(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 106295348 A
(43)申请公布日 2017.01.04

(21)申请号 201510289736.X

(22)申请日 2015.05.29

(71)申请人 阿里巴巴集团控股有限公司
地址 英属开曼群岛大开曼资本大厦一座四层847号邮箱

(72)发明人 孙伟超

(74)专利代理机构 北京博浩百睿知识产权代理有限责任公司 11134
代理人 宋子良

(51)Int.Cl.
G06F 21/57(2013.01)

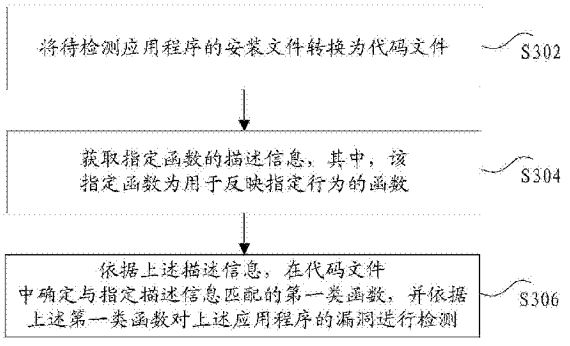
权利要求书2页 说明书12页 附图5页

(54)发明名称

应用程序的漏洞检测方法及装置

(57)摘要

本发明公开了一种应用程序的漏洞检测方法及装置。其中，该方法包括：将待检测应用程序的安装文件转换为代码文件；获取指定函数的描述信息，其中，所述指定函数为用于反映指定行为的函数；依据所述描述信息，在所述代码文件中确定与指定描述信息匹配的第一类函数，并依据所述第一类函数对所述应用程序的漏洞进行检测。通过上述技术方案，解决了漏洞检测方案存在效率低、且检测结果不全面的技术问题。



CN 106295348 A

1.一种应用程序的漏洞检测方法，其特征在于，包括：

将待检测应用程序的安装文件转换为代码文件；

获取指定函数的描述信息，其中，所述指定函数为用于反映指定行为的函数；

依据所述描述信息，在所述代码文件中确定与指定描述信息匹配的第一类函数，并依据所述第一类函数对所述应用程序的漏洞进行检测。

2.根据权利要求1所述的方法，其特征在于，依据所述第一类函数对所述应用程序的漏洞进行检测，包括：

构建所述第一类函数所在分支的第一控制流图CFG，并统计所有所述第一类函数中具有指定特征值的第二类函数；

在所述第一CFG中查找所述第二类函数所在的分支，并判断所述第二类函数所在的分支是否进行了异常处理，在判断结果为是时，则确定所述应用程序不存在漏洞；在所述判断结果为否时，则确定所述应用程序存在漏洞。

3.根据权利要求2所述的方法，其特征在于，所述异常处理包括：

设置用于指示在当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

4.根据权利要求2所述的方法，其特征在于，在所述第一CFG中查找所述第二类函数所在的分支，包括：

按照预设规则对所述第一CFG中的分支进行过滤，得到第二CFG；

在所述第二CFG中查找所述第二类函数所在的分支。

5.根据权利要求4所述的方法，其特征在于，按照预设规则对所述第一CFG中的分支进行过滤，包括：

删除所述第一CFG中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。

6.根据权利要求2所述的方法，其特征在于，所述指定特征值包括：所述第一类函数的返回值。

7.根据权利要求1所述的方法，其特征在于，获取指定函数的描述信息，包括：

从网络侧的开源文档中获取所述指定函数的描述信息。

8.根据权利要求7所述的方法，其特征在于，从网络侧的开源文档中获取所述指定函数的描述信息，包括：

通过网络爬虫的方式从所述开源文档中获取所述描述信息。

9.根据权利要求1至8中任一项所述的方法，其特征在于，所述用于反映指定行为的函数为API函数，和/或所述漏洞为拒绝服务DOS漏洞。

10.一种应用程序的漏洞检测装置，其特征在于，包括：

转换模块，用于将待检测应用程序的安装文件转换为代码文件；

获取模块，用于获取指定函数的描述信息，其中，所述指定函数为用于反映指定行为的函数；

检测模块，用于依据所述描述信息，在所述代码文件中确定与指定描述信息匹配的第一类函数，并依据所述第一类函数对所述应用程序的漏洞进行检测。

11.根据权利要求10所述的装置，其特征在于，所述检测模块，包括：

构建单元，构建所述第一类函数所在分支的第一控制流图CFG；

统计单元，用于统计所有所述第一类函数中具有指定特征值的第二类函数；

检测单元，用于在所述第一 CFG 中查找所述第二类函数所在的分支，并判断所述第二类函数所在的分支是否进行了异常处理，在判断结果为是时，则确定所述应用程序不存在漏洞；在所述判断结果为否时，则确定所述应用程序存在漏洞。

12. 根据权利要求 11 所述的装置，其特征在于，所述异常处理包括：设置用于指示在当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

13. 根据权利要求 11 所述的装置，其特征在于，所述检测单元，还用于按照预设规则对所述第一 CFG 中的分支进行过滤，得到第二 CFG；以及在所述第二 CFG 中查找所述第二类函数所在的分支。

14. 根据权利要求 13 所述的装置，其特征在于，所述检测单元，用于删除所述第一 CFG 中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。

15. 根据权利要求 10 所述的装置，其特征在于，所述获取模块，还用于从网络侧的开源文档中获取指定函数的描述信息。

应用程序的漏洞检测方法及装置

技术领域

[0001] 本发明涉及漏洞检测领域，具体而言，涉及一种应用程序的漏洞检测方法及装置。

背景技术

[0002] 随着智能移动终端的快速发展，基于移动操作系统的应用程序也层出不穷，但是，由于开发者众多，应用程序中不可避免地存在安全漏洞。例如，在Android应用程序存在的漏洞中，有一类影响面比较广的漏洞就是Java空指针(Null Pointer)拒绝服务(Denial Of Service，简称为DOS)漏洞(以下简称DOS漏洞)。这一类漏洞多是由于程序在调用一些系统API的时候处理参数不当造成的程序崩溃，导致正常功能无法使用从而导致DOS。

[0003] 目前检测这类漏洞的主要方式是模糊测试(Fuzz Testing)，即通过向目标程序的所有可能入口发送随机数据，并观察程序是否会出现异常。如图1所示，Fuzz监测程序是否存在漏洞的原理如下；

[0004] 用于做畸变的样本(sample)被读入Fuzz框架(framework)，经过分析交给转换模块(mutation)进行随机化的变异，之后通过桥接部分(bridge)传递给目标程序(target)，此时目标应用程序运行在自己的平台(Platform runtime)上，Fuzz框架通过监测模块(monitor)监测目标应用程序的运行状态，发现异常时记录到日志(logger)中。

[0005] 这种方式由于依赖程序的具体执行情况，不能保证遍历所有程序的代码分支，发现漏洞的效率较低，并且由于每秒可能产生几百甚至上千个Fuzz用例，即使检测到目标程序的异常也需要花费很多的精力重现异常以确定漏洞具体存在的点以及能够产生的影响。而且由于目标程序是在目标平台上执行，有些比较深的程序分支很难被查找到。

