(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 106295353 B
(45)授权公告日 2020.04.07

(21)申请号 201610643329.9

(22)申请日 2016.08.08

(65)同一申请的已公布的文献号
申请公布号 CN 106295353 A

(43)申请公布日 2017.01.04

(73)专利权人 腾讯科技(深圳)有限公司
地址 518000 广东省深圳市福田区振兴路赛格科技园2栋东403室

(72)发明人 张蓓 邹越 袁明凯 严明 魏学峰

(74)专利代理机构 深圳市深佳知识产权代理事务所(普通合伙) 44285
代理人 王仲凯

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

(56)对比文件
CN 104537309 A,2015.04.22,权利要求1-10,说明书第0028、0032-0037、0046段.
CN 103473509 A,2013.12.25,权利要求1-2.
CN 105827664 A,2016.08.03,全文.
US 2013298245 A1,2013.11.07,全文.

审查员 吴广平

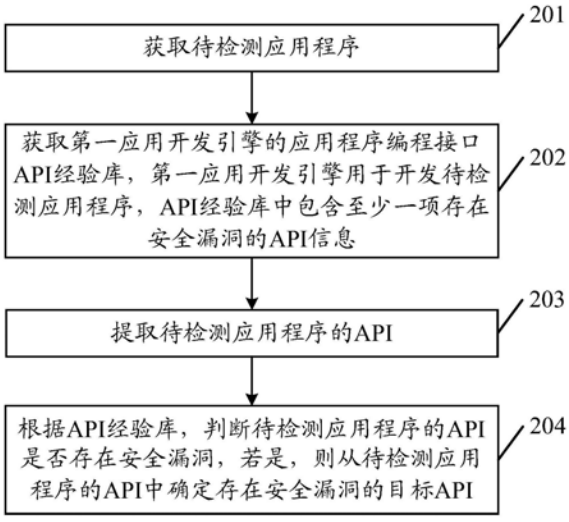
权利要求书3页 说明书14页 附图9页

(54)发明名称

一种引擎漏洞检测的方法以及检测装置

(57)摘要

本发明实施例公开了一种引擎漏洞检测的方法,包括:获取待检测应用程序;获取第一应用开发引擎的应用程序编程接口API经验库,所述第一应用开发引擎用于开发所述待检测应用程序,所述API经验库中包含至少一项存在安全漏洞的API信息;提取所述待检测应用程序的API;根据所述API经验库,判断所述待检测应用程序的API是否存在安全漏洞,若是,则从所述待检测应用程序的API中确定存在安全漏洞的目标API。本发明实施例还提供一种检测装置。本发明实施例无需为每个应用开发引擎编写测试用例,而是根据安全漏洞的API信息建立起应用开发引擎的API经验库,通过该API经验库扫描出待检测应用程序中存在的API漏洞,从而降低了API漏洞的检测成本。



CN 106295353 B

1.一种引擎漏洞检测的方法,其特征在于,包括:

获取待检测应用程序;

获取第一应用开发引擎的应用程序编程接口经验库,所述第一应用开发引擎用于开发所述待检测应用程序,所述应用程序编程接口经验库中包含至少一项存在安全漏洞的应用程序编程接口信息,且所述存在安全漏洞的应用程序编程接口信息是从多个版本所对应的应用开发引擎的源代码中获取到的,所述第一应用开发引擎属于所述多个版本中的其中一个版本,所述第一应用开发引擎为Cocos开源框架,所述应用程序编程接口信息包括出现安全漏洞的应用程序编程接口名称以及在整段源代码中出现的位置;

提取所述待检测应用程序的应用程序编程接口;

根据所述应用程序编程接口经验库,判断所述待检测应用程序的应用程序编程接口是否存在安全漏洞,若是,则从所述待检测应用程序的应用程序编程接口中确定存在安全漏洞的目标应用程序编程接口;

定位到高危险性应用程序编程接口的代码行。

2.根据权利要求1所述的方法,其特征在于,所述获取第一应用开发引擎的应用程序编程接口经验库之前,所述方法还包括:

获取所述第一应用开发引擎的源代码;

查找所述第一应用开发引擎的源代码中存在安全漏洞的应用程序编程接口;

根据所述存在安全漏洞的应用程序编程接口,建立所述应用程序编程接口经验库。

3.根据权利要求2所述的方法,其特征在于,所述查找所述第一应用开发引擎的源代码中存在安全漏洞的应用程序编程接口,包括:

检测所述第一应用开发引擎的源代码中存在空指针的应用程序编程接口;

获取所述存在空指针的应用程序编程接口所对应的应用程序编程接口名称和参数位置。

4.根据权利要求3所述的方法,其特征在于,所述根据所述存在安全漏洞的应用程序编程接口,建立所述应用程序编程接口经验库,包括:

根据所述存在空指针的应用程序编程接口所对应的应用程序编程接口名称和参数位置,建立所述第一应用开发引擎的应用程序编程接口经验库。

5.根据权利要求2至4中任一项所述的方法,其特征在于,所述获取第一应用开发引擎的应用程序编程接口经验库之后,所述方法还包括:

当所述第一应用开发引擎更新为第二应用开发引擎时,获取所述第二应用开发引擎的源代码;

查找所述第二应用开发引擎的源代码中存在安全漏洞的应用程序编程接口;

根据所述存在安全漏洞的应用程序编程接口,将所述第一应用开发引擎的应用程序编程接口经验库更新为所述第二应用开发引擎的应用程序编程接口经验库。

6.根据权利要求4所述的方法,其特征在于,所述根据所述应用程序编程接口经验库,判断所述待检测应用程序的应用程序编程接口是否存在安全漏洞,包括:

遍历所述待检测应用程序的应用程序编程接口名称;

判断所述待检测应用程序的应用程序编程接口名称是否与所述应用程序编程接口经验库中包含的应用程序编程接口名称一致,若是,则根据所述应用程序编程接口经验库获

取所述待检测应用程序中所述应用程序编程接口名称对应的参数位置；

若所述参数位置上对应的参数为空指针，则确定所述待检测应用程序的应用程序编程接口存在安全漏洞。

7.根据权利要求1所述的方法，其特征在于，所述从所述待检测应用程序的应用程序编程接口中确定存在安全漏洞的目标应用程序编程接口之后，所述方法还包括：

输出所述待检测应用程序中的风险信息，所述风险信息包括存在所述目标应用程序编程接口的待检测应用程序的文件名、代码位置、调用的应用程序编程接口名称以及报错信息中的至少一项。

8.一种检测装置，其特征在于，包括：

第一获取模块，用于获取待检测应用程序；

第二获取模块，用于获取第一应用开发引擎的应用程序编程接口经验库，所述第一应用开发引擎用于开发所述第一获取模块获取的所述待检测应用程序，所述应用程序编程接口经验库中包含至少一项存在安全漏洞的应用程序编程接口信息，且所述存在安全漏洞的应用程序编程接口信息是从多个版本所对应的应用开发引擎的源代码中获取到的，所述第一应用开发引擎属于所述多个版本中的其中一个版本，所述第一应用开发引擎为Cocos开源框架，所述应用程序编程接口信息包括出现安全漏洞的应用程序编程接口名称以及在整段源代码中出现的位置；

提取模块，用于提取所述第一获取模块获取的所述待检测应用程序的应用程序编程接口；

确定模块，用于根据所述第二获取模块获取的所述应用程序编程接口经验库，判断所述提取模块提取的所述待检测应用程序的应用程序编程接口是否存在安全漏洞，若是，则从所述待检测应用程序的应用程序编程接口中确定存在安全漏洞的目标应用程序编程接口；

定位到高危险性应用程序编程接口的代码行。

9.根据权利要求8所述的检测装置，其特征在于，所述检测装置还包括：

第三获取模块，用于所述第二获取模块获取第一应用开发引擎的应用程序编程接口经验库之前，获取所述第一应用开发引擎的源代码；

第一查找模块，用于查找所述第三获取模块获取的所述第一应用开发引擎的源代码中存在安全漏洞的应用程序编程接口；

第一建立模块，用于根据所述第一查找模块查找的所述存在安全漏洞的应用程序编程接口，建立所述应用程序编程接口经验库。

10.根据权利要求9所述的检测装置，其特征在于，所述第一查找模块包括：

检测单元，用于检测所述第一应用开发引擎的源代码中存在空指针的应用程序编程接口；

第一获取单元，用于获取所述检测单元检测的所述存在空指针的应用程序编程接口所对应的应用程序编程接口名称和参数位置。

11.根据权利要求10所述的检测装置，其特征在于，所述第一建立模块包括：

建立单元，用于根据所述第一获取单元获取的所述存在空指针的应用程序编程接口所对应的应用程序编程接口名称和参数位置，建立所述第一应用开发引擎的应用程序编程接

口经验库。

12.根据权利要求9至11中任一项所述的检测装置，其特征在于，所述检测装置还包括：

第四获取模块，用于所述第二获取模块获取第一应用开发引擎的应用程序编程接口经验库之后，当所述第一应用开发引擎更新为第二应用开发引擎时，获取所述第二应用开发引擎的源代码；

第二查找模块，用于查找所述第四获取模块获取的所述第二应用开发引擎的源代码中存在安全漏洞的应用程序编程接口；

更新模块，用于根据所述第二查找模块查找的所述存在安全漏洞的应用程序编程接口，将所述第一应用开发引擎的应用程序编程接口经验库更新为所述第二应用开发引擎的应用程序编程接口经验库。

13.根据权利要求11所述的检测装置，其特征在于，所述确定模块包括：

遍历单元，用于遍历所述待检测应用程序的应用程序编程接口名称；

第二获取单元，用于判断所述遍历单元遍历得到的所述待检测应用程序的应用程序编程接口名称，是否与所述应用程序编程接口经验库中包含的应用程序编程接口名称一致，若是，则根据所述应用程序编程接口经验库获取所述待检测应用程序中所述应用程序编程接口名称对应的参数位置；

确定单元，用于若所述第二获取单元获取的所述参数位置上对应的参数为空指针，则确定所述待检测应用程序的应用程序编程接口存在安全漏洞。

14.根据权利要求8所述的检测装置，其特征在于，所述检测装置还包括：

输出模块，用于所述确定模块从所述待检测应用程序的应用程序编程接口中确定存在安全漏洞的目标应用程序编程接口之后，输出所述待检测应用程序中的风险信息，所述风险信息包括存在所述目标应用程序编程接口的待检测应用程序的文件名、代码位置、调用的应用程序编程接口名称以及报错信息中的至少一项。

一种引擎漏洞检测的方法以及检测装置

技术领域

[0001] 本发明涉及计算机技术领域以及信息安全技术领域，尤其涉及一种引擎漏洞检测的方法以及检测装置。

背景技术

[0002] 如今，随着信息技术的不断发展，应用程序也日益普及化。为了丰富人们的日常生活，开发人员开发了多种多样的应用程序，例如社交类应用、游戏类应用、电商类应用、搜索类应用以及论坛类应用等。然而，在开发应用程序的过程中应用程序编程接口(英文全称：Application Programming Interface，英文缩写：API)可能会出现漏洞，从而导致该应用程序在运行过程中发生宕机的情况。

[0003] 为了降低应用程序在运行过程中出现宕机的频率，目前可以采用一种手段对API是否存在漏洞进行检测。

[0004] 具体如图1所示，图1为现有技术中检测应用程序编程接口漏洞的流程示意图。步骤101中开发人员需要先熟悉应用程序核心功能和架构，然后设计可能存在性能问题的场景。步骤102中，开发人员可以为开发引擎中的每条API编写接口测试用例，使得在步骤103中能够采用每条接口测试用例对相应的每个API进行接口测试。由于修改开发引擎底层API风险较高，且不利于更新和维护引擎代码，一般需要通过步骤104来调用接口的上层逻辑代码。最后，在步骤105中，将输出步骤104中定位得到的高危API代码行，开发人员可以根据输出的高危API代码行进行及时修复。

