(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104462952 B
(45)授权公告日 2017.11.10

(21)申请号 201410855681.X

(22)申请日 2014.12.31

(65)同一申请的已公布的文献号
申请公布号 CN 104462952 A

(43)申请公布日 2015.03.25

(73)专利权人 北京奇虎科技有限公司
地址 100088 北京市西城区新街口外大街28号D座112室(德胜园区)
专利权人 奇智软件(北京)有限公司

(72)发明人 宋振涛 符传坚

(74)专利代理机构 北京市立方律师事务所 11330
代理人 王增鑫

(51)Int.Cl.
G06F 21/51(2013.01)

(56)对比文件
CN 104077521 A,2014.10.01,
CN 101620529 A,2010.01.06,
CN 103279706 A,2013.09.04,
CN 104123162 A,2014.10.29,

审查员 刘义乐

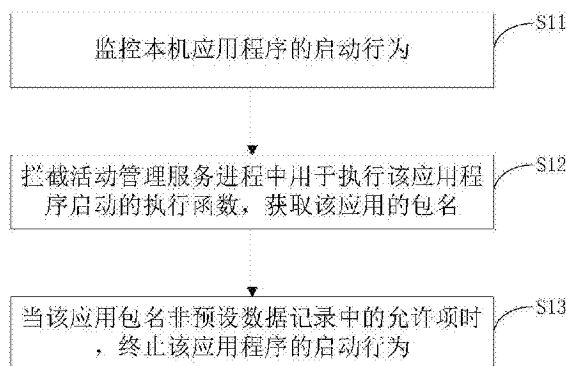
权利要求书2页 说明书9页 附图1页

(54)发明名称

一种禁止应用自启动的方法及装置

(57)摘要

本发明提供一种禁止应用自启动的方法,包括如下步骤:监控本机应用程序的启动行为;拦截活动管理服务进程中用于执行该应用程序启动的执行函数,获取该应用的包名;当该应用包名非预设数据记录中的允许项时,终止该应用程序的启动行为。相应的,本发明还提供一种禁止应用自启动的装置。本发明可以禁止应用的自启动行为以及顽固自启动行为,从而减少移动终端的耗电,同时提高运行速度。



CN 104462952 B

1.一种禁止应用自启动的方法,其特征在于,包括以下步骤:

监控本机应用程序的启动行为,所述监控本机应用程序的启动行为采用Hook技术;

拦截活动管理服务进程中用于执行该应用程序启动的执行函数,获取该应用的包名;

当该应用包名非预设数据记录中的允许项时,终止该应用程序的启动行为,所述数据库具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系,所述数据记录既包含允许自启动的应用的记录,又包含禁止自启动的应用的记录;或者,仅仅包含禁止自启动的应用的记录。

2.根据权利要求1任意一项所述的一种禁止应用自启动的方法,其特征在于,所述数据记录还包括用于记录反复自启动应用信息的顽固自启日志。

3.根据权利要求1所述的一种禁止应用自启动的方法,其特征在于,所述方法还提供一个用户界面,用于显示顽固自启日志记录的信息。

4.根据权利要求1所述的一种禁止应用自启动的方法,其特征在于,所述活动管理服务进程具体指ActivityManagerService所在的进程。

5.根据权利要求1所述的一种禁止应用自启动的方法,其特征在于,所述应用程序启动的执行函数具体指应用启动时ActivityManagerService调用的成员函数startProcessLocked()。

6.根据权利要求1所述的一种禁止应用自启动的方法,其特征在于,所述终止应用程序启动的具体方法为调用函数killProcess()。

7.根据权利要求1所述的一种禁止应用自启动的方法,其特征在于,所述终止应用程序启动的方法还包括强制切断顽固自启动应用的被唤醒路径。

8.一种禁止应用自启动的装置,其特征在于,包括:

监控单元:用于监控机型应用程序的启动行为,所述监控本机应用程序的启动行为采用Hook技术;

拦截单元:拦截活动管理服务进程中用于执行该应用程序启动的执行函数,获取该应用的包名;