[0006] 针对上述的问题，目前尚未提出有效的解决方案。

发明内容

[0007] 本发明实施例提供了一种应用程序的漏洞检测方法及装置，以至少解决漏洞检测方案存在效率低、且检测结果不全面等技术问题。

[0008] 根据本发明实施例的一个方面，提供了一种应用程序的漏洞检测方法，包括：将待检测应用程序的安装文件转换为代码文件；获取指定函数的描述信息，其中，所述指定函数为用于反映指定行为的函数；依据所述描述信息，在所述代码文件中确定与指定描述信息匹配的第一类函数，并依据所述第一类函数对所述应用程序的漏洞进行检测。

[0009] 根据本发明实施例的另一方面，还提供了一种应用程序的漏洞检测装置，包括：转换模块，用于将待检测应用程序的安装文件转换为代码文件；获取模块，用于获取指定函数的描述信息，其中，所述指定函数为用于反映指定行为的函数；检测模块，用于依据所述描述信息，在所述代码文件中确定与指定描述信息匹配的第一类函数，并依据所述第一类函数对所述应用程序的漏洞进行检测。

[0010] 在本发明实施例中，采用将应用程序的安装文件转换为代码文件并从该代码文件中查找用于反映指定行为的函数，以及依据描述信息与指定描述信息匹配的第一类函数对

应用程序的漏洞进行检测的方式，达到了通过静态分析的方式进行漏洞检测的目的，同时，由于是以应用程序的代码文件为基础进行检测，可以对代码文件中的各个分支进行遍历检测，因此，可以使得检测结果更加全面，进而解决了漏洞检测方案存在的效率低且检测结果不全面等技术问题。

附图说明

[0011] 此处所说明的附图用来提供对本发明的进一步理解，构成本申请的一部分，本发明的示意性实施例及其说明用于解释本发明，并不构成对本发明的不当限定。在附图中：

[0012] 图 1 是根据相关技术的一种利用 Fuzz 监测应用程序的原理示意图；

[0013] 图 2 是本发明实施例的一种用于实现应用程序的漏洞检测方法的计算机终端的硬件结构框图；

[0014] 图 3 是根据本发明实施例的一种可选的应用程序的漏洞检测方法的示意图；

[0015] 图 4 是根据本发明实施例的一种可选的第一 CFG 的生成过程示意图；

[0016] 图 5 是根据本发明实施例的一种 CFG 的示意图；

[0017] 图 6 是根据本发明实施例的一种可选的应用程序的漏洞检测方法的另一示意图；

[0018] 图 7 是根据本发明实施例的一种可选的应用程序的漏洞检测装置的示意图；

[0019] 图 8 是根据本发明实施例的一种可选的应用程序的漏洞检测装置的另一示意图；

[0020] 图 9 是根据本发明实施例的一种计算机终端的结构框图。

具体实施方式

[0021] 为了使本技术领域的人员更好地理解本发明方案，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本发明一部分的实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例，都应当属于本发明保护的范围。

[0022] 需要说明的是，本发明的说明书和权利要求书及上述附图中的术语“第一”、“第二”等是用于区别类似的对象，而不必用于描述特定的顺序或先后次序。应该理解这样使用的数据在适当情况下可以互换，以便这里描述的本发明的实施例能够以除了在这里图示或描述的那些以外的顺序实施。此外，术语“包括”和“具有”以及他们的任何变形，意图在于覆盖不排他的包含，例如，包含了一系列步骤或单元的过程、方法、系统、产品或设备不必限于清楚地列出的那些步骤或单元，而是可包括没有清楚地列出的或对于这些过程、方法、产品或设备固有的其它步骤或单元。

[0023] 实施例 1

[0024] 根据本发明实施例，还提供了一种应用程序的漏洞检测方法的方法实施例，需要说明的是，在附图的流程图示出的步骤可以在诸如一组计算机可执行指令的计算机系统中执行，并且，虽然在流程图中示出了逻辑顺序，但是在某些情况下，可以以不同于此处的顺序执行所示出或描述的步骤。

[0025] 本申请实施例一所提供的方法实施例可以在移动终端、计算机终端或者类似的运算装置中执行。以运行在计算机终端上为例，图 2 是本发明实施例的一种用于实现应用程

序的漏洞检测方法的计算机终端的硬件结构框图。如图 2 所示，计算机终端 20 可以包括一个或多个（图中仅示出一个）处理器 202（处理器 202 可以包括但不限于微处理器 MCU 或可编程逻辑器件 FPGA 等的处理装置）、用于存储数据的存储器 204、以及用于通信功能的传输装置 206。本领域普通技术人员可以理解，图 2 所示的结构仅为示意，其并不对上述电子装置的结构造成限定。例如，计算机终端 20 还可包括比图 2 中所示更多或者更少的组件，或者具有与图 2 所示不同的配置。

[0026] 存储器 204 可用于存储应用软件的软件程序以及模块，如本发明实施例中的应用程序的漏洞检测方法对应的程序指令 / 模块，处理器 202 通过运行存储在存储器 204 内的软件程序以及模块，从而执行各种功能应用以及数据处理，即实现上述的应用程序的漏洞检测方法。存储器 204 可包括高速随机存储器，还可包括非易失性存储器，如一个或者多个磁性存储装置、闪存、或者其他非易失性固态存储器。在一些实例中，存储器 204 可进一步包括相对于处理器 202 远程设置的存储器，这些远程存储器可以通过网络连接至计算机终端 20。上述网络的实例包括但不限于互联网、企业内部网、局域网、移动通信网及其组合。

[0027] 传输装置 206 用于经由一个网络接收或者发送数据。上述的网络具体实例可包括计算机终端 20 的通信供应商提供的无线网络。在一个实例中，传输装置 206 包括一个网络适配器 (Network Interface Controller，NIC)，其可通过基站与其他网络设备相连从而可与互联网进行通讯。在一个实例中，传输装置 206 可以为射频 (Radio Frequency，RF) 模块，其用于通过无线方式与互联网进行通讯。

[0028] 在上述运行环境下，本申请提供了如图 3 所示的应用程序的漏洞检测方法。图 3 是根据本发明实施例一的应用程序的漏洞检测方法的流程图。如图 3 所示，该方法包括：

[0029] 步骤 S302，将待检测应用程序的安装文件转换为代码文件；

[0030] 对于该处理步骤，可以通过相关技术中的解决方案实现，例如，对于安卓应用程序的安装包文件，可以通过 APKTool 将应用程序的安装包 (Android Package，简称为 APK) 文件转换成 smali 代码；其中，APKTool 是谷歌 (GOOGLE) 提供的 APK 编译工具，能够反编译及回编译 apk，同时安装反编译系统 apk 所需要的 framework-res 框架，清理反编译文件等功能，smali 为安卓系统中 Java 虚拟机 (Dalvik) 所使用的一种 .dex 格式文件的汇编器。通过步骤 S302，实现了将安装包文件转换为代码文件，为后续的静态分析提供了依据。

