基于 Xposed 的 Android 透明文件加密系统的研究

朱天楠[1],施 勇[1],薛 质[1,2]
(1. 上海交通大学 信息安全工程学院,上海 200240;
2. 上海市信息安全综合管理技术研究重点实验室,上海 200240)

摘 要:随着移动处理器技术水平的高速发展,智能设备的计算能力不断加强,人们对智能手机的依赖性也不断增加。通过安装各类应用,手机可以具有丰富的功能,但使用过程中往往会需要记录用户的隐私数据,保护存储在智能设备上的用户隐私数据不被恶意应用随意获取的需求日益加大。结合当前流行的透明文件加密技术与 Android 自身的一些特点,提出了一种基于 Xposed 框架的透明文件加解密方案。其以 SharedUserId 和开发者签名信息为标识自动生成密钥,将各个 APP 的数据以不同的密钥加密处理,这样即使在恶意 APP 获取到了 Root 权限,仍能保护各 APP 的隐私数据不被非法获取,从而提升了 Android 设备的安全性。该过程自动完成,无需应用开发者和用户参与,无需改变开发与使用习惯。

关键词:隐私安全;Xposed 框架;透明加密;Android
中图分类号:TP309 **文献标识码**:A **文章编号**:1673-629X(2017)02-0064-05
doi:10.3969/j.issn.1673-629X.2017.02.015

Research on Android Transparent Encryption File System Based on Xposed

ZHU Tian-nan[1], SHI Yong[1], XUE Zhi[1,2]
(1. College of Information Security and Engineering, Shanghai Jiaotong University, Shanghai 200240, China;
2. Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, Shanghai 200240, China)

Abstract: With the constant progress of SOC technology, mobile devices are more and more powerful, and the people relies increasingly on them. Richer applications enhance the capability of mobile devices, but malicious APP which aim to steal users' private information are also spring up. To protect users' privacy, an encrypted on-the-fly solution based on Xposed is proposed which is combination of a popular Hook framework on Android. This solution uses SharedUserId and developer's signature as identity to calculate the secret key for encryption, so that every different APP has a different key. Hence even the malicious app runs as root user, it still cannot obtain other APP private data which improves the security of Android devices. The process is done automatically, without application developers and users to participate in, don't need to change the habit of development and use.

Key words: privacy security; Xposed; encrypt on-the-fly; Android

0 引 言

随着智能手机的普及,人们对手机的依赖不断加大。各种 APP 出于功能上的需求会悄悄在手机上存储各种数据文件,其中可能会包含使用者的隐私信息,加密技术无疑是保护这类数据的有效方式。透明文件加密技术多年的发展表明,这是一种较好的隐私保护安全机制。

透明文件加密技术按实现的位置可以分为用户态的实现与内核态实现。早期用户态的透明加密主要是通过钩子函数来 Hook 文件打开与关闭的操作,在打开文件时,将磁盘上的密文解密到一个临时的隐藏文件中,然后将隐藏文件返回给应用,用户对文件的修改会反馈到隐藏文件中,当关闭文件时系统再将隐藏文件整体加密,替换磁盘上原先的密文,最后删除掉隐藏文件。这种静态加密的方法实现简单,但是每次都要对整个文件做加密解密操作,整体效率较低,同时隐藏文件的出现对整体安全性构成威胁[1]。内核态的加密系统有的通过堆栈式文件系统在 VFS 与底层操作之间完成加解密[2],也有通过 LSM 框架 Hook 相关内核函数[3]等实现方法,内核态的加密技术在速度与稳定

收稿日期:2015-08-15 **修回日期**:2016-01-05 **网络出版时间**:2017-01-10
基金项目:国家自然科学基金资助项目(61332010)
作者简介:朱天楠(1988-),男,硕士,研究方向为移动设备安全。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20170110.1010.024.html

性上具有优势[4],但是出于安全性考虑,目前有些 Android 移动设备不具备动态加载驱动模块的功能。这将使得内核态的加密技术难以在现有设备上推广。

由于移动处理器性能增速远快于磁盘性能的增长,用户态上的加密技术对性能影响的权重将会逐渐减少。同时文中使用 SharedUserId 和 APP 的签名作为标识来选取密钥,而从 Android 上层获取这些信息比较方便。因此,使用 Xposed 框架来实现用户态的动态透明文件加解密系统。该系统能够自动为各个应用生成不同的加密密钥,并加密数据。整个过程都对用户与开发者透明。

