**针对Android文件管理器删除数据的安全性研究**

摘 要 随着移动互联网的飞速发展，智能移动终端已经成为人们个人隐私数据的重要载体之一。但是对于目前的移动智能终端来说，系统所提供的数据删除接口并没有达到预想中的安全性。在现有的Android设计模型特别是安全模型中存在一些难以回避的缺陷，正是这些缺陷导致了这些不易察觉的残留数据存在风险，而这些风险被以往的安全分析所忽略，甚至主观的认为这些风险不存在。在这种情况下，攻击者能够利用安全防范的疏忽发起攻击，进而更为严重地威胁到设备相关使用者的隐私和数据安全。以Android系统为目标，利用AOP切片技术，对存在于Android系统中的多种文件管理器删除数据的方法进行了研究，研究表明其中部分方法存在着严重地安全隐患。着重分析了Android用户比较常用的通过文件管理器删除数据的操作。实验成功地从Android设备中恢复出大量的通过上述方式删除的数据，证明了目前的文件管理器对数据的删除是不安全的。提出名为PureEnc的解决方案来保护Android设备中的数据以对抗数据残留问题。PureEnc可以实现对Android应用程序的自动修改，以达到保护数据的目的，不需要对Android系统进行修改，可以很方便地部署并有效地保护应用程序的数据，以对抗数据残留漏洞。

关键词 Android，数据删除，数据残留，文件管理器，AOP

1 引言

移动只能终端设备的流行改变了人们处理个人信息的方式。越来越多的人选择使用手机等移动设备处理个人事务和工作事务。移动应用开发者们提供了各种各样的程序来帮助人们更好地处理生活中使用到的各类信息，比如使用社交应用来和朋友保持联系，使用银行应用来处理自己的银行事务，使用照相机应用来记录生活中的美好瞬间。这种趋势的出现导致了移动智能终端设备中存储了越来越多的和用户紧密相关的个人隐私数据，同时也导致了个人隐私数据泄露的事件频频发生。

作为目前最流行的移动智能终端操作系统，在很长一段时间内，Android都是个人隐私数据泄露事件发生的主要源头[1]，导致这种情况出现的主要原因在于Android系统和应用程序对于个人隐私数据的处理存在安全隐患，尤其是在进行数据删除操作时，没有满足预先设想的安全性。虽然在闪存介质[2]上的数据残留问题已经被研究得很多了，但是在Android系统中，这个问题仍然没有得到文件系统和操作系统的重视。在Android应用程序的开发文档中，并没有针对该问题进行特殊的说明，也没有警告开发者使用一般数据删除接口时可能会存在的安全问题，这导致了绝大部分Android应用程序都使用了不安全的数据删除接口处理敏感的数据，最终导致了数据泄露的发生。

本文旨在对Android系统上使用文件管理器进行数据删除的接口的安全情况进行深入的调查研究，并针对可能出现的安全问题制定合理的解决方案。

2 背景

2.1 Android数据存储

和传统的桌面电脑不同，搭载Android系统的智能移动终端往往选择闪存作为自身的存储介质。对于闪存来说，最重要的一大特性是闪存上所有的数据读写都是以块为单位的。一般情况下，一个数据块的大小是几百到几千个字节。在进行数据删除操作时，即使要删除的数据只是一个块的一部分，也需要将整个块上的全部数据都擦除。因此，为了提高磁盘读写的效率，当删除数据时，闪存的控制芯片不会真正把数据块里的数据全部置0，而是将这个数据块标记为未使用，当下一次需要向该数据块内写入数据时，才真正把数据擦除，这就导致了在闪存介质上，被删除的数据会继续残留在介质上一段时间，这个时间和介质的大小以及磁盘读写的频率有关[3]。

2.2 数据擦除

一般来说，Android系统中有两种主要的数据擦除方式。第一种是通过调用系统API直接删除文件，通过这种方式删除的文件，其内容会继续残留在磁盘上。通过对磁盘进行全盘镜像提取，就可以获取这些残留的数据内容，之后通过一些取证手段就能够恢复出原有数据的内容。在Android系统中，绝大多数的数据擦除操作都属于这种方式，虽然在系统中呈现的接口不尽相同，但是在底层均是调用Linux删除文件的系统来实现文件的擦除。Android系统中另一种数据擦除方式就是恢复出厂设置，通过触发系统设置界面中的相关接口，Android系统会重启到Recovery模式。在Recovery模式下，系统会自动对整个userdata分区进行格式化。一般说来，恢复出厂设置的过程中，系统会调用闪存介质底层接口来重写整个分区，确保分区上所有的数据都被安全地擦除。

