(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102663285 A
(43)申请公布日 2012.09.12

(21)申请号 201210076721.1

(22)申请日 2012.03.21

(71)申请人 奇智软件(北京)有限公司
地址 100016 北京市朝阳区酒仙桥路 14 号兆维大厦 4 层东侧单元

(72)发明人 王栒 张旭

(74)专利代理机构 北京润泽恒知识产权代理有限公司 11319
代理人 赵娟

(51)Int.Cl.
G06F 21/00(2006.01)

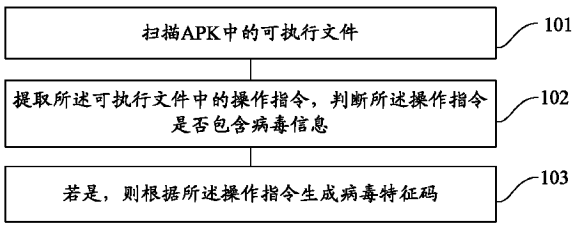
权利要求书 3 页 说明书 16 页 附图 3 页

(54)发明名称

一种 APK 病毒特征码的提取方法及装置

(57)摘要

本申请提供了一种 APK 病毒特征码的提取方法及装置，其中所述方法包括：扫描 Android 安装包 APK 中的指定文件；提取所述指定文件中的操作指令，判断所述操作指令是否包含病毒信息；若是，则根据所述操作指令生成病毒特征码。本申请可以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，从而提高 APK 应用的安全性。



CN 102663285 A

1. 一种 APK 病毒特征码的提取方法，其特征在于，包括：

扫描 Android 安装包 APK 中的指定文件；

提取所述指定文件中的操作指令，判断所述操作指令是否包含病毒信息；

若是，则根据所述操作指令生成病毒特征码。

2. 如权利要求 1 所述的方法，其特征在于，所述指定文件包括可执行文件，所述可执行文件包括 Dex 文件，所述 Dex 文件包括 classes.dex 文件，扩展名为 .jar 的文件，以及，Dex 格式的文件。

3. 如权利要求 2 所述的方法，其特征在于，还包括：

提取所述可执行文件的常量池中的常量，判断所述常量中是否包含病毒信息；

若是，则根据所述常量生成病毒特征码。

4. 如权利要求 3 所述的方法，其特征在于，还包括：

提取所述可执行文件的头部信息，判断所述头部信息中是否包含病毒信息；

若是，则根据所述头部信息生成病毒特征码。

5. 如权利要求 1、2、3 或 4 所述的方法，其特征在于，所述操作指令包括操作码和操作数，所述判断操作指令是否包含病毒信息的步骤包括：

判断所述操作数中是否包含预定义的非法操作数；

和 / 或，

判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

6. 如权利要求 5 所述的方法，其特征在于，所述根据操作指令生成病毒特征码的步骤包括：

将所述操作指令本身作为病毒特征码；

和 / 或，

将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

7. 如权利要求 3 所述的方法，其特征在于，所述可执行文件的常量池中的常量包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，所述判断常量中是否包含病毒信息的步骤包括：

判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息；

和 / 或，

判断所述类型 types、域 fields 和方法 methods 中的常量是否有调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名；

所述根据常量生成病毒特征码的步骤为，将所述常量中的病毒信息作为病毒特征码。

8. 如权利要求 4 所述的方法，其特征在于，所述可执行文件的头部信息中包括摘要信息 checksum 和签名信息 Signature，所述判断头部信息中是否包含病毒信息的步骤包括：

判断所述摘要信息 checksum 和 / 或签名信息 Signature 中是否包含预定义的非法字符串；

所述根据头部信息生成病毒特征码的步骤为，将所述摘要信息 checksum 和 / 或签名信息 Signature 作为病毒特征码。

9. 如权利要求 1 所述的方法，其特征在于，所述指定文件还包括文本文件，所述操作指

令为 linux 命令，

所述判断操作指令是否包含病毒信息的步骤为，判断所述 linux 命令是否符合预置的恶意 linux 命令；

所述根据操作指令生成病毒特征码的步骤为，将所述 linux 命令作为病毒特征码。

10. 如权利要求 1、2、3、4、6、7、8、9 或 10 所述的方法，其特征在于，还包括：

将所述病毒特征码保存至数据库中。

11. 如权利要求 10 所述的方法，其特征在于，所述病毒特征码包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；所述将病毒特征码保存至数据库中的步骤包括：

将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；

或者，

将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

12. 一种 APK 病毒特征码的提取装置，其特征在于，包括：

扫描模块，用于扫描 Android 安装包 APK 中的指定文件；

指令提取模块，用于提取所述指定文件中的操作指令；

指令判断模块，用于判断所述操作指令是否包含病毒信息；

第一特征码生成模块，用于在所述操作指令包含病毒信息时，根据所述操作指令生成病毒特征码。

13. 如权利要求 12 所述的装置，其特征在于，所述指定文件包括可执行文件，所述可执行文件包括 Dex 文件，所述 Dex 文件包括 classes.dex 文件，扩展名为 .jar 的文件，以及，Dex 格式的文件。

14. 如权利要求 13 所述的装置，其特征在于，还包括：

常量提取模块，用于提取所述可执行文件常量池中的常量；

常量判断模块，用于判断所述常量中是否包含病毒信息；

第二特征码生成模块，用于在所述常量中包含病毒信息时，根据所述常量生成病毒特征码。

15. 如权利要求 14 所述的装置，其特征在于，还包括：

头部信息提取模块，用于提取所述可执行文件的头部信息；

头部信息判断模块，用于判断所述头部信息中是否包含病毒信息；

第三特征码生成模块，用于在所述头部信息中包含病毒信息时，根据所述头部信息生成病毒特征码。

16. 如权利要求 12、13、14 或 15 所述的装置，其特征在于，所述操作指令包括操作码和操作数两部分，所述指令判断模块包括：

第一判断子模块，用于判断所述操作数中是否包含预定义的非法操作数；

和 / 或，

第二判断子模块，用于判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

17. 如权利要求 16 所述的装置，其特征在于，所述第一特征码生成模块包括：

直接生成子模块，用于将所述操作指令本身作为病毒特征码；

和 / 或，

转换生成子模块，用于将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

18. 如权利要求 14 所述的装置，其特征在于，所述可执行文件中常量池中的常量包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，所述常量判断模块包括：

第三判断子模块，用于判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息；

和 / 或，

第四判断子模块，用于判断所述类型 types、域 fields 和方法 methods 中的常量是否有调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名。

19. 如权利要求 12 所述的装置，其特征在于，所述指定文件还包括文本文件，所述操作指令为 linux 命令，所述指令判断模块包括：

第五判断子模块，用于判断所述 linux 命令是否符合预置的恶意 linux 命令。

20. 如权利要求 12、13、14、16、17、18 或 19 所述的装置，其特征在于，还包括：

存储模块，用于将所述病毒特征码保存至数据库中。

21. 如权利要求 20 所述的装置，其特征在于，所述病毒特征码包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；所述存储模块包括：