[0005] 然而，由于开发引擎的API有很多，且存在较多的开发引擎版本，每个版本的API又存在着一定的差异，因此需要为每个开发版本编写大量的测试用例，从而耗费巨大的API漏洞检测成本。

发明内容

[0006] 本发明实施例提供了一种引擎漏洞检测的方法以及检测装置，可以不用为每个应用开发引擎编写测试用例，而是根据应用开发引擎中存在安全漏洞的API信息建立起API经验库，通过该API经验库扫描出待检测应用程序中存在的API漏洞，从而降低了API漏洞的检测成本。

[0007] 有鉴于此，本发明第一方面提供一种引擎漏洞检测的方法，包括：

[0008] 获取待检测应用程序；

[0009] 获取第一应用开发引擎的API经验库，所述第一应用开发引擎用于开发所述待检测应用程序，所述API经验库中包含至少一项存在安全漏洞的API信息；

[0010] 提取所述待检测应用程序的应用程序编程接口API；

[0011] 根据所述API经验库，判断所述待检测应用程序的API是否存在安全漏洞，若是，则从所述待检测应用程序的API中确定存在安全漏洞的目标API。

[0012] 第二方面，本方面实施例还提供一种检测装置，包括：

[0013] 第一获取模块，用于获取待检测应用程序；

[0014] 第二获取模块，用于获取第一应用开发引擎的API经验库，所述第一应用开发引擎用于开发所述第一获取模块获取的所述待检测应用程序，所述API经验库中包含至少一项存在安全漏洞的API信息；

[0015] 提取模块，用于提取所述第一获取模块获取的所述待检测应用程序的应用程序编程接口API；

[0016] 确定模块，用于根据所述第二获取模块获取的所述API经验库，判断所述提取模块提取的所述待检测应用程序的API是否存在安全漏洞，若是，则从所述待检测应用程序的API中确定存在安全漏洞的目标API。

[0017] 从以上技术方案可以看出，本发明实施例具有以下优点：

[0018] 本发明实施例中，提供了一种检测应用程序中API漏洞的方法，检测装置先获取待检测应用程序以及第一应用开发引擎的API经验库，其中，第一应用开发引擎用于开发待检测应用程序，该API经验库中包含至少一项存在安全漏洞的API信息，然后检测装置提取待检测应用程序的API，最后根据API经验库，判断待检测应用程序的API是否存在安全漏洞，若是，则从待检测应用程序的API中确定存在安全漏洞的目标API。采用上述方式，无需为每个应用开发引擎编写测试用例，而是根据安全漏洞的API信息建立起应用开发引擎的API经验库，通过该API经验库扫描出待检测应用程序中存在的API漏洞，从而降低了API漏洞的检测成本。

附图说明

[0019] 图1为现有技术中检测应用程序编程接口漏洞的流程示意图；

[0020] 图2为本发明实施例中引擎漏洞检测的方法一个实施例示意图；

[0021] 图3为本发明实施例中检测应用程序中API漏洞的流程示意图；

[0022] 图4为本发明实施例中建立API经验库的流程示意图；

[0023] 图5为本发明实施例中实现应用开发引擎检测项的流程示意图；

[0024] 图6为应用场景中CocosCheck工具检测前的界面显示图；

[0025] 图7为应用场景中CocosCheck工具检测时的界面显示图；

[0026] 图8为应用场景中CocosCheck工具检测后的界面显示图；

[0027] 图9为本发明实施例中检测装置一个实施例示意图；

[0028] 图10为本发明实施例中检测装置另一个实施例示意图；

[0029] 图11为本发明实施例中检测装置另一个实施例示意图；

[0030] 图12为本发明实施例中检测装置另一个实施例示意图；

[0031] 图13为本发明实施例中检测装置另一个实施例示意图；

[0032] 图14为本发明实施例中检测装置另一个实施例示意图；

[0033] 图15为本发明实施例中检测装置另一个实施例示意图；

[0034] 图16为本发明实施例中检测装置一个结构示意图。

具体实施方式

[0035] 本发明实施例提供了一种引擎漏洞检测的方法以及检测装置，无需为每个应用开

发引擎编写测试用例，而是根据应用开发引擎中存在安全漏洞的API信息建立起API经验库，通过该API经验库扫描出待检测应用程序中存在的API漏洞，从而降低了API漏洞的检测成本。

[0036] 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”、“第三”、“第四”等(如果存在)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例如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。

[0037] 应理解，本发明实施例中的应用开发引擎主要是一种基于开源软件许可协议(英文全称:Massachusetts Institute of Technology，英文缩写:MIT)的开源框架，具体可以为Cocos的开源框架。Cocos提供了全套的引擎和开发工具，涵盖从前期设计、资源制作、开发调试和打包上线全套的解决方案。Cocos用于构建游戏、应用程序和其他图形界面交互应用，可以重点优化工作流，规范了整个开发流程，降低沟通成本，提高开发效率。

[0038] Cocos的开源框架的功能主要有流程控制，流程控制可以非常容易地管理不同场景之间的流程控制。除了流程控制的功能，还具有以下功能，例如：

[0039] (1)动作(英文全称:Sprites)功能，可组合的动作如移动、旋转和缩放等；

[0040] (2)特效(英文全称:Effects)功能，包括波浪、旋转和透镜等；

[0041] (3)平面地图(英文全称:Tiled Maps)功能，支持包括矩形和六边形平面地图；

[0042] (4)转换(英文全称:Transitions)功能，从一个场景移动到另外一个不同风格的场景；

[0043] (5)菜单(英文全称:Menus)功能，创建内部菜单；

[0044] (6)文本渲染(英文全称:Text Rendering)功能，支持标签和超级文本标记语言(英文全称:HyperText Markup Language，英文缩写:HTML)；

[0045] (7)文档(英文全称:Documents功能，编程指南、API参考、视频教学以及很多教用户如何使用的简单测试例子。

[0046] 在实际应用中，Cocos的开源框架作为交互式应用程序的引擎开发工具，还可能会包括或者增强更多的功能，此处不一一穷举。

[0047] 应理解，本发明实施例中的待检测应用程序可以是游戏应用，但并不仅限于游戏应用，还可以是其他类型的应用，例如社交类应用、媒体播放类应用或者服务类应用等，此处不作限定。

[0048] 而本发明中的检测装置具体可以是一款针对Cocos开源框架而采用的检测工具，该检测工具可称为CocosCheck，CocosCheck在终端上显示为一个客户端，通过CocosCheck的客户端将待检测应用程序包含的相关文件进行添加、检测以及输出，以使得用户开发人员能够迅速定位待检测应用程序中出现API漏洞的代码位置，并且及时地进行修复，以恢复待检测应用程序的正常运行。

[0049] 此外，本发明能够应用于微软公司开发的视窗操作系统(英文全称:Windows Operation System，英文缩写:Windows OS)，Windows OS采用了图形用户界面(英文全称:

Graphical User Interface,英文缩写:GUI)。随着电脑硬件和软件的不断升级,微软的Windows也在不断升级,需要说明的是,本方案能够应用于视窗操作体验版(英文全称:Windows Experience,英文缩写:Windows XP)OS、Windows 7OS、Windows 8OS或者Windows 10OS,还可以应用于其他类型的Windows OS,此处不做限定。

[0050] 请参阅图2,本发明实施例中引擎漏洞检测的方法一个实施例包括:

[0051] 201、获取待检测应用程序;

[0052] 本实施例中,检测装置获取待检测应用程序,其中,该待检测应用程序是通过第一应用开发引擎开发而成的,第一应用开发引擎可以是Cocos的开源框架,而检测装置则可以是针对Cocos开源框架设计的一个用于检测API漏洞的工具。

[0053] 202、获取第一应用开发引擎的API经验库,第一应用开发引擎用于开发待检测应用程序,API经验库中包含至少一项存在安全漏洞的API信息;

[0054] 本实施例中,检测装置进而调用并获取第一应用开发引擎对应的API经验库。其中,API经验库中包含了至少一项存在安全漏洞的API信息,存在安全漏洞的API信息即为存在高危险性的API信息,通过该API信息可以确定高危险性API类型的提炼规则。

[0055] 具体地,例如API信息包括了出现安全漏洞的API名称以及在整段源代码中出现的位置,高危险性API类型的提炼规则即为根据API名称可以定位到该API在整段源代码中出现的位置。由此,检测装置能够利用这一提炼规则找出待检测应用程序中可能出现API漏洞的代码位置。

[0056] 203、提取待检测应用程序的应用程序编程接口API;

[0057] 本实施例中,检测装置可以利用关键字提取待检测应用程序的API。

[0058] API是一些预先定义的函数,目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力,而又无需访问源代码或者理解内部工作机制的细节。

[0059] 204、根据API经验库,判断待检测应用程序的API是否存在安全漏洞,若是,则从待检测应用程序的API中确定存在安全漏洞的目标API。

[0060] 本实施例中,检测装置在获取到第一应用开发引擎所对应的API经验库后,判断待检测应用程序的API是否存与API经验库中出现安全漏洞的API信息一致,例如是否具有相同的API名称,如果有,那么就认为该待检测应用程序的API中确定存在安全漏洞的目标API。

[0061] 为了便于理解,请参阅图3,图3为本发明实施例中检测应用程序中API漏洞的流程示意图,如图所示,具体为:

[0062] 步骤301中,通过静态扫描分析的手段,分析应用开发引擎的源代码,其中,对于应用开发引擎而言通常会出现多个版本,当前最新的版本即为第一应用开发引擎。具体为,挖掘各个版本Cocos开源框架的源代码,然后找出里头存在风险的高危险性API;

[0063] 步骤302中,通过步骤301中静态扫描分析结果,按照版本建立Cocos的高危险性API经验库,可选地,也可以每次将更新版本对应的高危险性API添加至至API经验库,即不需对每个版本的Cocos都建立各自的API经验库,而是不断地更新API经验库,从而节省计算机存储资源;

[0064] 步骤303中,CocosChec工具根据API经验库中的高危险性API类型提炼规则,实现CocosCheck工具的检查项;

[0065] 步骤304中，使用CocosCheck工具扫描待检测应用程序的源代码；

[0066] 步骤305中，CocosCheck工具扫描已定位到调用高危险性API的代码行，开发人员直接进行修改即可。

[0067] 本发明实施例中，无需为每个应用开发引擎编写测试用例，而是根据安全漏洞的API信息建立起应用开发引擎的API经验库，通过该API经验库扫描出待检测应用程序中存在的API漏洞，从而降低了API漏洞的检测成本。

[0068] 可选地，在上述图2对应的实施例的基础上，本发明实施例提供的引擎漏洞检测的方法第一个可选实施例中，获取第一应用开发引擎的API经验库之前，还可以包括：

[0069] 获取第一应用开发引擎的源代码；

[0070] 查找第一应用开发引擎的源代码中存在安全漏洞的API；

[0071] 根据存在安全漏洞的API，建立API经验库。

[0072] 本实施例中，在检测装置获取第一应用开发燃具的API经验库之前，已经提前建立起了该API经验库。

[0073] 具体地，请参阅图4，图4为本发明实施例中建立API经验库的流程示意图，如图所示，通过以下流程中的步骤就可以建立起第一应用开发引擎的API经验库。

[0074] 步骤401中，先获取第一应用开发引擎，即Cocos的各个版本源代码，如果只有一个版本，获取该版本的源代码即可。

[0075] 步骤402中，然后遍历第一应用开发引擎的各个版本源代码，所谓遍历就是指沿着某条搜索路线，依次对每个结点均做一次且仅做一次访问。

[0076] 步骤403中，在步骤402中已经遍历了第一应用开发引擎的各个版本源代码，从中可以找到最新版本对应的源代码，通过静态代码分析当前版本的源代码中存在空指针风险的API。