2.3 Android数据保护

Google从Android 3．0开始在Android系统中加入了可供选择的全磁盘加密功能，全磁盘加密功能基于Linux dm-crypt特性修改而来。用户可以在系统设置中选择为自己的设备开启全磁盘加密，开启了全磁盘加密后，每次系统启动前都需要用户输入用于解密的根密钥。全磁盘加密是一次性的不可逆的过程，这意味着一旦用户开启了全磁盘加密，除非格式化整个设备，否则没有其他的方法来解除磁盘的加密状态。

在Android系统中，为了减少全磁盘加密对系统性能的影响，Google去除了dm-erypt方案中所有用于对抗取证的安全策略，这在一定程度上削弱了全磁盘加密的安全性。另一方面，dm-crypt方案的on-the-fly eneryption/decryption特性决定了全磁盘加密无法对抗来自系统内部的攻击，当系统已经处于运行状态时，磁盘上所有的数据将不再收到全磁盘加密的保护，拥有数据访问权限的任何用户和App都能够读取解密过的数据。在Android设备root非常普遍的今天，全磁盘加密并不能有效地保护磁盘数据的安全。

此外，由于方案设计上的一些缺陷，导致了全磁盘加密并没有带来预想中的安全性。Android系统中的全磁盘加密密钥由用户的锁屏PIN码生成，同时加解密磁盘所需要的加密参数(encryption parameters)都存储在本地磁盘对应分区的尾部的crypto footer结构中，通过解析这个结构体，攻击者就能够获得对全磁盘加密保护的磁盘发起暴力穷举攻击所需要的参数，即穷举PIN码。同时，作为一个日常使用的手持设备，绝大多数Android用户不会使用过于复杂的PIN码来锁定自己的手机，对于Android默认的6位PIN码来说，任何普通的家用电脑都能在1～2小时内完成破解，而如果使用专业的大型计算机来进行攻击，可破解的PIN码范围将大大提升。在Android4.4中，Google升级了FDE所使用的密钥生成算法[4]，导致进行暴力破解密钥所花费的时间有了大幅度的提升，但是FDE本身的机制并没有变化，仍然为攻击者留下了可能的攻击窗口，过于简单的PIN仍然能够在可以接受的时间内被碰撞出来。

全磁盘加密的这些不安全特性会在磁盘数据残留问题的影响下被大大放大，对于被遗弃或意外丢失的Android设备来说，即使开启了全磁盘加密，也无法阻止攻击者利用磁盘上残留的数据来进行上述的攻击，甚至发现在一些使用了第三方recovery的设备上，由于recovery开发者的疏忽，想要格式化全磁盘加密过的磁盘都变得非常困难。

若要攻击开启了全磁盘加密的设备，首先需要将磁盘的相关分区提取出来，包括存储了加密数据的data分区和存储了crypto footer的特殊分区；之后解析crypto footer获取计算master key所必须的参数；然后使用一些暴力破解脚本对用户的PIN码进行暴力穷举，尝试使用不同的PIN码去解密data分区的第一个区块，如果解密后的数据符合ext4文件系统的标准格式，那么说明找到了正确的PIN码，之后就可以使用这个PIN码解密整个data分区，得到明文的数据。值得一提的是，磁盘上发生的数据残留是存在概率的，并不是每一个字节都会残留在磁盘上，这意味着从磁盘上提取到的残留信息可能破损的、不完整的，如果这些残缺恰好发生在data分区的头部或是保存crypot footer的部分，则关键数据的缺失将导致暴力破解无法进行。

基于上述的原因，本文认为传统的Android全磁盘加密方案无法达到其预想中的安全强度，也无法在存在数据残留的情况下保护磁盘数据的安全。

3 Android数据删除的缺陷

在一个安全的操作系统中，对数据安全的保护应该贯穿数据的整个生命周期，从数据生成开始，覆盖数据的运算、传输、存储，最后是数据的删除。上述阶段中任何一个阶段出现安全问题，数据的安全就会遭到破坏。在Android系统中，我们将数据可能被删除的方式归纳为以下4种：