[0031] 需要说明的是，上述代码文件并不限于汇编文件，还可以表现为源代码文件等底层代码文件（即可执行的代码文件）。在实际应用中，由于源代码文件不易获取，可以优先转换为汇编代码文件等。

[0032] 步骤 S304，获取指定函数的描述信息，其中，该指定函数为用于反映指定行为的函数。

[0033] 可选地，在执行步骤 S304 的过程中，可以包括一个查找上述指定函数的步骤，即可以从上述代码文件中查找用于反映指定行为的函数；当然也可以在具体的漏洞检测过程中不执行该查找步骤。

[0034] 此处的指定行为可以表现为函数所执行的功能类型，即函数类型。对于该处理步骤，针对不同类型的漏洞检测，可以使用用于反映不同行为的函数，例如在检测 dos 漏洞时，可以在利用应用程序安装包转换得到的汇编文件中的 API 函数进行检测。

[0035] 可选地，对于上述指定函数的描述信息的获取方式有多种，例如可以从网络侧的

开源文档中获取上述函数的描述信息，具体可以通过网络爬虫的方式从上述开源文档中爬取上述描述信息。为便于理解，以下以利用安卓应用程序中的 API 函数检测 dos 漏洞为例进行说明：

[0036] 在 Google Android API 文档中找出可能返回空值 (NULL) 的 API。由于 Google Android 的 API 文档是公开的，并且格式统一，描述规范，因此，可以通过抓取这些文档并分析其中的内容来找出可能返回 NULL 的 API。

[0037] 例如：利用以下程序中的返回值描述信息确定所需要的 API：

[0038] public Bundle getBundleExtra(String name)

[0039] Retrieve extended data from the intent.

[0040] Parameters

[0041] name The name of the des ired item.

[0042] Returns

[0043] the value of an item that previous ly added wi th putExtra()or null if no Bundle value was found.

[0044] See Also

[0045] putextra(String,Bundle)

[0046] 以上是一段 Google API 文档的中的程序，文档中描述了 getBundleExtra 这一函数的参数、功能以及返回值。其中返回值部分（划线部分）明确指出此函数可能返回 null，由于 Google 的 API 文档格式是比较标准和规范的，所有可能返回 null 的 API 都会在返回值描述中说明（见上述程序代码中的划线部分），因此，可以通过简单的文本搜索方式找出可能返回 null 的 API，以用于后续验证是否存在 DOS 漏洞。

[0047] 在确定完使用的开源文档后，可选地，可以通过以下步骤确定可能返回 null 的 API 函数：1. 通过爬虫抓取 Android API 文档；2. 分析每个 API 的返回值，找到可能返回 NULL 的 API。

[0048] 步骤 S306，依据上述描述信息，在代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行检测。

[0049] 仍然以步骤 S304 中识别 dos 漏洞为例进行说明，在该步骤 S306 中的指定描述信息可以表现为返回值为 null，上述第一类函数可以表现为返回值可能为 null 的 API 函数，但不限于此。

[0050] 可选地，在步骤 S306 中，依据上述第一类函数对上述应用程序的漏洞进行检测可以通过以下方式实现，但不限于此：构建上述第一类函数所在分支的第一控制流图 CFG（简称：第一 CFG），并统计所有上述第一类函数中具有指定特征值的第二类函数；在上述第一 CFG 中查找上述第二类函数所在的分支，并判断上述分支是否进行了异常处理，在判断结果为是时，则确定上述应用程序不存在漏洞；在上述判断结果为否时，则确定上述应用程序存在漏洞。可选地，上述异常处理可以表现为以下形式，但不限于此：设置用于指示当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

[0051] 需要说明的是，上述指定特征值可以表现为同一类函数所具有的相同特征值，例如函数的返回值，例如可以通过查找返回值有可能为空的特征值确定上述第二类函数，但不限于此种表现形式。

[0052] 在一个可选实施例中，第一 CFG 的生成过程为：将 Android 应用程序安装包 (APK) 转化成 Smali 代码，并通过静态代码分析生成应用程序的 CFG。其中，CFG 是一个以应用程序代码为节点的有向图，边的方向代表调用方向，即程序的执行方向。如图 4 所示，包括以下步骤：

[0053] 步骤 S402. 通过 apktool 解出 APK 中的 smali 代码；

[0054] 步骤 S404. 基于 Smali 中的函数调用关系以及代码的分支逻辑生成 CFG。具体地，该步骤可以通过以下过程实现：

[0055] 1. 将 smali 代码打碎成很多个块 (chunk)。一个 chunk 是代码被顺序执行的最大单元。即代码中遇到分支跳转（循环也是条件分支跳转的一种），函数调用等能够改变程序执行流程的指令时即结束当前 chunk，并开始下一个 chunk。每一个 chunk 都有一个唯一的 id，也是它们的入口点，是一个相对于函数起始地址的偏移。对于跳转指令或函数调用指令可以通过分析指令的操作数来计算出当前 chunk 在执行时可能的后续 chunk id。

[0056] 2. 将这些 chunk 通过自身 id 和后续 chunk id 的值对接在一起，即构造出 CFG。

[0057] 其中，基于上述处理过程，可以构建的一个 CFG 的示意图，需要说明的是，此处为便于理解，以下 CFG 的各个分支采用了自然语言的描述，并未用代码表示，在实际应用时，各个步骤可以表现为实现以下功能的代码。如图 5 所示：

[0058] 1、获取传入的 intent，检查 intent 是否包含参数，如果是，转步骤 2，否则转步骤 4；

[0059] 2、从 intent 中获取参数，强制转换或自定义类；

[0060] 3、调用类的方法；

[0061] 4、结束程序。

[0062] 在一个可选实施例中，在上述第一 CFG 中查找上述第二类函数所在的分支时，可以通过以下过程实现，但不限于此：按照预设规则对上述第一 CFG 中的分支进行过滤，得到第二 CFG；在上述第二 CFG 中查找上述第二类函数所在的分支。其中，上述预设规则可以根据实际情况灵活设定，例如可以按照以下方式上述第一 CFG 中的分支进行过滤：删除上述第一 CFG 中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。以基于安卓系统的应用程序为例进行说明。

[0063] 对分支进行过滤，又称为 CFG 剪枝。Android 系统上的应用程序都有特定的函数入口，需要找出 CFG 中只有出度没有入度的节点。如果这个节点的函数不是已知的程序入口，就将所有只能被这个节点连通的点去除。这样就能够保证 CFG 中剩下的节点都是可以被外部程序（如攻击者的程序）调用到的。具体可以通过以下步骤实现：

[0064] 1. 总结 Android 系统的应用程序入口函数

[0065] 2. 找出 CFG 中只有出度没有入度的点

[0066] 3. 判断这个点是不是入口函数

[0067] 4. 遍历当前节点的所有子结点，删除所有只能被这个节点连通的点

[0068] 在一个可选实施例中，判断 CFG 的分支是否做了异常处理可以通过以下方式实现：遍历所有 CFG 中的分支；查找包含了可能返回 NULL 的分支；判断这个分支是否做了异常处理。