[0077] 其中，采用静态代码分析不不用运行代码，只是通过对代码的静态扫描对程序进行分析，执行速度快、效率高。目前成熟的代码静态分析工具每秒可扫描上万行代码，相对于动态分析，具有检测速度快、效率高的特点。

[0078] 需要说明的是，本方案采用的静态代码分析技术可以是词法分析，或者语法分析，或者抽象语法树分析，还可以是语义分析、控制流分析或者数据流分析，在实际应用中，还可以采用其他类型的静态代码分析技术，此处不作限定。

[0079] 步骤404中，经过步骤403中的静态代码分析后，得到最新版本对应的源代码中存在高危API风险的API名称和相应的参数位置，并记录下这些高危API信息。

[0080] 步骤405中，根据步骤404记录的API信息建立起最新版本对应的API经验库。由于第一应用开发引擎Cocos的版本会不断更新，因此完成步骤405后还将再次执行步骤402，以此保证API经验库覆盖所有第一应用开发引擎的版本以及对应的API信息。

[0081] 可选地，静态代码分析还可以进一步通过代码扫描检测不同脚本语言在调用参数时的合法性，如果脚本语言之间互相调用时存在参数不匹配，或者调用的API不存在时，也可能存在风险。

[0082] 其次，本发明实施例中，还可以通过对第一应用开发引擎的源代码查找存在安全漏洞的API，以此建立API经验库，使得该API经验库中的API信息更加完整，从而提升方案的可行性和实用性。

[0083] 可选地，在上述图2对应的第一个实施例的基础上，本发明实施例提供的引擎漏洞检测的方法第二个可选实施例中，查找第一应用开发引擎的源代码中存在安全漏洞的API，可以包括：

[0084] 检测第一应用开发引擎的源代码中存在空指针的API；

[0085] 获取存在空指针的API所对应的API名称和参数位置。

[0086] 本实施例中，检测装置在查找第一应用开发引擎Cocos开源框架中的源代码是否存在具有安全漏洞API，一种可行方式为，检测源代码中是否存在空指针的API，如果存在，就获取存在空指针API的API名称和参数位置。

[0087] 其中，空指针一般的文档中倾向于用NULL表示，对于指针类型来说，返回NULL和返回0是等价的，因为NULL和0都表示“null pointer(空指针)”。

[0088] 关于自动定位空指针异常的方式，具体可以为，首先结合程序的静态分析技术，利用程序运行时的堆栈信息指导程序切片，然后对得到的切片进行空指针分析及别名分析，得出引发空指针异常的可疑语句集合，最终给出错误定位报告。在实际应用中，可以存在其他方式定位空指针，此处不做限定。

[0089] 再次，本发明实施例中，具体定义了检测API漏洞的依据可以是检测源代码中的API是否存在空指针，由于空指针的出现是引起宕机的关键因素，因此将其作为API漏洞检测的标准更有利于提升方案的实用性，同时，检测空指针的方式也较为简便，从而提升了方案的检测效率。

[0090] 可选地，在上述图2对应的第二个实施例的基础上，本发明实施例提供的引擎漏洞检测的方法第三个可选实施例中，根据存在安全漏洞的API，建立API经验库，可以包括：

[0091] 根据存在空指针的API所对应的API名称和参数位置，建立第一应用开发引擎的API经验库。

[0092] 本实施例中，检测装置对于存在安全漏洞的API，将建立起对应的API经验库，该API经验库中包括了高危险性API的相关信息，例如存在空指针的API所对应的API名称和参数位置。

[0093] 以下是一种Cocos的高危API经验库示例，如下表1所示：

[0094] 表1

[0095]

Cocos 版本	高危 API	参数位置	优化建议
2.1.1	initWithMaskSprite	1	调用该 API 时保证传入的第 1 个参数不会为 NULL
2.1.1	cpSpaceEachConstraint	3	调用该 API 时保证传入的 3 个参数不会为 NULL
2.1.4	kmGLGetMatrix	2	调用该 API 时保证传入的第 2 个参数不会为 NULL
2.1.4	cpRecenterPoly	2	调用该 API 时保证传入的第 2 个参数不会为 NULL

[0096] 进一步地，本发明实施例中，示意了API经验库建立的依据，其中存在空指针的API

被认为是高危API，于是在API经验库中会记录这些高危API的相关信息，以便后续进行漏洞检测，以此提升方案的可行性。

[0097] 可选地，在上述图2对应的第一、第二或第三个实施例的基础上，本发明实施例提供的引擎漏洞检测的方法第四个可选实施例中，获取第一应用开发引擎的API经验库之后，还可以包括：

[0098] 当第一应用开发引擎更新为第二应用开发引擎时，获取第二应用开发引擎的源代码；

[0099] 查找第二应用开发引擎的源代码中存在安全漏洞的API；

[0100] 根据存在安全漏洞的API，将第一应用开发引擎的API经验库更新为第二应用开发引擎的API经验库。

[0101] 本实施例中，假设第一应用开发引擎进行了版本更新，得到更新后的第二应用开发引擎，换言之，第二应用开发引擎与第一应用开发引擎都为同一款软件，但是版本不同，其中，第二应用开发引擎比第一应用开发引擎的版本更新。

[0102] 具体地，检测装置将不断遍历第一应用开发引擎的版本，当发现存在信息版本时，可以确定第一应用开发引擎更新为第二应用开发引擎，于是检测装置再从应用开发引擎的官网上调取第二应用开发引擎的源代码。接下来，检测装置将查找第二应用开发引擎的源代码中存在安全漏洞的API，即查找是否存在空指针的API，如果存在就认为该API存在安全漏洞。

[0103] 最后，检测装置根据存在安全漏洞的API，将第一应用开发引擎的API经验库更新为第二应用开发引擎的API经验库，也就是将第二应用开发引擎的高危API信息添加至上一个版本对应的API经验库。

[0104] 更进一步地，本发明实施例中，当应用开发引擎的版本发生更新时，检测装置可以找出更新后的应用开发引擎中存在安全漏洞的API，并且直接更新API经验库，不需要再次建立更新后应用开发引擎对应的API经验库，而是将新信息添加至上一个版本的API经验库即可，不但可以提高建立API经验库的效率，而且还节省了检测装置的存储资源。

[0105] 可选地，在上述图2对应的第三个实施例的基础上，本发明实施例提供的引擎漏洞检测的方法第五个可选实施例中，根据API经验库，判断待检测应用程序的API是否存在安全漏洞，可以包括：

[0106] 遍历待检测应用程序的API名称；

[0107] 判断待检测应用程序的API名称是否与API经验库中包含的API名称一致，若是，则根据API经验库获取待检测应用程序中API名称对应的参数位置；

[0108] 若参数位置上对应的参数为空指针，则确定待检测应用程序的API存在安全漏洞。

[0109] 本实施例中，检测装置根据API经验库中的高危API信息，判断待检测应用程序的API是否存在安全漏洞的具体实现方式如下介绍：

[0110] 请参阅图5，图5为本发明实施例中实现应用开发引擎检测项的流程示意图，步骤501中，检测装置先获取待扫描应用程序的使用的第一应用开发引擎版本号，具体可以从cocos2d.cpp的cocos2dVersion函数中提取，其提取的代码如下所示：

[0111] #include“cocos2d.h”

[0112] NS_CC_BEGIN

[0113] Const char*cocos2dversion()

[0114] {

[0115] {return“2.2.1”;

[0116] }

[0117] }

[0118] 通过上述代码可以提取版本号,可选地,也可以直接提取最新版本对应的版本号。

[0119] 步骤502中,查看是否存在对应版本的API经验库,该API经验库中包含了高危API的信息,若存在,则继续进入步骤503,否则跳转至步骤510;

[0120] 步骤503中,根据第一应用开发引擎的版本号找到对应的API经验库,并提取该API经验库;

[0121] 步骤504中,对待检测应用程序中的每一个文件都进行扫描,这些文件即为待检测应用程序的源代码,遍历源代码中的每一处的函数调用。

[0122] 步骤505中,检测装置在步骤504中遍历待检测应用程序中的每一处调用后,得到各个API的函数名。

[0123] 步骤506中,接下来,检测装置将步骤505中获取到API的函数名与API经验库中的函数名进行对比,从而判断是否为高危API,如果函数名一致,则可以认为是高危API,于是进入步骤507,否则则跳转至步骤504,即继续遍历源代码中的每一处函数调用。

[0124] 步骤507中,检测装置根据API经验库中存储的高危API信息,可以查找到API函数名所对应的参数信息,例如其出现的位置以及代码内容等。

[0125] 步骤508中,判断调用API位置的参数是否为NULL,若是,则进入步骤509,否则就跳转至步骤504,,即继续遍历源代码中的每一处函数调用。

[0126] 步骤509中,输出待检测应用程序的风险信息,包括出现漏洞的文件名、代码行号、调用的API名称以及报错信息等。

[0127] 步骤510中,结束本次API漏洞的检测。

[0128] 再进一步地,本发明实施例中,提供了利用API经验库对待检测应用程序进行API漏洞检测的方式,通过API经验库中包含的高危API信息,确定待检测应用程序中可能出现的高危API,相对于现有技术,无需在定位到的高危API基础上,再查找上层调用该API的参数位置,而是直接定位到高危API的参数位置,从而极大地降低了挖掘高危API的成本。

[0129] 可选地,在上述图2对应的实施例的基础上,本发明实施例提供的引擎漏洞检测的方法六个可选实施例中,从待检测应用程序的API中确定存在安全漏洞的目标API之后,还可以包括:

[0130] 输出待检测应用程序中的风险信息,风险信息包括存在目标API的待检测应用程序的文件名、代码位置、调用的API名称以及报错信息中的至少一项。

[0131] 本实施例中,检测装置在确定待检测应用程序中存在安全漏洞的目标API后,还可以进一步地将目标API的风险信息显示在前端界面上,以供开发人员查看并及时进行修复。

[0132] 需要说明的是,目标API的风险信息包括待检测应用程序的文件名、代码位置、调用的API名称以及报错信息中的至少一项,然而在实际应用中,包括但不仅限于上述的信息。

[0133] 其中,风险信息中待检测应用程序的文件名,表示待检测应用程序中出现高危API

的文件是哪个;风险信息中代码位置,表示在待检测应用程序的源代码中出现高危API的代码位置;风险信息中调用的API名称,表示高危API的名称;风险信息中报错信息就是一个消息提示,用于提示开发人员当前的待检测应用程序出现了API漏洞。

[0134] 其次,本发明实施例中,检测装置可以展示关于待检测应用程序一系列的风险信息,使得开发人员能够及时获取到出现API漏洞的信息,从而提升修复待检测应用程序的效率。

[0135] 为便于理解,下面以一个具体应用场景对本发明中一种引擎漏洞检测的方法进行详细描述,具体为:

[0136] 开发人员甲采用Cocos开源框架开发了一款游戏,为了保证这款游戏的顺利上线,开发人员甲采用CocosCheck工具检测这款游戏是否存在API漏洞。其检测步骤如下:

[0137] 首先,开发人员甲双击电脑上的“CocosCheck.exe”工具,于是进入CocosCheck工具的使用界面,请参阅图6,图6为应用场景中CocosCheck工具检测前的界面显示图,菜单栏中的“Check”为检测功能,“File”为文件选择功能,“Edit”为编辑功能,“View”为视图功能,“Help”为帮助功能。

[0138] 接着,开发人员甲选择菜单栏中的检测功能,即点击“Check”,然后再选择“Check”中的“Directory”选项。选择后CocosCheck工具的界面上就会出现待检测的游戏源代码目录,具体如图7,图7为应用场景中CocosCheck工具检测时的界面显示图,开发人员甲选择了该游戏中所有的源代码文件进行扫描。