1　现有 Android 平台的数据安全服务

在数据加密方面:自 Android3.0 之后,谷歌通过 deivce-mapper 提供了磁盘加密机制 dm-crypt[5],deivce-mapper 是 Linux 中的一个框架,Linux 中的 RAID、LVM 等功能都基于该框架。它可以将一个虚拟的块设备映射到一个或多个实际的物理设备,在映射过程中,可以对交互的数据进行修改[6]。该技术在 Android 中可以用于全盘加密,下文提到的磁盘数据校验中也依赖该框架。

dm-crypt 磁盘加密技术主要是对整个分区,例如把/data 对应的分区进行加密,在系统启动时 init 进程通过挂载/data 目录来判断磁盘是否加密。如果分区加密则会提示用户输入密码,并重启 Android framework,重新挂载分区。这种方式控制粒度较粗,一旦解密,所有 APP 都能像以前一样访问/data 目录,因此无法防止恶意 APP 获取其他 APP 的私有数据。

在数据完整性方面:为了应对 RootKit 等攻击,Android 在 4.4 中引入了 Verified Boot 机制来保证启动磁盘的完整性,其基于 Linux 内核 dm-verity(device-mapper-verity,3.4 版加入到 Linux 中)机制。在启动过程中会对块设备进行完整性校验。校验过程中使用的 RSA 公钥存储在启动分区中。校验出错将会返回 I/O 异常。dm-verity 通过构建一个多叉树状结构来保存对应分区中所有块的哈希值,这棵树的叶子节点表示物理设备上各块的哈希值,中间节点保存其子节点的哈希值,这样物理设备上任何数据的变化都将导致该树子节点哈希值的变化,同时这个变化会影响到其所有祖先节点,最终改变根节点的哈希值。这样只需比较根节点的哈希值就可以知道数据是否被篡改[7]。相较于 I/O 操作,哈希计算所造成的迟延不是十分明显。dm-verity 对磁盘所进行的校验工作是由 Linux 内核完成的,因此首先需要保证 Linux 内核的完整性,防止其被篡改。通常移动设备制造商会在一块物理存储设备上固化一把校验密钥,在设备启动时可以通过 TrustZone 技术首先使用这把密钥校验 bootloader 和 kernel 的完整性。由于在可写的情况下,如磁盘挂载时间一类的元数据都将被系统修改,因此该技术要求被保护的磁盘只能以只读的方式挂载,Android 中主要用来保护/system 分区,而/data 以及外部存储设备这种本身就会被不断修改的分区将无法获得保护。

在数据访问控制方面:Android 继承并发展了传统 Linux 下的访问控制机制,将传统 Linux 下的用户的概念应用到 APP 上。UID 不再代表手机使用者,而是用来区分各个 APP,每一个 APP 获得一个 UID,这样传统 Linux 在对用户的"读-写-执行"权限控制机制就顺利地延续到了 APP 上[8]。同时组的概念用来分配系统资源,APP 要访问系统资源例如网络、摄像头、外部存储等,需要加入相应的组。通过这种机制各个 APP 自己的数据(主要指/data/data/下的数据)相互隔离开来。但是一旦恶意应用获取到 root 权限,还是可以访问其他应用的数据。

2　应用标识的选取

为了实现各 APP 的数据使用不同的密钥加密,就需要选取一个合适的标识,来区分哪些 APP 不能共享数据。Android 应用的 AndroidManifest.xml 中有一个称为 SharedUserId 的标识符(没定义的话为空)。对于具有相同的 SharedUserId,并使用相同的开发者证书签名的应用可以相互访问各自私有数据[9]。Android 系统在安装应用时,如果有多个具有相同 SharedUserId 但证书不同的 APP,只有最初的那一个能安装成功。因此选取 SharedUserId 与开发者的证书信息作为标识符合该系统的需求。