分区存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；

或者，

标签存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

一种 APK 病毒特征码的提取方法及装置

技术领域

[0001] 本申请涉及计算机安全的技术领域，特别是涉及一种 APK 病毒特征码的提取方法，以及，一种 APK 病毒特征码的提取装置。

背景技术

[0002] Android 是一种以 Linux 为基础的开放源码操作系统，主要使用于手机等移动终端，目前尚未有统一中文名称。Android 平台由操作系统、中间件、用户界面和应用软件组成。

[0003] APK 是 Android application package file 的缩写，即 Android 安装包，也可以理解为 Android 终端上安装的应用软件。APK 是类似 Symbian Sis 或 Sisx 的文件格式。通过将 APK 文件直接传到 Android 模拟器或 Android 终端中执行即可安装。apk 文件和 sis 一样，把 android sdk 编译的工程打包成一个安装程序文件，格式为 apk。APK 文件其实是 zip 格式，但后缀名被修改为 apk，通过 UnZip 解压后，可以看到 Dex 文件，Dex 是 DalvikVM executes 的全称，即 Android Dalvik 执行程序，并非标准的 Java 字节码而是 Dalvik 字节码。Android 在运行一个程序时首先需要 UnZip，然后类似 Symbian 那样直接运行，和 Windows Mobile 中的 PE 文件有区别。

[0004] 具体而言，APK 文件的结构如下表所示：

[0005]

META-INF\	存放CERT.RSA、CERT.SF、MANIFEST.MF文件
res\	存放APK相关的资源文件
AndroidManifest.xml	APK全局配置文件
classes.dex	Dalvik Executable(Dalvik虚拟机可执行文件)
resources.arsc	编译后的二进制资源文件

[0006]

[0007] 在具体应用时，APK 可以通过数据线导入移动终端，或者，直接通过 market（工具软件，如安卓市场）、网页等方式下载安装。随着 Android 终端的普及和发展，各种各样的 APK 应运而生，这其中就包括了病毒 APK，例如，一些 APK 通过诸如短信定制付费服务、拨打付费电话、备份用户手机中的敏感数据至特定服务器等恶意行为来损害用户的权益。

[0008] 目前，已经出现了一些专门针对移动终端的安全软件（如手机杀毒软件）来对这些病毒 APK 进行查杀。这些现有的安全软件查杀病毒 APK 的方法主要有以下两种：

[0009] 第一种是通过 APK 文件的 HASH、签名、Package 名字来对病毒 APK 进行识别，其原理是通过对 APK 使用 HASH 算法提取 KEY，之后即可依据此 KEY 去识别病毒 APK，或者，通过病毒 APK 制作者的 APK 数字签名、包名等对其进行识别。

[0010] 然而，上述现有的基于 APK 文件的 HASH 进行识别的方式，很容易通过重新混淆、或者，在 APK 文件中添加新的资源文件乃至修改代码等方式，使通过 HASH 算法提取 KEY 发生改变，进而导致无法识别；上述现有的基于签名的识别方式可以通过更换签名的方式绕过；上述现有的基于 Package 名字来识别的方式也可通过修改包名的方式来绕过。而且更改混淆方式，修改 APK 文件（添加删除资源，代码等）或者更换签名对病毒制造者而言都很容易，所以病毒制造者很轻易地就可以制造新的病毒变种从而绕过安全软件的识别。

[0011] 第二种是通过 APK 文件中的 classes.dex 中的类名对其进行识别，其原理是通过分析 classes.dex 中的类然后从中提取出若干个类的名字作为病毒特征码，之后即可解析病毒 APK 的 classes.dex 文件，看其中是否包含特定的类名来对其进行识别。

[0012] 然而，这种通过扫描类名来进行识别的方式，一方面因为仅仅检查类名从而容易误报，另一方面也很容易被病毒制造者通过混淆或者直接修改类名而绕过。

[0013] 因此，目前需要本领域技术人员解决的一个技术问题就是，提供一种病毒 APK 特征码的提取机制，用以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，从而提高 APK 应用的安全性。

发明内容

[0014] 本申请提供一种 APK 病毒特征码的提取方法，用以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，从而提高 APK 应用的安全性。

[0015] 本申请还提供了一种 APK 病毒特征码的提取装置，用以保证上述方法在实际中的应用及实现。

[0016] 为了解决上述问题，本申请公开了一种 APK 病毒特征码的提取方法，包括：

[0017] 扫描 Android 安装包 APK 中的指定文件；

[0018] 提取所述指定文件中的操作指令，判断所述操作指令是否包含病毒信息；

[0019] 若是，则根据所述操作指令生成病毒特征码。

[0020] 优选的，所述指定文件包括可执行文件，所述可执行文件包括 Dex 文件，所述 Dex 文件包括 classes.dex 文件，扩展名为 .jar 的文件，以及，Dex 格式的文件。

[0021] 优选的，所述的方法，还包括：

[0022] 提取所述可执行文件的常量池中的常量，判断所述常量中是否包含病毒信息；

[0023] 若是，则根据所述常量生成病毒特征码。

[0024] 优选的，所述的方法，还包括：

[0025] 提取所述可执行文件的头部信息，判断所述头部信息中是否包含病毒信息；

[0026] 若是，则根据所述头部信息生成病毒特征码。

[0027] 优选的，所述操作指令包括操作码和操作数，所述判断操作指令是否包含病毒信息的步骤包括：

[0028] 判断所述操作数中是否包含预定义的非法操作数；

[0029] 和 / 或，

[0030] 判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

[0031] 优选的，所述根据操作指令生成病毒特征码的步骤包括：

[0032] 将所述操作指令本身作为病毒特征码；

[0033] 和/或，

[0034] 将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

[0035] 优选的，所述可执行文件的常量池中的常量包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，所述判断常量中是否包含病毒信息的步骤包括：

[0036] 判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息；

[0037] 和/或，

[0038] 判断所述类型 types、域 fields 和方法 methods 中的常量是否有调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名；

[0039] 所述根据常量生成病毒特征码的步骤为，将所述常量中的病毒信息作为病毒特征码。

[0040] 优选的，所述可执行文件的头部信息中包括摘要信息 checksum 和签名信息 Signature，所述判断头部信息中是否包含病毒信息的步骤包括：

[0041] 判断所述摘要信息 checksum 和/或签名信息 Signature 中是否包含预定义的非法字符串；

[0042] 所述根据头部信息生成病毒特征码的步骤为，将所述摘要信息 checksum 和/或签名信息 Signature 作为病毒特征码。

[0043] 优选的，所述指定文件还包括文本文件，所述操作指令为 linux 命令，

[0044] 所述判断操作指令是否包含病毒信息的步骤为，判断所述 linux 命令是否符合预置的恶意 linux 命令；

[0045] 所述根据操作指令生成病毒特征码的步骤为，将所述 linux 命令作为病毒特征码。

[0046] 优选的，所述的方法，还包括：

[0047] 将所述病毒特征码保存至数据库中。

[0048] 优选的，所述病毒特征码包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；所述将病毒特征码保存至数据库中的步骤包括：