[0139] CocosCheck工具扫描完成后,将显示界面如图8所示,图8为应用场景中CocosCheck工具检测后的界面显示图,开发人员甲可以看到扫描完成后的结果,结果中展示调用高危API的代码文件名为“QGAuto.cpp”,对应的代码行数为247行,这个高危API名称为“initWithMaskSprite”。

[0140] 于是,开发人员迅速找到出现高危API的文件,并根据代码位置找到出错的地方,然后及时进行了修复,从而保证这款游戏顺利上线。

[0141] 下面对本发明中的检测装置进行详细描述,请参阅图9,本发明实施例中的检测装置包括:

[0142] 第一获取模块601,用于获取待检测应用程序;

[0143] 第二获取模块602,用于获取第一应用开发引擎的API经验库,所述第一应用开发引擎用于开发所述第一获取模块601获取的所述待检测应用程序,所述API经验库中包含至少一项存在安全漏洞的API信息;

[0144] 提取模块603,用于提取所述第一获取模块601获取的所述待检测应用程序的应用程序编程接口API;

[0145] 确定模块604,用于根据所述第二获取模块602获取的所述API经验库,判断所述提取模块603提取的所述待检测应用程序的API是否存在安全漏洞,若是,则从所述待检测应用程序的API中确定存在安全漏洞的目标API。

[0146] 本实施例中,第一获取模块601获取待检测应用程序,第二获取模块602获取第一应用开发引擎的API经验库,所述第一应用开发引擎用于开发所述第一获取模块601获取的所述待检测应用程序,所述API经验库中包含至少一项存在安全漏洞的API信息,提取模块603提取所述第一获取模块601获取的所述待检测应用程序的应用程序编程接口API,确定

模块604根据所述第二获取模块602获取的所述API经验库，判断所述提取模块603提取的所述待检测应用程序的API是否存在安全漏洞，若是，则从所述待检测应用程序的API中确定存在安全漏洞的目标API。

[0147] 本发明实施例中，无需为每个应用开发引擎编写测试用例，而是根据安全漏洞的API信息建立起应用开发引擎的API经验库，通过该API经验库扫描出待检测应用程序中存在的API漏洞，从而降低了API漏洞的检测成本。

[0148] 可选地，在上述图9所对应的实施例的基础上，请参阅图10，本发明实施例提供的检测装置的另一实施例中，所述检测装置还包括：

[0149] 第三获取模块605A，用于所述第二获取模块602获取第一应用开发引擎的API经验库之前，获取所述第一应用开发引擎的源代码；

[0150] 第一查找模块605B，用于查找所述第三获取模块605A获取的所述第一应用开发引擎的源代码中存在安全漏洞的API；

[0151] 第一建立模块605C，用于根据所述第一查找模块605B查找的所述存在安全漏洞的API，建立所述API经验库。

[0152] 其次，本发明实施例中，还可以通过对第一应用开发引擎的源代码查找存在安全漏洞的API，以此建立API经验库，使得该API经验库中的API信息更加完整，从而提升方案的可行性和实用性。

[0153] 可选地，在上述图10所对应的实施例的基础上，请参阅图11，本发明实施例提供的检测装置的另一实施例中，

[0154] 所述第一查找模块605B包括：

[0155] 检测单元605B1，用于检测所述第一应用开发引擎的源代码中存在空指针的API；

[0156] 第一获取单元605B2，用于获取所述检测单元605B1检测的所述存在空指针的API所对应的API名称和参数位置。

[0157] 再次，本发明实施例中，具体定义了检测API漏洞的依据可以是检测源代码中的API是否存在空指针，由于空指针的出现是引起宕机的关键因素，因此将其作为API漏洞检测的标准更有利于提升方案的实用性，同时，检测空指针的方式也较为简便，从而提升了方案的检测效率。

[0158] 可选地，在上述图11所对应的实施例的基础上，请参阅图12，本发明实施例提供的检测装置的另一实施例中，

[0159] 所述第一建立模块605C包括：

[0160] 建立单元605C1，用于根据所述第一获取单元605B2获取的所述存在空指针的API所对应的API名称和参数位置，建立所述第一应用开发引擎的API经验库。

[0161] 进一步地，本发明实施例中，示意了API经验库建立的依据，其中存在空指针的API被认为是高危API，于是在API经验库中会记录这些高危API的相关信息，以便后续进行漏洞检测，以此提升方案的可行性。

[0162] 可选地，在上述图10、图11或图12所对应的实施例的基础上，请参阅图13，本发明实施例提供的检测装置的另一实施例中，

[0163] 所述检测装置60还包括：

[0164] 第四获取模块606A，用于所述第二获取模块602获取第一应用开发引擎的API经验

库之后，当所述第一应用开发引擎更新为第二应用开发引擎时，获取所述第二应用开发引擎的源代码；

[0165] 第二查找模块606B，用于查找所述第四获取模块606A获取的所述第二应用开发引擎的源代码中存在安全漏洞的API；

[0166] 更新模块606C，用于根据所述第二查找模块606B查找的所述存在安全漏洞的API，将所述第一应用开发引擎的API经验库更新为所述第二应用开发引擎的API经验库。

[0167] 更进一步地，本发明实施例中，当应用开发引擎的版本发生更新时，检测装置可以找出更新后的应用开发引擎中存在安全漏洞的API，并且直接更新API经验库，不需要再次建立更新后应用开发引擎对应的API经验库，而是将新信息添加至上一个版本的API经验库即可，不但可以提高建立API经验库的效率，而且还节省了检测装置的存储资源。

[0168] 可选地，在上述图12所对应的实施例的基础上，请参阅图14，本发明实施例提供的检测装置的另一实施例中，

[0169] 所述确定模块604包括：

[0170] 遍历单元6041，用于遍历所述待检测应用程序的API名称；

[0171] 第二获取单元6042，用于判断所述遍历单元6041遍历得到的所述待检测应用程序的API名称，是否与所述API经验库中包含的API名称一致，若是，则根据所述API经验库获取所述待检测应用程序中所述API名称对应的参数位置；

[0172] 确定单元6043，用于若所述第二获取单元6042获取的所述参数位置上对应的参数为空指针，则确定所述待检测应用程序的API存在安全漏洞。

[0173] 再进一步地，本发明实施例中，提供了利用API经验库对待检测应用程序进行API漏洞检测的方式，通过API经验库中包含的高危API信息，确定待检测应用程序中可能出现的高危API，相对于现有技术，无需在定位到的高危API基础上，再查找上层调用该API的参数位置，而是直接定位到高危API的参数位置，从而极大地降低了挖掘高危API的成本。

[0174] 可选地，在上述图9所对应的实施例的基础上，请参阅图15，本发明实施例提供的检测装置的另一实施例中，

[0175] 所述检测装置60还包括：

[0176] 输出模块607，用于所述确定模块604从所述待检测应用程序的API中确定存在安全漏洞的目标API之后，输出所述待检测应用程序中的风险信息，所述风险信息包括存在所述目标API的待检测应用程序的文件名、代码位置、调用的API名称以及报错信息中的至少一项。

[0177] 其次，本发明实施例中，检测装置可以展示关于待检测应用程序一系列的风险信息，使得开发人员能够及时获取到出现API漏洞的信息，从而提升修复待检测应用程序的效率。

[0178] 本发明实施例还提供了另一种检测装置，如图16所示，为了便于说明，仅示出了与本发明实施例相关的部分，具体技术细节未揭示的，请参照本发明实施例方法部分。该终端可以为包括个人电脑(英文全称:Personal Computer，英文缩写:PC)、手机、平板电脑、个人数字助理(英文全称:Personal Digital Assistant，英文缩写:PDA)、销售终端(英文全称:Point of Sales，英文缩写:POS)、车载电脑等任意终端设备，以终端为PC为例：

[0179] 图16示出的是与本发明实施例提供的终端相关的PC的部分结构的框图。参考图

16,PC包括:射频(英文全称:Radio Frequency,英文缩写:RF)电路710、存储器720、输入单元730、显示单元740、传感器750、音频电路760、无线保真(英文全称:wireless fidelity,英文缩写:WiFi)模块770、处理器780、以及电源790等部件。本领域技术人员可以理解,图16中示出的PC结构并不构成对PC的限定,可以包括比图示更多或更少的部件,或者组合某些部件,或者不同的部件布置。

[0180] 下面结合图16对PC的各个构成部件进行具体的介绍:

[0181] RF电路710可用于收发信息或通话过程中,信号的接收和发送,特别地,将基站的下行信息接收后,给处理器780处理;另外,将设计上行的数据发送给基站。通常,RF电路710包括但不限于天线、至少一个放大器、收发信机、耦合器、低噪声放大器(英文全称:Low Noise Amplifier,英文缩写:LNA)、双工器等。此外,RF电路710还可以通过无线通信与网络和其他设备通信。上述无线通信可以使用任一通信标准或协议,包括但不限于全球移动通讯系统(英文全称:Global System of Mobile communication,英文缩写:GSM)、通用分组无线服务(英文全称:General Packet Radio Service,GPRS)、码分多址(英文全称:Code Division Multiple Access,英文缩写:CDMA)、宽带码分多址(英文全称:Wideband Code Division Multiple Access,英文缩写:WCDMA)、长期演进(英文全称:Long Term Evolution,英文缩写:LTE)、电子邮件、短消息服务(英文全称:Short Messaging Service,SMS)等。

[0182] 存储器720可用于存储软件程序以及模块,处理器780通过运行存储在存储器720的软件程序以及模块,从而执行PC的各种功能应用以及数据处理。存储器720可主要包括存储程序区和存储数据区,其中,存储程序区可存储操作系统、至少一个功能所需的应用程序(比如声音播放功能、图像播放功能等)等;存储数据区可存储根据PC的使用所创建的数据(比如音频数据、电话本等)等。此外,存储器720可以包括高速随机存取存储器,还可以包括非易失性存储器,例如至少一个磁盘存储器件、闪存器件、或其他易失性固态存储器件。

[0183] 输入单元730可用于接收输入的数字或字符信息,以及产生与PC的用户设置以及功能控制有关的键信号输入。具体地,输入单元730可包括触控面板731以及其他输入设备732。触控面板731,也称为触摸屏,可收集用户在其上或附近的触摸操作(比如用户使用手指、触笔等任何适合的物体或附件在触控面板731上或在触控面板731附近的操作),并根据预先设定的程式驱动相应的连接装置。可选的,触控面板731可包括触摸检测装置和触摸控制器两个部分。其中,触摸检测装置检测用户的触摸方位,并检测触摸操作带来的信号,将信号传送给触摸控制器;触摸控制器从触摸检测装置上接收触摸信息,并将它转换成触点坐标,再送给处理器780,并能接收处理器780发来的命令并加以执行。此外,可以采用电阻式、电容式、红外线以及表面声波等多种类型实现触控面板731。除了触控面板731,输入单元730还可以包括其他输入设备732。具体地,其他输入设备732可以包括但不限于物理键盘、功能键(比如音量控制按键、开关按键等)、轨迹球、鼠标、操作杆等中的一种或多种。

[0184] 显示单元740可用于显示由用户输入的信息或提供给用户的信息以及PC的各种菜单。显示单元740可包括显示面板741,可选的,可以采用液晶显示器(英文全称:Liquid Crystal Display,英文缩写:LCD)、有机发光二极管(英文全称:Organic Light-Emitting Diode,英文缩写:OLED)等形式来配置显示面板741。进一步的,触控面板731可覆盖显示面板741,当触控面板731检测到在其上或附近的触摸操作后,传送给处理器780以确定触摸事