[0069] 通过上述实施例可以看出，本发明实施例提供的应用程序的漏洞检测方法可以用

于检测 DOS 漏洞，此时，本发明实施例中用于反映指定行为的函数可以为 API 函数。

[0070] 需要说明的是，本发明实施例提供的技术方案可以运行于不同的移动终端操作系统，即可以用于检测基于不同操作系统的应用程序的漏洞，该操作系统包括但不限于：安卓(Android)操作系统、iOS 操作系统、Symbian、Windows Phone 操作系统以及 BlackBerry OS 操作系统等。

[0071] 在本实施例中，采用将应用程序的安装文件转换为代码文件并从该代码文件中查找用于反映指定行为的函数，以及依据描述信息与指定描述信息匹配的第一类函数对应用程序的漏洞进行检测的方式，达到了通过静态分析的方式进行漏洞检测的目的，同时，由于可以对上述代码文件中的各个分支进行遍历检测，因此，可以使得检测结果更加全面，进而解决了漏洞检测方案存在效率低、且检测结果不全面等技术问题。

[0072] 需要说明的是，对于前述的各方法实施例，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，本发明并不受所描述的动作顺序的限制，因为依据本发明，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发明所必须的。

[0073] 通过以上的实施方式的描述，本领域的技术人员可以清楚地了解到根据上述实施例的方法可借助软件加必需的通用硬件平台的方式来实现，当然也可以通过硬件，但很多情况下前者是更佳的实施方式。基于这样的理解，本发明的技术方案本质上或者说对现有技术做出贡献的部分可以以软件产品的形式体现出来，该计算机软件产品存储在一个存储介质（如 ROM/RAM、磁碟、光盘）中，包括若干指令用以使得一台终端设备（可以是手机，计算机，服务器，或者网络设备等）执行本发明各个实施例所述的方法。

[0074] 实施例 2

[0075] 本实施例以检测基于安卓操作系统的应用程序的 DOS 漏洞为例进行说明，但是需要说明的是，本实施例中的方案并不限于应用于安卓操作系统的应用检测，也不限于 DOS 漏洞。本实施例的主要设计思想在于，通过官方文档（即开源文档）来过滤出特定 API，并以此作为检测 DOS 漏洞的依据，即对于开源的操作系统，根据开源文档中描述的系统 API 的行为来找出感兴趣的特定 API，基于这些 API 结合一些其它方法实现特定的功能（如本实施例中的 DOS 漏洞检测）。主要包括以下过程：1. 基于静态代码产生 CFG(Controll Flow Graph)；2. 通过静态分析剪枝，过滤出可能被外界调用到的 API 分支；3. 结合 Google Android API 文档识别潜在导致 DOS 漏洞的 API；4. 判断分支是否进行了异常处理。具体地，如图 6 所示，本发明实施例提供的应用程序的漏洞检测方法包括以下处理步骤：

[0076] 步骤 S602，使用 apktool 将 apk 转换成 smali 代码；

[0077] 步骤 S604，通过分析 smali 代码构建 CFG。该步骤主要将 Android 应用程序 (APK) 转化成 Smali 代码，并通过静态代码分析生成应用程序的 CFG。CFG 是一个以应用程序代码为节点的有向图，边的方向代表调用方向，即程序的执行方向。具体实现过程如下：1、将 smali 代码打碎成很多个 chunk。一个 chunk 是代码被顺序执行的最大单元。即代码中遇到分支跳转（循环也是条件分支跳转的一种），函数调用等能够改变程序执行流程的指令时即结束当前 chunk，并开始下一个 chunk。每一个 chunk 都有一个唯一的 id，也是入口点，是一个相对于函数起始地址的偏移。对于跳转指令或函数调用指令可以通过分析指令的操

作数来计算出当前 chunk 在执行时可能的后续 chunk id。2. 将这些 chunk 通过自身 id 和后续 chunk id 的值对接在一起，即构造出 CFG。

[0078] 步骤 S606，抽取出所有入口函数所在的分支，组成新的 CFG。Android 系统上的应用程序都有特定的函数入口，找出 CFG 中只有出度没有入度的节点。如果这个节点的函数不是已知的程序入口，就将所有只能被这个节点连通的点去除。这样就能够保证 CFG 中剩下的节点都是可以被外部程序（如攻击者的程序）调用到的。

[0079] 步骤 S608，通过 HTTP 请求爬取 Google 的 Android API 文档。这部分主要从 Google Android API 文档中找出可能返回 NULL 的 API。由于 Google Android 的 API 文档是公开的，并且格式统一，描述规范，所以可以通过抓取这些文档并分析其中的内容来找出可能返回 NULL 的 API，例如可以通过文档中的返回值描述信息来确定返回 NULL 的 API 函数。

[0080] 步骤 S610，将文档存入本地数据库（可以简单以文件形式存储）。该步骤也可以通过缓存的形式实现，即将文档存入缓存中，并设置文档的存活时间等。

[0081] 步骤 S612，通过字符串搜索找出可能返回 NULL 的 API。

[0082] 步骤 S614，找出目标 API 列表 (target API list)，即利用这些 API 构建出可能导致 DOS 漏洞的 API 列表。

[0083] 步骤 S616，进行分支过滤 (branch filter)，即在 CFG 中找出包含危险 API 的分支。

[0084] 步骤 S618，进行 try/catch parser 处理，即判断此分支是否做了异常处理。

[0085] 本发明实施例通过静态的方式结合 Google API 文档发现存在漏洞的入口点。由于是静态分析，所以可以保证遍历到所有的应用程序分支，并且相对于 Fuzz 方式，静态分析拥有更高的效率，检测到的漏洞可以精确定位。本发明实施例的每一步的运算都是确定的，相对于 Fuzz 依赖程序的具体运行状况，本发明可以保证找到所有可能产生 DOS 漏洞的点。

[0086] 实施例 3

[0087] 根据本发明实施例，还提供了一种用于实施上述方法的应用程序的漏洞检测装置，该装置可以运行于实施例 1 中所述的移动终端、计算机终端或者类似的运算装置中，但不限于实施例 1 中上述运算装置的功能或结构。如图 7 所示，该装置包括：

[0088] 转换模块 70，用于将待检测应用程序的安装文件转换为代码文件。对于该模块实现的功能，可以通过相关技术中的解决方案实现，例如，对于安卓应用程序的安装包文件，可以通过 APKTool 将应用程序的安装包 (Android Package，简称为 APK) 文件转换成 smali 代码，但不限于该种实现方式。

[0089] 获取模块 72，连接至转换模块 70，用于获取指定函数的描述信息。此处的指定行为可以表现为函数所执行的功能类型，即函数类型。对于该处理步骤，针对不同类型的漏洞检测，可以使用用于反映不同行为的函数，例如在检测 dos 漏洞时，可以在利用应用程序安装包转换得到的汇编文件中的 API 函数进行检测。可选地，获取模块 72 获取上述指定函数的描述信息的有多种，例如可以从网络侧的开源文档中获取上述函数的描述信息，具体可以通过网络爬虫的方式从上述开源文档中爬取上述描述信息，此时，为便于抓取上述描述信息，可以利用描述信息比较标准和规范的文档。

[0090] 检测模块 74，连接至获取模块 72，用于依据上述描述信息，在上述代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行

