(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 115904385 A
(43)申请公布日 2023.04.04

(21)申请号 202211400031.7

(22)申请日 2022.11.09

(71)申请人 支付宝(杭州)信息技术有限公司
地址 310000 浙江省杭州市西湖区西溪路556号8层B段801-11

(72)发明人 江冠儒

(74)专利代理机构 北京亿腾知识产权代理事务所(普通合伙) 11309
专利代理师 陈霁 周良玉

(51)Int.Cl.
G06F 8/41(2018.01)
G06F 11/36(2006.01)
G06F 21/62(2013.01)

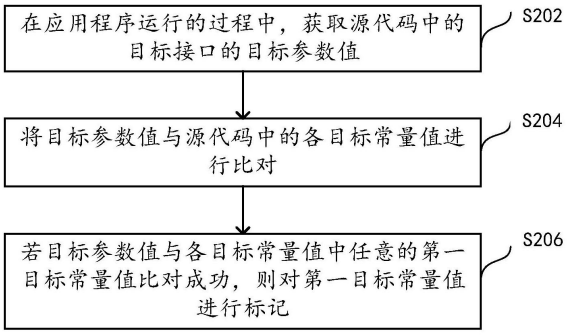
权利要求书2页 说明书7页 附图2页

(54)发明名称

代码检测方法及装置

(57)摘要

本说明书实施例提供一种代码检测方法及装置，在代码检测方法中，在应用程序运行的过程中，获取源代码中的目标接口的目标参数值。该目标接口为敏感数据的操作接口。将目标参数值与源代码中的各目标常量值进行比对。若目标参数值与各目标常量值中任意的第一目标常量值比对成功，则对第一目标常量值进行标记。



CN 115904385 A

1.一种代码检测方法,用于检测应用程序的源代码的安全性;所述方法包括:

在所述应用程序运行的过程中,获取所述源代码中的目标接口的目标参数值;所述目标接口为敏感数据的操作接口;

将所述目标参数值与所述源代码中的各目标常量值进行比对;

若所述目标参数值与所述各目标常量值中任意的第一目标常量值比对成功,则对所述第一目标常量值进行标记。

2.根据权利要求1所述的方法,其中,所述目标参数值包括,输入参数值和输出参数值中的至少一项。

3.根据权利要求1所述的方法,其中,所述源代码基于目标语言编写;所述各目标常量值存储于所述目标语言对应的常量池中;所述常量池用于存储所述源代码中被实际使用的各常量值。

4.根据权利要求1所述的方法,其中,所述各目标常量值中的至少部分目标常量值为编码后的常量值;所述方法还包括:

若所述目标参数值与所述各目标常量值均比对不成功,则按照预设的编码方式,对所述目标参数值进行编码;

将编码后的目标参数值与所述各目标常量值进行比对,若与任意的第二目标常量值比对成功,则对所述第二目标常量值进行标记。

5.根据权利要求1所述的方法,其中,所述方法通过静态或者运行时注入到所述应用程序的字节码中的自定义代码执行;所述字节码通过对所述源代码编译得到。

6.根据权利要求5所述方法,其中,所述自定义代码的注入位置至少基于类名、接口名和参数名确定。

7.根据权利要求1所述的方法,其中,所述敏感数据包括以下中的至少一项:用户口令、密钥、认证凭据、证书私钥和个人隐私数据。

8.一种代码检测装置,用于检测应用程序的源代码的安全性;所述装置包括:

获取单元,用于在所述应用程序运行的过程中,获取所述源代码中的目标接口的目标参数值;所述目标接口为敏感数据的操作接口;

比对单元,用于将所述目标参数值与所述源代码中的各目标常量值进行比对;

标记单元,用于若所述目标参数值与所述各目标常量值中任意的第一目标常量值比对成功,则对所述第一目标常量值进行标记。

9.根据权利要求8所述的装置,其中,所述目标参数值包括,输入参数值和输出参数值中的至少一项。

10.根据权利要求8所述的装置,其中,所述源代码基于目标语言编写;所述各目标常量值存储于所述目标语言对应的常量池中;所述常量池用于存储所述源代码中被实际使用的各常量值。