[0049] 将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；

[0050] 或者，

[0051] 将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

[0052] 本申请实施例还公开了一种 APK 病毒特征码的提取装置，包括：

[0053] 扫描模块，用于扫描 Android 安装包 APK 中的指定文件；

[0054] 指令提取模块，用于提取所述指定文件中的操作指令；

[0055] 指令判断模块，用于判断所述操作指令是否包含病毒信息；

[0056] 第一特征码生成模块，用于在所述操作指令包含病毒信息时，根据所述操作指令生成病毒特征码。

[0057] 优选的，所述指定文件包括可执行文件，所述可执行文件包括 Dex 文件，所述 Dex 文件包括 classes.dex 文件，扩展名为 .jar 的文件，以及，Dex 格式的文件。

[0058] 优选的，所述的装置，还包括：

[0059] 常量提取模块，用于提取所述可执行文件常量池中的常量；

[0060] 常量判断模块，用于判断所述常量中是否包含病毒信息；

[0061] 第二特征码生成模块，用于在所述常量中包含病毒信息时，根据所述常量生成病毒特征码。

[0062] 优选的，所述的装置，还包括：

[0063] 头部信息提取模块，用于提取所述可执行文件的头部信息；

[0064] 头部信息判断模块，用于判断所述头部信息中是否包含病毒信息；

[0065] 第三特征码生成模块，用于在所述头部信息中包含病毒信息时，根据所述头部信息生成病毒特征码。

[0066] 优选的，所述操作指令包括操作码和操作数两部分，所述指令判断模块包括：

[0067] 第一判断子模块，用于判断所述操作数中是否包含预定义的非法操作数；

[0068] 和 / 或，

[0069] 第二判断子模块，用于判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

[0070] 优选的，所述第一特征码生成模块包括：

[0071] 直接生成子模块，用于将所述操作指令本身作为病毒特征码；

[0072] 和 / 或，

[0073] 转换生成子模块，用于将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

[0074] 优选的，所述可执行文件中常量池中的常量包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，所述常量判断模块包括：

[0075] 第三判断子模块，用于判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息；

[0076] 和 / 或，

[0077] 第四判断子模块，用于判断所述类型 types、域 fields 和方法 methods 中的常量是否有调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名。

[0078] 优选的，所述指定文件还包括文本文件，所述操作指令为 linux 命令，所述指令判断模块包括：

[0079] 第五判断子模块，用于判断所述 linux 命令是否符合预置的恶意 linux 命令。

[0080] 优选的，所述的装置，还包括：

[0081] 存储模块，用于将所述病毒特征码保存至数据库中。

[0082] 优选的，所述病毒特征码包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；所述存储模块包括：

[0083] 分区存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令

特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；

[0084] 或者，

[0085] 标签存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

[0086] 与现有技术相比，本申请具有以下优点：

[0087] 本申请通过扫描分析 APK 文件中的指定文件，如可执行文件、文本文件等，针对包含病毒信息的指令、常量或头部信息按预置规则生成相应的病毒特征码，并汇编成病毒特征码库，以供后续 APK 病毒识别使用。从而可以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，提高 APK 应用的安全性。

附图说明

[0088] 图 1 是本申请的一种 APK 病毒特征码的提取方法实施例 1 的流程图；

[0089] 图 2 是本申请的一种 APK 病毒特征码的提取方法实施例 2 的流程图；

[0090] 图 3 是本申请的一种 APK 病毒特征码的提取方法实施例 3 的流程图；

[0091] 图 4 是本申请的一种 APK 病毒特征码的提取方法实施例 4 的流程图；

[0092] 图 5 是本申请的一种 APK 病毒特征码的提取装置实施例的结构框图。

具体实施方式

[0093] 为使本申请的上述目的、特征和优点能够更加明显易懂，下面结合附图和具体实施方式对本申请作进一步详细的说明。

[0094] 本申请实施例的核心构思之一在于，通过扫描分析 APK 文件中的指定文件，如可执行文件、文本文件等，针对包含病毒信息的指令、常量或头部信息按预置规则生成相应的病毒特征码，并汇编成病毒特征码库，以供后续 APK 病毒识别使用。

[0095] 参考图 1，示出了本申请的一种 APK 病毒特征码的提取方法实施例 1 的步骤流程图，具体可以包括如下步骤：

[0096] 步骤 101、扫描 APK 中的可执行文件；

[0097] 在本申请的一种优选实施列中，所述可执行文件包括 Dex 文件，Dex 文件主要是 APK 中的 classes.dex 文件，即 Dalvik Executable(Dalvik 虚拟机可执行文件)。公知的是，Dalvik 是用于 Android 平台的 Java 虚拟机。Dalvik 虚拟机 (Dalvik VM) 是 Android 移动设备平台的核心组成部分之一。它可以支持已转换为 .dex(即 Dalvik Executable) 格式的 Java 应用程序的运行，.dex 格式是专为 Dalvik 设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

[0098] 更为优选的是，所述可执行文件还可以包括扩展名为 .jar 的文件。Android 安装包中的 JAR 文件其实就是 Dex 文件，只不过其扩展名为 .jar，对于 APK 中除 classes.dex 之外的其他文件，只要判定其为 Dex 文件即可决定是否进行扫描。

[0099] 在实际应用中，所述 Dex 文件还可以包括其它 Dex 格式的文件。

[0100] 步骤 102、提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信

息；

[0101] 在具体实现中，所述操作指令包括操作码（opcode）和操作数，在这种情况下，可以通过如下子步骤判断所述操作指令是否包含病毒信息：

[0102] 子步骤 S11、判断所述操作数中是否包含预定义的非法操作数；

[0103] 和 / 或，

[0104] 子步骤 S12、判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

[0105] 步骤 103、若是，则根据所述操作指令生成病毒特征码。

[0106] 在本申请的一种优选实施例中，所述步骤 103 可以包括如下子步骤：

[0107] 子步骤 S21、将所述操作指令本身作为病毒特征码；

[0108] 和 / 或，

[0109] 子步骤 S22、将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

[0110] 在本实施例中，所述病毒特征码可以包括操作数特征码、指令特征码、指令特征码序列。

[0111] 例如，某个 APK 的 classes.dex 文件中的 Instructions（操作指令）如下所示：

[0112]

1a0c bb08 |009b: const-string v12, "tiger" // string@08bb

1a0d 1e03 |009d: const-string v13, "P5" // string@031e

7120 1404 dc00 |009f: invoke-static {v12, v13}, Lcom/androidkernel/flash/util/LogUtil;.i:(Ljava/lang/String;Ljava/lang/String;)V // method@0414

2205 9700 |00a2: new-instance v5, Lcom/androidkernel/flash/http/base/DlStruct; // type@0097

7010 1603 0500 |00a4: invoke-direct {v5},

[0113]

Lcom/androidkernel/flash/http/base/DlStruct;.<init>:()V // method@0316

1a0c 7200 |00a7: const-string v12, "AA" // string@0072

7020 f402 ce00 |00a9: invoke-direct {v14, v12}, Lcom/androidkernel/flash/helper/Tiger;.getUrl:(Ljava/lang/String;)Ljava/lang/String; // method@02f4

