(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 105243019 A
(43)申请公布日 2016.01.13

(21)申请号 201510708735.4

(22)申请日 2015.10.27

(71)申请人 北京神州绿盟信息安全科技股份有限公司
地址 100089 北京市海淀区北洼路4号益泰大厦三层
申请人 北京神州绿盟科技有限公司

(72)发明人 廖新喜

(74)专利代理机构 北京同达信恒知识产权代理有限公司 11291
代理人 朱佳

(51)Int.Cl.
G06F 11/36(2006.01)

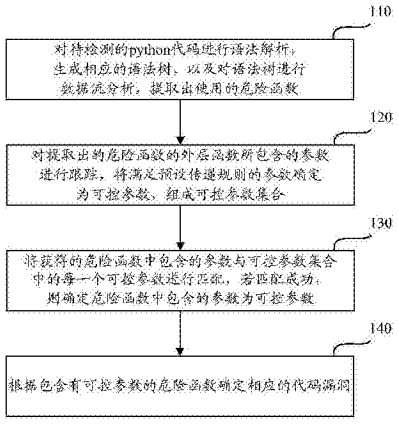
权利要求书2页 说明书8页 附图3页

(54)发明名称

一种检测python代码漏洞的方法及装置

(57)摘要

本发明涉及计算机技术，公开了一种检测python代码漏洞的方法及装置，用于提高python代码漏洞的检测精准度。该方法为：基于待检测python代码生成相应的语法树，提取出危险函数；再对危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合，以及确定危险函数中包含的参数归属于上述可控参数组合时，确定危险函数中包含有可控参数；最后，再根据包含有可控参数的危险函数确定相应的代码漏洞。这样，大大提高了python代码漏洞检测的准确性，解决了python代码语法灵活维护困难的问题，由于是自动化的跟踪参数，降低了操作复杂度。



CN 105243019 A

1.一种检测python代码漏洞的方法，其特征在于，包括：

对待检测的python代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数；

对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合；

将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数；

根据包含有可控参数的危险函数确定相应的代码漏洞。

2.如权利要求1所述的方法，其特征在于，进一步包括：

在对python代码进行语法解析的过程中，根据指示删除非必要代码。

3.如权利要求1所述的方法，其特征在于，进一步包括，

在进行数据流跟踪的过程中，基于输入import机制，从数据流以及python代码的环境变量中获取引入的第三方模块或/和第三方函数，并对所述第三方模块或/和第三方函数进行解析；

根据解析结果判断所述第三方模块或/和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

4.如权利要求1、2或3任一项所述的方法，其特征在于，将赋值类型满足预设传递规则的参数确定为可控参数，具体包括：

将赋值类型满足以下规则中的一种或任意组合的参数确定为可控参数：

参数的赋值类型为：指定的属性取值；

参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数；

参数的赋值类型为：分片符取值；

参数的赋值类型为：列表解析式，且所述列表解析式基于至少一个可控因子进行迭代，或者，且所述列表解析式为包含有可控因子的列表相加；

参数为赋值类型为：经字符串操作函数处理的返回值，或/和，经未过滤函数处理的返回值。

5.如权利要求4所述的方法，其特征在于，进一步包括：

对所述语法树进行类分析，在分析过程中将各个类的初始化参数归属于所述可控参数集合，以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定所述任一危险函数包含有可控参数。

6.如权利要求4所述的方法，其特征在于，确定任一危险函数中包含的参数为可控参数后，在根据所述任一危险函数确定相应的代码漏洞之前，进一步包括：

采用用户预设的安全函数或者python自带的安全函数对所述任一危险函数包含的可控参数进行处理，确定经处理后的可控参数仍可控时，最终判定所述任一危险函数为真正的危险函数。

7.一种检测python代码漏洞的装置，其特征在于，包括：

提取单元，用于对待检测的python代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数；

处理单元，用于对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型

满足预设传递规则的参数确定为可控参数，组成可控参数集合；

匹配单元，用于将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数；

确定单元，用于根据包含有可控参数的危险函数确定相应的代码漏洞。

8. 如权利要求 7 所述的装置，其特征在于，所述提取单元进一步用于：

在对 python 代码进行语法解析的过程中，根据指示删除非必要代码。

9. 如权利要求 7 所述的装置，其特征在于，所述处理单元进一步用于，

在进行数据流跟踪的过程中，基于输入 import 机制，从数据流以及 python 代码的环境变量中获取引入的第三方模块或 / 和第三方函数，并对所述第三方模块或 / 和第三方函数进行解析；