11.根据权利要求8所述的装置,其中,所述各目标常量值中的至少部分目标常量值为编码后的常量值;所述装置还包括:

编码单元,用于若所述目标参数值与所述各目标常量值均比对不成功,则按照预设的编码方式,对所述目标参数值进行编码;

所述比对单元,还用于将编码后的目标参数值与所述各目标常量值进行比对;

所述标记单元，还用于若编码后的目标参数值与任意的第二目标常量值比对成功，则对所述第二目标常量值进行标记。

12.根据权利要求8所述的装置，其中，所述装置中设置有通过静态或者运行时注入到所述应用程序的字节码中的自定义代码；所述字节码通过对所述源代码编译得到。

13.根据权利要求12所述装置，其中，所述自定义代码的注入位置至少基于类名、接口名和参数名确定。

14.根据权利要求8所述的装置，其中，所述敏感数据包括以下中的至少一项：用户口令、密钥、认证凭据、证书私钥和个人隐私数据。

15.一种计算机可读存储介质，其上存储有计算机程序，其中，当所述计算机程序在计算机中执行时，令计算机执行权利要求1-7中任一项所述的方法。

16.一种计算设备，包括存储器和处理器，其中，所述存储器中存储有可执行代码，所述处理器执行所述可执行代码时，实现权利要求1-7中任一项所述的方法。

代码检测方法及装置

技术领域

[0001] 本说明书一个或多个实施例涉及安全技术领域，尤其涉及一种代码检测方法及装置。

背景技术

[0002] 软件源代码(简称源代码)在一个软件系统(也称应用程序)中扮演了最基础的地位，所有程序的运行都是通过软件源代码编译、打包、分发、部署安装之后才能实现。因此如果软件源代码中包含了敏感数据(也称隐私数据)，攻击者就有机会通过逆向分析编译后的软件安装包得到这些敏感数据，部分敏感数据甚至可以作为凭据或者口令访问其他系统，这样的话整个系统的安全性就难以保证。

发明内容

[0003] 本说明书一个或多个实施例描述了一种代码检测方法及装置，可以准确检测出软件源代码中包含的会对软件系统造成危险的敏感数据。

[0004] 根据第一方面，提供了一种代码检测方法，包括：

[0005] 在所述应用程序运行的过程中，获取所述源代码中的目标接口的目标参数值；所述目标接口为敏感数据的操作接口；

[0006] 将所述目标参数值与所述源代码中的各目标常量值进行比对；

[0007] 若所述目标参数值与所述各目标常量值中任意的第一目标常量值比对成功，则对所述第一目标常量值进行标记。

[0008] 在一种可能的实现方式中，所述目标参数值包括，输入参数值和输出参数值中的至少一项。

[0009] 在一种可能的实现方式中，所述源代码基于目标语言编写；所述各目标常量值存储于所述目标语言对应的常量池中；所述常量池用于存储所述源代码中被实际使用的各常量值。

[0010] 在一种可能的实现方式中，所述各目标常量值中的至少部分目标常量值为编码后的常量值；所述方法还包括：

[0011] 若所述目标参数值与所述各目标常量值均比对不成功，则按照预设的编码方式，对所述目标参数值进行编码；

[0012] 将编码后的目标参数值与所述各目标常量值进行比对，若与任意的第二目标常量值比对成功，则对所述第二目标常量值进行标记。

[0013] 在一种可能的实现方式中，所述方法通过静态或者运行时注入到所述应用程序的字节码中的自定义代码执行；所述字节码通过对所述源代码编译得到。

[0014] 在一种可能的实现方式中，所述自定义代码的注入位置至少基于类名、接口名和参数名确定。

[0015] 在一种可能的实现方式中，所述敏感数据包括以下中的至少一项：用户口令、密

钥、认证凭据、证书私钥和个人隐私数据。

[0016] 根据第二方面，提供了一种代码检测装置，包括：

[0017] 获取单元，用于在所述应用程序运行的过程中，获取所述源代码中的目标接口的目标参数值；所述目标接口为敏感数据的操作接口；

[0018] 比对单元，用于将所述目标参数值与所述源代码中的各目标常量值进行比对；

[0019] 标记单元，用于若所述目标参数值与所述各目标常量值中任意的第一目标常量值比对成功，则对所述第一目标常量值进行标记。

[0020] 在一种可能的实现方式中，所述目标参数值包括，输入参数值和输出参数值中的至少一项。

[0021] 在一种可能的实现方式中，所述源代码基于目标语言编写；所述各目标常量值存储于所述目标语言对应的常量池中；所述常量池用于存储所述源代码中被实际使用的各常量值。