禁止单元:当该应用包名非预设数据记录中的允许项时,终止该应用程序的启动行为,所述数据库具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系,所述数据记录既包含允许自启动的应用的记录,又包含禁止自启动的应用的记录;或者,仅仅包含禁止自启动的应用的记录。

9.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述数据记录还包括用于记录反复自启动应用信息的顽固自启日志。

10.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述装置还包括一个顽固自启日志生成单元,生成记录反复自启动应用信息的日志。

11.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述监控单元监控本机应用程序的启动行为采用Hook技术。

12.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述活动管理服务进程具体指ActivityManagerService所在的进程。

13.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于所述应用程序启动的执行函数具体指应用启动时ActivityManagerService调用的成员函数

startProcessLocked()。

14.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述禁止单元中的终止应用程序启动的具体方法为调用函数killProcess()。

15.根据权利要求8所述的一种禁止应用自启动的装置,其特征在于,所述装置还包括唤醒路径切断单元,用于切断顽固自启动应用的被唤醒路径。

一种禁止应用自启动的方法及装置

技术领域

[0001] 本发明涉及移动设备的应用自启动控制技术，具体而言，本发明涉及一种禁止应用自启动的方法及装置。

背景技术

[0002] 应用自启动，是指Android系统上应用在非用户主动运行的情况下自行启动的行为。现在很多厂商为谋取APP商业化让其APP时刻在它需要自启的时刻自启，以谋求自己的利益。

[0003] 自启分为常规自启和顽固自启，当用户通过自启管理软件关闭顽固自启应用时，顽固自启应用会立刻想办法进行自我修复，虽然目前有许多自启管理的安全软件，但仅能禁止通过broadcast receiver方式自启动的应用，不能禁止service、provider的自启动以及顽固自启动的应用，而这些应用过多会造成用户的手机耗电快，卡顿，频繁弹窗广告等现象。本发明提供禁止自启的方案，不仅可以禁止通过broadcast receiver方式的自启动应用，还能够有效精确地禁止service、provider的自启动以及顽固自启动的应用。

发明内容

[0004] 本发明的目的旨在解决应用自启动的问题，特别是解决应用的顽固自启动行为。

[0005] 本发明提供一种禁止自启动的方法，包括以下步骤：

[0006] 监控本机应用程序的启动行为；

[0007] 拦截活动管理服务进程中用于执行该应用程序启动的执行函数，获取该应用的包名；

[0008] 当该应用包名非预设数据记录中的允许项时，终止该应用程序的启动行为。

[0009] 具体的，所述数据记录以数据库的形式予以表征，该数据库具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系。

[0010] 较佳的，所述数据记录既包含允许自启动的应用的记录，又包含禁止自启动的应用的记录；

[0011] 或者，仅仅包含禁止自启动的应用的记录。

[0012] 进一步的，所述数据记录还包括用于记录反复自启动应用信息的顽固自启日志。

[0013] 较佳的，所述方法还提供一个用户界面，用于显示顽固自启日志记录的信息。

[0014] 进一步的，所述终止应用程序启动的方法还包括强制切断顽固自启动应用的被唤醒路径。

[0015] 一种禁止应用自启动的装置，包括：

[0016] 监控单元：用于监控机型应用程序的启动行为；

[0017] 拦截单元：拦截活动管理服务进程中用于执行该应用程序启动的执行函数，获取该应用的包名；

[0018] 禁止单元：当该应用包名非预设数据记录中的允许项时，终止该应用程序的启动

行为。

[0019] 具体的，所述数据记录以数据库的形式予以表征，该数据库具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系。

[0020] 进一步的，所述装置还包括一个顽固自启日志生成单元，生成记录反复自启动应用信息的日志。

[0021] 进一步的，所述装置还包括唤醒路径切断单元，用于切断顽固自启动应用的被唤醒路径。

[0022] 相比现有技术，本发明提供的方案有以下优点：

[0023] 1、本发明提供一种禁止应用自启动方法，通过拦截应用启动时的ActivityManagerService调用的成员函数startProcessLocked()，获取该应用的包名、pid、uid等信息，通过应用包名确定用户禁止自启动的应用，终止所述应用的进程，实现禁止自启动。