根据解析结果判断所述第三方模块或 / 和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

10. 如权利要求 7、8 或 9 任一项所述的装置，其特征在于，将赋值类型满足预设传递规则的参数确定为可控参数时，所述处理单元具体用于：

将赋值类型满足以下规则中的一种或任意组合的参数确定为可控参数：

参数的赋值类型为：指定的属性取值；

参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数；

参数的赋值类型为：分片符取值；

参数的赋值类型为：列表解析式，且所述列表解析式基于至少一个可控因子进行迭代，或者，且所述列表解析式为包含有可控因子的列表相加；

参数为赋值类型为：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

11. 如权利要求 10 所述的装置，其特征在于，所述处理单元进一步用于：

对所述语法树进行类分析，在分析过程中将各个类的初始化参数归属于所述可控参数集合，以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定所述任一危险函数包含有可控参数。

12. 如权利要求 10 所述的装置，其特征在于，确定任一危险函数中包含的参数为可控参数后，在根据所述任一危险函数确定相应的代码漏洞之前，所述处理单元进一步用于：

采用用户预设的安全函数或者 python 自带的安全函数对所述任一危险函数包含的可控参数进行处理，确定经处理后的可控参数仍可控时，最终判定所述任一危险函数为真正的危险函数。

一种检测 python 代码漏洞的方法及装置

技术领域

[0001] 本发明涉及网络安全领域，特别涉及一种检测 python 代码漏洞的方法及装置。

背景技术

[0002] 检测 python 代码漏洞是系统运维的必备手段，准确检测出 python 代码漏洞，可以有效防止非法攻击，从而保障系统的平稳运行。

[0003] 现有技术下，检测 python 代码漏洞都是通过正则匹配方式实现的，即使用字符串来描述、匹配一系列符合某个句法规则的字符串。当通过正则匹配方式匹配了危险函数之后则确定选取到危险代码，即检测到 python 代码漏洞。

[0004] 然而，并非每一种危险函数都对应着代码漏洞，部分危险函数包含的参数属于不可控参数，不存在危险性，而采用正则匹配方式仅仅是对代码做粗暴的分析，如果代码中存在“换行”，“注释”等规则难以描述的内容，则无法对其中的参数进行跟踪，无法确认当前参数是否可控。

[0005] 显然，通过正则匹配方式进行 python 代码漏洞检测，误报率非常高，在很大程度上也浪费了系统资源。

发明内容

[0006] 本发明实施例提供一种检测代码漏洞的方法及装置，用以解决现有技术下提高 python 代码漏洞检测的精准度。

[0007] 本发明实施例提供的具体技术方案如下：

[0008] 一种检测 python 代码漏洞的方法，包括：

[0009] 对待检测的 python 代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数；

[0010] 对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合；

[0011] 将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数；

[0012] 根据包含有可控参数的危险函数确定相应的代码漏洞。

[0013] 这样，基于语法树对 python 代码进行语法分析，避免了代码中的复杂内容对分析过程所造成的干扰（如，“换行”、“注释”所来来的干扰），能够在语法树中识别出参数，并且能根据预设传递规则判断参数在流动的过程中是否可控，如果在整个解析范围内可控，则认定为可控参数，若识别出的危险函数中存在可控参数，则认为相应的一段代码存在漏洞，这样，大大提高了 python 代码漏洞检测的准确性，解决了 python 代码语法灵活正则表达式维护困难的问题，由于是自动化的跟踪参数，免去了人工单步跟踪参数，降低了操作复杂度。

[0014] 较佳的，进一步包括：

[0015] 在对 python 代码进行语法解析的过程中，根据指示删除非必要代码。

[0016] 较佳的，进一步包括，

[0017] 在进行数据流跟踪的过程中，基于输入 import 机制，从数据流以及 python 代码的环境变量中获取引入的第三方模块或 / 和第三方函数，并对所述第三方模块或 / 和第三方函数进行解析 ；