①常规数据删除；

②应用程序数据清除；

③应用程序卸载；

④恢复出厂设置。

上述4种数据删除方式，除了常规数据删除操作，其他3种都不受应用程序自身控制，也就是说应用程序无法对这4种数据删除方式作出主动的安全性补充，本文只探讨前面3种删除操作，并进行相应的恢复。我们的研究表明，前面3种数据删除操作均存在安全问题，以下会详细讨论这3种安全缺陷。

3.1常规数据删除缺陷

根据Android开发者文档，Google推荐开发者在开发应用程序时将需要使用的文件存放在userdata分区的私有目录中。私有目录受到Android沙盒机制的保护[3]，仅文件的创建者自己自己能够访问这些文件。然而沙盒并不能保证这些私有文件在数据的整个生命周期中的安全。根据上文所述的闪存介质和文件系统的缺陷，当文件删除时，文件系统仅删除对应的文件节点，而不是删除删除文件的内容。

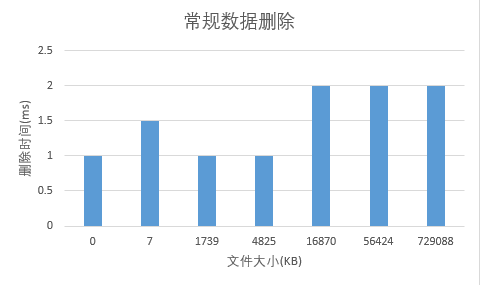


图 1 删除数据时间分析

利用AOP技术监测文件管理器对不同大小的文件进行常规数据删除花费的时间，分析删除花费时间与文件大小得知，花费时间并不会随着文件增大而变长，以上统计数据也间接地说明了删除文件时只删除了文件节点。因此，当私有目录的数据被删除后，就不再受沙盒的保护，一旦终端发生遗失或是被攻击者取得root权限，这些残留在磁盘上的数据都有泄露的风险。

3.2 不安全的应用程序数据清空

Android系统和一般的文件管理器为用户提供了“清除应用程序数据”的功能，用户可以在应用程序管理界面或者文件管理其中找到这个清除数据的接口。该功能能够帮助用户清除指定应用程序在userdata下私有目录。为了分析该方式删除数据的安全性，对几款文件管理器进行了源码分析，发现该数据删除功能调用了Android系统底层API：

|  |
| --- |
| public int delete(String table, String whereClause, String[] whereArgs) {  acquireReference();  try {  SQLiteStatement statement = new SQLiteStatement(this, "DELETE FROM " + table + (!TextUtils.isEmpty(whereClause) ? " WHERE " + whereClause : ""), whereArgs);  try {  return statement.executeUpdateDelete();  } finally {  statement.close();  }  } finally {  releaseReference();  }  } |

可以看出，“清除应用程序数据”这个功能在底层会调用executeUpdateDelete()这个函数删除数据库中的数据，通过查询Linux手册发现这个函数并没有针对数据残留问题采取任何缓解措施，也就是说通过该函数删除的数据都会出现数据残留问题。

4 PureEnc

上文分析了Android平台多个数据删除功能的安全性，发现绝大多数的数据擦除操作都会导致数据残留现象的发生。如何在这种情况下保护上层应用程序数据的安全变成了一个急需研究的问题。数据的生命周期从数据生成开始，直到数据被删除才结束。一个安全的数据保护方案应该完整地覆盖到数据的整个生命周期。通过前文的分析知道，在Android平台上，底层文件系统无法保证数据的安全擦除。因此，为了对抗必然存在的数据残留漏洞，提出名为PureEnc的Android平台应用程序数据保护技术，PureEnc是一个根植于系统上层的数据保护方案，不依赖于底层的文件系统支持，能够有效地保护数据的安全。

在保护数据的同时，PureEnc对应用程序开发者和用户透明，不影响应用程序的正常使用。PureEnc通过hook应用程序所使用的文件读写API，在数据被写入磁盘前进行自动加密，以此保证数据不会被明文保存到磁盘上，从而规避数据残留漏洞带来的影响。在对加密文件进行读取时，PureEnc会自动地对文件内容进行解密，从而保证不影响程序的正常使用。4.1 设计

通过上文的分析，对于应用程序来说，Android底层的文件系统是不安全性、不可信的，闪存介质上的数据残留问题不可避免，需要设计一个对于用户透明的应用级数据保护方案，该方案中数据的安全不会受到底层不安全性的影响。