[0022] 在一种可能的实现方式中，所述各目标常量值中的至少部分目标常量值为编码后的常量值；所述装置还包括：

[0023] 编码单元，用于若所述目标参数值与所述各目标常量值均比对不成功，则按照预设的编码方式，对所述目标参数值进行编码；

[0024] 所述比对单元，还用于将编码后的目标参数值与所述各目标常量值进行比对；

[0025] 所述标记单元，还用于若编码后的目标参数值与任意的第二目标常量值比对成功，则对所述第二目标常量值进行标记。

[0026] 在一种可能的实现方式中，所述装置中设置有通过静态或者运行时注入到所述应用程序的字节码中的自定义代码；所述字节码通过对所述源代码编译得到。

[0027] 在一种可能的实现方式中，所述自定义代码的注入位置至少基于类名、接口名和参数名确定。

[0028] 在一种可能的实现方式中，所述敏感数据包括以下中的至少一项：用户口令、密钥、认证凭据、证书私钥和个人隐私数据。

[0029] 根据第三方面，提供了一种计算机可读存储介质，其上存储有计算机程序，当所述计算机程序在计算机中执行时，令计算机执行第一方面的方法。

[0030] 根据第四方面，提供了一种计算设备，包括存储器和处理器，所述存储器中存储有可执行代码，该处理器执行所述可执行代码时，实现第一方面的方法。

[0031] 本说明书一个或多个实施例提供的代码检测方法及装置，通过将敏感数据操作接口的目标参数值与源代码中的各目标常量值进行比对，来实现对源代码的安全检测，由此可以挖掘出深藏在软件源代码中的敏感数据。此外，由于上述目标参数值代表着真实流转的敏感数据，从而相比较于传统的静态检测方案，本方案可以减少误报率。

附图说明

[0032] 为了更清楚地说明本说明书实施例的技术方案，下面将对实施例描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本说明书的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其它的附图。

[0033] 图1示出在一个例子中自定义代码的执行方法示意图；

[0034] 图2示出根据一个实施例的代码检测方法流程图；

[0035] 图3示出根据另一个实施例的代码检测方法流程图；

[0036] 图4示出根据一个实施例的代码检测装置示意图。

具体实施方式

[0037] 下面结合附图，对本说明书提供的方案进行描述。

[0038] 如前所述，软件源代码在整个软件系统中发挥着重要作用，但是由于种种原因软件源代码中通常会包含一系列的敏感数据。比如，开发人员本身安全意识不够，在开发的时候直接写入。再比如，开发人员无意识地通过软件供应链的方式引入带有敏感数据的上游软件源代码。还比如，原本应该是用于测试的敏感数据没有及时删除从而遗留在软件源代码中。因此，需要对软件源代码进行安全检测。

[0039] 传统技术中，基于静态检测的方法进行安全检测，即借助一定的代码扫描工具对软件源代码进行文本扫描，根据关键字(比如，用于检测用户口令的“password”，用于检测认证凭据的“Token”等)、代码特征(比如，常见的用户口令“123456”)识别出可疑的敏感数据，可以初步地找到软件源代码中的敏感数据。

[0040] 但是上述静态检测的方法存在如下缺点：

[0041] 1.查找不准确。敏感数据可能藏于软件源代码的任何地方，仅靠关键字查找并不能够全面地找到软件源代码中的所有敏感数据。

[0042] 2.误报率高。静态检测的方法是对软件源代码的文本进行扫描和匹配，源代码的文本不能够体现软件真实运行空间，即使在软件源代码的文本中找到了可疑的敏感数据，但是在实际运行中这段代码并没有执行或者这些敏感数据并没有被使用，从而其不会造成安全问题，因此不应被统计。实验发现，上述静态检测的方法的误报率在70％以上。