0c0b |00ac: move-result-object v11

[0114] 若判定上述操作码和操作数的组合符合预定义的非法组合规则，或者，上述操作数中包含预定义的非法操作数时，可按如下方式生成特征码：

[0115] 方式一：

[0116] 1a0cbb081a0d1e0371201404dc00220597007010160305001a0c72007020f402ce000

c0b

[0117] 方式二：

[0118] 1a$tiger$1a$P5$71$Lcom/androidkernel/flash/util/LogUtil ;.i :(Ljava/lang/String ;Ljava/lang/String ;)V22Lcom/androidkernel/flash/http/base/DlStruct ;70Lcom/androidkernel/flash/http/base/DlStruct ;.<init> :()V$1a$AA70Lcom/androidkernel/flash/helper/Tiger ;.getUrl :(Ljava/lang/String ;)Ljava/lang/String ;$0c$*

[0119] 方式三：

[0120] 1a0cbb08$1a$P5$71201404dc00$22$*$70$Lcom/androidkernel/flash/http/base/DlStruct ;.<init> :()V$1a$AA70Lcom/androidkernel/flash/helper/Tiger ;.getUrl :(Ljava/lang/String ;)Ljava/lang/String ;$0c$*

[0121] 参考图 2，示出了本申请的一种 APK 病毒特征码的提取方法实施例 2 的步骤流程图，具体可以包括如下步骤：

[0122] 步骤 201、扫描 APK 中的可执行文件；

[0123] 步骤 202、提取所述可执行文件的常量池中的常量，判断所述常量中是否包含病毒信息；

[0124] 在本申请的一种优选实施例中，所述指定文件的常量池中的常量可以包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，在这种情况下，可以通过以下子步骤判断常量中是否包含病毒信息：

[0125] 子步骤 S31、判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息等恶意信息；

[0126] 和 / 或，

[0127] 子步骤 S32、判断所述类型 types、域 fields 和方法 methods 中的常量是否调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名。

[0128] 步骤 203、若是，则根据所述常量生成病毒特征码。

[0129] 在具体应用中，可以直接将所述常量中的病毒信息作为病毒特征码。

[0130] 在本实施例中，所述病毒特征码还包括常量特征码，类名函数名特征码。

[0131] 例如，某个 APK 的 classes.dex 文件中的常量池中包含如下字符串：

[0132] com.noshufbu.android.su

[0133] /system/app/com.google.update.apk

[0134] 在判定其为病毒信息后，可直接将其作为病毒特征码。

[0135] 例如，某个 APK 的 classes.dex 文件中的常量池中包含如下 method：

[0136] Lcom/android/main/SmsReceiver ;

[0137] Lcom/android/main/ActionReceiver ;

[0138] 在判定其为病毒信息后，可直接将其作为病毒特征码。

[0139] 例如，某个 APK 的 classes.dex 文件中的常量池中包含如下 type：

[0140] Lcom/androidkernel/flash/Main$1 ;

[0141] 在判定其为病毒信息后，可直接将其作为病毒特征码。

[0142] 例如，某个 APK 的 classes.dex 文件中的常量池中包含如下 field：

[0143] Lcom/androidkernel/flash/b/br$1 ;.this$0 :Lcom/androidkernel/flash/b/br ;

[0144] 在判定其为病毒信息后，可直接将其作为病毒特征码。

[0145] 步骤 204、提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信息；

[0146] 步骤 205、若是，则根据所述操作指令生成病毒特征码。

[0147] Dalvik VM 是基于寄存器设计的，程序中使用的数据如 strings，types，fields 和 methods 保存在专门的数据存储区（常量池）中，在程序当中通过对应的索引来引用，而字符文字常量则直接保存在 instructions（操作指令）中，其操作码 (opcode) 分为两类：

[0148] 一类将指定的数据放入寄存器，如参见如下例 1 至例 4：

[0149] 例 1：

[0150] 13036100 |0000 :const/16v3，#int 97//#61

[0151] 将整数 97 放入寄存器 v3 中。

[0152] 例 2：

[0153] 1700 0000 0040 |0049 :const-wide/32 v0，#float 2.000000//#40000000

[0154] 将浮点数 2.000000 放入寄存器 v0 中。

[0155] 例 3：

[0156] 1a00 7d00 |000b :const-string v0，″ %.2fMB″ //string@007d

[0157] 将字符串″ %.2fMB″ 放入寄存器 v0 中。

[0158] 例 4：

[0159] 1c03 6e04 |0015 :const-class v3，Lcom/qihoo360/mobilesafe/service/NetTrafficService ;//type@046e

[0160] 将类 com.qihoo360.mobilesafe.service.NetTrafficService 放入寄存器 v3 中。

[0161] 另外一类则基于寄存器进行操作，如参见如下例 5 至例 10：

[0162] 例 5：

[0163] 3100 0305 |0042 :cmp-long v0，v3，v5

[0164] 比较寄存器 v3 和 v5 中的 long 值，将比较结果存入寄存器 v0。

[0165] 例 6：

[0166] 3221 0400 |001a :if-eq v1，v2，001e//+0004

[0167] 条件 if，根据 v1 和 v2 是否相等来决定执行流程。

[0168] 例 7：

[0169] 3800 1500 |001e :if-eqz v0，0033//+0015

[0170] 条件 if，判断 v0 是否等于 0 来决定执行流程。

[0171] 例 8：

[0172] 6e10 0e29 0500 |0006 :invoke-virtual{v5}，Ljava/io/File ;.length :()J//method@290e

[0173] 调用 File 的 length() 函数。

[0174] 例 9 :

[0175] 7010 042a 0800 |011d :invoke-direct {v8}, Ljava/lang/StringBuilder ;.<init> :()V//method@2a04

[0176] 调用 StringBuilder 的 init 函数。

[0177] 例 10 :

[0178] b021 |0035 :add-int/2addr v1, v2

[0179] 将 v1+v2 的结果保存在 v1 中。

[0180] APK 中的 classes.dex 中的用户类名,函数名,字符串会受到混淆或者修改而发生改变,但 Dalvik VM 的指令以及对 Android 系统 SDK 提供的类的调用不会受到用户类名,函数名,变量名等被混淆或者修改的影响,因此可以通过一组有序的特定指令来识别 APK。因为 Dalvik VM 是基于寄存器的,因此其指令本身只能操作寄存器,字符文字常量,数据存储区,而寄存器地址是可变的,因此识别时要模糊匹配也即通过识别指令中的固定部分——opcode 及其相关的字符文字常量参数或者数据存储区中的 strings, types, fields 和 methods 等,当然也可以直接使用指令及其操作数本身作为病毒特征码。

[0181] 特征码生成方案一 :

[0182] 直接使用 APK 中的 classes.dex 中的特定指令集本身作为病毒特征码。

[0183] 例如,上述例 1 的病毒特征码可以为 13036100,例 2 的病毒特征码可以为 170000000040,例 3 的病毒特征码可以为 1a007d00,例 4 的病毒特征码可以为 1c036e04,例 5 的病毒特征码可以为 31000305,例 6 的病毒特征码可以为 32210400,例 7 的病毒特征码可以为 38001500,例 8 的病毒特征码可以为 6e100e290500,例 9 的病毒特征码可以为 7010042a 0800,例 10 的病毒特征码可以为 b021。