[0024] 2、本发明对反复自启的应用记录一个顽固自启日志，通过切断所述顽固自启动应用被其友盟应用程序唤醒的路径，可以实现精确禁止被唤醒的应用自启动行为。

[0025] 3、本发明提供一个数据记录，以数据库的形式予以表征，该数据库具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系，可以通过该映射关系确定被用户禁止的应用，与拦截到的应用包名对比，快速确定拦截到的应用是否为被禁止的应用。

[0026] 4、本发明生成用于记录反复自启动的应用信息的顽固自启日志，并将该顽固自启日志以界面的形式呈现给用户，由用户参照列出的应用信息选择禁止的应用，展示给用户更加友好交互界面，实现最优化禁止自启动。

[0027] 本发明附加的方面和优点将在下面的描述中部分给出，这些将从下面的描述中变得明显，或通过本发明的实践了解到。

附图说明

[0028] 本发明上述的和/或附加的方面和优点从下面结合附图对实施例的描述中将变得明显和容易理解，其中：

[0029] 图1为一种禁止应用自启动的方法流程图

[0030] 图2为一种禁止应用自启动的装置原理框图

具体实施方式

[0031] 下面详细描述本发明的实施例，所述实施例的示例在附图中示出，其中自始至终相同或类似的标号表示相同或类似的元件或具有相同或类似功能的元件。下面通过参考附图描述的实施例是示例性的，仅用于解释本发明，而不能解释为对本发明的限制。

[0032] 本技术领域技术人员可以理解，除非特意声明，这里使用的单数形式“一”、“一个”、“所述”和“该”也可包括复数形式。应该进一步理解的是，本发明的说明书中使用的措辞“包括”是指存在所述特征、整数、步骤、操作、元件和/或组件，但是并不排除存在或添加一个或多个其他特征、整数、步骤、操作、元件、组件和/或它们的组。应该理解，当我们称元件被“连接”或“耦接”到另一元件时，它可以直接连接或耦接到其他元件，或者也可以存在

中间元件。此外，这里使用的“连接”或“耦接”可以包括无线连接或无线耦接。这里使用的措辞“和/或”包括一个或更多个相关联的列出项的全部或任一单元和全部组合。

[0033] 本技术领域技术人员可以理解，除非另外定义，这里使用的所有术语(包括技术术语和科学术语)，具有与本发明所属领域中的普通技术人员的一般理解相同的意义。还应该理解的是，诸如通用字典中定义的那些术语，应该被理解为具有与现有技术的上下文中的意义一致的意义，并且除非像这里一样被特定定义，否则不会用理想化或过于正式的含义来解释。

[0034] 本技术领域技术人员可以理解，这里所使用的“终端”、“终端设备”既包括无线信号接收器的设备，其仅具备无发射能力的无线信号接收器的设备，又包括接收和发射硬件的设备，其具有能够在双向通信链路上，执行双向通信的接收和发射硬件的设备。这种设备可以包括：蜂窝或其他通信设备，其具有单线路显示器或多线路显示器或没有多线路显示器的蜂窝或其他通信设备；PCS(Personal Communications Service，个人通信系统)，其可以组合语音、数据处理、传真和/或数据通信能力；PDA(Personal Digital Assistant，个人数字助理)，其可以包括射频接收器、寻呼机、互联网/内联网访问、网络浏览器、记事本、日历和/或GPS(Global Positioning System，全球定位系统)接收器；常规膝上型和/或掌上型计算机或其他设备，其具有和/或包括射频接收器的常规膝上型和/或掌上型计算机或其他设备。这里所使用的“终端”、“终端设备”可以是便携式、可运输、安装在交通工具(航空、海运和/或陆地)中的，或者适合于和/或配置为在本地运行，和/或以分布形式，运行在地球和/或空间的任何其他位置运行。这里所使用的“终端”、“终端设备”还可以是通信终端、上网终端、音乐/视频播放终端，例如可以是PDA、MID(Mobile Internet Device，移动互联网设备)和/或具有音乐/视频播放功能的移动电话，也可以是智能电视、机顶盒等设备。