3　Xposed Hook 原理

在 Android 中,所有的 APP 进程都通过 Zygote 进程创建。Zygote 在启动过程中会加载部分资源,这样通过其创建的所有 APP 都将继承这些资源,而无需重新加载,从而减少了启动时间。Zygote 进程实际上是在系统启动过程中/system/bin/app_process 程序通过系统调用更换名称得到的[10],为此,Xposed 将自己定制的 app_process 替换到目标设备中(需要 root 权限),通过 Hook Zygote,从而达到 Hook 所有 APP 的目的。Xposed 在这个定制的 app_process 中添加了 XposedBridge.jar 库文件,系统会将需要 Hook 的方法指向 XposedBridge 中的本地方法 xposedCallHandler,而后 xposedCallHandler 会调用 handleHookedMethod 来回调对应的 beforeHookedMethod 和 afterHookedMethod(分别在被 Hook 方法前后调用),在这两种方法中可以修改传给被 Hook 方法的参数及其返回值。

4 加密算法的选择

现代加密算法按照加解密密钥的类型可以分为对称加密与非对称加密两大类[11]。由于非对称加密算法计算量相对较大,出于性能上的考虑,该系统采用对称加密算法。对称加密技术又可分为块加密与流加密技术。文中通过对比常见的DES、AES、RC4来选取适合的方案。

DES(Data Encryption Standard)是一种块加密算法,它每次使用64 bit的密钥(除去校验位实际只有56 bit),以64 bit(8字节)数据为单位加密数据,加密后的密文块也是64位。

AES(Advanced Encryption Standard)又称为Rijndael算法,它使用的数据分组长度为128 bit,密钥长度可以为128/192/256 bit[12]。

RC4是一种密钥长度可变的流加密算法[12],该算法实现十分简单。通过密钥来初始化一个大小为2^n的S-Box(n一般为8),在对明文进行加密的同时,S-Box也在不断变化,这样即使出现相同字符,解密结果也不一定相同。作为流加密算法,可以使得密文长度与明文长度相同。

4.1 从安全性角度进行比较

暴力破解作为最直接的攻击方法,适用于各种加密算法。因此保证数据在一定时间内难以破解是评估密码算法的一项重要指标。相比于RC4和AES,默认的DES算法密钥有效长度只有56 bit,较易被攻击[13]。2006年,鲁尔大学与基尔大学用FPGA开发出硬件破解设备COPACOBANA,它主要针对64位以内的密钥进行暴力破解。2007年,该设备平均6.4天就可以破解一个DES密钥[14]。相对来说,AES和RC4的密钥所对应的空间要远远高于标准的DES算法,暴力破解成本较高。

4.2 从加密前后数据长度变化角度进行比较

块加密算法,一般都是一次对固定长度n的平文进行加密操作得到密文。DES算法每块数据长度为8个字节,加密后得到8个字节密文,AES算法一般使用16字节的平文数据块,加密后得到16字节密文。使用块加密方法如果平文长度不足n,一般会进行填充操作,从而使得加解密前后数据长度发生变化。而如RC4这样的流加密算法,可以每次以一个字节为单位进行加密处理,不会改变数据长度。

4.3 从误差传播角度进行比较

存储介质在使用过程中都会渐渐出现数据的损坏,因此,在设计透明文件系统时就需要考虑存储介质上一个存储单位的损坏会对解密后明文的结果造成的影响。

块加密的工作方式有多种,如ECB(Electronic CodeBook)、CBC(Cipher Block Chaining)、PCBC(Propagating Cipher-Block Chaining)、CFB(Cipher-FeedBack)、OFB(Output-FeedBack)等[15]。其中ECB最为简单,其工作原理如图1所示。使用同一把密钥分别对各块数据进行加密,各块数据相互独立。因此,磁盘上密文的损坏只会影响到其所在的数据块,不会将这种影响扩散到其他数据块中。

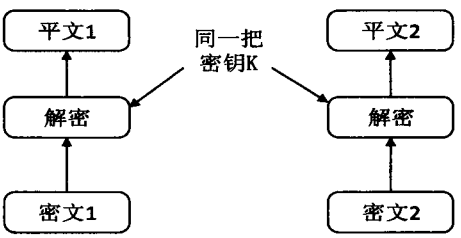

图1 ECB解密原理图

CBC和简单的CFB在解密过程中都是用前一块密文作为初始化向量参与到解密过程中,因此一个字节的损坏会影响到自己以及后一块数据的解密,对其他数据没有影响。CFB解密过程如图2所示。