[0043] 3.可靠性低。比如，如果开发人员对软件源代码中的敏感数据进行简单的编码，则上述静态检测的方法无法检测出该敏感数据，检出率大幅下降。

[0044] 因此，静态检测的方法虽然能够检测出软件源代码中的一部分敏感数据，但是上述的几个缺点会导致需要大量的人力物力和时间对扫描出的结果进行清洗、分析、整合，然后反复多次检测，进而影响检测效率。

[0045] 为此，本申请的发明人提出，通过动态捕获所有正在使用的敏感数据，并将捕获的敏感数据与源代码中的常量值进行比对，以检测出隐藏在软件源代码中的敏感数据，由此可以大大提升代码检测的准确性和可靠性。

[0046] 以下对本方案进行详细说明。

[0047] 需要说明，本方案通过静态或运行时注入到应用程序的字节码中的自定义代码(也称埋点代码)执行，这里的字节码通过对应用程序的源代码编译得到。其中的静态注入是指在应用程序启动之前，将自定义代码注入到应用程序的字节码中。运行时注入是指在应用程序启动之后，动态注入到应用程序的字节码中。

[0048] 以下对自定义代码的执行方法进行说明。

[0049] 图1示出在一个例子中自定义代码的执行方法示意图。图1中，在注入自定义代码之前，函数main()直接调用接口1。而当在调用接口1之前注入自定义代码时，则先执行自

定义代码,之后再调用接口1。

[0050] 图2示出根据一个实施例的代码检测方法流程图。如图2所示,该方法可以包括如下步骤:

[0051] 步骤202,在应用程序运行的过程中,获取源代码中的目标接口的目标参数值。

[0052] 上述的目标接口为敏感数据的操作接口,其通常也称为敏感数据流转的出入口。这里的敏感数据包括以下中的至少一项:用户口令、密钥、认证凭据、证书私钥和个人隐私数据(比如,身份标识、住址、电话、银行账号、邮箱等等)等。

[0053] 在一个例子中,假设密码(password)为敏感数据,且源代码中存在如下的接口:void verify(String username,String password),该接口用于验证用户名和密码,那么接口verify即为目标接口。

[0054] 此外,上述目标参数值包括输入参数值和输出参数值中的至少一项。在前述例子中,目标参数值为输入参数值。

[0055] 如上所述,本方案的执行主体是自定义代码。该自定义代码的注入位置可以根据所需获取的目标接口和/或目标参数值的不同而不同。

[0056] 在一个示例中,上述自定义代码的注入位置基于类名、接口名和参数名确定。

[0057] 在前述例子中,假设接口verify所属的类为:Verifier,那么自定义代码的注入位置可以为:Verifier.verify(param1,param2),其含义是:将自定义代码注入到类Verifier的接口verify中。

[0058] 当然,在其他例子中,假设目标参数值为输出参数值,那么可以将自定义代码注入到目标接口调用完成之后。

[0059] 总之,本方案可以通过注入到字节码中的自定义代码,来获取源代码中的目标接口的目标参数值。或者说,通过埋点的方式来获取目标接口的目标参数值,也即这里的目标接口为埋点接口。

[0060] 步骤204,将目标参数值与源代码中的各目标常量值进行比对。

[0061] 在上述源代码基于Java语言编写时,上述各目标常量值为存储于Java语言对应的常量池中的各常量值。存储于常量池中的各目标常量值属于源代码的一部分数据,其在被实际使用时自动加载到常量池中。也就是说,Java语言对应的常量池用于存储源代码中被实际使用的各常量值。这里的实际使用是指被接口、函数或方法使用。

[0062] 在源代码基于除Java语言外的其他语言编写时,上述各目标常量值可以为源代码中所包含的所有常量值。比如,可以通过代码注入的能力(即向应用程序的字节码中注入不同于上述自定义代码的其它自定义代码),显式地将源代码的所有常量值加载到自定义常量池中。之后,将目标参数值与自定义常量池中的各常量值进行比对。