[0018] 根据解析结果判断所述第三方模块或 / 和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

[0019] 较佳的，将赋值类型满足预设传递规则的参数确定为可控参数，具体包括 ：

[0020] 将赋值类型满足以下规则中的一种或任意组合的参数确定为可控参数 ：

[0021] 参数的赋值类型为 ：指定的属性取值 ；

[0022] 参数的赋值类型为 ：字符串拼接，且被拼接的字符串中包含已认定的可控参数 ；

[0023] 参数的赋值类型为 ：分片符取值 ；

[0024] 参数的赋值类型为 ：列表解析式，且所述列表解析式基于至少一个可控因子进行迭代，或者，且所述列表解析式为包含有可控因子的列表相加 ；

[0025] 参数为赋值类型为 ：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

[0026] 较佳的，进一步包括 ：

[0027] 对所述语法树进行类分析，在分析过程中将各个类的初始化参数归属于所述可控参数集合，以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定所述任一危险函数包含有可控参数。

[0028] 较佳的，确定任一危险函数中包含的参数为可控参数后，在根据所述任一危险函数确定相应的代码漏洞之前，进一步包括 ：

[0029] 采用用户预设的安全函数或者 python 自带的安全函数对所述任一危险函数包含的可控参数进行处理，确定经处理后的可控参数仍可控时，最终判定所述任一危险函数为真正的危险函数。

[0030] 一种检测 python 代码漏洞的装置，包括 ：

[0031] 提取单元，用于对待检测的 python 代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数 ；

[0032] 处理单元，用于对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合 ；

[0033] 匹配单元，用于将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数 ；

[0034] 确定单元，用于根据包含有可控参数的危险函数确定相应的代码漏洞。

[0035] 这样，基于语法树对 python 进行语法分析，避免了代码中的复杂内容对分析过程所造成的干扰（如，“换行”、“注释”所来来的干扰），能够在语法树中识别出参数，并且能根据预设传递规则判断参数在流动的过程中是否可控，如果在整个解析范围内可控，则认定为可控参数，若识别出的危险函数中存在可控参数，则认为相应的一段代码存在漏洞，这样，大大提高了 python 代码漏洞检测的准确性，解决了 python 代码语法灵活维护困难的问题，由于是自动化的跟踪参数，免去了人工单步跟踪参数，降低了操作复杂度。

[0036] 较佳的，所述提取单元进一步用于：

[0037] 在对 python 代码进行语法解析的过程中，根据指示删除非必要代码。

[0038] 较佳的，所述处理单元进一步用于，

[0039] 在进行数据流跟踪的过程中，基于输入 import 机制，从数据流以及 python 代码的环境变量中获取引入的第三方模块或 / 和第三方函数，并对所述第三方模块或 / 和第三方函数进行解析；

[0040] 根据解析结果判断所述第三方模块或 / 和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

[0041] 较佳的，将赋值类型满足预设传递规则的参数确定为可控参数时，所述处理单元具体用于：

[0042] 将赋值类型满足以下规则中的一种或任意组合的参数确定为可控参数：

[0043] 参数的赋值类型为：指定的属性取值；

[0044] 参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数；

[0045] 参数的赋值类型为：分片符取值；

[0046] 参数的赋值类型为：列表解析式，且所述列表解析式基于至少一个可控因子进行迭代，或者，且所述列表解析式为包含有可控因子的列表相加；

[0047] 参数为赋值类型为：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

[0048] 较佳的，所述处理单元进一步用于：

[0049] 对所述语法树进行类分析，在分析过程中将各个类的初始化参数归属于所述可控参数集合，以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定所述任一危险函数包含有可控参数。

[0050] 较佳的，确定任一危险函数中包含的参数为可控参数后，在根据所述任一危险函数确定相应的代码漏洞之前，所述处理单元进一步用于：

[0051] 采用用户预设的安全函数或者 python 自带的安全函数对所述任一危险函数包含的可控参数进行处理，确定经处理后的可控参数仍可控时，最终判定所述任一危险函数为真正的危险函数。