[0184] 特征码生成方案二 :

[0185] 使用 APK 中的 classes.dex 中的特定 opcode 及其操作数的字符串或通配符作为病毒特征码。

[0186] 例如,上述例 1 的病毒特征码可以为 13$*(其中 * 代表模糊匹配,下同,需要说明的是,此处的“*”仅用作举例,实际中可以使用任意字符),例 2 的病毒特征码可以为 17$*,例 3 的病毒特征码可以为 1a$,例 4 的病毒特征码可以为 1c$ Lcom/qihoo360/mobilesafe/service/NetTrafficService,例 5 的病毒特征码可以为 31$*,例 6 的病毒特征码可以为 32$*,例 7 的病毒特征码可以为 38$*,例 8 的病毒特征码可以为 6e$Ljava/io/File ;.length :(),例 9 的病毒特征码可以为 70$Ljava/lang/StringBuilder ;.<init>,例 10 的病毒特征码可以为 b0$*。

[0187] 特征码选择方案三 :

[0188] 混合使用上述方案一和方案二。即将上述 APK 中的 classes.dex 中的特定指令集本身,以及, APK 中的 classes.dex 中的特定 opcode 及其操作数的字符串或通配符全部作为病毒特征码。

[0189] 为使本领域技术人员更好地理解本申请,以下通过一个具体示例进行说明。

[0190] 针对提取 classes.dex 中的常量池 (string、type、field 和 meth) 当中的常量提取的病毒特征码如下 :例如,某病毒在其字符串常量池当中包括以下特征字符串 :

[0191] zjphonecall.txt 和 zjsms.txt,在这 2 个文件中包括了恶意电话号码以及特服短

信号码则可提取其作为病毒特征码。

[0192] 针对反汇编 classes.dex 提取的病毒特征码如下：

[0193] 例如，病毒 X 卧底 .apk 中包含以下指令用以备份用户隐私数据至 http://www.mybackup.me，按照其出现的先后顺序列举如下：

[0194] 2200 f600 |0000 :new-instance v0, Ljava/lang/StringBuilder ;// type@00f6

[0195] 提取其病毒特征码为 :2200f600 或 22$Ljava/lang/StringBuilder

[0196] 7010 9804 0000 |0002 :invoke-direct {v0}, Ljava/lang/StringBuilder ;.<init> :()V//method@0498

[0197] 提取其病毒特征码为 :701098040000 或 70$Ljava/lang/StringBuilder ;.<init>

[0198] 1a01 5506 |0005 :const-string v1," http://www.mybackup.me" // string@0655

[0199] 提取其病毒特征码为 :701098040000 或 1a$http://www.mybackup.me

[0200] 6e20 9e04 1000 |0007 :invoke-virtual{v0, v1}, Ljava/lang/StringBuilder ;.append :(Ljava/lang/String ;)Ljava/lang/StringBuilder ;// method@049e

[0201] 提取其病毒特征码为 :6e209e041000 或 6e$Ljava/lang/StringBuilder ;.append

[0202] 3902 0900 |0005 :if-nez v2,000e//+0009

[0203] 提取其病毒特征码为 :39020900 或 39$*

[0204] 0c02 |0003 :move-result-object v2

[0205] 提取其病毒特征码为 :0c02 或 0c$*

[0206] 最终获得的病毒特征码为：

[0207] 病毒特征码选择方案一：

[0208] 2200f600701098040000701098040000 6e209e041000390209000c02

[0209] 病毒特征码选择方案二：

[0210] 22$Ljava/lang/StringBuilder$70$Ljava/lang/StringBuilder ;.<init>$1a$http://www.mybackup.me$6e$Ljava/lang/StringBuilder ;.append39*$0c$*

[0211] 病毒特征码选择方案三：

[0212] 22$Ljava/lang/StringBuilder$701098040000$1a$http://www.mybackup.me$6e$Ljava/lang/StringBuilder ;.append39*$0c02

[0213] 参考图 3，示出了本申请的一种 APK 病毒特征码的提取方法实施例 3 的步骤流程图，具体可以包括如下步骤：

[0214] 步骤 301、扫描 APK 中的可执行文件；

[0215] 步骤 302、提取所述可执行文件常量池中的常量，判断所述常量中是否包含病毒信息；

[0216] 步骤 303、若是，则根据所述常量生成病毒特征码。

[0217] 步骤 304、提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信息；

[0218] 步骤 305、若是，则根据所述操作指令生成病毒特征码；

[0219] APK 中的 classes.dex 文件和 JAR 文件中的用户类名，函数名，字符串会受到混淆或者修改而发生改变，但Dalvik VM的指令以及对Android系统SDK提供的类的调用不会受到用户类名，函数名，变量名等被混淆或者修改的影响，因此可以通过一组有序的特定指令来识别 APK。因为 Dalvik VM 是基于寄存器的，因此其指令本身只能操作寄存器，字符文字常量，数据存储区，而寄存器地址是可变的，因此识别时要模糊匹配也即通过识别指令中的固定部分——opcode 及其相关的字符文字常量参数或者数据存储区中的 strings，types，fields 和 methods 等，当然也可以直接使用指令及其操作数本身作为病毒特征码。

[0220] 病毒特征码生成方案一：

[0221] 直接使用 APK 中的 classes.dex 文件和 JAR 文件中的特定指令集本身作为病毒特征码。

[0222] 病毒特征码生成方案二：

[0223] 使用 APK 中的 classes.dex 文件和 JAR 文件中的特定 opcode 及其操作数的字符串或通配符作为病毒特征码。

[0224] 病毒特征码选择方案三：

[0225] 混合使用上述方案一和方案二。即将上述 APK 中的 classes.dex 文件和 JAR 文件中的特定指令集本身，以及，APK 中的 classes.dex 文件和 JAR 文件中的特定 opcode 及其操作数的字符串或通配符全部作为病毒特征码。

[0226] 步骤 306、提取所述可执行文件的头部信息，判断所述头部信息中是否包含病毒信息；

[0227] 在具体实现中，所述可执行文件的头部信息中包括摘要信息 checksum 和签名信息 Signature，在这种情况下，所述步骤 306 可以为：

[0228] 判断所述摘要信息 checksum 和签名信息 Signature 中是否包含预定义的非法字符串。

[0229] 步骤 307、若是，则根据所述头部信息生成病毒特征码。

[0230] 在具体应用中，可以将所述摘要信息 checksum 和 / 或签名信息 Signature 均可作为病毒特征码。

[0231] 在本实施例中，所述病毒特征码包括：头部信息特征码。

[0232] 例如，APK 中的 classes.dex 文件头部信息 header 的 checksum 为：11f26cac；Signature 为：2911621AD071F675ADF0F590C3F1AFB5443BEBBE，在判定其为木马病毒后，直接将 11f26cac 和 2911621AD071F675ADF0F590C3F1AFB5443BEBBE 提取为病毒特征码。