[0063] 步骤206,若目标参数值与各目标常量值中任意的第一目标常量值比对成功,则对第一目标常量值进行标记。

[0064] 上述对第一目标常量值进行标记,也可以理解为是对源代码中第一目标常量值进行标记,以指示该第一目标常量值的使用是违规的。

[0065] 对于违规使用的目标常量值,还可以将其输出到敏感数据池中,以进行后置处理。

[0066] 此外,在实际应用中,加载到Java语言对应的常量池中的部分常量值,是经过编码后的常量值,因此,在目标参数值与各目标常量值均比对不成功的情况下,还可以执行如下

步骤：

[0067] 按照预设的编码方式，对目标参数值进行编码。将编码后的目标参数值与各目标常量值进行比对，若与任意的第二目标常量值比对成功，则对第二目标常量值进行标记。

[0068] 首先，上述预设的编码方式可以包括以下中的任一种：Base64编码、Hex编码以及Base58编码等。

[0069] 此外，上述对第二目标常量值进行标记，也可以理解为是对源代码中编码前的第二目标常量值进行标记，以指示编码前的第二目标常量值的使用是违规的。

[0070] 以下结合例子对上述静态检测的方法和本方案进行对比说明。

[0071] 假设应用程序的源代码如下：

```
public class Verifier {
    private static final String username1 = "小明";
    private static final String password1 = "小明的密码";
    private static final String username2 = "小红";
    private static final String password2 = "小红的密码";
    public void verify(String username, String password) {
```

[0072]

```
        ……
    }
    public static void main(String[] args) {
        new Verifier().verify(username1, password1);
        //new Verifier().verify(username2,password2);
    }
}
```

[0073] 其中，上述代码中的“//”为注释符号，即在程序运行过程中，该段代码不会被执行。

[0074] 需要说明，对于上述的源代码，如果利用静态检测的方法，那么会将其中的“小明的密码”和“小红的密码”均检测为敏感数据(比如，通过关键字“password”识别)。而事实上，在程序运行的过程中，“小红的密码”不会被使用，因为调用该敏感数据的代码被注释掉了。实验证明，这些未被使用的敏感数据通常是一些过期数据，因此即便这些敏感数据被窃取，也不会对软件系统造成危险。然而静态检测的方法则会将过期的数据也识别为敏感数据，由此会导致误报的情况。

[0075] 而本方案通过动态捕获所有正在使用的敏感数据(也称实际流转的敏感数据)，可以避免将过期的数据也识别为敏感数据。比如，在上述例子中，由于“小明的密码”被实际使用了，从而本方案只会捕获到“小明的密码”，并将其与各目标常量值进行比对，而不会捕获“小红的密码”，由此可以提升检测准确率。

[0076] 综上，本说明书实施例提供的代码检测方法，通过代码注入的方式，可以动态精准

地捕获所有正在使用的敏感数据，由此解决了敏感数据难以获取的问题。此外，对于这些真实流转的敏感数据，再借助常量池(包括自定义常量池)，可以挖掘出深藏在软件源代码中的敏感数据。

[0077] 相较于静态检测的方法，本方案所有检测出的敏感数据都是真实流转的敏感数据，能够大大减少误报率，并且本方案通过注入代码的方式，不会遗漏任何关键出入口的敏感数据，而且如果软件源代码中的常量值是经过编码的场景，可以先对出入口获取的敏感数据进行同样的编码再进行比对，这样也能够克服遗漏率高的问题。

[0078] 图3示出根据另一个实施例的代码检测方法流程图。如图3所示，该方法可以包括如下步骤：

[0079] 步骤302，在应用程序运行的过程中，获取源代码中的埋点接口的目标参数值。

[0080] 这里的埋点接口可以理解为注入的自定义代码所需获取输入/输出参数值的接口，其为敏感数据流转的接口。

[0081] 步骤304，判断目标参数值是否存在于常量池中。

[0082] 这里的常量池包括程序内置的常量池(比如，Java语言)，以及自定义的常量池。

[0083] 步骤306，如果存在于常量池，则对源代码中的对比一致的常量值进行标记，以进行后续的违规处理。

[0084] 步骤308，如果不存在于常量池，对目标参数值进行编码。

[0085] 步骤310，判断编码后的目标参数值是否存在于常量池中。

[0086] 步骤312，如果是，则对源代码中的对应常量值进行标记。

[0087] 步骤314，如果否，则将目标参数值判别为正常敏感数据，不作任何特殊处理。

[0088] 与上述代码检测方法对应地，本说明书一个实施例还提供的一种代码检测装置，用于检测应用程序的源代码的安全性。该装置中设置有通过静态或者运行时注入到应用程序的字节码中的自定义代码，该字节码通过对应用程序的源代码编译得到。