检测。

[0091] 可选地，如图 8 所示，检测模块 74，还可以包括以下处理单元，但不限于此：

[0092] 构建单元 740，构建上述第一类函数所在分支的第一控制流图 CFG；

[0093] 以基于安卓操作系统的应用程序为例进行说明，在一个可选实施例中，第一 CFG 的构建过程为：将 Android 应用程序安装包 (APK) 转化成 Smali 代码，并通过静态代码分析生成应用程序的 CFG。其中，CFG 是一个以应用程序代码为节点的有向图，边的方向代表调用方向，即程序的执行方向。具体可以表现为以下处理步骤，但不限于此：通过 apktool 解出 APK 中的 smali 代码；基于 Smali 中的函数调用关系以及代码的分支逻辑生成 CFG。具体地，该步骤可以通过以下过程实现：

[0094] 1. 将 smali 代码打碎成很多个块 (chunk)。一个 chunk 是代码被顺序执行的最大单元。即代码中遇到分支跳转（循环也是条件分支跳转的一种），函数调用等能够改变程序执行流程的指令时即结束当前 chunk，并开始下一个 chunk。每一个 chunk 都有一个唯一的 id，也是它们的入口点，是一个相对于函数起始地址的偏移。对于跳转指令或函数调用指令可以通过分析指令的操作数来计算出当前 chunk 在执行时可能的后续 chunk id。2. 将这些 chunk 通过自身 id 和后续 chunk id 的值对接在一起，即构造出 CFG。

[0095] 统计单元 742，用于统计所有上述第一类函数中具有指定特征值的第二类函数；可选地，上述指定特征值可以表现为同一类函数所具有的相同特征值，例如函数的返回值，例如可以通过查找返回值有可能为空的特征值确定上述第二类函数，但不限于此种表现形式。

[0096] 检测单元 744，连接至构建单元 740 和统计单元 742，用于在上述第一 CFG 中查找上述第二类函数所在的分支，并判断上述第二类函数所在的分支是否进行了异常处理，在判断结果为是时，则确定上述应用程序不存在漏洞；在上述判断结果为否时，则确定上述应用程序存在漏洞。可选的，上述异常处理可以表现为以下形式，但不限于此：设置用于指示在当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

[0097] 对于检测单元 744 在第一 CFG 中查找第二类函数所在分支，为了提高效率和检测的准确性，检测单元 744，还用于按照预设规则对上述第一 CFG 中的分支进行过滤，得到第二 CFG；以及在上述第二 CFG 中查找上述第二类函数所在的分支。以下以基于安卓系统的应用程序为例进行说明。

[0098] 对分支进行过滤，又称为 CFG 剪枝。Android 系统上的应用程序都有特定的函数入口，需要找出 CFG 中只有出度没有入度的节点。如果这个节点的函数不是已知的程序入口，就将所有只能被这个节点连通的点去除。这样就能够保证 CFG 中剩下的节点都是可以被外部程序（如攻击者的程序）调用到的。具体可以通过以下步骤实现：

[0099] 1. 总结 Android 系统的应用程序入口函数；

[0100] 2. 找出 CFG 中只有出度没有入度的点；

[0101] 3. 判断这个点是不是入口函数；

[0102] 4. 遍历当前节点的所有子结点，删除所有只能被这个节点连通的点。

[0103] 在一个可选实施例中，判断 CFG 的分支是否做了异常处理可以通过以下方式实现：遍历所有 CFG 中的分支；查找包含了可能返回 NULL 的分支；判断这个分支是否做了异常处理。

[0104] 例如，检测单元744，还用于删除上述第一CFG中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。这样，便对第一CFG进行了剪枝处理，提高了检索效率和准确性。

[0105] 在一个可选实施例中，获取模块72，还用于从网络侧的开源文档中获取指定函数的描述信息。

[0106] 本实施例中所涉及的各个模块是可以通过相应地软件或硬件来实现的，对于后者，例如可以采用以下方式实现，但不限于此：转换模块70、获取模块72和检测模块74均位于同一处理器中；或者，转换模块70、获取模块72和检测模块74分别位于第一处理器、第二处理器和第三处理器中；或者，转换模块70和获取模块72位于同一处理器中，检测模块74位于另一处理器中；或者，获取模块72和检测模块74位于同一处理器中，而转换模块70位于另一处理器中，但不限于上述组合方式。

[0107] 通过本发明实施例提供的应用程序的漏洞检测装置，同样可以达到通过静态分析的方式进行漏洞检测的目的，同时，由于每一步均是确定的，因此，可以使得检测结果更加全面，进而解决了漏洞检测方案存在效率低、且检测结果不全面等技术问题。

[0108] 实施例4

[0109] 本发明的实施例可以提供一种计算机终端，该计算机终端可以是计算机终端群中的任意一个计算机终端设备。可选地，在本实施例中，上述计算机终端也可以替换为移动终端等终端设备。

[0110] 可选地，在本实施例中，上述计算机终端可以位于计算机网络的多个网络设备中的至少一个网络设备。

[0111] 在本实施例中，上述计算机终端可以执行应用程序的漏洞检测方法中以下步骤的程序代码：将待检测应用程序的安装文件转换为代码文件；获取指定函数的描述信息，其中，上述指定函数为用于反映指定行为的函数；依据上述描述信息，在上述代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行检测。

[0112] 可选地，图9是根据本发明实施例的一种计算机终端的结构框图。如图9所示，该计算机终端A可以包括：一个或多个（图中仅示出一个）处理器91、存储器93、以及传输装置95。

[0113] 其中，存储器93可用于存储软件程序以及模块，如本发明实施例中的安全漏洞检测方法和装置对应的程序指令/模块，处理器91通过运行存储在存储器93内的软件程序以及模块，从而执行各种功能应用以及数据处理，即实现上述的系统漏洞攻击的检测方法。存储器93可包括高速随机存储器，还可以包括非易失性存储器，如一个或者多个磁性存储装置、闪存、或者其他非易失性固态存储器。在一些实例中，存储器93可进一步包括相对于处理器91远程设置的存储器，这些远程存储器可以通过网络连接至终端A。上述网络的实例包括但不限于互联网、企业内部网、局域网、移动通信网及其组合。

[0114] 上述的传输装置95用于经由一个网络接收或者发送数据。上述的网络具体实例可包括有线网络及无线网络。在一个实例中，传输装置95包括一个网络适配器（Network Interface Controller，NIC），其可通过网线与其他网络设备与路由器相连从而可与互联网或局域网进行通讯。在一个实例中，传输装置95为射频（Radio Frequency，RF）模块，其

用于通过无线方式与互联网进行通讯。

[0115] 其中，具体地，存储器 93 用于存储预设动作条件和预设权限用户的信息、以及应用程序。

[0116] 处理器 91 可以通过传输装置调用存储器 93 存储的信息及应用程序，以执行下述步骤：将待检测应用程序的安装文件转换为代码文件；获取指定函数的描述信息，其中，上述指定函数为用于反映指定行为的函数；依据上述描述信息，在上述代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行检测。