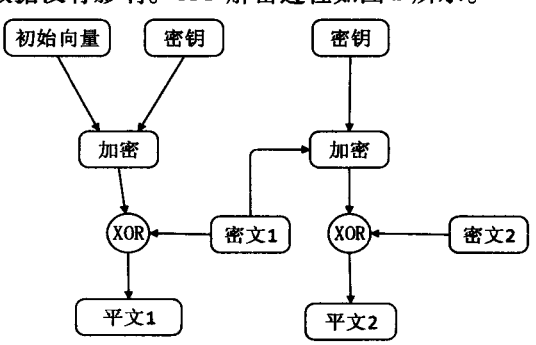

图2 CFB解密数据原理图

PCBC这种方式下密文中出现的误差能够尽可能影响到后续结果,这与透明文件加密的需求不符。

OFB可以让密文与平文同一位置的数据位同时翻转。这个特性利于奇偶校验,密文出错时,在解密前就可以验证出问题。但一位数据的损坏,同样会造成一个数据块的解密失败。

RC4算法在解密过程中S-Box可以不受密文或平文的影响,存储介质上一个字节的损坏不会影响到其他字节。

4.4 从数据加解密速度进行比较

为了比较三种加密算法的加解密速度,使用一台配备Exynos4210(双核 频率1.5 GHz)CPU的Android设备,分别处理1 k,10 k,100 k,1 M,10 M数据,测量消耗时间。其中DES和AES使用Java中自带库来实现(使用ECB/PKCS5Padding),RC4实现简单,实验中自己实现相关加解密函数,得到的结果如表1所示。

综合上述结果,选取RC4算法实现透明文件加密系统。由于RC4的S-Box随着加密过程不断变化,如果要往一个长度为n的文件追加内容,首先需要对S-

Box 的初始化进行 n 次变换。在处理较大文件时效率低,也无法发挥多核 CPU 的优势。为此,以 4 k 为单位划分文件数据,每到 4 k 数据就将 S-Box 复位,这样每 4k 数据相互独立,可以多线程处理。

表 1　各加密算法在实验平台上的性能测试　ms

算法	1 k	10 k	100 k	1 M	10 M
DES	0.245	2.42	22.8	213	2 037
AES	0.493	2.65	18.5	160	1 441
RC4	0.125	1.16	9.1	147	733

5　系统整体框架

系统采用应用中的 SharedUserId 以及对应的签名信息作为 APP 的标识,通过主密钥加密处理后,得到加解密数据的实际密钥。这样 APP 无法访问通过其他标识加密的数据,可以在不影响用户与原有应用的情况下为系统提供透明的文件加密服务。其中数据加解密功能是通过 Xposed 框架来 Hook 各个 APP 中 Java 文件读写操作,工作原理如图 3 所示。

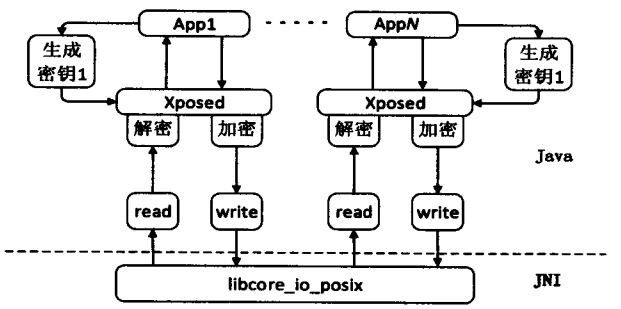

图 3　透明文件加密工作原理图

6　系统实现

6.1　加密策略

从 Android 设计的角度来看,其对各个 APP 数据访问的限制都是针对内部存储的,因此一般开发者默认将私有的数据保存在自己的存储区中,将共享的或无需加密的数据放在外部存储空间中,因此系统默认为应用在/data/data 下的文件提供加解密服务,采用由如图 4 所示的加密策略。