附图说明

[0052] 图 1 为本发明实施例中检测 python 代码漏洞流程图；

[0053] 图 2 为本发明实施例中生成可控参数集合流程图；

[0054] 图 3 为本发明实施例中检测 python 代码漏洞的装置功能结构示意图。

具体实施方式

[0055] 为了提高 python 代码漏洞检测的精准度，本发明实施例中，针对 python 代码进行分析，生成一个 python 语法树，然后对该语法树进行分析，跟踪数据流，当发现危险函数之后，再继续跟踪危险函数包含参数是否可控，若为可控参数，则认定检测到危险函数对应的漏洞。

[0056] 下面结合附图对本发明优选的实施方式进行详细说明。

[0057] 参阅图 1 所示，本发明实施例中，检测 python 代码漏洞的详细流程如下：

[0058] 步骤 110：对待检测的 python 代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数。

[0059] 实际应用中，用于解析 python 代码的工具很多，可以采用任意一种工具对 python 代码进行语法解析，生成相应的语法树。在进行语法分析时，包含但不限于对代码中包含的类，函数，import，控制语句等等进行分析。

[0060] 进一步地，在对 python 代码进行语法解析的过程中，需要根据指示删除非必要代码，从而去语法树中不必要的枝干，降低代码分析的难度，降低干扰。

[0061] 另一方面，存在漏洞的场景，很大一部分是由注入引起的，常见的注入场景有操作系统 (Operating System，OS) 命令注入，代码注入，结构化查询语言 (Structured Query Language，SQL) 注入，任意文件读取，下载等等。在这些注入场景中，所有的漏洞都涉及到使用危险函数，并且该危险函数中包含的参数是可控的。因此，在执行步骤 110 时，经语法解析，获得 python 的语法树后，得到其语法树，再跟踪语法树产生的数据流，从而提取出 python 代码中可能使用到的危险函数。

[0062] 筛选危险函数的过程较简单，可选的，可以通过学习累积维护一个危险函数列表，若语法树中发现了函数调用并且该函数名称在上述危险函数列表中，就可以标注出相应的代码行，表示查找到危险函数。

[0063] 进一步地，在进行数据流跟踪的过程中，基于输入 (import) 机制，从数据流以及 python 代码的环境变量中获取引入的第三方模块或 / 和第三方函数，并对该第三方模块或 / 和第三方函数进行解析，包括一些简单的函数处理，具体包含但不限于以下两种处理方式的任意一种或组合：

[0064] 1、判断引入的第三方模块或 / 和第三方函数是否为危险函数，确定为危险函数时，将该第三方模块或 / 和第三方函数包含的参数归属至可控参数集合中；

[0065] 2、根据解析结果判断引入的第三方模块或 / 和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

[0066] 例如，输入参数 A 后，返回值仍为参数 A，则认为参数 A 可控。

[0067] 又例如：将已知的可控参数赋值给参数 B，那么则认为参数 B 可控。

[0068] 之所以这样处理，是第三方可能会引入可控参数，那么，这些参数在后续的流程中经过各类危险函数处理后，仍然可以是可控参数，因此，需要加强防范。

[0069] 步骤 120：对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合。

[0070] 具体的，在提取出危险函数后，难点在于跟踪危险函数包含的参数。

[0071] 本发明实施例中，较佳的，默认危险函数的外层函数所包含的参数为可控参数，那么，只需要对外层函数包含的参数进行跟踪，分析其传递过程，即可确定最终的取值是否可控，从而可以根据分析结果获得相应的可控参数集合。

[0072] 可选的，本发明实施例中，只需要确定外层函数包含的参数的赋值类型符合以下传递规则中的任意一条，即可确定其为可控参数。

[0073] 1、参数的赋值类型为：指定的属性取值。

[0074] 例如，对一个变量取属性，比如：request 的 GET 属性、POST 属性和 FILES 属性。若

参数的赋值类型为上述属性取值，或者，进一步地，为上述属性的属性取值，则认为参数是可控参数。

[0075] 此外，若参数的赋值类型是 request 的其他字段，如 META、user、session、url 等等，则认为不是可控参数。

[0076] 2、参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数。

[0077] 由于被可控参数赋值后的值也被认定是可控的，因此，需要考虑到各种拼接情况，如，采用 +，%，format 函数等等方式进行拼接。

[0078] 3、参数的赋值类型为：分片符取值。

[0079] 通常情况下一般认为分片后的值也是可控的。

[0080] 4、参数的赋值类型为：列表解析式，且该列表解析式基于至少一个可控因子进行迭代，或者，且该列表解析式为包含有可控因子的列表相加。

[0081] 在这种情况下，默认赋值后的列表也是可控的。

[0082] 5、参数的赋值类型为：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

[0083] 例如，若参数经过简单的字符串操作函数（如，str，unicode，strip，encode 等）处理，或 / 和，参数经简单的未过滤函数处理，那么，则认定函数的返回参数是可控的。