[0117] 可选的，上述处理器 91 还可以执行如下步骤的程序代码：构建上述第一类函数所在分支的第一控制流图 CFG，并统计所有上述第一类函数中具有指定特征值的第二类函数；在上述第一 CFG 中查找上述第二类函数所在的分支，并判断上述第二类函数所在的分支是否进行了异常处理，在判断结果为是时，则确定上述应用程序不存在漏洞；在上述判断结果为否时，则确定上述应用程序存在漏洞。可选地，此处的“异常处理”可以表现为：设置用于指示在当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

[0118] 可选的，上述处理器 91 还可以执行如下步骤的程序代码：按照预设规则对上述第一 CFG 中的分支进行过滤，得到第二 CFG；在上述第二 CFG 中查找上述第二类函数所在的分支。

[0119] 可选的，上述处理器 91 还可以执行如下步骤的程序代码：删除上述第一 CFG 中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。

[0120] 可选的，上述处理器 91 还可以执行如下步骤的程序代码：从网络侧的开源文档中获取上述指定函数的描述信息，例如可以通过网络爬虫的方式从开源文档中爬取上述描述信息。

[0121] 采用本发明实施例，提供了一种利用静态代码结合其中所涉及函数的描述信息检测应用程序的漏洞的方案。解决了漏洞检测方案存在效率低、且检测结果不全面的技术问题。

[0122] 本领域普通技术人员可以理解，图 9 所示的结构仅为示意，计算机终端也可以是智能手机（如 Android 手机、iOS 手机等）、平板电脑、掌声电脑以及移动互联网设备(Mobile Internet Devices,MID)、PAD 等终端设备。图 9 其并不对上述电子装置的结构造成限定。例如，计算机终端 A 还可包括比图 9 中所示更多或者更少的组件（如网络接口、显示装置等），或者具有与图 9 所示不同的配置。

[0123] 本领域普通技术人员可以理解上述实施例的各种方法中的全部或部分步骤是可以通过程序来指令终端设备相关的硬件来完成，该程序可以存储于一计算机可读存储介质中，存储介质可以包括：闪存盘、只读存储器(Read-Only Memory,ROM)、随机存取器(Random Access Memory,RAM)、磁盘或光盘等。

[0124] 实施例 4

[0125] 本发明的实施例还提供了一种存储介质。可选地，在本实施例中，上述存储介质可以用于保存上述实施例一所提供的应用程序的漏洞检测方法所执行的程序代码。

[0126] 可选地，在本实施例中，上述存储介质可以位于计算机网络中计算机终端群中的任意一个计算机终端中，或者位于移动终端群中的任意一个移动终端中。

[0127] 可选地，在本实施例中，存储介质被设置为存储用于执行以下步骤的程序代码：将待检测应用程序的安装文件转换为代码文件；获取指定函数的描述信息，其中，上述指定函数为用于反映指定行为的函数；依据上述描述信息，在上述代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行检测。

[0128] 可选的，上述存储介质还可以执行如下步骤的程序代码：构建上述第一类函数所在分支的第一控制流图CFG，并统计所有上述第一类函数中具有指定特征值的第二类函数；在上述第一CFG中查找上述第二类函数所在的分支，并判断上述第二类函数所在的分支是否进行了异常处理，在判断结果为是时，则确定上述应用程序不存在漏洞；在上述判断结果为否时，则确定上述应用程序存在漏洞。可选地，此处的“异常处理”可以表现为：设置用于指示在当前指令异常时，跳转至其他指令的跳转指令或者调用指令。

[0129] 可选的，上述存储介质还可以执行如下步骤的程序代码：按照预设规则对上述第一CFG中的分支进行过滤，得到第二CFG；在上述第二CFG中查找上述第二类函数所在的分支。

[0130] 可选的，上述存储介质还可以执行如下步骤的程序代码：删除上述第一CFG中的指定节点以及仅能被该指定节点连通的节点，其中，该指定节点所对应的函数为仅有出度且没有入度的函数。

[0131] 可选的，上述存储介质还可以执行如下步骤的程序代码：从网络侧的开源文档中获取上述指定函数的描述信息，例如可以通过网络爬虫的方式从开源文档中爬取上述描述信息。

[0132] 此处需要说明的是，上述计算机终端群中的任意一个可以与网站服务器和扫描器建立通信关系，扫描器可以扫描计算机终端上php执行的web应用程序的值命令。

[0133] 上述本发明实施例序号仅仅为了描述，不代表实施例的优劣。

[0134] 在本发明的上述实施例中，对各个实施例的描述都各有侧重，某个实施例中没有详述的部分，可以参见其他实施例的相关描述。

[0135] 在本申请所提供的几个实施例中，应该理解到，所揭露的实体设备，可通过其它的方式实现。其中，以上所描述的装置实施例仅仅是示意性的，例如所述单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，例如多个单元或组件可以结合或者可以集成到另一个系统，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些接口，单元或模块的间接耦合或通信连接，可以是电性或其它的形式。

[0136] 所述作为分离部件说明的单元可以是或者也可以不是物理上分开的，作为单元显示的部件可以是或者也可以不是物理单元，即可以位于一个地方，或者也可以分布到多个网络单元上。可以根据实际的需要选择其中的部分或者全部单元来实现本实施例方案的目的。

[0137] 另外，在本发明各个实施例中的各功能单元可以集成在一个处理单元中，也可以是各个单元单独物理存在，也可以两个或两个以上单元集成在一个单元中。上述集成的单元既可以采用硬件的形式实现，也可以采用软件功能单元的形式实现。

[0138] 所述集成的单元如果以软件功能单元的形式实现并作为独立的产品销售或使用时，可以存储在一个计算机可读取存储介质中。基于这样的理解，本发明的技术方案本质上

或者说对现有技术做出贡献的部分或者该技术方案的全部或部分可以以软件产品的形式体现出来，该计算机软件产品存储在一个存储介质中，包括若干指令用以使得一台计算机设备（可为个人计算机、服务器或者网络设备等）执行本发明各个实施例所述方法的全部或部分步骤。而前述的存储介质包括：U盘、只读存储器（ROM，Read-Only Memory）、随机存取存储器（RAM，Random Acces s Memory）、移动硬盘、磁碟或者光盘等各种可以存储程序代码的介质。