6.2　获取 SharedUserId 以及签名信息

在 Xposed 模块里可以通过 handleLoadPackage 传递进来的参数获取当前应用的名称,再通过 PackageManager 获取相关的 SharedUserId。但再次之前需要获取当前的 Context 对象才行,但是通常 Xposed 的 Hook 模块是在一个单独的类,本身没有当前程序的 Context 对象。所以需要 Hook 当前 APP 的 getApplicationContext 方法,再保存其返回 Context 对象,相关代码如下:

```
findAndHookMethod("android.content.ContextWrapper", lpparam.classLoader, "getApplicationContext", new XC_MethodHook
())
{
    protected void afterHookedMethod(MethodHookParam param)
throws Throwable {
    storedContext=(Context) param.getResult();
    .....
    if(pInfo.packageName.equals(packageName)){
    sharedUserId=pInfo.sharedUserId;
    signature=pInfo.signatures[0].toCharsString());
    }
    .....
});
```

然后通过 SharedUserId,签名信息,以及主密钥计算出加密该 APP 数据的实际密钥。

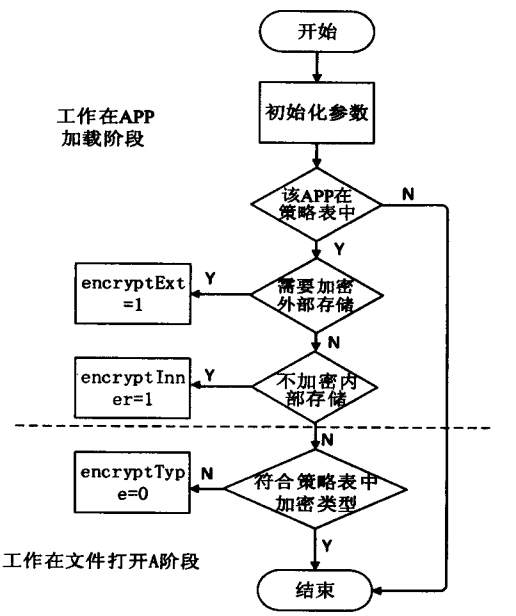

图 4　加密策略

6.3　Hook 相关方法

6.3.1　Hook 文件操作的构造方法

由于 Java 中与文件读写相关的类很多,这里以 FileInputStream 和 FileOutputStream 为例(下同)。Java 中对文件操作没有显示的 Open 操作,实际上在 FileInputStream 之类的构造方法中会调用相关的 Open 函数,为此 Hook 相关构造方法即可获得与 Hook open 方法类似的效果。

在构造方法中,首先判断这个文件是否需要加密处理,如果需要的话在全局的 HashMap 中保存一个以构造函数创建的对象为 key,index 为值的数据项。index表示目前创建的对象在文件中将要读写的偏移量,当发现文件长度比 index 小时说明有进程或其他对象对文件内容做过删减操作,当前的 S-Box 需要更新,同时也用来辅助实现按 4k 分割数据复位 S-Box 的功能。

6.3.2　Hook read 相关方法

FileInputStream 的 read 相关方法主要有 read(),

read(byte b[]),read(byte b[],int off,int len)。其中,前两种相对简单,这里以第三种为例。

解密操作主要是在read方法从磁盘上读到文件之后进行的,因此需要实现XC_MethodHook中的afterHookedMethod方法。首先通过afterHookedMethod方法的参数param.thisObject得到实际调用read方法的对象,然后从全局HashMap中得到对应的index值,在后续读过程中按需复位S-Box。由于这个read方法是将读取到的内容保存到参数b数组中,因此还需要通过param.args获取到该数组和对应偏移量。虽然参数中有要读取的长度信息len,但实际读取的数据量可能比len要小,所以还要通过param.getResult()方法得到read的返回值即实际读取到的数据大小。然后结合index,b,off,实际数据量,对b中的数据进行加密。最后更新HashMap中的index。