[0084] 另一方面，可选的，还可以进一步对上述 python 语法树进行类分析，在分析过程中将各个类的初始化参数归属于可控参数集合（便于后续匹配流程使用），以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定该任一危险函数包含有可控参数。

[0085] 步骤 130：将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数。

[0086] 实际应用中，在确定任一危险函数中包含的参数为可控参数后，在根据上述任一危险函数确定相应的代码漏洞之前，进一步地，采用用户预设的安全函数或者 python 自带的安全函数对上述任一危险函数包含的可控参数进行处理，确定经处理后的参数仍为可控参数时，最终判定所述任一危险函数为真正的危险函数。

[0087] 这是因为，用户可以根据实际需求自行定义安全函数或者用户可以采用 python 自带的安全函数，如果任一危险函数（以下称为危险函数 X）包含的可控参数，经过安全函数处理之后由可控变成了不可控，那么，危险函数 X 则被排除，安全函数通常是程序员对输入进行安全检查的另外一种封装，相应的，若采用用户预设的安全函数对危险函数 X 包含的参数进行处理后，参数仍可控，则最终判定危险函数 X 为真正的危险函数，即需要检测出危险函数 X 对应的代码漏洞。

[0088] 当然，对危险函数进行分析的前提是危险函数中包含有参数，若有，才会进一步判断参数是否可控，否则，则直接排除相应的危险函数，但需要将该危险函数加入危险函数列表中，以便分析其他函数时使用。

[0089] 步骤 140：根据包含有可控参数的危险函数确定相应的代码漏洞。

[0090] 进一步地，在检测到所有的代码漏洞后，根据代码漏洞的类型，对应记录相应级别的系统日志，在不同的级别下输出的系统日志也是不一样的。

[0091] 如，代码漏洞可以是 sql 注入，命令注入等等，相应的，可以选择仅记录系统漏洞

类型的方式生成系统日志，也可以选择既记录系统漏洞类型又记录解决方案的方式生成系统日志。

[0092] 基于上述实施例中，参阅图 2 所示，在执行步骤 120 时，生成可控参数集合的具体过程如下：

[0093] 步骤 1200：获取 body 内容。

[0094] body 是语法树中的结构，可以理解为在对危险函数的外层函数进行排查。body 之中还可以嵌入 body，遍历每个 body 才能实现数据流的跟踪。

[0095] 步骤 1201：遍历 body 内容。

[0096] 步骤 1202：对 body 进行结构类分析，如，分析 orelse 结构、handles 结构、tests 结构等等。

[0097] 步骤 1203：判断是否获得赋值语句？若是，则执行步骤 1204；否则，执行 1207。

[0098] 在执行步骤 1203 时，可以理解为在外层函数排查的过程中，检测到了外层函数所包含的参数，那么需要进一步检测这些参数所对应的赋值语句，以便进一步判断这些参数是否可控。

[0099] 步骤 1204：判断获得的赋值语句是否为预设情况中的一种？若则，则执行步骤 1205；否则，执行 1207。

[0100] 执行步骤 1204，即是在判断外层函数包含的参数的赋值类型是否为上述五种情况中的一种，具体为：

[0101] 1、参数的赋值类型为：指定的属性取值。

[0102] 2、参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数。

[0103] 3、参数的赋值类型为：分片符取值。

[0104] 4、参数的赋值类型为：列表解析式，且该列表解析式基于至少一个可控因子进行迭代。

[0105] 5、参数的赋值类型为：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

[0106] 若属于上述五种条件中的一种或任意组合，则确定相应的赋值后的参数为可控参数。

[0107] 步骤 1205：将赋值后的参数归属至可控参数集合中。

[0108] 步骤 1206：判断当前解析的 body 是否为空？即判断当前解析的 body 中是否还有尚未完成的内容，若是，则执行步骤 1207；否则，返回步骤 1201。