[0233] 步骤 308、将所述病毒特征码保存至数据库中。

[0234] 在本申请实施例中，所述病毒特征码可以包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；在这种情况下，可以将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；或者，还可以将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

[0235] 当然，上述保存病毒特征码的方式仅仅用作示例，本领域技术人员根据实际情况

采用任一种保存方式都是可行的，本申请对此无需加以限制。

[0236] 本领域技术人员易于理解的是，上述操作指令、常量池和头部信息的扫描并无先后顺序的限制，本领域技术人员根据实际情况任意设定上述三者的扫描顺序都是可行的，本申请对此无需加以限制。

[0237] 上述申请实施例通过扫描分析 APK 文件中的可执行文件，针对包含病毒信息的指令、常量或头部信息按预置规则生成相应的病毒特征码，并汇编成病毒特征码库，以供后续 APK 病毒识别使用。从而可以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，提高 APK 应用的安全性。

[0238] 参考图 4，示出了本申请的一种 APK 病毒特征码的提取方法实施例 4 的步骤流程图，具体可以包括如下步骤：

[0239] 步骤 401、扫描 APK 中的文本文件；

[0240] 步骤 402、提取所述文本文件中的 linux 命令，判断所述 linux 命令是否包含病毒信息；

[0241] 在具体实现中，可以通过判断所述 linux 命令是否符合预置的恶意 linux 命令确定所述 linux 命令是否包含病毒信息。

[0242] 步骤 403、若是，则根据所述 linux 命令生成病毒特征码。

[0243] 在具体实现中，可以将所述包含病毒信息的 linux 命令直接作为病毒特征码。

[0244] 例如，从 APK 中的文本文件中提取相应的 linux 命令如下：

[0245]

```
cat /system/bin/sh > /data/data/$1/files/sh.new
chown 0.0 /data/data/$1/files/sh.new
chmod 4755 /data/data/$1/files/sh.new
rm -f /data/data/$1/files/sh
```

[0246]

```
mount -o remount system /system

mkdir /system/xbin/$1
myuid=$2
if [ "$myuid" == "" ]; then
myuid="0"
fi
chown ${myuid} /system/xbin/$1
chmod 700 /system/xbin/$1

cat /system/bin/sh > /system/xbin/$1/sh
chown 0.0 /system/xbin/$1/sh
chmod 4755 /system/xbin/$1/sh

sync
mount -o remount,ro system /system
#/system/bin/stop vold
#/system/bin/start vold
echo "+++ending+++"
```

[0247] 在判定上述 linux 命令符合预置的恶意 linux 命令时，将上述命令作为病毒特征码写入病毒特征库中。

[0248] 上述申请实施例通过扫描分析 APK 文件中的文本文件，针对包含病毒信息的 linux 命令生成相应的病毒特征码，并汇编成病毒特征码库，以供后续 APK 病毒识别使用。从而可以准确、有效地提取出病毒 APK 特征码，以帮助提高病毒 APK 及其变种识别的效率和准确性，提高 APK 应用的安全性。

[0249] 本申请实施例还适用于 APK 中嵌套 APK 的情形，即当 APK 中还包含其它 APK 时，同样可应用本申请实施例，对 APK 及其嵌套 APK 中的可执行文件、文本文件等进行解析和病毒提取，例如，在某个 1.APK 中嵌入了一个 root.apk 用以获取 root 权限，应用本申请实施例，除从 1.APK 提取病毒特征码，还会从 root.apk 中提取病毒特征码。本领域技术人员易于想到的是，对于多重嵌套 APK 的情形，本申请实施例亦同样适用，本申请在此不作限制。

[0250] 在具体实现中，本申请实施例可以对文本文件和可执行文件都进行扫描以及进行提取病毒特征码的处理，即上述方法实施例 4 与方法实施例 1、方法实施例 2 或方法实施例 3 任意组合均是可行的，本申请实施例旨在分实施例重点说明不同应用的情形，对上述方法

实施例的组合应用不作限制。

[0251] 需要说明的是，本申请实施例不仅适用于各种 Android 终端，即使用 Android 平台（操作系统）的终端，包括计算机、PC、笔记本电脑、手机、平板电脑等等；还适用于在其他计算机系统（例如 Windows、Linux）之上使用的病毒特征码提取方案。

[0252] 对于方法实施例，为了简单描述，故将其都表述为一系列的动作组合，但是本领域技术人员应该知悉，本申请并不受所描述的动作顺序的限制，因为依据本申请，某些步骤可以采用其他顺序或者同时进行。其次，本领域技术人员也应该知悉，说明书中所描述的实施例均属于优选实施例，所涉及的动作和模块并不一定是本申请所必须的。

[0253] 参考图 5，其示出了本申请的一种 APK 病毒特征码的提取装置实施例的结构框图，具体可以包括以下模块：

[0254] 扫描模块 501，用于扫描 Android 安装包 APK 中的指定文件；

[0255] 指令提取模块 502，用于提取所述指定文件中的操作指令；

[0256] 指令判断模块 503，用于判断所述操作指令是否包含病毒信息；

[0257] 第一特征码生成模块 504，用于在所述操作指令包含病毒信息时，根据所述操作指令生成病毒特征码。

[0258] 在具体实现中，所述可执行文件可以包括 Dex 文件，所述指定文件包括可执行文件，所述 Dex 文件可以包括 classes.dex 文件，扩展名为 .jar 的文件，以及，采用其他扩展名或无扩展名但文件格式为 Dex 格式的文件。

[0259] 在本申请的一种优选实施例中，还可以包括如下模块：

[0260] 常量提取模块 505，用于提取所述可执行文件常量池中的常量；

[0261] 常量判断模块 506，用于判断所述常量中是否包含病毒信息；

[0262] 第二特征码生成模块 507，用于在所述常量中包含病毒信息时，根据所述常量生成病毒特征码。

[0263] 在本申请的一种优选实施例中，还可以包括如下模块：

[0264] 头部信息提取模块 508，用于提取所述可执行文件的头部信息；

[0265] 头部信息判断模块 509，用于判断所述头部信息中是否包含病毒信息；

[0266] 第三特征码生成模块 510，用于在所述头部信息中包含病毒信息时，根据所述头部信息生成病毒特征码。

[0267] 作为本申请实施例具体应用的一种示例，所述操作指令包括操作码和操作数两部分，在这种情况下，所述指令判断模块 403 具体可以包括如下子模块：

[0268] 第一判断子模块，用于判断所述操作数中是否包含预定义的非法操作数；

[0269] 和 / 或，

[0270] 第二判断子模块，用于判断所述操作码和操作数的组合是否符合预定义的非法组合规则。

[0271] 在具体实现中，所述第一特征码生成模块 404 可以包括如下子模块：

[0272] 直接生成子模块，用于将所述操作指令本身作为病毒特征码；

[0273] 和 / 或，

[0274] 转换生成子模块，用于将所述操作指令的操作码，以及，操作数的字符串或通配符作为病毒特征码。

[0275] 在具体实现中，所述可执行文件中常量池中的常量包括字符串 strings、类型 types、域 fields 和方法 methods 中的常量，在这种情况下，所述常量判断模块 406 可以包括如下子模块：