[0139] 以上所述仅是本发明的优选实施方式，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。

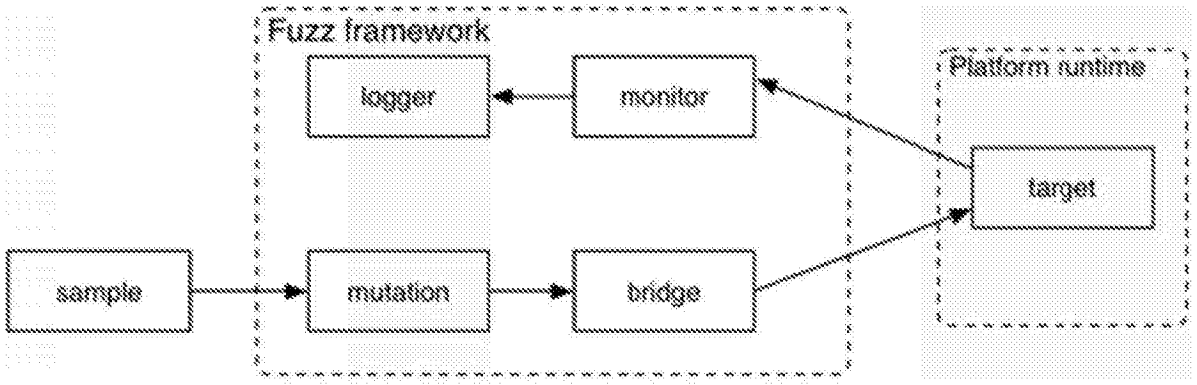

Fuzz framework
logger
monitor
Platform runtime
target
sample
mutation
bridge

图 1

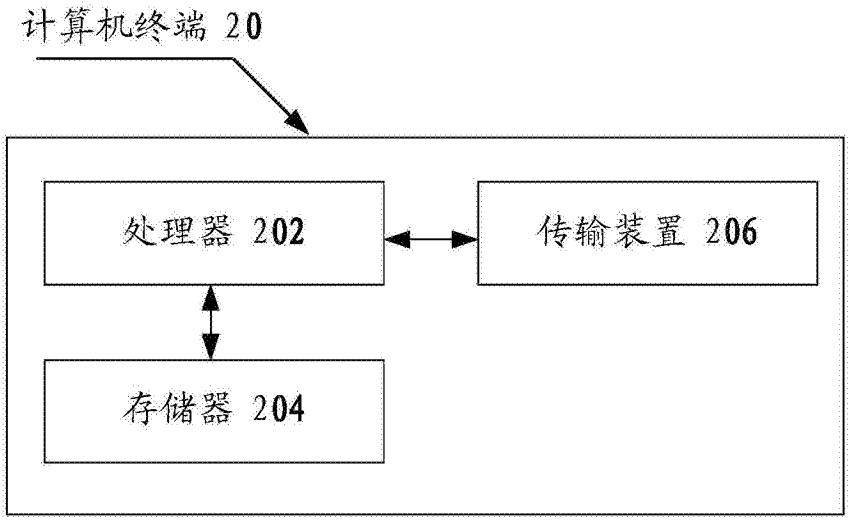

计算机终端 20
处理器 202
传输装置 206
存储器 204

图 2

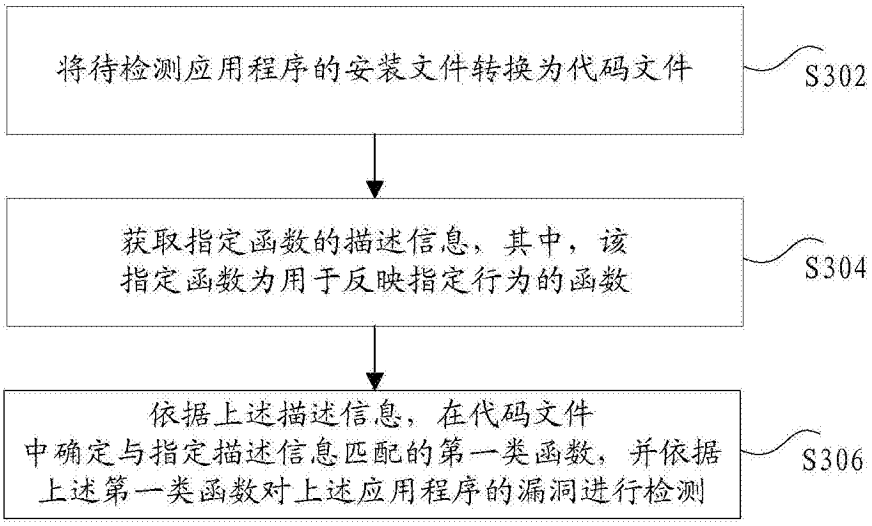

将待检测应用程序的安装文件转换为代码文件
S302
获取指定函数的描述信息，其中，该指定函数为用于反映指定行为的函数
S304
依据上述描述信息，在代码文件中确定与指定描述信息匹配的第一类函数，并依据上述第一类函数对上述应用程序的漏洞进行检测
S306