[0109] 步骤 1207：判断待解析的 body 结构、orelse 结构、handles 结构、tests 结构等等是否为空？若是，则结束当前流程；否则，返回步骤 1200。

[0110] 上述流程即是通过递归方式遍历语法树结构，跟踪危险函数的外层函数所包含的参数的赋值类型，如果赋值类型属于上述五种情况中的一种且输入的参数是可控的，则认为赋值后的变量还是可控的，这样，就得到了整个外层函数的可控变量，从而组成可控参数集合。

[0111] 那么，再判断危险函数包含的参数是否归属于上述可控参数集合，即可以获知危险函数包含的参数是否为可控参数。

[0112] 用以解决现有技术下提高 python 代码漏洞检测的精准度。

[0113] 基于上述实施例，参阅图 3 所示，本发明实施例中，用于检测 python 漏洞的装置包括提取单元 30、处理单元 31、匹配单元 32 和确定单元 33，其中，

[0114] 提取单元 30，用于对待检测的 python 代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数；

[0115] 处理单元 31，用于对提取出的危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合；

[0116] 匹配单元 32，用于将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数；

[0117] 确定单元 33，用于根据包含有可控参数的危险函数确定相应的代码漏洞。

[0118] 较佳的，提取单元 30 进一步用于：

[0119] 在对 python 代码进行语法解析的过程中，根据指示删除非必要代码 .

[0120] 较佳的，处理单元 31 进一步用于，

[0121] 在进行数据流跟踪的过程中，基于输入 import 机制，从数据流以及 python 代码的环境变量中获取引入的第三方模块或 / 和第三方函数，并对第三方模块或 / 和第三方函数进行解析；

[0122] 根据解析结果判断所述第三方模块或 / 和第三方函数包含的参数在整个函数处理过程是否发生改变，将未发生改变的参数归属于可控参数集合中。

[0123] 较佳的，将赋值类型满足预设传递规则的参数确定为可控参数时，处理单元 31 具体用于：

[0124] 将赋值类型满足以下规则中的一种或任意组合的参数确定为可控参数：

[0125] 参数的赋值类型为：指定的属性取值；

[0126] 参数的赋值类型为：字符串拼接，且被拼接的字符串中包含已认定的可控参数；

[0127] 参数的赋值类型为：分片符取值；

[0128] 参数的赋值类型为：列表解析式，且列表解析式基于至少一个可控因子进行迭代，或者，且列表解析式为包含有可控因子的列表相加。

[0129] 参数为赋值类型为：经字符串操作函数处理的返回值，或 / 和，经未过滤函数处理的返回值。

[0130] 较佳的，处理单元 31 进一步用于：

[0131] 对语法树进行类分析，在分析过程中将各个类的初始化参数归属于可控参数集合，以及对各个类的变量赋值进行跟踪，若任一危险函数中使用了类的变量，则确定任一危险函数包含有可控参数。

[0132] 较佳的，确定任一危险函数中包含的参数为可控参数后，在根据任一危险函数确定相应的代码漏洞之前，处理单元 31 进一步用于：

[0133] 采用用户预设的安全函数或者 python 自带的安全函数对任一危险函数包含的可控参数进行处理，确定经处理后的可控参数仍可控时，最终判定任一危险函数为真正的危险函数。

[0134] 本发明实施例中，基于待检测 python 代码生成相应的语法树，提取出危险函数；再对危险函数的外层函数所包含的参数进行跟踪，将赋值类型满足预设传递规则的参数确定为可控参数，组成可控参数集合，以及确定危险函数中包含的参数归属于上述可控参数

组合时，确定危险函数中包含有可控参数；最后，再根据包含有可控参数的危险函数确定相应的代码漏洞。

[0135] 这样，基于语法树对 python 进行语法分析，避免了代码中的复杂内容对分析过程所造成的干扰（如，“换行”、“注释”所来来的干扰），能够在语法树中识别出参数，并且能根据预设传递规则判断参数在流动的过程中是否可控，如果在整个解析范围内可控，则认定为可控参数，若识别出的危险函数中存在可控参数，则认为相应的一段代码存在漏洞，这样，大大提高了 python 代码漏洞检测的准确性，解决了 python 代码语法灵活维护困难的问题，由于是自动化的跟踪参数，免去了人工单步跟踪参数，降低了操作复杂度。