相比修改Android系统或是将系统升级到更加安全的版本，通过修改应用程序来获得更高的安全性是一个更容易实施的方案。因此本文设计了PureEnc，通过修改应用程序的文件读写接口，对写入磁盘的数据进行加密，保证数据不会以明文的方式出现在磁盘上，以此来对抗数据残留漏洞。对于不同的应用程序使用不同的加密密钥，密钥仅出现在应用程序的内存中，使用后立刻销毁。同时，PureEnc支持应用程序对需要加密的文件进行配置，确保只加密需要保护的文件，保证应用程序的运行效率不受到影响。当使用了PureEnc的应用程序被卸载时，残留在磁盘上的都是加密后的密文，在无法获得密钥的情况下，攻击者无法从这些密文中恢复出任何有用的信息。

除了主动加密数据之外，PureEnc还提供针对共享文件的强制安全擦除功能。在Android系统中，很多应用程序之间会互相共享数据，比如照相类应用会在Sdcard上共享一个用来存放照片的文件夹。在这种情况下，使用独立密钥对数据进行加密会导致应用程序之间无法共享数据，对应用程序易用性造成了破坏。为此专门针对这些文件提供强制安全擦除功能，PureEnc通过Hook接管系统的删除API，在文件删除前使用正确的安全擦除算法对数据进行多次擦写，保证尽可能少的数据残留在磁盘上，PureEnc架构图如图1所示。

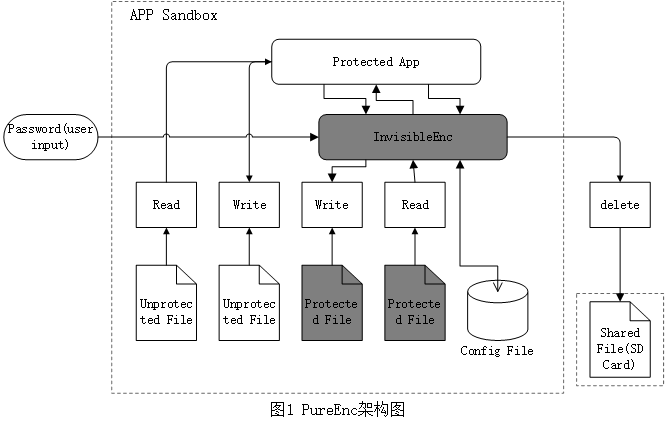


图 2 PureEnc 架构图

4.2实现

PureEnc是为Android应用程序开发者设计的，应该在应用程序的开发阶段被部署在应用程序中。通过在代码中部署PureEnc，应用程序可以获得加密所有文件读写的能力，同时不需要应用程序申请任何的额外权限。对于应用程序开发者来说，部署PureEnc仅仅需要在应用程序的代码中添加一行代码，用以加载我们设计的．SO库到应用程序的内存中，这个．SO库在加载后会自动完成PureEnc后续的所有工作，对应用程序开发者来说非常易于部署。搭载了PureEnc的应用程序在使用上和一般的应用程序没有任何的不同，用户不会感觉到有任何的异常，只不过这个应用程序写入磁盘的所有数据都已经被PureEnc加密了。

对于PureEnc来说，我们保护的文件范围限制在那些在应用程序安装之后才会被创建或者修改的，对于那些在应用程序安装前就已经存在于应用程序安装包里的文件以及在安装后不会被修改的文件，认为这些文件的机密性在应用程序安装包传输时就已经被破坏了，在安装后再进行保护不仅不能提升安全性，反而会增加本文方案部署的复杂性。PureEnc所使用的最主要的技术是inlinehook。

Hook 可以拦截指定函数或者某条汇编指令的功能，针对函数头注入got hook。而inline hook可以直接针对某个函数特定位置进行hook。它可以直接修改hook位置的指令，让其跳转到桩函数，在处理完桩函数之后回调如原指令继续执行。在这个过程中需要注意跳转指令的构建、inline hook指令的覆盖顺序等过程具体的执行过程如图3