[0276] 第三判断子模块，用于判断所述字符串 strings 中的常量是否包含预定义的恶意网址信息、恶意文件名或恶意号码信息等恶意信息；

[0277] 和 / 或，

[0278] 第四判断子模块，用于判断所述类型 types、域 fields 和方法 methods 中的常量是否有调用自定义的类名、自定义的函数名或 Android 系统 SDK 类名、Android 系统函数名。

[0279] 在本申请实施例的具体应用中，可以将常量中的病毒信息，以及，所述摘要信息 checksum 和 / 或签名信息 Signature 中的病毒信息直接提取为病毒特征码。

[0280] 在本申请的一种优选实施例中，所述指定文件还可以包括文本文件，所述操作指令可以为 linux 命令，在这种情况下，所述指令判断模块 403 可以包括如下子模块：

[0281] 第五判断子模块，用于判断所述 linux 命令是否符合预置的恶意 linux 命令。

[0282] 为便于后续的 APK 病毒识别，本申请实施例还可以包括如下模块：

[0283] 存储模块 511，用于将所述病毒特征码保存至数据库中。

[0284] 作为本申请实施例具体应用的一种示例，所述病毒特征码可以包括：头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码；在这种情况下，所述存储模块 511 可以包括如下子模块：

[0285] 分区存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码分别保存在数据库中不同的存储区域；

[0286] 或者，

[0287] 标签存储子模块，用于将所述头部信息特征码、常量特征码、操作数特征码、指令特征码、指令特征码序列、类名函数名特征码保存在数据库中，并分别标记分类标签。

[0288] 由于所述装置实施例基本相应于前述图 1、图 2 和图 3 所示的方法实施例，故本实施例的描述中未详尽之处，可以参见前述实施例中的相关说明，在此就不赘述了。

[0289] 本领域内的技术人员应明白，本申请的实施例可提供为方法、系统、或计算机程序产品。因此，本申请可采用完全硬件实施例、完全软件实施例、或结合软件和硬件方面的实施例的形式。而且，本申请可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质（包括但不限于磁盘存储器、CD-ROM、光学存储器等）上实施的计算机程序产品的形式。

[0290] 本申请是参照根据本申请实施例的方法、设备（系统）、和计算机程序产品的流程图和 / 或方框图来描述的。应理解可由计算机程序指令实现流程图和 / 或方框图中的每一流程和 / 或方框、以及流程图和 / 或方框图中的流程和 / 或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理设备的处理器以产生一个机器，使得通过计算机或其他可编程数据处理设备的处理器执行的指令产生用于实现在流程图一个流程或多个流程和 / 或方框图一个方框或多个方框中指定的功能的装置。

[0291] 这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理设备以特定方式工作的计算机可读存储器中，使得存储在该计算机可读存储器中的指令产生包括指

令装置的制造品，该指令装置实现在流程图一个流程或多个流程和 / 或方框图一个方框或多个方框中指定的功能。

[0292] 这些计算机程序指令也可装载到计算机或其他可编程数据处理设备上，使得在计算机或其他可编程设备上执行一系列操作步骤以产生计算机实现的处理，从而在计算机或其他可编程设备上执行的指令提供用于实现在流程图一个流程或多个流程和 / 或方框图一个方框或多个方框中指定的功能的步骤。

[0293] 尽管已描述了本申请的优选实施例，但本领域内的技术人员一旦得知了基本创造性概念，则可对这些实施例做出另外的变更和修改。所以，所附权利要求意欲解释为包括优选实施例以及落入本申请范围的所有变更和修改。

[0294] 最后，还需要说明的是，在本文中，诸如第一和第二等之类的关系术语仅仅用来将一个实体或者操作与另一个实体或操作区分开来，而不一定要求或者暗示这些实体或操作之间存在任何这种实际的关系或者顺序。而且，术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、物品或者设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、物品或者设备所固有的要素。在没有更多限制的情况下，由语句“包括一个 ……”限定的要素，并不排除在包括所述要素的过程、方法、物品或者设备中还存在另外的相同要素。

[0295] 以上对本申请所提供的一种 APK 病毒特征码的提取方法，以及，一种 APK 病毒特征码的提取装置进行了详细介绍，本文中应用了具体个例对本申请的原理及实施方式进行了阐述，以上实施例的说明只是用于帮助理解本申请的方法及其核心思想；同时，对于本领域的一般技术人员，依据本申请的思想，在具体实施方式及应用范围上均会有改变之处，综上所述，本说明书内容不应理解为对本申请的限制。

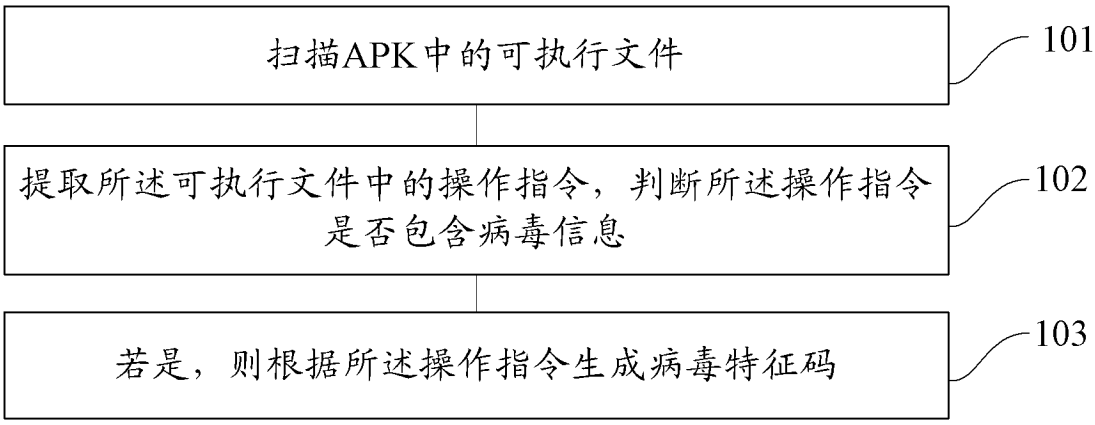

扫描APK中的可执行文件
101
提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信息
102
若是，则根据所述操作指令生成病毒特征码
103