[0089] 在一些实施例中，上述自定义代码的注入位置至少基于类名、接口名和参数名确定。

[0090] 如图4所示，该装置可以包括：

[0091] 获取单元402，用于在应用程序运行的过程中，获取源代码中的目标接口的目标参数值，该目标接口为敏感数据的操作接口。

[0092] 其中，上述目标参数值包括，输入参数值和输出参数值中的至少一项。

[0093] 其中，上述敏感数据包括以下中的至少一项：用户口令、密钥、认证凭据、证书私钥和个人隐私数据。

[0094] 比对单元404，用于将目标参数值与源代码中的各目标常量值进行比对。

[0095] 其中，上述源代码基于目标语言编写，上述各目标常量值存储于目标语言对应的常量池中，该常量池用于存储源代码中被实际使用的各常量值。

[0096] 标记单元406，用于若目标参数值与各目标常量值中任意的第一目标常量值比对成功，则对第一目标常量值进行标记。

[0097] 在一些实施例中，上述各目标常量值中的至少部分目标常量值为编码后的常量值，该装置还包括：

[0098] 编码单元408，用于若目标参数值与各目标常量值均比对不成功，则按照预设的编

码方式,对目标参数值进行编码;

[0099] 比对单元404,还用于将编码后的目标参数值与各目标常量值进行比对;

[0100] 标记单元406,还用于若编码后的目标参数值与任意的第二目标常量值比对成功,则对第二目标常量值进行标记。

[0101] 本说明书上述实施例装置的各功能模块的功能,可以通过上述方法实施例的各步骤来实现,因此,本说明书一个实施例提供的装置的具体工作过程,在此不复赘述。

[0102] 本说明书一个实施例提供的代码检测装置,可以准确检测出软件源代码中包含的会对软件系统造成危险的敏感数据。根据另一方面的实施例,还提供一种计算机可读存储介质,其上存储有计算机程序,当所述计算机程序在计算机中执行时,令计算机执行结合图2或图3所描述的方法。

[0103] 根据再一方面的实施例,还提供一种计算设备,包括存储器和处理器,所述存储器中存储有可执行代码,所述处理器执行所述可执行代码时,实现结合图2或图3所描述的方法。

[0104] 本说明书中的各个实施例均采用递进的方式描述,各个实施例之间相同相似的部分互相参见即可,每个实施例重点说明的都是与其他实施例的不同之处。尤其,对于设备实施例而言,由于其基本相似于方法实施例,所以描述的比较简单,相关之处参见方法实施例的部分说明即可。

[0105] 结合本说明书公开内容所描述的方法或者算法的步骤可以硬件的方式来实现,也可以是由处理器执行软件指令的方式来实现。软件指令可以由相应的软件模块组成,软件模块可以被存放于RAM存储器、闪存、ROM存储器、EPROM存储器、EEPROM存储器、寄存器、硬盘、移动硬盘、CD-ROM或者本领域熟知的任何其它形式的存储介质中。一种示例性的存储介质耦合至处理器,从而使处理器能够从该存储介质读取信息,且可向该存储介质写入信息。当然,存储介质也可以是处理器的组成部分。处理器和存储介质可以位于ASIC中。另外,该ASIC可以位于服务器中。当然,处理器和存储介质也可以作为分立组件存在于服务器中。

[0106] 本领域技术人员应该可以意识到,在上述一个或多个示例中,本发明所描述的功能可以用硬件、软件、固件或它们的任意组合来实现。当使用软件实现时,可以将这些功能存储在计算机可读介质中或者作为计算机可读介质上的一个或多个指令或代码进行传输。计算机可读介质包括计算机存储介质和通信介质,其中通信介质包括便于从一个地方向另一个地方传送计算机程序的任何介质。存储介质可以是通用或专用计算机能够存取的任何可用介质。