件的类型，随后处理器780根据触摸事件的类型在显示面板741上提供相应的视觉输出。虽然在图16中，触控面板731与显示面板741是作为两个独立的部件来实现PC的输入和输入功能，但是在某些实施例中，可以将触控面板731与显示面板741集成而实现PC的输入和输出功能。

[0185] PC还可包括至少一种传感器750，比如光传感器、运动传感器以及其他传感器。具体地，光传感器可包括环境光传感器及接近传感器，其中，环境光传感器可根据环境光线的明暗来调节显示面板741的亮度，接近传感器可在PC移动到耳边时，关闭显示面板741和/或背光。作为运动传感器的一种，加速计传感器可检测各个方向上(一般为三轴)加速度的大小，静止时可检测出重力的大小及方向，可用于识别PC姿态的应用(比如横竖屏切换、相关游戏、磁力计姿态校准)、振动识别相关功能(比如计步器、敲击)等；至于PC还可配置的陀螺仪、气压计、湿度计、温度计、红外线传感器等其他传感器，在此不再赘述。

[0186] 音频电路760、扬声器761，传声器762可提供用户与PC之间的音频接口。音频电路760可将接收到的音频数据转换后的电信号，传输到扬声器761，由扬声器761转换为声音信号输出；另一方面，传声器762将收集的声音信号转换为电信号，由音频电路760接收后转换为音频数据，再将音频数据输出处理器780处理后，经RF电路710以发送给比如另一PC，或者将音频数据输出至存储器720以便进一步处理。

[0187] WiFi属于短距离无线传输技术，PC通过WiFi模块770可以帮助用户收发电子邮件、浏览网页和访问流式媒体等，它为用户提供了无线的宽带互联网访问。虽然图16示出了WiFi模块770，但是可以理解的是，其并不属于PC的必须构成，完全可以根据需要在不改变发明的本质的范围内而省略。

[0188] 处理器780是PC的控制中心，利用各种接口和线路连接整个PC的各个部分，通过运行或执行存储在存储器720内的软件程序和/或模块，以及调用存储在存储器720内的数据，执行PC的各种功能和处理数据，从而对PC进行整体监控。可选的，处理器780可包括一个或多个处理单元；优选的，处理器780可集成应用处理器和调制解调处理器，其中，应用处理器主要处理操作系统、用户界面和应用程序等，调制解调处理器主要处理无线通信。可以理解的是，上述调制解调处理器也可以不集成到处理器780中。

[0189] PC还包括给各个部件供电的电源790(比如电池)，优选的，电源可以通过电源管理系统与处理器780逻辑相连，从而通过电源管理系统实现管理充电、放电、以及功耗管理等功能。

[0190] 尽管未示出，PC还可以包括摄像头、蓝牙模块等，在此不再赘述。

[0191] 在本发明实施例中，该终端所包括的处理器780还具有以下功能：

[0192] 获取待检测应用程序；

[0193] 获取第一应用开发引擎的API经验库，所述第一应用开发引擎用于开发所述待检测应用程序，所述API经验库中包含至少一项存在安全漏洞的API信息；

[0194] 提取所述待检测应用程序的应用程序编程接口API；

[0195] 根据所述API经验库，判断所述待检测应用程序的API是否存在安全漏洞，若是，则从所述待检测应用程序的API中确定存在安全漏洞的目标API。

[0196] 所属领域的技术人员可以清楚地了解到，为描述的方便和简洁，上述描述的系统，装置和单元的具体工作过程，可以参考前述方法实施例中的对应过程，在此不再赘述。

[0197] 在本申请所提供的几个实施例中，应该理解到，所揭露的系统，装置和方法，可以通过其它的方式实现。例如，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，装置或单元的间接耦合或通信连接，可以是电性，机械或其它的形式。

[0198] 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。

[0199] 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。

[0200] 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(可以是个人计算机，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、移动硬盘、只读存储器(英文全称：Read-Only Memory，英文缩写：ROM)、随机存取存储器(英文全称：Random Access Memory，英文缩写：RAM)、磁碟或者光盘等各种可以存储程序代码的介质。

[0201] 以上所述，以上实施例仅用以说明本发明的技术方案，而非对其限制；尽管参照前述实施例对本发明进行了详细的说明，本领域的普通技术人员应当理解：其依然可以对前述各实施例所记载的技术方案进行修改，或者对其中部分技术特征进行等同替换；而这些修改或者替换，并不使相应技术方案的本质脱离本发明各实施例技术方案的精神和范围。

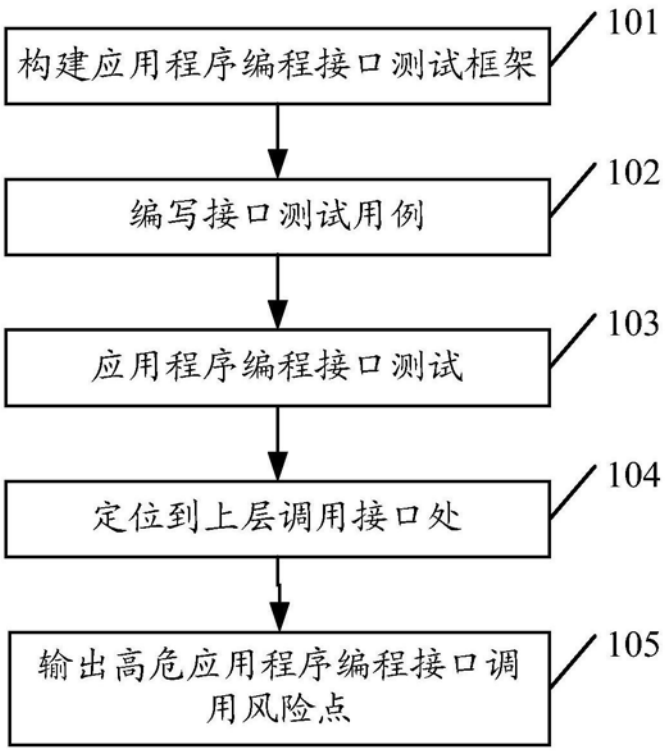

构建应用程序编程接口测试框架
101
编写接口测试用例
102
应用程序编程接口测试
103
定位到上层调用接口处
104
输出高危应用程序编程接口调用风险点
105

图1

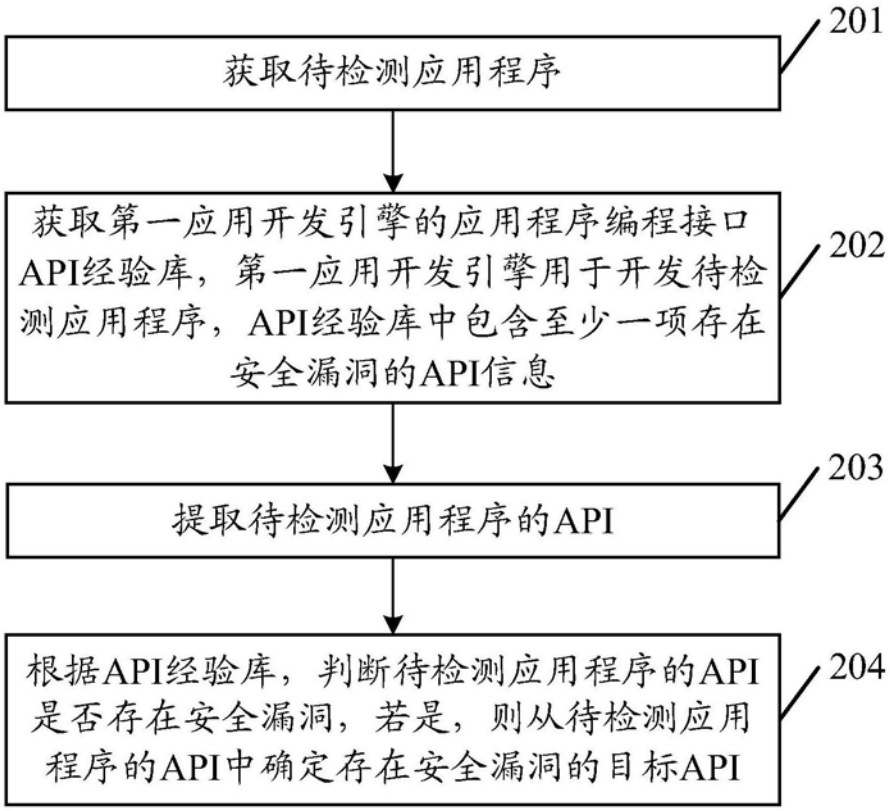

获取待检测应用程序
201
获取第一应用开发引擎的应用程序编程接口API经验库，第一应用开发引擎用于开发待检测应用程序，API经验库中包含至少一项存在安全漏洞的API信息
202
提取待检测应用程序的API
203
根据API经验库，判断待检测应用程序的API是否存在安全漏洞，若是，则从待检测应用程序的API中确定存在安全漏洞的目标API
204

图2

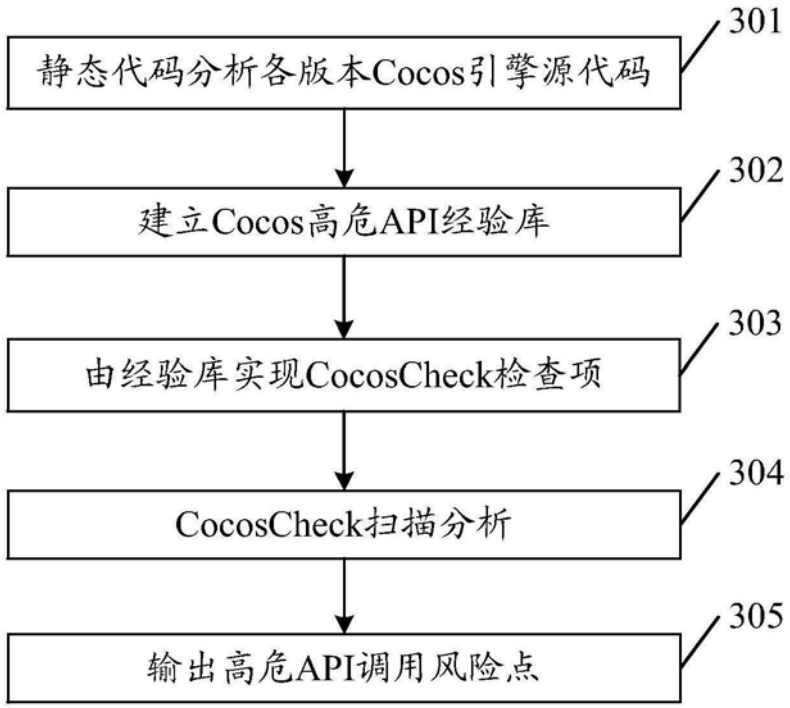

静态代码分析各版本Cocos引擎源代码
301
建立Cocos高危API经验库
302
由经验库实现CocosCheck检查项
303
CocosCheck扫描分析
304
输出高危API调用风险点
305