[0035] 本技术领域技术人员可以理解，这里所使用的远端网络设备，其包括但不限于计算机、网络主机、单个网络服务器、多个网络服务器集或多个服务器构成的云。在此，云由基于云计算(Cloud Computing)的大量计算机或网络服务器构成，其中，云计算是分布式计算的一种，由一群松散耦合的计算机集组成的一个超级虚拟计算机。本发明的实施例中，远端网络设备、终端设备与WNS服务器之间可通过任何通信方式实现通信，包括但不限于，基于3GPP、LTE、WIMAX的移动通信、基于TCP/IP、UDP协议的计算机网络通信以及基于蓝牙、红外传输标准的近距无线传输方式。

[0036] 本发明的有关方法和装置，是基于Android操作系统实现的。为了实现本发明所述方案的功能，需要获取Android操作系统的Root权限。

[0037] Root是指Linux系统中的Root账户，该账户可以操作系统中的所有对象，类似于Windows系统中的Administrator账户。获取Root权限可以进行诸如备份系统、修改系统的内部程序、获取文件目录、静默安装应用程序、卸载应用程序等操作。针对Android系统的移动设备进行Root权限的获取，可以通过向移动设备植入SU和superuser.apk两个文件实现，SU文件放入手机的/system/bin目录下，superuser.apk文件放入/system/app目录下，两者运行时相互配合实现有效的Root权限管理。

[0038] 上述获取Root权限的方法为本领域的公知技术，现实中移动终端可以通过多种方式实现Root，部分厂商也为用户开放了系统Root权限或为获取Root权限提供了便利手段。因此，不应将Root权限的获取方式视为影响本发明实施的必要构件。

[0039] 以Android系统为应用平台，移动设备中的应用自启方式主要有以下三种方式：

[0040] 1、通过broadcast receiver组件实现自启

[0041] broadcast receiver是Android系统中接受并响应广播通知的一类组件，不包含任何用户界面，但可以启动一个activity或service响应它们收到的信息，或者用NotificationManager通知用户。注册广播接收者有两种方式:动态注册和静态注册。

[0042] 通过广播实现自启是目前最普遍的也是使用最多的应用自启方式。当目标应用采用静态注册的方式通过对指定的广播在配置文件AndroidManifest.xml中注册广播组件，在配置文件中添加代码如下所示：

[0043]

```
</receiver>
<!-- 注册系统静态广播接收器 -->
  <receiver android:name=".BroadcastReceiver">
  <intent-filter>
    <action
android:name="android.intent.action.BOOT_COMPLETED" />
    <category android:name="android.intent.category.HOME" />
  </intent-filter>
</receiver>
```

[0044] 由此，配置文件中注册了名称为Broadcast Receiver的广播组件，当系统启动时，会发送一条系统广播给所有应用，该广播名称为android.intent.action.BOOT_COMPLETED，注册了systemReceiver组件的应用就会接收到这条广播，在移动设备开始时自动运行起来，实现自启动。

[0045] 2、通过service组件实现自启

[0046] Service是Android系统的服务组件，没有用户界面，是运行在后台的应用，无法与用户直接进行交互，必须由用户或其他程序显示启动，它的优先级较高，比处于前台的应用优先级低，但是比后台的其他应用优先级高，可以分为本地service和远程service。

[0047] 本地service就是和当前应用在同一个进程中的service，彼此之间拥有共同的内存区域，可以方便和简单地共享某些数据；

[0048] 远程service主要是不同进程之间的service访问，通过AIDL工具使不同进程之间可以使用一般方法共享数据。

[0049] 应用通过服务service实现自启动的方式，需要先在配置文件AndroidManifest.xml中注册服务，否则系统无法找到service。添加代码如下：

[0050] <!--注册服务>

[0051] <service android:name＝”.MyService”>

[0052] 然后启动该注册服务，启动服务的方式有两种：

[0053] 一、通过startService()启动

[0054] 应用程序通过startService()启动一个service时,这个service将处于启动状态,一旦启动,该service就可以在后台一直运行下去,即使启动它的应用程序已经退出。

