5 Azab AM, Swidowski K, Bhutkar R, *et al*. SKEE: A lightweight secure kernel-level execution environment for ARM. Proceedings of Network and Distributed System Security Symposium. San Diego, CA, USA. 2016.

6 Azab AM, Ning P, Shah J, *et al*. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. Proceedings of 2014 ACM SIGSAC Conference on Computer and Communications Security. Scottsdale, AZ,

USA. 2014. 90–102.
7 张英骏, 冯登国, 秦宇, 等. 基于 Trustzone 的强安全需求环境下可信代码执行方案. 计算机研究与发展, 2015, 52(10): 2224–2238.
8 Feng DG, Qin Y, Wang D, *et al*. Research on trusted computing technology. Journal of Computer Research and Development, 2011, 48(8): 1332–1349.
9 Shen CX, Zhang HG, Feng DG, *et al*. Survey of information security. Science in China Series F: Information Sciences, 2007, 50(3): 273–298. [doi: 10.1007/s11432-007-0037-2]
10 杨波, 冯登国, 秦宇, 等. 基于可信移动平台的直接匿名证明方案研究. 计算机研究与发展, 2014, 51(7): 1436–1445. [doi: 10.7544/issn1000-1239.2014.20131768]
11 Zhang XW, Acıiçmez O, Seifert JP. Building efficient integrity measurement and attestation for mobile phone platforms. Proceedings of the First International ICST Conference on Security and Privacy in Mobile Information and Communication Systems. Turin, Italy. 2009. 71–82.
12 Nauman M, Khan S, Zhang XW, *et al*. Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. Proceedings of the Third International Conference on Trust and Trustworthy Computing. Berlin, Germany. 2010. 1–15.
13 Asokan N, Ekberg J E, Kostiainen K, *et al*. Mobile trusted computing. Proceedings of the IEEE, 2014, 102(8): 1189–1206. [doi: 10.1109/JPROC.2014.2332007]
14 刘孜文, 冯登国. 基于可信计算的动态完整性度量架构. 电子与信息学报, 2010, 32(4): 875–879.
15 Sailer R, Zhang XL, Jaeger T, *et al*. Design and implementation of a TCG-based integrity measurement architecture. Proceedings of the 13th Conference on Usenix Security Symposium. San Diego, CA, USA. 2004. 16.