图 3

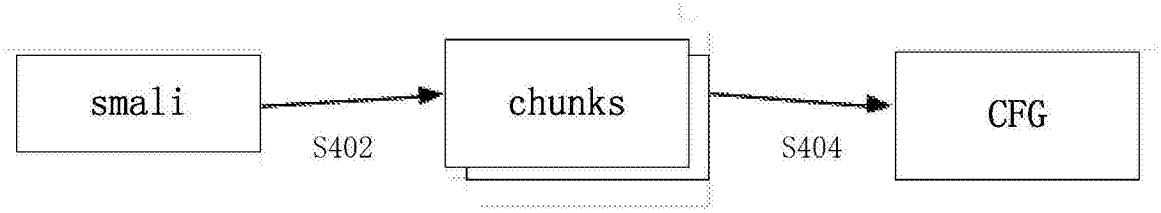

smali
S402
chunks
S404
CFG

图 4

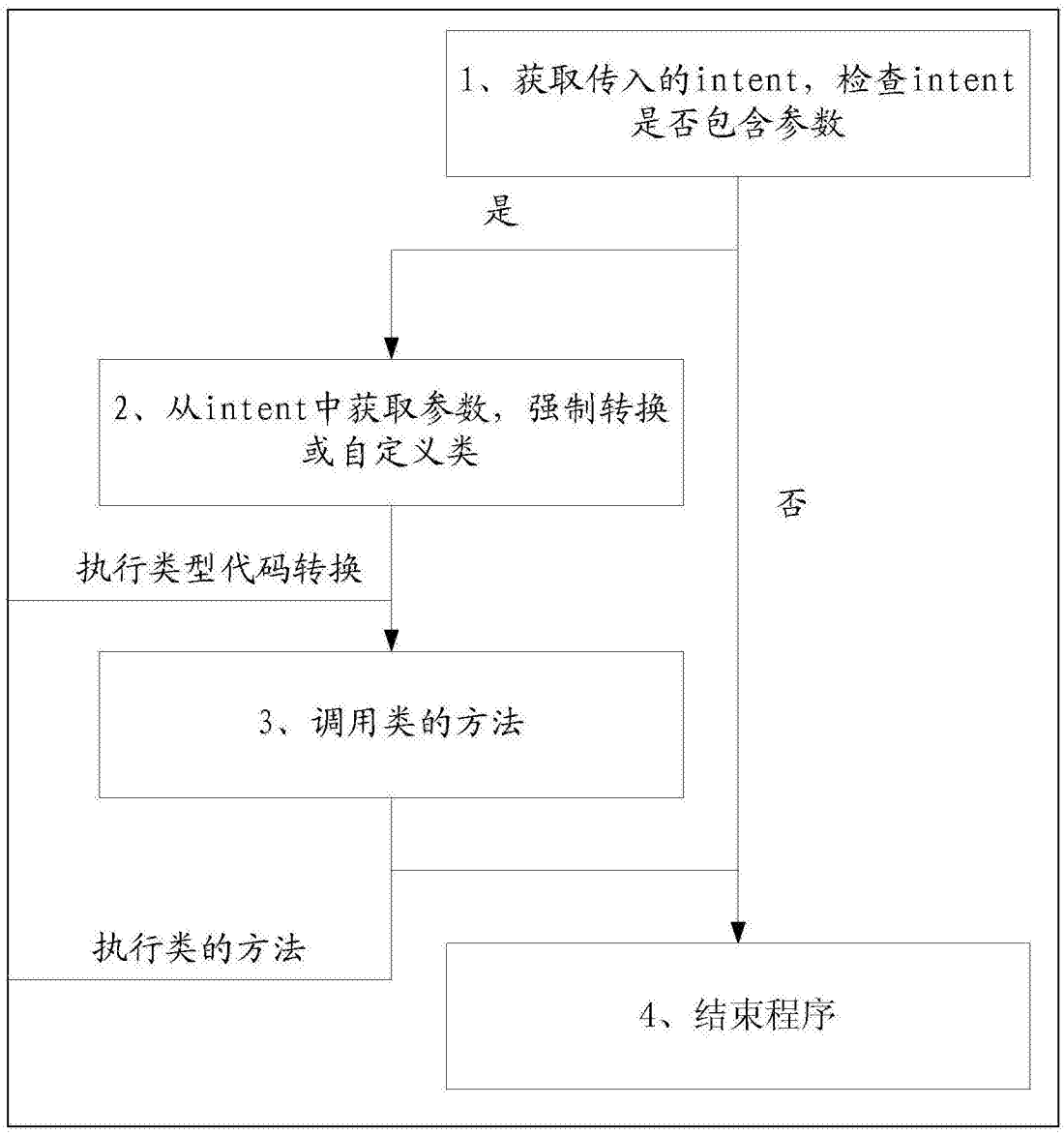

1、获取传入的intent，检查intent是否包含参数
是
否
2、从intent中获取参数，强制转换或自定义类
执行类型代码转换
3、调用类的方法
执行类的方法
4、结束程序

图 5

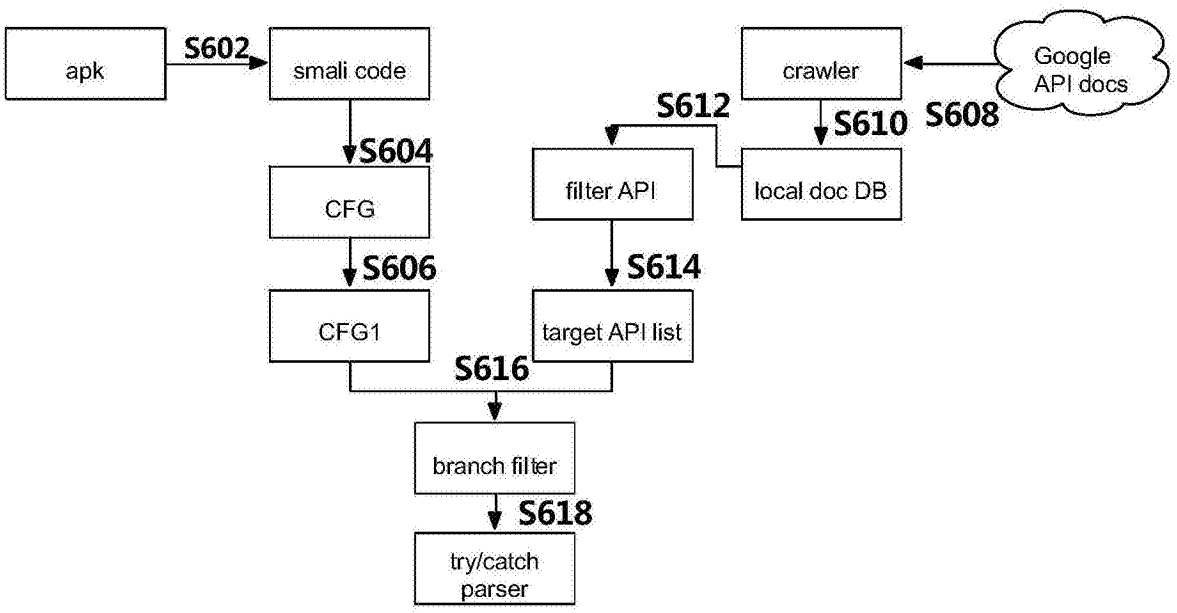

apk
S602
smali code
S604
CFG
S606
CFG1
S616
branch filter
S618
try/catch parser
crawler
Google API docs
S608
S610
S612
filter API
local doc DB
S614
target API list

图 6

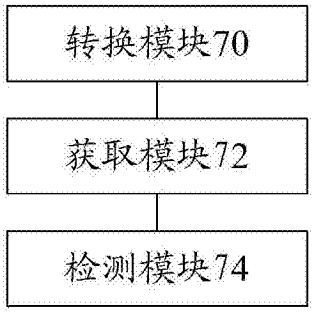

转换模块70
获取模块72
检测模块74

图 7

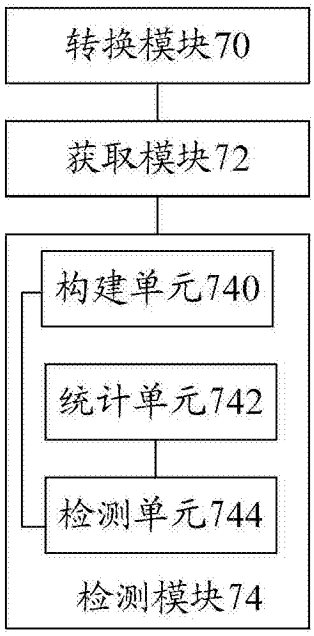

转换模块70
获取模块72
构建单元740
统计单元742
检测单元744
检测模块74

图 8

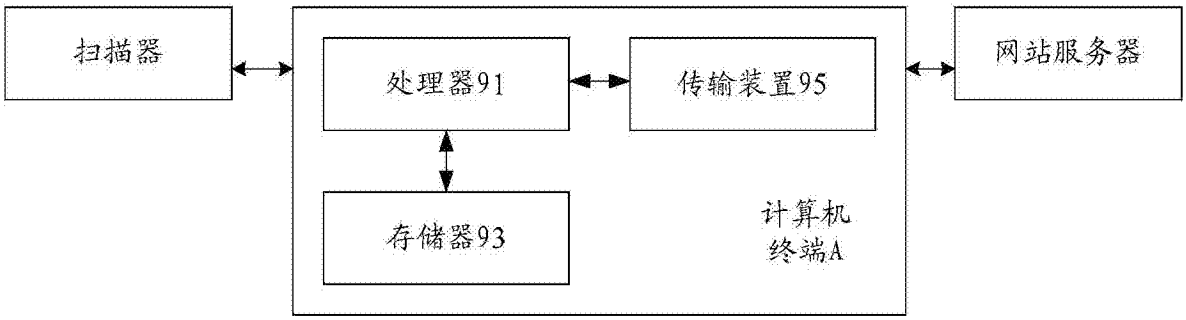


图 9