图3

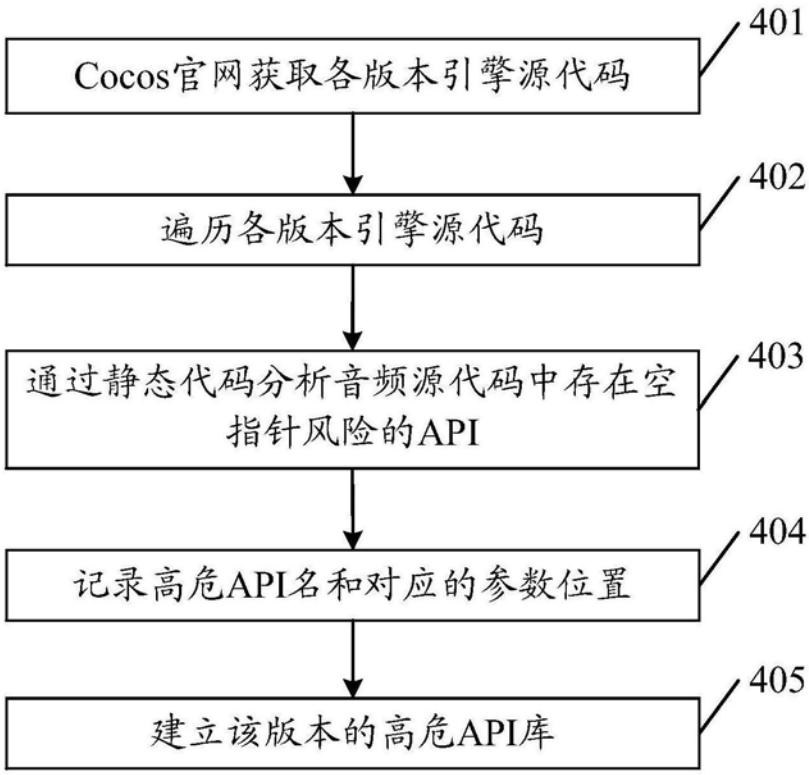

Cocos官网获取各版本引擎源代码
401
遍历各版本引擎源代码
402
通过静态代码分析音频源代码中存在空指针风险的API
403
记录高危API名和对应的参数位置
404
建立该版本的高危API库
405

图4

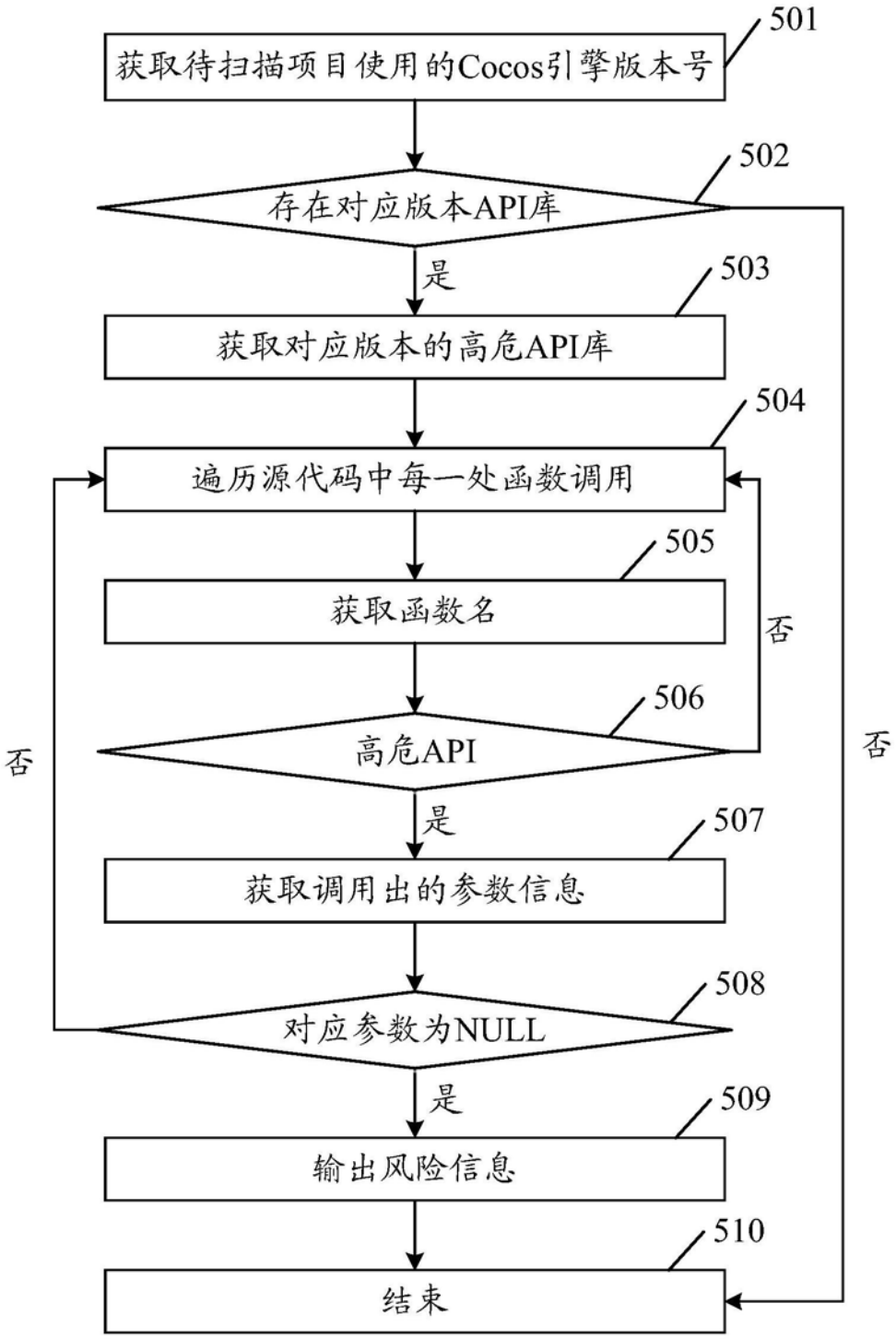

501
获取待扫描项目使用的Cocos引擎版本号
502
存在对应版本API库
是
否
503
获取对应版本的高危API库
504
遍历源代码中每一处函数调用
505
获取函数名
506
高危API
否
是
否
507
获取调用出的参数信息
508
对应参数为NULL
是
509
输出风险信息
510
结束

图5

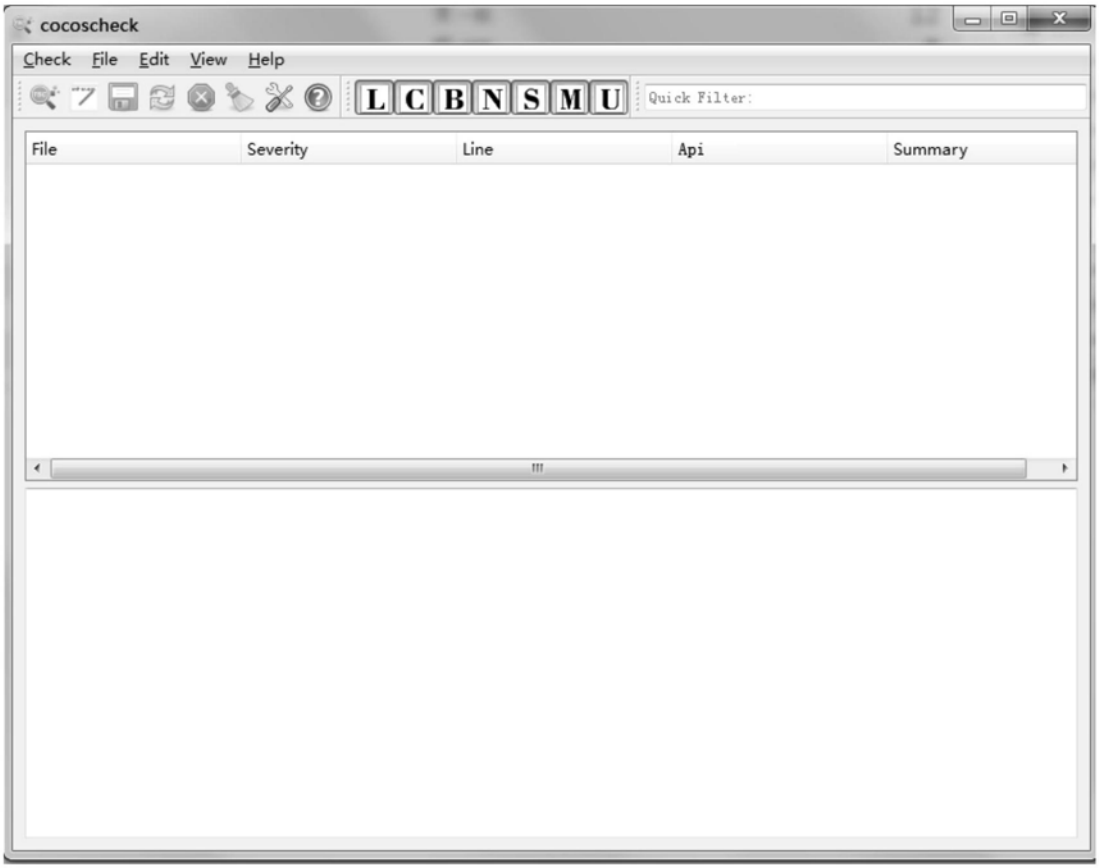

cocoscheck
Check File Edit View Help
L C B N S M U
Quick Filter:
File
Severity
Line
Api
Summary

图6

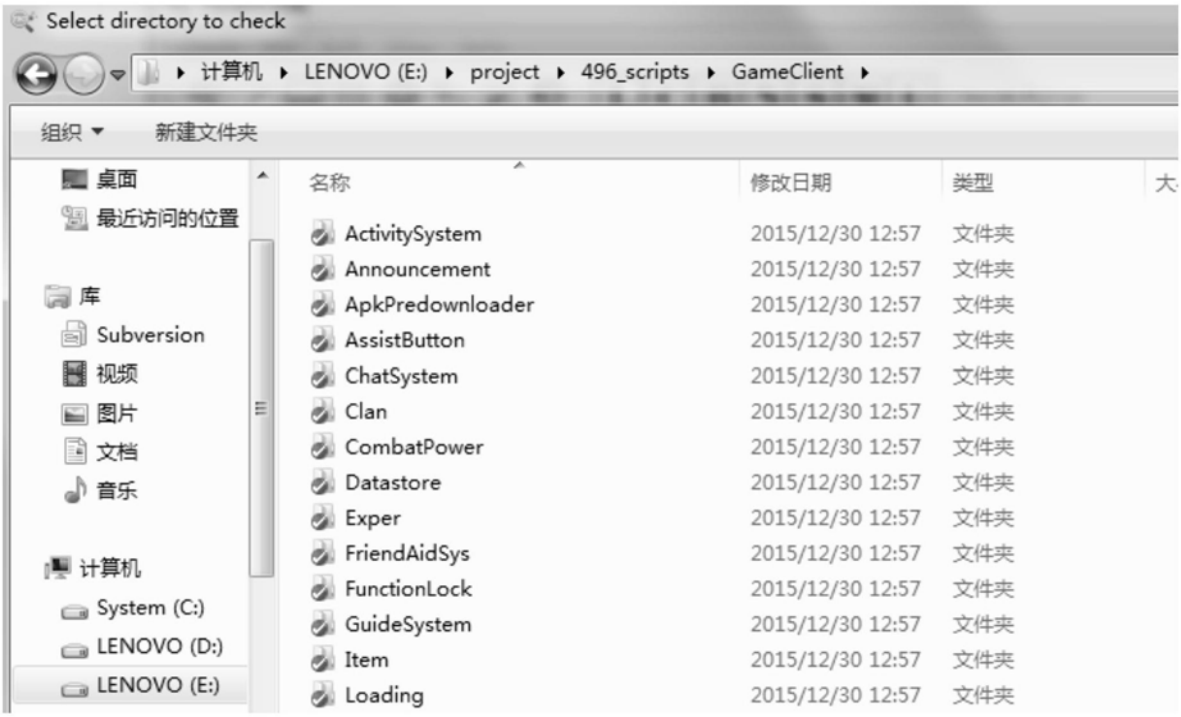

Select directory to check
计算机 ▸ LENOVO (E:) ▸ project ▸ 496_scripts ▸ GameClient ▸
组织
新建文件夹
桌面
最近访问的位置
库
Subversion
视频
图片
文档
音乐
计算机
System (C:)
LENOVO (D:)
LENOVO (E:)
名称
修改日期
类型
ActivitySystem 2015/12/30 12:57 文件夹
Announcement 2015/12/30 12:57 文件夹
ApkPredownloader 2015/12/30 12:57 文件夹
AssistButton 2015/12/30 12:57 文件夹
ChatSystem 2015/12/30 12:57 文件夹
Clan 2015/12/30 12:57 文件夹
CombatPower 2015/12/30 12:57 文件夹
Datastore 2015/12/30 12:57 文件夹
Exper 2015/12/30 12:57 文件夹
FriendAidSys 2015/12/30 12:57 文件夹
FunctionLock 2015/12/30 12:57 文件夹
GuideSystem 2015/12/30 12:57 文件夹
Item 2015/12/30 12:57 文件夹
Loading 2015/12/30 12:57 文件夹