[0055] 二、通过bindService()启动

[0056] bindService是绑定service服务,执行service服务中的逻辑流程。客户端建立一个与service的连接,并使用此连接与service进行通话,通过Context.bindService()方法绑定服务,Context.unbindService()方法关闭服务。这种模式下的service开始于调用Context.StartService(),停止于Context.stopService()。不管调用多少次StartService(),只需调用一次stopService()就可以停止service。

[0057] 应用程序可以启动service实现自启,也可以通过绑定service实现自启,由其他应用通过service唤醒。

[0058] 3、通过content provider组件实现自启

[0059] content provider是Android系统中的组件之一,是Android系统提供的共享数据的机制,通过提供标准的数据访问接口,使得应用程序可以通过content provider访问其他应用程序的一些私有数据。共享的数据可以是存储在文件系统中、SQLite数据库中或其他的一些媒体中。

[0060] 应用程序创建自己的Content Provider,为其他应用程序提供信息共享服务。当其他应用需要调用该provider时,该应用程序就会被其他应用调起来,从而实现自启。具体步骤为:每个Content Provider在配置文件AndroidManifest.xml中注册自己,在配置文件中添加的代码如下所示:

[0061] <provider android:name=".SomeProvider"

[0062] android:authorities="com.your-company.SomeProvider"/>

[0063] 由此,注册了由SomeProvider提供的Content Provider,并为其授权,授权的基础URI为"com.your-company.SomeProvider",根据该授权信息系统可以准确定位到具体的Content Provider,从而使访问者能够获取到指定的信息。将应用的启动事件作为授权的URI,当其他应用访问该URI时,所述应用就会自动被调起,实现自启动。

[0064] 请参阅图1,本发明提供一种可以禁止以上所述三种方式的自启动应用的方法,具体实施方式如下所述:

[0065] S11、监控本机应用程序的启动行为

[0066] 本发明采用公知技术Hook函数实现对应用的自启动行为的监控,首先获取移动设备的系统root权限,向系统注册一个通信服务进程,该服务进程通过Android系统提供的Binder通信机制,与其监控的应用服务进程之间建立C/S架构的通信管道。

[0067] 其中Hook技术,是一种用于改变API执行结果的技术,是Windows消息处理机制的一个平台,应用程序可以在这个平台上面设置自己的子进程,用于监视指定窗口的某种消息,所述监视窗口可以是其他进程创建的。当消息到达时,在目标窗口处理函数之前处理它。Hook,译为钩子,实际是一个处理消息的程序段,通过系统调用挂入系统,当特定的消息发出时,在没有到达目的窗口前,钩子函数就先捕获该消息,可以对该消息进行加工,也可以不作处理直接放行,还可以强制结束消息的传递。

[0068] API Hook包括两种方式,一种是基于PE文件的导入表,一种是修改前5个字节直接JMP的inline Hook。其中,第一种方式的原理是PE文件有个导入表,代表该模块调用了哪些

外部API,模块被加载到内存后,PE加载器会修改该导入表,将地址改为外部API重新定位后的真实地址,直接将里面的地址改为新函数的地址,就可以完成对相应API的Hook;第二种方式的原理是解析函数开头的指令,并将该些指令复制到数组保存,替换所述指令,由新函数处理完毕再执行保存的原指令,调回取指令之后的地址执行。

[0069] 具体而言,获取系统的Root权限,向系统服务进程注入监控模块,具体步骤为:

[0070] 首先,暂停系统服务进程;

[0071] 然后,将修改后的系统服务进程的库文件覆盖原库文件;其中,修改后的库文件中的函数包含监控模块的功能代码、jar包、.so文件等。从而将监控模块注入系统的服务进程,监控系统中其他应用的服务进程。