[0107] 上述对本说明书特定实施例进行了描述。其它实施例在所附权利要求书的范围内。在一些情况下,在权利要求书中记载的动作或步骤可以按照不同于实施例中的顺序来执行并且仍然可以实现期望的结果。另外,在附图中描绘的过程不一定要求示出的特定顺序或者连续顺序才能实现期望的结果。在某些实施方式中,多任务处理和并行处理也是可以的或者可能是有利的。

[0108] 以上所述的具体实施方式,对本说明书的目的、技术方案和有益效果进行了进一步详细说明,所应理解的是,以上所述仅为本说明书的具体实施方式而已,并不用于限定本说明书的保护范围,凡在本说明书的技术方案的基础之上,所做的任何修改、等同替换、改进等,均应包括在本说明书的保护范围之内。

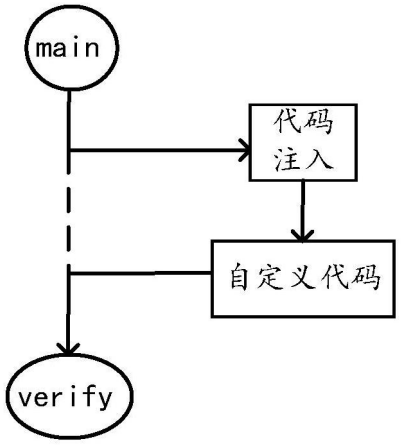

main
代码
注入
自定义代码
verify

图1

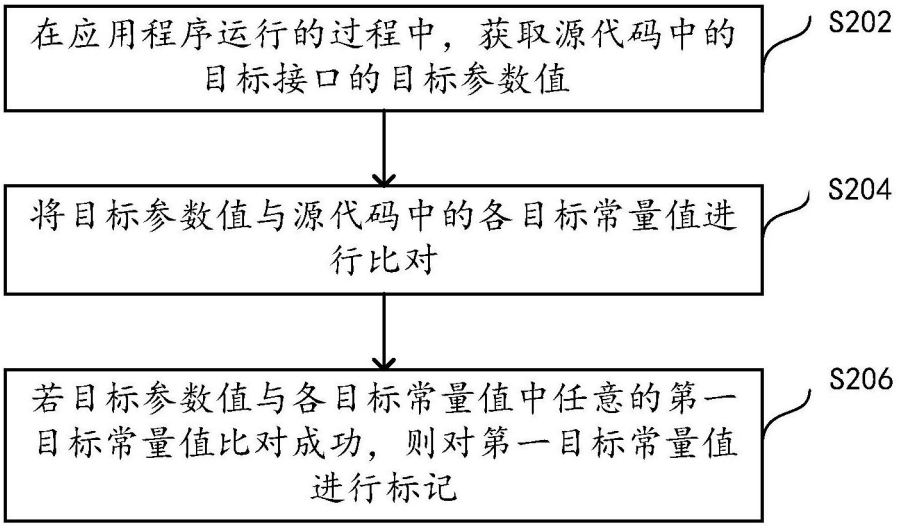

在应用程序运行的过程中，获取源代码中的目标接口的目标参数值
S202
将目标参数值与源代码中的各目标常量值进行比对
S204
若目标参数值与各目标常量值中任意的第一目标常量值比对成功，则对第一目标常量值进行标记
S206