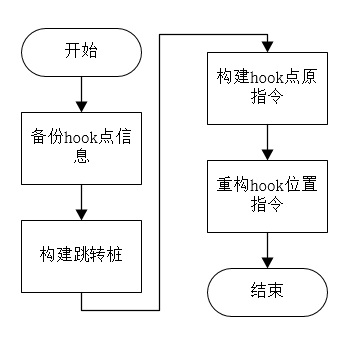


图 3 Hook执行过程

当PureEnc被加载的时候，会自动地hook应用程序内存空间中所有的文件写函数，然后将这些函数头部的部分指令修改为一系列跳转指令，使得每一次对这些函数的调用都会先跳转到事先准备好的预处理函数中，在预处理函数中对数据进行加密，加密完成后才会调用原有的文件写函数将密文写入磁盘上。

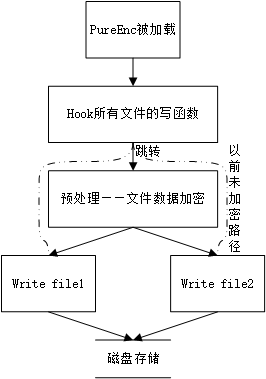


图 4 对文件加密写入

对于文件读函数的处理和写函数类似，只不过这次的预处理函数会在读出密文后对密文进行解密，将明文返回给函数调用者。

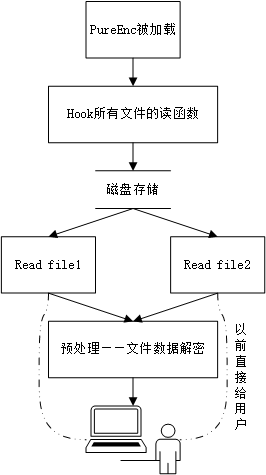


图 5 对文件解密读出

当某个文件被删除时，和该文件相关的密钥就被系统自动抛弃，此时残留在磁盘上的数据将无法被任何人解密。在Android平台上，绝大部分的文件读写API在底层依赖于libc中的write/read函数，所以在我们的原型系统中hook了这两个系列的函数。实验中，通过hook这两个系列的函数就能够实现本文的目标。

在部署PureEnc的过程中，需要考虑的另一方面是加密方案的选取和密钥管理。

密钥管理，对于每一个加载了本文保护方案的App，在App启动时都会要求用户输人对应的根密钥。根密钥在App第一次启动时由用户设置，根密钥是系统对文件进行加解密操作的基础，输人错误的根密钥将无法启动应用程序。根密钥只存在于内存中，使用后立刻从内存中抹去，确保了根密钥的安全性。对于被保护的每一个文件，都会由系统自动地从根密钥生成一个不同的文件密钥，保证了每一个文件之间和每个应用程序之间安全的独立性。由于存在文件名被修改的情况，因此使用文件在磁盘上对应的inode号来区分不同的文件，而不是文件名。由于应用程序对文件的读写是不可预测的，每次从文件中读取或写入的数据在文件中的位置会发生变化，并且为了不让明文的文件出现在磁盘上，使用流密码对文件进行加密保护，确保每一个bit的数据都能够被正确地加解密，因此我们从每个文件对应的文件密钥生成一个密钥流，根据写入/读取数据在文件中的位置来使用密钥流进行正确的加密/解密操作。在整个加密/解密过程中，使用到的根密钥、文件密钥、密钥流均只存在于内存中，不保存在磁盘上，用后即焚，保证了所有密钥的安全性。整个方案中，文件系统上数据的安全性由成熟的密码学加密方案来进行保护，密钥的安全则基于根密钥的安全性，在我们方案的保护下，无论攻击者收集到多少残留数据，都无法将这些加密过的数据还原成有用的信息。而对于应用程序开发者来说，使用我们的方案所需要做的仅仅是在应用程序初始化时加载我们的hook框架，在Android应用程序开发中，仅需要一行代码就能够完成。同时所提保护方案也能够通过重打包技术保护那些没有源码支持的应用程序。4.3 实验

为了评估这套方案在真实设备上的使用效果，选取华为荣耀6手机进行了磁盘读写效率的测试。使用同一款Android APP FileManager，分别在正常情况和加载我们系统的情况下进行磁盘数据的读写。选取了4个大小不同的数据样本，分别是10KB，100KB，1MB和10MB随机生成的数据。对每一个数据样本，分别在正常情况和加载我们系统的情况下进行100次磁盘写入操作，选取20次操作的平均值作为该数据样本的写入时间。对于数据读取性能的测试则使用相同的数据样本，在完成数据样本的写入后，对4个不同的数据样本进行数据读取操作，同样是在两种情况下分别读取100次，取20次的平均时间作为该数据样本的读取时间。数据写入和读取实验的详细结果如表1所列。