[0072] 其次,当应用启动时调用ActivityManagerService,监控模块中的钩子函数监控目标进程的初始化函数,钩子函数监控到组件调用指令,是指监控到目标应用进程中该组件调用指令(函数)的入口点,这时组件调用指令虽有运行的特征,但未有执行该组件调用指令内部功能的事实发生。钩子函数在其入口点挂钩,对地址指针进行修改,使目标应用进程转入新的地址,执行钩子函数本身,钩子函数完成处理后再转向挂钩的入口点执行原来的指令,也可以不进行任何处理直接释放该进程,或者可以直接强制杀死该进程,从而实现对目标应用的监控与控制。

[0073] S12、拦截活动管理服务进程中用于执行该应用程序启动的执行函数,获取该应用的包名

[0074] 在Android系统中,ActivityManagerService负责管理应用程序的创建,运行在独立的进程SystemServer中,当系统要在一个新的进程中启动一个Activity或Service时,为应用创建一个新的进程,ActivityManagerService调用成员函数startProcessLocked()为应用程序启动一个新的进程。成员函数startProcessLocked()定义在frameworks/base/services/java/com/android/se rver/am/ActivityManagerService.java文件中,该函数方法实现的部分代码如下所示:

[0075]

```
private final void startProcessLocked(processRecord app, String hostingType, String hostingNameStr)
{
    ......//此处省略
    try
    {
        int uid = app.info.uid;
        int[] gids = null;
        try
```

[0076]

```
        {
          gids = mContext.getPackageManager().getPackageGids(app.inf
        o.packagename);
        }
        catch(PackageManager.NameNotFoundException e)
        {
            ……//此处省略
         }
         int debugFlags = 0;
         ……
         int pid = Process.start("android.app.ActivityThread",mSimplePr
        ocessManagement?app.processName:null,uid,uid,gids,debugFlags,
        null);
         ……//此处省略
      }
      catch(RuntimeException e)
      {
        ……//此处省略
      }
  }
```

[0077] 拦截上述startProcessLocked()函数，获取自启动应用的包信息app.info，得到应用包名app.info.packagename、uid、pid等信息。

[0078] 可以通过抓取日志信息获取应用进程的相关信息，具体实施方式可以为：自定义服务watchservice，在watchservice.onStart函数中建立新的logcat进程，重写onStart函数用于响应服务启动、创建动作，Hook系统的日志信息。具体代码如下：

[0079] mLogcat＝safeExec("logcat-c")；

[0080] mLogcat＝safeExec("logcat-v raw ActivityManager:I*:S")；

[0081] 然后对mLogcat进程输出信息进行监控，将其重定向，按行提取日志信息并对日志信息分析处理，具体为：

[0082] is＝mLogcat.getInputStream()；

[0083] isr＝new InputStreamReader(is)；

[0084] reader＝new BufferedReader(isr,0x400)；

[0085] 提取流信息并进行解析，得到应用程序的包名、pid、uid、应用程序的启动方式，其中如果该应用由其他应用唤醒启动，则还包括执行调用的应用包名，以及两个应用之间的调用路径等信息。

[0086] S13、当该应用包名非预设数据记录中的允许项时，终止该应用程序的启动行为

[0087] 将拦截获取的自启动应用包名分别与数据记录中的应用包名进行对比，判断所述被拦截的自启动应用包名是否为数据记录中被禁止的应用包名。具体过程为注入系统中的服务进程对数据记录中的信息进行扫描，获取被设置为禁止状态的应用包名，将拦截到的应用包名分别与所述被设置为禁止状态的应用包名一一对比，如果拦截到的应用包名为数据记录中的被禁止项，则禁止该应用自启动，否则不禁止该应用自启动。

[0088] 其中，所述数据记录以数据库的形式予以表征，具有应用包名与是否允许该应用包名所对应的应用程序启动的状态项之间的映射关系，既包含允许自启动的应用的记录，又包含禁止自启动的应用的记录，或者仅仅包含禁止自启动的应用的记录；包括应用包名、启动次数、状态项等信息。

[0089] 对获取的禁止自启动应用包名进行禁止，具体实施方式为：当所述拦截获取的自启动应用包名存在于所述数据库中，且该应用自启动行为通过broadcast receiver方式实现，则执行Process.killProcess(pid)函数结束应用的进程，禁止该应用的自启动行为。