图2

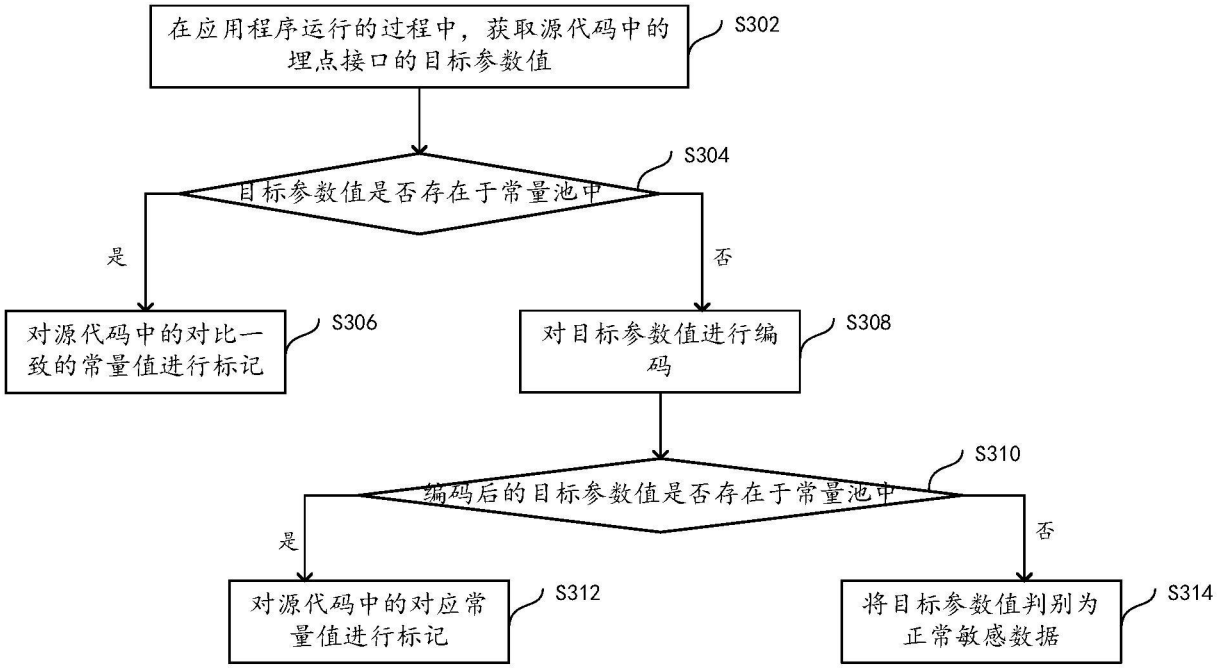

在应用程序运行的过程中，获取源代码中的埋点接口的目标参数值
S302
目标参数值是否存在于常量池中
S304
是
否
对源代码中的对比一致的常量值进行标记
S306
对目标参数值进行编码
S308
编码后的目标参数值是否存在于常量池中
S310
是
否
对源代码中的对应常量值进行标记
S312
将目标参数值判别为正常敏感数据
S314

图3

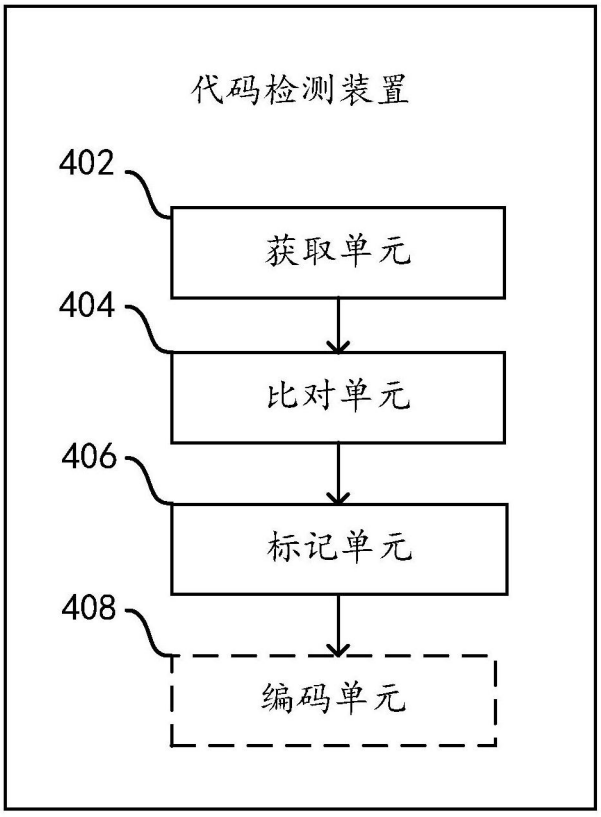

代码检测装置
402
获取单元
404
比对单元
406
标记单元
408
编码单元

图4