图7

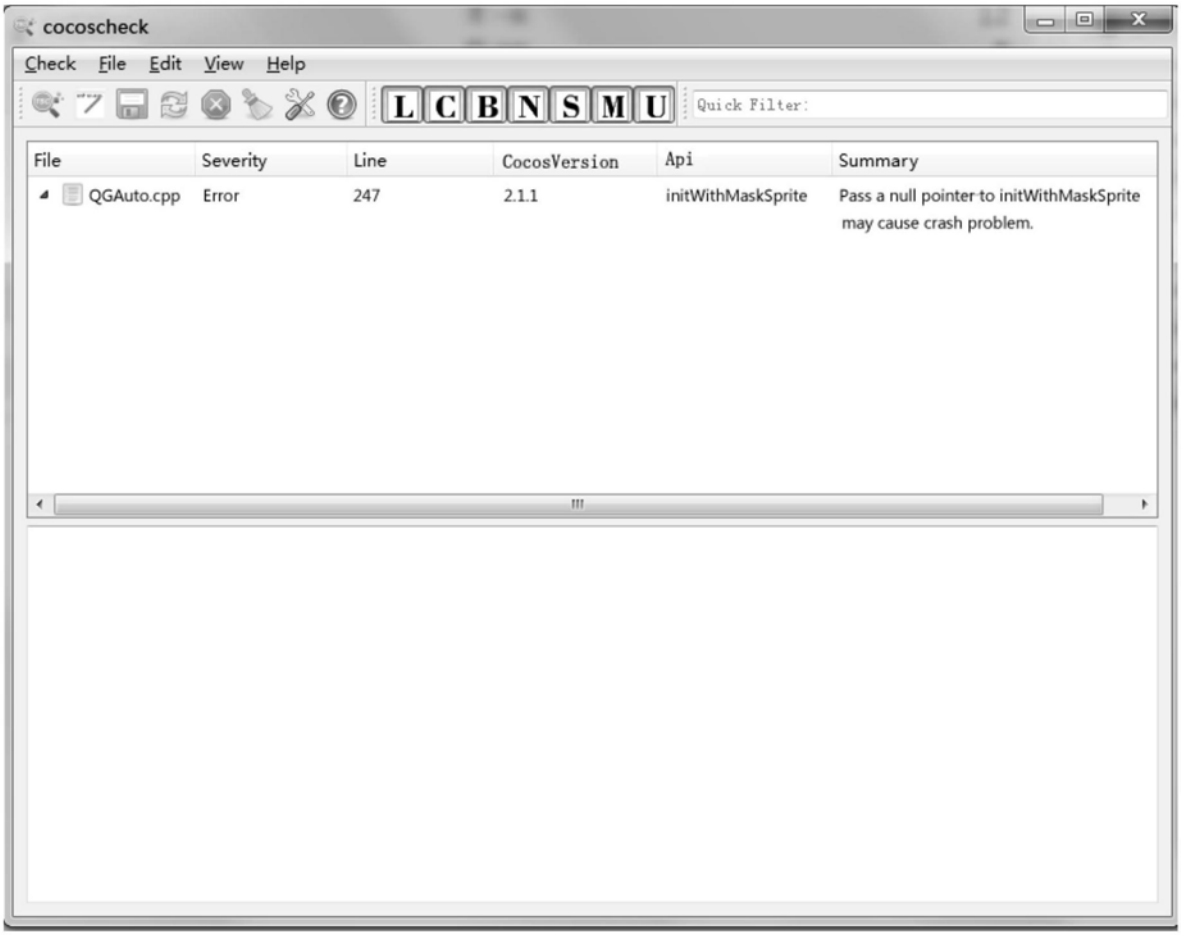

cocoscheck
Check File Edit View Help
L C B N S M U
Quick Filter:
File Severity Line CocosVersion Api Summary
QGAuto.cpp Error 247 2.1.1 initWithMaskSprite Pass a null pointer to initWithMaskSprite may cause crash problem.

图8

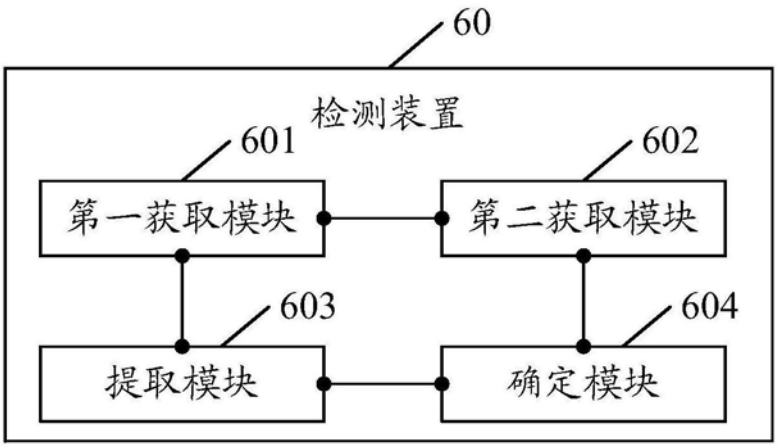

60
检测装置
601
第一获取模块
602
第二获取模块
603
提取模块
604
确定模块

图9

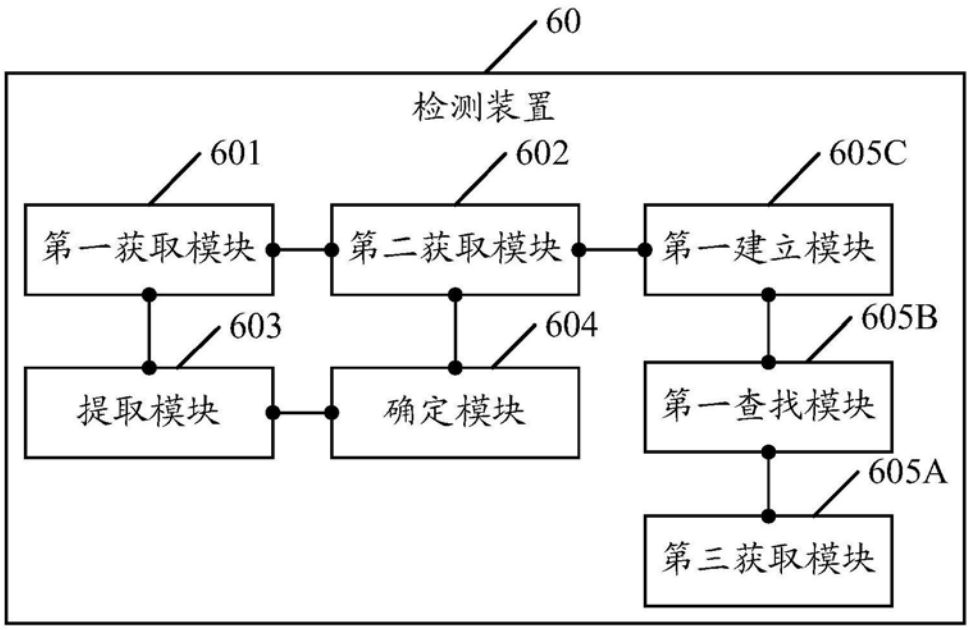

60
检测装置
601
602
605C
第一获取模块
第二获取模块
第一建立模块
603
604
605B
提取模块
确定模块
第一查找模块
605A
第三获取模块

图10

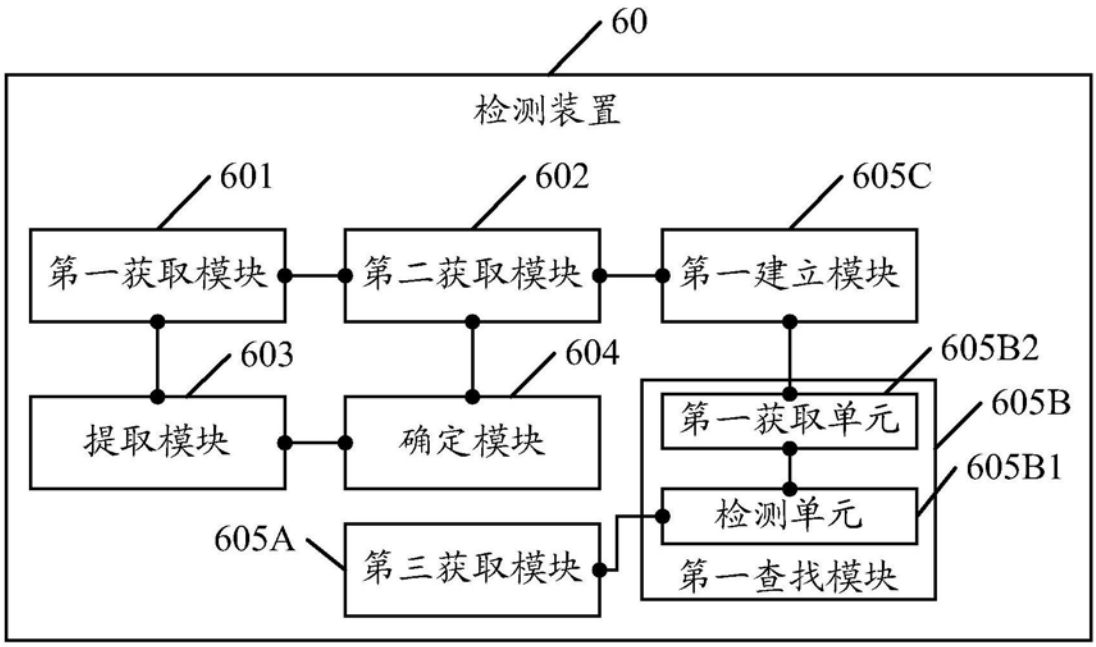

60
检测装置
601
602
605C
第一获取模块
第二获取模块
第一建立模块
603
604
605B2
605B
605B1
提取模块
确定模块
第一获取单元
检测单元
605A
第三获取模块
第一查找模块