图 1

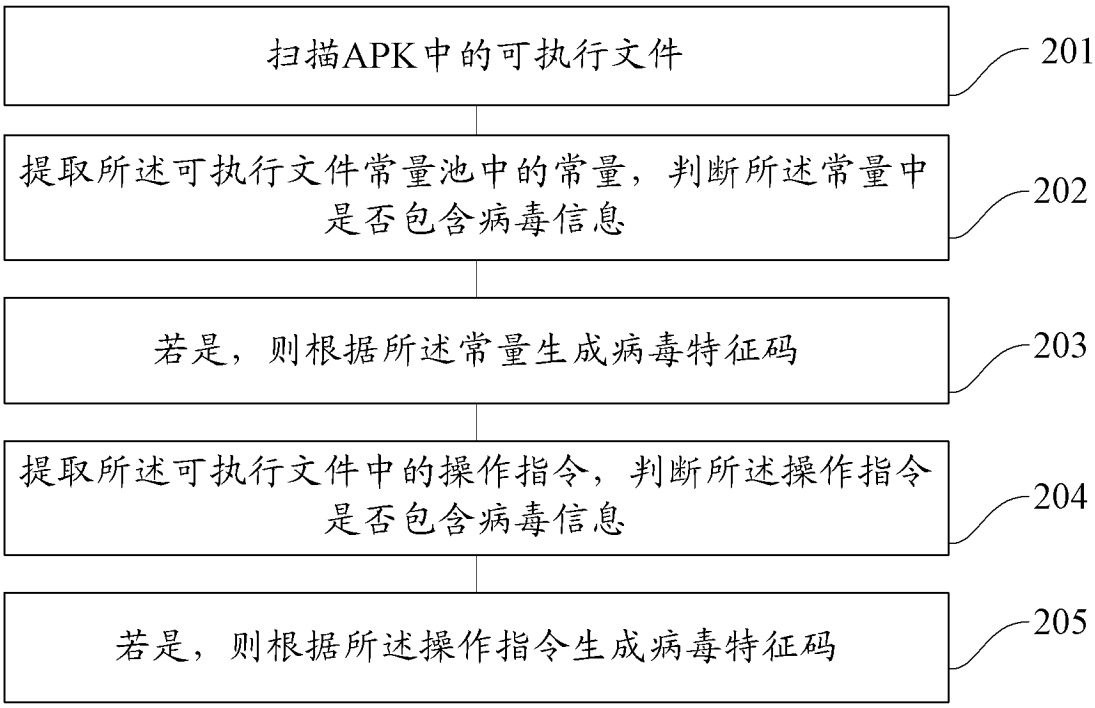

扫描APK中的可执行文件
201
提取所述可执行文件常量池中的常量，判断所述常量中是否包含病毒信息
202
若是，则根据所述常量生成病毒特征码
203
提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信息
204
若是，则根据所述操作指令生成病毒特征码
205

图 2


扫描APK中的可执行文件
301
提取所述可执行文件常量池中的常量，判断所述常量中是否包含病毒信息
302
若是，则根据所述常量生成病毒特征码
303
提取所述可执行文件中的操作指令，判断所述操作指令是否包含病毒信息
304
若是，则根据所述操作指令生成病毒特征码
305
提取所述可执行文件的头部信息，判断所述头部信息中是否包含病毒信息
306
若是，则根据所述头部信息生成病毒特征码
307
将所述病毒特征码保存至数据库中
308

图 3

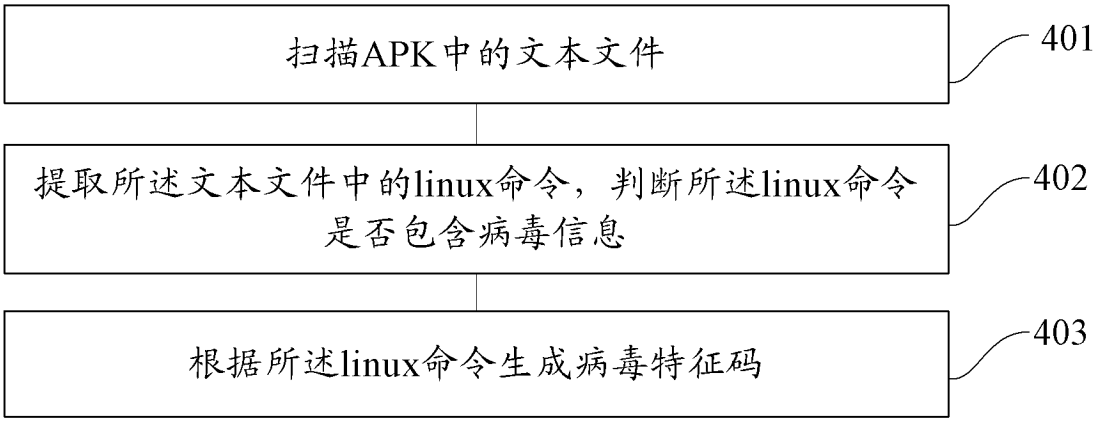

扫描APK中的文本文件
401
提取所述文本文件中的linux命令，判断所述linux命令是否包含病毒信息
402
根据所述linux命令生成病毒特征码
403

图 4

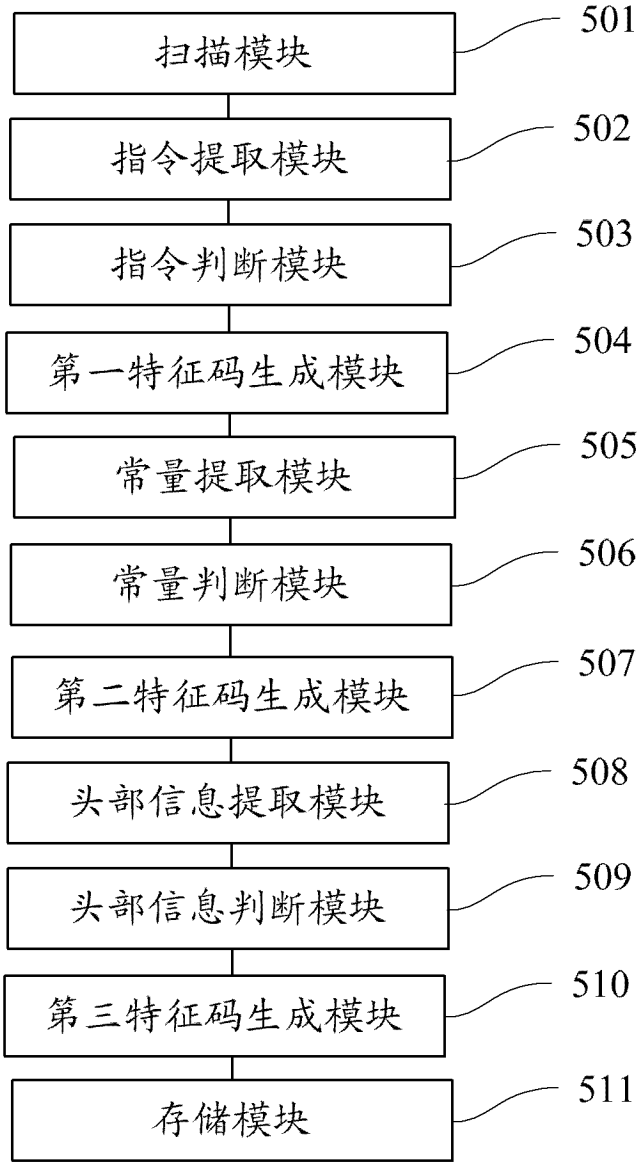

扫描模块
501
指令提取模块
502
指令判断模块
503
第一特征码生成模块
504
常量提取模块
505
常量判断模块
506
第二特征码生成模块
507
头部信息提取模块
508
头部信息判断模块
509
第三特征码生成模块
510
存储模块
511

图 5