[0090] 此外，本发明还记录一个顽固自启动日志，记录反复自启动的应用，不包括系统的自启动应用。所述顽固自启日志记录反复自启动应用的应用包名、启动次数、禁止该应用自启动的用户统计比例、状态项等。其中，应用包名通过拦截应用启动时的活动管理服务ActivityManagerService进程中的执行函数startProcessLocked()获取。

[0091] 反复自启动的应用启动方式一般为通过service或content provider启动，由于broadcast receiver是通过接收系统发送的广播实现自启，不同于通过service或content provider而相互调用启动，自启动应用程序被调用它的应用程序反复唤醒调起，调用它的应用处于激活状态就会唤醒与它有调用关系的应用程序，实现被调用应用程序的自启动。通过broadcast receiver自启动的应用进程可以一次被杀掉，一般不会出现反复自启动。而通过service或content provider方式自启动的应用可能是由友盟应用或者同系应用通过调用唤醒自启动，造成某些应用的顽固自启动，故而所述顽固自启动日志一般记录由上述启动方式引起的自启动应用信息。

[0092] 启动次数可以通过监控应用调用的onResume和onPause方法的次数统计，调用一次onResume和onPause方法记录为一次应用启动，通过注入系统的服务进程统计应用的自启动次数。

[0093] 禁止该应用自启动的用户统计比例由云端服务器统计并推送给注入系统中的后台服务进程，所述后台服务进程向远程服务器发送顽固自启的应用包名，由服务器与预先记录的应用被禁止的用户比例统计表一一匹配，将匹配得到的结果推送到所述后台服务进程，并记录到顽固自启日志中。

[0094] 将顽固自启日志以界面的形式展示给用户，用户可以根据提示的应用启动信息选择要禁止的应用，如果该应用的状态项设为禁止状态，则注入的后台服务进程会根据所设应用的状态项禁止该应用的自启。例如友盟应用，即应用之间相互写入友盟SDK，通过互相

调用实现自启动;以及通过service唤醒自启的同系应用,将应用进程启动的相关信息转发到注入系统的service进程里,service进程判断启动的应用包名,被启动的应用包名,分析应用之间的唤醒路径;当用户将该应用的状态项设为禁止状态时,注入系统的service进程会通过调用forcestopPackage()函数切断应用之间的相互启动路径,强制关闭被启动的应用进程。

[0095] 需要注意,对于相互唤醒启动的应用只禁止通过service方式启动的应用,对于provider方式的应用不禁止,因为provider传递的是数据信息,如果禁止会影响正常数据的传输,故不禁止该种形式的自启动应用。此外,当一个应用在一个预设的时间阈值内多次重启,如1秒钟之内重启三次,则判断该应用为系统应用,对系统应用的自启动行为不禁止,否则容易引起耗电。

[0096] 相应的,参见图2,本发明还提供一种禁止应用自启动的装置,包括监控单元11、拦截单元12、禁止单元13。此外,还包括顽固自启日志生成单元14和唤醒路径切断单元15。其中,

[0097] 监控单元11用于监控移动终端应用的自启动行为,通过注入系统一个服务进程,hook函数对应用启动时的执行函数挂钩,钩子函数在其入口点挂钩,对地址指针进行修改,使目标应用进程转入新的地址,执行钩子函数本身,钩子函数完成处理后再转向挂钩的入口点执行原来的指令,也可以不进行任何处理直接释放该进程,或者可以直接强制杀死该进程,从而实现对目标应用的监控。

[0098] 拦截单元12用于拦截自启动应用的执行函数,获取其包名;当hook的应用进程启动时,对ActivityManagerService创建应用进程时执行的成员函数startProcessLocked()进行拦截,获取拦截应用的启动方式、应用包名等信息。