6.3.3 Hook write相关方法

加密操作是在平文数据发送给实际write方法之前进行,因此要实现XC_MethodHook中的beforeHookedMethod方法。具体实现与Hook read中所述类似。此外write操作时要注意同步问题,例如当对象A以非追加的方式打开并写入文件,相当于清空数据,此时S-Box应该复位,如果没有同步机制的话,可能对象B正在通过之前计算的S-Box加密数据然后调用原生的write写入文件,导致使用错误的S-Box加密数据。

6.3.4 Hook skip方法

skip方法用于在读文件的过程中跳过一部分数据,该系统实现beforeHookedMethod方法,由于skip(long n)不一定能跳过n个字节,所以要用原skip函数的返回值来更新index和S-Box。

6.3.5 Hook close方法

程序调用close方法后会释放相关的文件资源,通过实现beforeHookedMethod方法,删除index等自己创建的资源。

6.4 实验结果

该系统可以对Java层的应用实现透明的文件加解密,保护数据安全。为测试该系统对Android读写性能的影响,使用在一台CPU为Exynos4210(双核cpu频率1.5 GHz)的设备为实验平台,对比使用透明加密与没有使用透明加密时在性能上的区别。

实验结果如表2所示,加解密数据会对读写性能造成一定的影响。但当应用不是频繁读写磁盘数据时,对使用者使用体验影响不明显。

表2 透明加密对性能的影响 ms

文 件	1 k	10 k	100 k	1 M	10 M
未加密读	3	6	19	107	485
未加密写	4	9	33	185	1 603
加密读	5	11	59	211	1 296
加密写	6	17	214	391	2 027

7 结束语

透明文件加密可以有效保护用户数据安全,基于Xposed的透明文件加密方案灵活性较高,可以方便地部署在已经发布的移动设备中,使用该系统可以在一定程度上保证用户隐私安全。虽然在当前移动设备硬件条件下,性能会有一定损失,但目前移动GPU厂商纷纷将异构计算引入到移动设备中,多线程操作加上硬件性能的提升将减缓加密造成的性能损耗。

参考文献:

[1] 赵铭伟,毛 锐,江荣安.基于过滤驱动的透明加密文件系统模型[J].计算机工程,2009,35(1):150-152.

[2] Halcrow M A. eCryptfs:an enterprise-class encrypted filesystem for Linux[C]//Proceedings of the 2005 Linux symposium.[s.l.]:[s.n.],2005:201-218.

[3] 陈莉君,于运超.基于LSM的轻量级透明加密设计与实现[J].西安邮电大学学报,2014,19(1):78-81.

[4] 王全民,周 清,刘宇明,等.文件透明加密技术研究[J].计算机技术与发展,2010,20(3):147-150.

[5] Müller T,Spreitzenbarth M. Frost[C]//Applied cryptography and network security. Berlin:Springer,2013:373-388.

[6] 宋振华.虚拟化技术中的存储管理问题研究[D].合肥:中国科学技术大学,2010.

[7] 艾 祝.基于iSCSI的数据完整性研究与实现[D].兰州:兰州大学,2014.

[8] 吴 倩,赵晨啸,郭 莹.Android安全机制解析与应用实践[M].北京:机械工业出版社,2013:26-29.

[9] Delac G,Silic M,Krolo J. Emerging security threats for mobile platforms[C]//Proceedings of the 34th international convention.[s.l.]:IEEE,2011:1468-1473.

[10] Shabtai A,Fledel Y,Elovici Y. Securing Android-powered mobile devices using SELinux[J]. IEEE Security & Privacy Magazine,2010,8(3):36-44.

[11] 张晓丰,樊启华,程红斌.密码算法研究[J].计算机技术与发展,2006,16(2):179-180.

[12] Singhal N,Raina J P S. Comparative analysis of AES and RC4 algorithms for better utilization[J]. International Journal of Computer Trends and Technology,2011,79(14):177-181.

[13] Wiener M J. Efficient DES key search[D]. Canada:Carleton University,1994.

[14] Schimmler M,Wienbrandt L,Guneysu T,et al. COPACOBANA:a massively parallel FPGA-based computer architecture[M]//Bioinformatics-high performance parallel computer architectures.[s.l.]:CRC Press,2010:223-262.

[15] 吴文玲,冯登国.分组密码工作模式的研究现状[J].计算机学报,2006,29(1):21-36.