图11

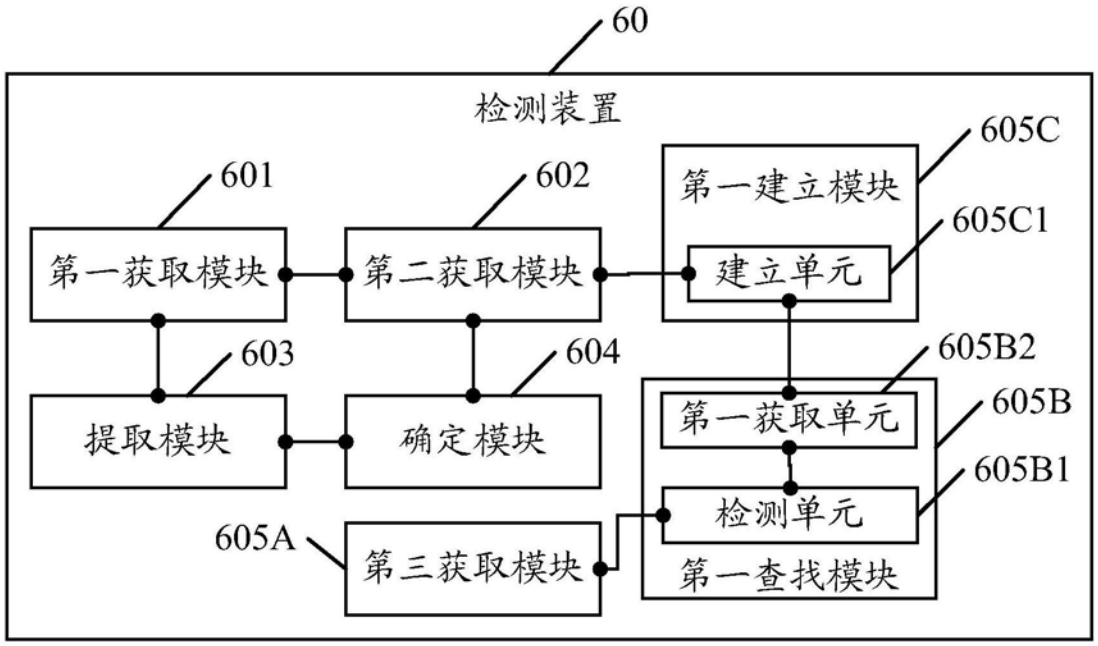

60
检测装置
601
602
605C
第一建立模块
605C1
第一获取模块
第二获取模块
建立单元
603
604
605B2
605B
提取模块
确定模块
第一获取单元
605B1
检测单元
605A
第三获取模块
第一查找模块

图12

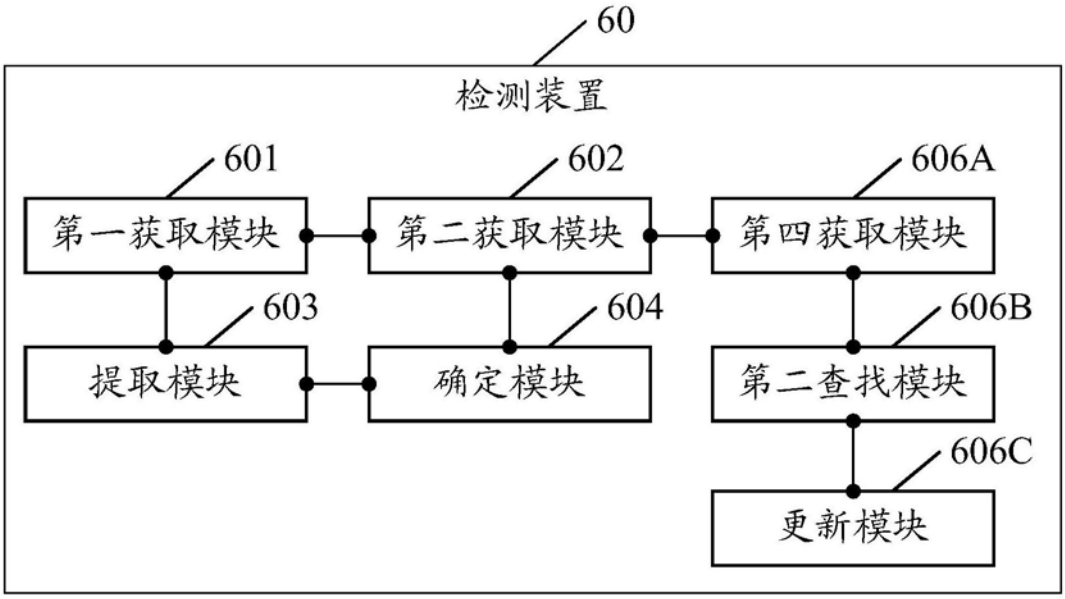

60
检测装置
601
602
606A
第一获取模块
第二获取模块
第四获取模块
603
604
606B
提取模块
确定模块
第二查找模块
606C
更新模块

图13

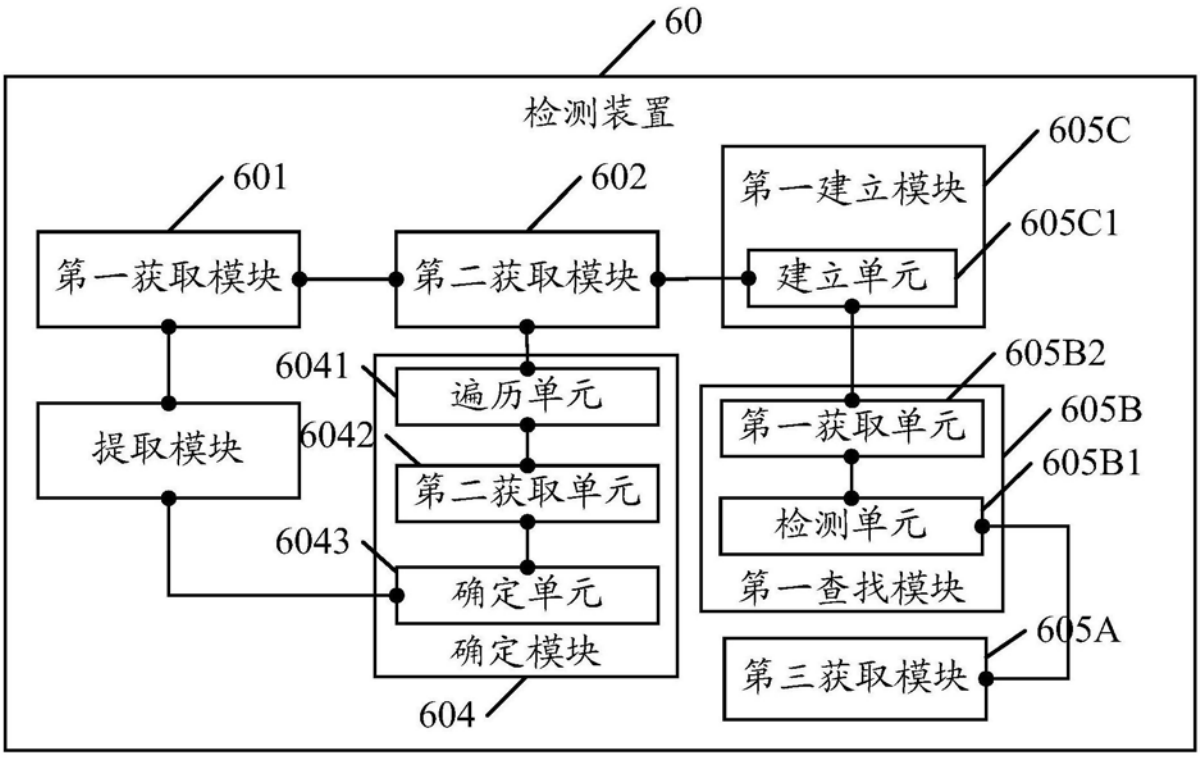

60
检测装置
601
602
605C
605C1
第一获取模块
第二获取模块
第一建立模块
建立单元
6041
遍历单元
605B2
605B
605B1
提取模块
6042
第二获取单元
第一获取单元
检测单元
6043
确定单元
第一查找模块
605A
确定模块
第三获取模块
604

图14

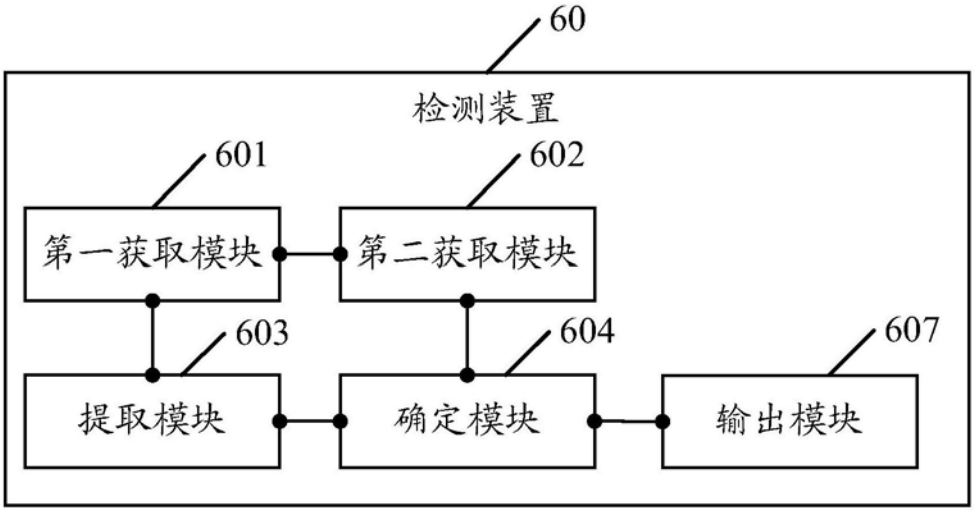

60
检测装置
601
602
第一获取模块
第二获取模块
603
604
607
提取模块
确定模块
输出模块

图15

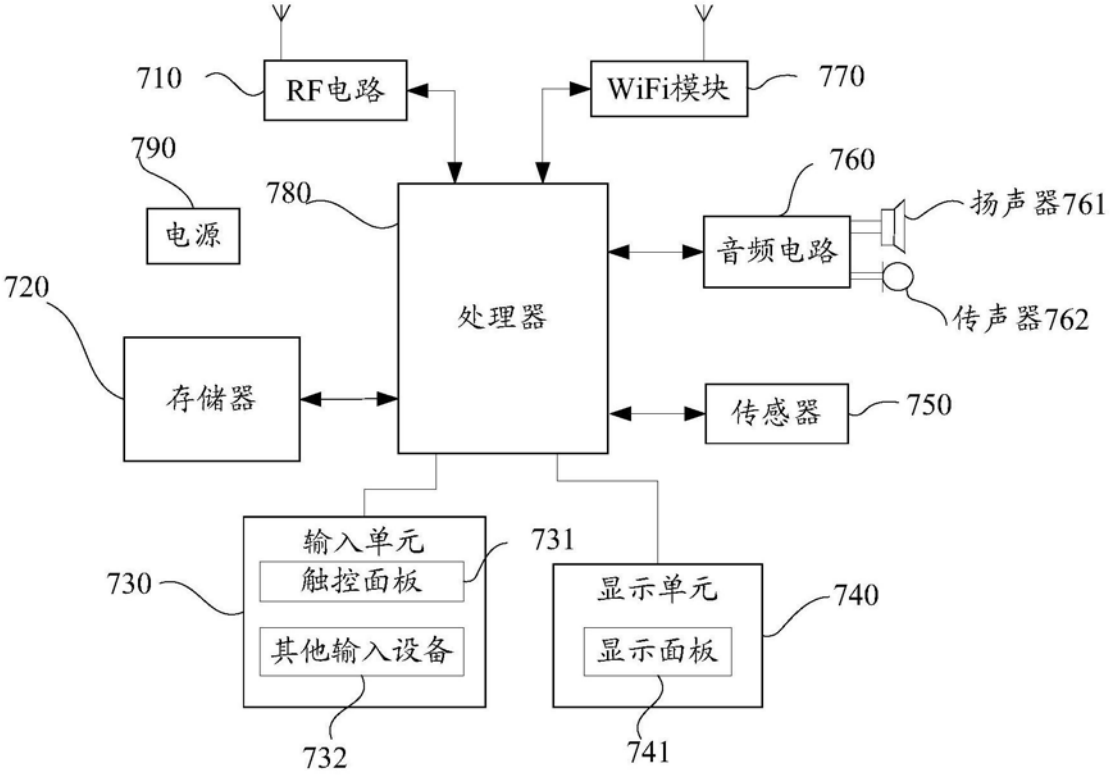

710
RF电路
WiFi模块
770
790
电源
780
处理器
760
音频电路
扬声器761
传声器762
720
存储器
传感器
750
输入单元
731
730
触控面板
其他输入设备
732
显示单元
740
显示面板
741

图16