[0099] 禁止单元13用于禁止用户选择禁止的应用自启动行为。将拦截到的应用包名与数据记录中的禁止应用包名一一对比,如果为数据记录中的禁止项则通过执行Process.killProcess(pid)函数结束应用的进程,禁止该应用自启动。同时由顽固日志生成单元14生成的顽固自启日志,用于记录反复自启动的应用。将顽固日志信息以界面的形式展示给用户,由用户选择禁止的顽固自启应用。对用户选择禁止的应用,由唤醒路径切断单元15将应用进程启动的相关信息转发到注入系统的service进程里,service进程判断启动应用的应用包名,被启动应用的应用包名,分析应用之间的唤醒路径,通过调用forcestopPackage()函数切断相互启动的路径,强制关闭被启动应用的进程,禁止该应用的自启动行为。

[0100] 以上所述仅是本发明的部分实施方式,应当指出,对于本技术领域的普通技术人员来说,在不脱离本发明原理的前提下,还可以做出若干改进和润饰,这些改进和润饰也应视为本发明的保护范围。

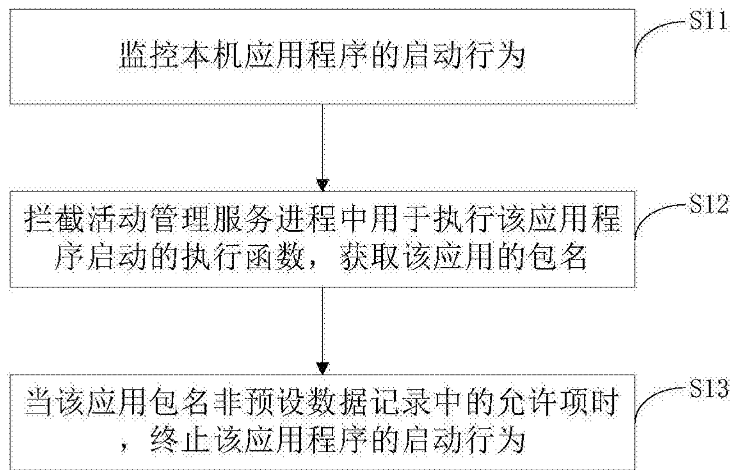


图1

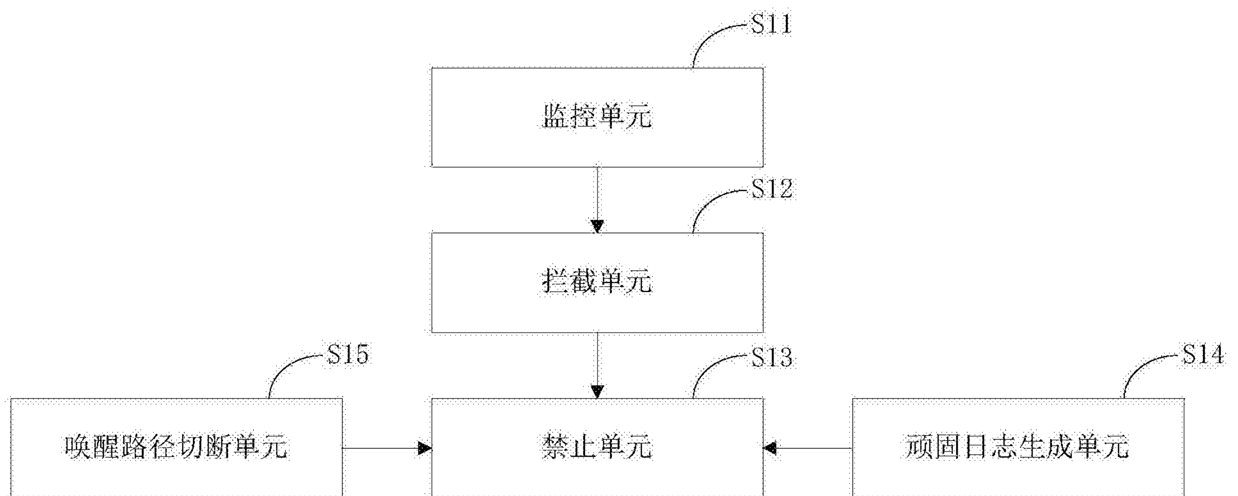


图2