表 1 PureEnc读写性能（ms）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 0.1MB | 1MB | 10MB | 100MB |
| 写 | 普通FileManager | 2.69 | 22.27 | 306.12 | 3454.57 |
| PureEnc | 6.81 | 61.16 | 777.75 | 7348.8 |
| 读 | 普通FileManger | 0.26 | 2.96 | 15.28 | 169.68 |
| PureEnc | 0.49 | 4.42 | 29.19 | 306.75 |

从实验结果中可以发现，我们的系统在文件写入方面会带来大约1．5倍的额外性能开销，在文件读取方面会带来约0．8倍的额外性能开销，考虑到在移动平台上，需要读写的数据量往往不是很大，在这种情况下，我们系统所带来的额外时间成本是非常低的，相较于该方案所提供的安全性，我们认为这样的额外时间成本是可以接受的。

为了评估所提数据安全擦除方案对文件删除操作性能的影响，设计了如下的实验方案。由于所提安全数据擦除方案所要保护的数据主要是保存在这杯中的公开资源，而这类公开资源文件最为常见的类型就是手机所拍摄的照片，对目前的移动设备来说，一张照片的大小不会超过10MB，因此选取了符合这类资源文件常规大小的4类不同大小的数据样本来进行实验，数据样本的大小分别是10KB，100KB，1MB和5MB。由于Andoroid底层文件系统在删除文件时的特殊性(仅删除文件索引节点，而不擦除数据)，因此Android系统在删除不同大小的文件时，所花费的时间是大致相同的。因此在这个实验中，选取安全擦除操作的绝对时间作为衡量性能的标准，而不是和常规操作的耗时进行对比。对于每一类数据样本，进行20次独立的删除操作，取20次删除所消耗时间的平均值作为安全擦除该数据样本的耗时。最终的实验结果如表2所列。

表 2 PureEnc数据安全擦除性能

|  |  |
| --- | --- |
| 文件大小 | 时间（s） |
| 10KB | 0.064 |
| 100KB | 0.350 |
| 1MB | 3.047 |
| 5MB | 13.007 |

从表2中的数据可以发现，使用了我们的安全擦除方案之后，在删除移动平台绝大部分数据文件时，所花费的时间均在可以接受的范围之内，删除一张正常照片需要的时间约为10s，这些时间主要损耗在Gutmann算法对数据的35次随机复写过程中，并不会为终端用户带来用户体验上的困扰。

为了测试使用了我们系统的APP卸载之后数据的保护效果，我们使用控制变量法进行对比。先对文件大小相同，但文件格式不同进行试验，利用使用了PureEnc系统的APP和未使用该系统的APP分别创建文件格式为.jpg、.png、.pdf、.doc、.ppt，大小都为1MB的文件各10个，之后卸载APP。利用数据恢复工具对APP创建的数据进行恢复，得到的统计结果如表3所列：

表 3 不同格式文件的恢复

|  |  |  |
| --- | --- | --- |
| 文件类型 | 使用PureEnc正常恢复数 | 未使用PureEnc正常回复数 |
| .jpg | 0 | 9 |
| .png | 0 | 10 |
| .pdf | 0 | 10 |
| .doc | 0 | 8 |
| .ppt | 0 | 9 |
| 平均恢复率： | 0% | 92% |

接下来对文件格式相同，但文件大小不同进行实验，同样利用使用了PureEnc系统的APP和未使用该系统的APP分别创建文件格式为.pdf，大小分别大约是1KB、10KB、100KB、1MB、10MB的文件各10个，之后卸载APP。利用数据恢复工具对APP创建的数据进行恢复，得到的统计结果如表4所列：

表 4 不同大小文件的恢复

|  |  |  |
| --- | --- | --- |
| 文件大小 | 使用PureEnc正常恢复数 | 未使用PureEnc正常回复数 |
| 1KB | 0 | 10 |
| 10KB | 0 | 10 |
| 100KB | 0 | 10 |
| 1MB | 0 | 8 |
| 10MB | 0 | 9 |
| 平均恢复率： | 0% | 94% |

从上面的实验数据中可以发现，文件的大小和文件的格式对文件恢复基本无影响，影响较大的是是否对文件进行了加密。使用了PureEnc系统的软件，在该软件删除之后，再次访问该软件产生的不同大小和不同格式的数据时，显示乱码或者是无法打开。对比没有使用PureEnc系统时，在该软件删除之后，再次访问该软件产生的不同大小和不同格式的数据时依然可以正常读取。本次实验证实了使用PureEnc系统对文件进行加密之后，即使攻击者拿到了残余数据，也无法恢复到原来的有效信息，这样很好地实现了数据删除之后的安全。

结束语 本文针对Android系统中的数据擦除操作的安全性进行了全面的分析，发现在目前的Android系统中，绝大部分数据擦除操作均会产生数据残留，可能导致数据的泄露。在这样的分析基础上，本文提出名为PureEnc的上层应用程序数据安全保护方案，能够有效地对抗数据残留问题。最后，本文将通过实验证明PureEnc不会影响系统的正常运行效率。

**Related Works：**

1、《基于 Android 的手机文件安全加密系统》。该系统是使用android平台使用AES、DES，MD5等安全算法，开发的一套手机安全加密系统。实现了文件加密，图像隐藏、数字签名、报文鉴别等功能。该系统由4大模块构成分别为文件加密、图像隐藏、数字签名和报文鉴别。在具体实现上采用分组加密，对明文按照以64位分组完成加密。

对比该系统我们设计的系统具有跟高的灵活性和可用性，它修改了底层的系统指令，在系统执行read或者是write操作时利用inline hook调用预处理函数进行加密和解密操作。

2《基于android平台智能终端文件加密的研究》。系统包含4个模块分别为文件加密、文件解密、文件编辑、文件删除。加密算法使用AES，该加密算法是一种代替DES算法的高级加密算法，加密数据块分组长度为128bit，必要长度最少为128为特点是运算速度快，安全级别高。

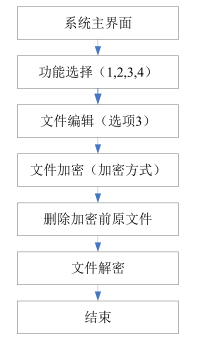


图 5 系统工作流程图

对比上一个系统它具有更高的安全性和更快的速度，和我们的设计依然具有局限性。它使得对文件的操作变得复杂。对一个文件的使用流程编程了解密-读取。而我们设计的系统在文件读写前使用预处理程序提高了处理的效率和可操作性。

3 《Android平台下文件透明加密技术的研究与实现》。透明加密技术是指在不改变用户操作习惯的情况下 完成文件的加密和解密过程。当系统监控到用户对受保护的文件进行读写操作是将文件进行解密操作，将明文传送该应用程序；当系统检测到用户对受保护的文件结束读写操作时，又将文件进行加密操作，将密文传送储存设备进行储存。实现这一技术主要有钩子透明加密技术和文件驱动透明加密技术。系统设计采用MVC框架，具有应用层模块以及内核层模块。其中内核层模块实现监控、加密/解密等操作。在具体的加密算法的选择上使用DES算法。

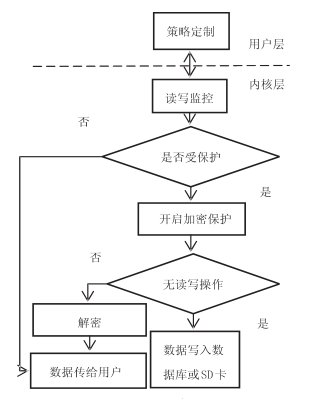


图 6 系统工作流程图

该系统与本系统的设计相似，实现了相似的功能。但我们的设计更接近系统的底层。因为使用了inline hook的技术所以在实现上较为简单，它能够识别对文件的操作并进行相应的文件加密解密操作。

[1]Baekes M，Gerling S，HammerC，et a1．AppGuard-enfordng user requirements on android apps[C]//International Confe-rence on Tools and Algorithms for the Construction and Analysis of Systems(TACAS’2013)．Rome：Springer Berlin Heidelberg，2013：543-54

[2] Flash memory[OL]．[2016-5-17]．http://en.wikipedia.org/w/index php?title=

Flash\_memory

[3] Bugiel S，Heuser S，Sadeghi A R. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies[C]//Usenix Conference on Security．2013 :131—146

[4]scrypt[OL].[2016-3-14].http://www.tarsnap.com/scrypt.html