[0136] 另一方面，由于基于 import 机制引入的第三方模块或第三方函数也进行了跟踪，因此，免去了人工跳转到其他文件审查代码，降低了操作复杂度，从而在漏洞报告中可以直接报告了代码位置及产生漏洞的原因，使漏洞报告无需修改就可使用。

[0137] 本领域内的技术人员应明白，本发明的实施例可提供为方法、系统、或计算机程序产品。因此，本发明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，本发明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。

[0138] 本发明是参照根据本发明实施例的方法、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。

[0139] 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。

[0140] 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。

[0141] 尽管已描述了本发明的优选实施例，但本领域内的技术人员一旦得知了基本创造性概念，则可对这些实施例作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发明范围的所有变更和修改。

[0142] 显然，本领域的技术人员可以对本发明实施例进行各种改动和变型而不脱离本发明实施例的精神和范围。这样，倘若本发明实施例的这些修改和变型属于本发明权利要求及其等同技术的范围之内，则本发明也意图包含这些改动和变型在内。

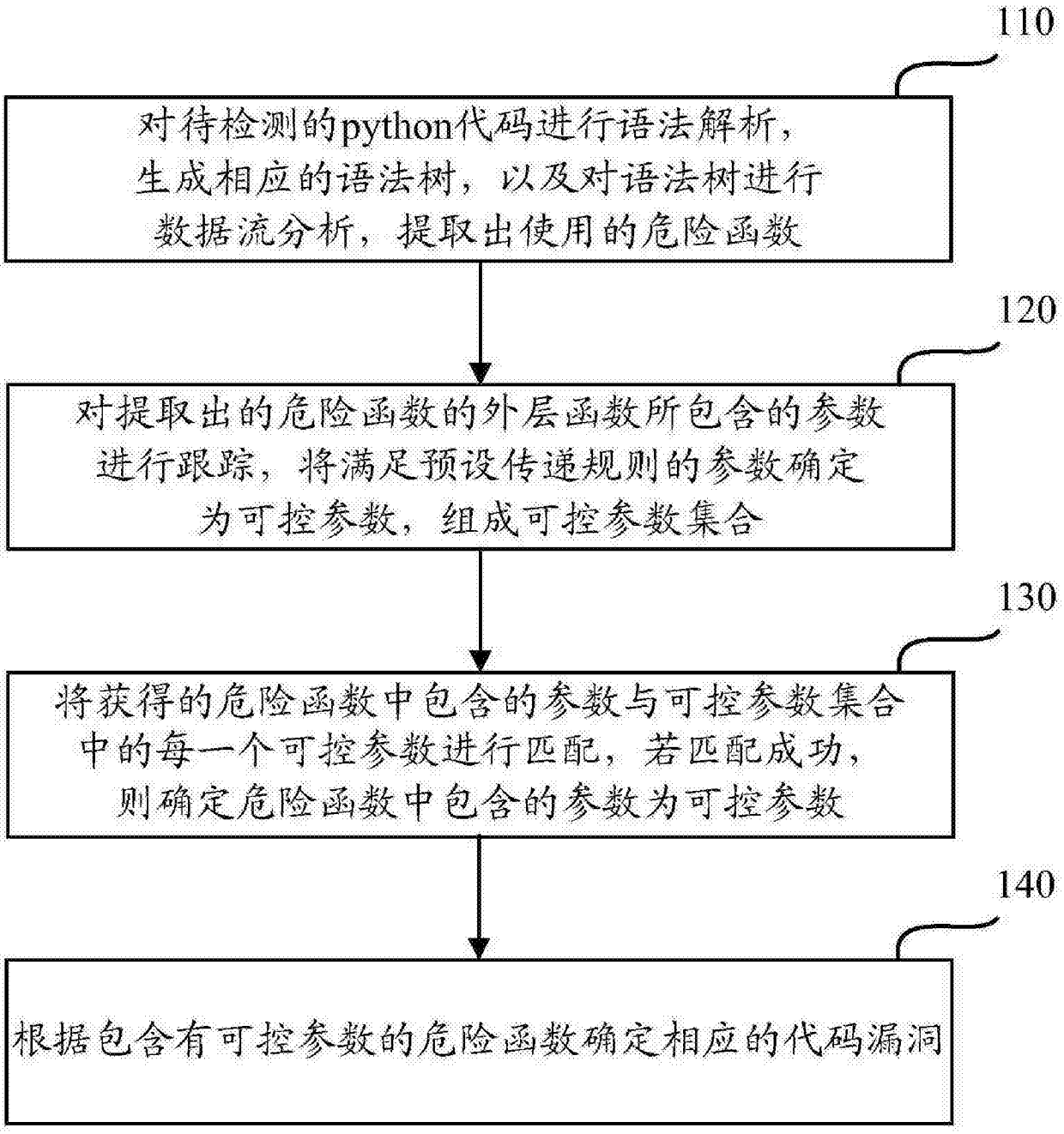

110
对待检测的python代码进行语法解析，生成相应的语法树，以及对语法树进行数据流分析，提取出使用的危险函数
120
对提取出的危险函数的外层函数所包含的参数进行跟踪，将满足预设传递规则的参数确定为可控参数，组成可控参数集合
130
将获得的危险函数中包含的参数与可控参数集合中的每一个可控参数进行匹配，若匹配成功，则确定危险函数中包含的参数为可控参数
140
根据包含有可控参数的危险函数确定相应的代码漏洞

图 1

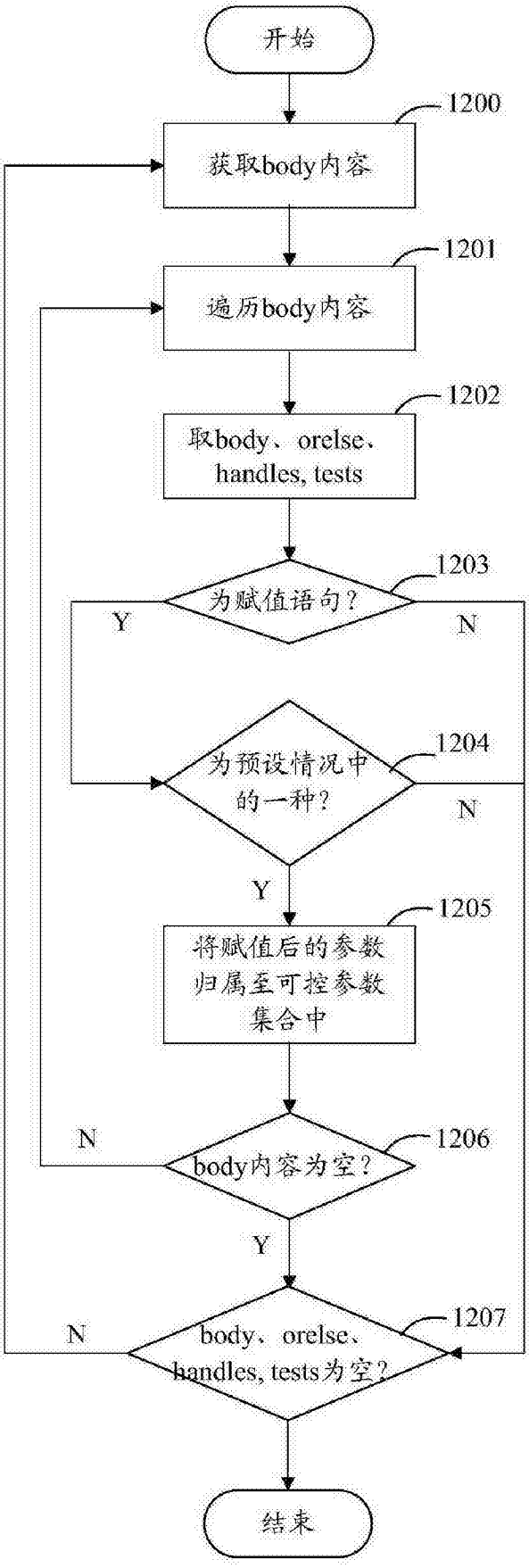

开始
1200
获取body内容
1201
遍历body内容
1202
取body、orelse、handles, tests
1203
为赋值语句？
Y
N
为预设情况中的一种？
1204
N
Y
1205
将赋值后的参数归属至可控参数集合中
N
body内容为空？
1206
Y
N
body、orelse、handles, tests为空？
1207
结束

图 2

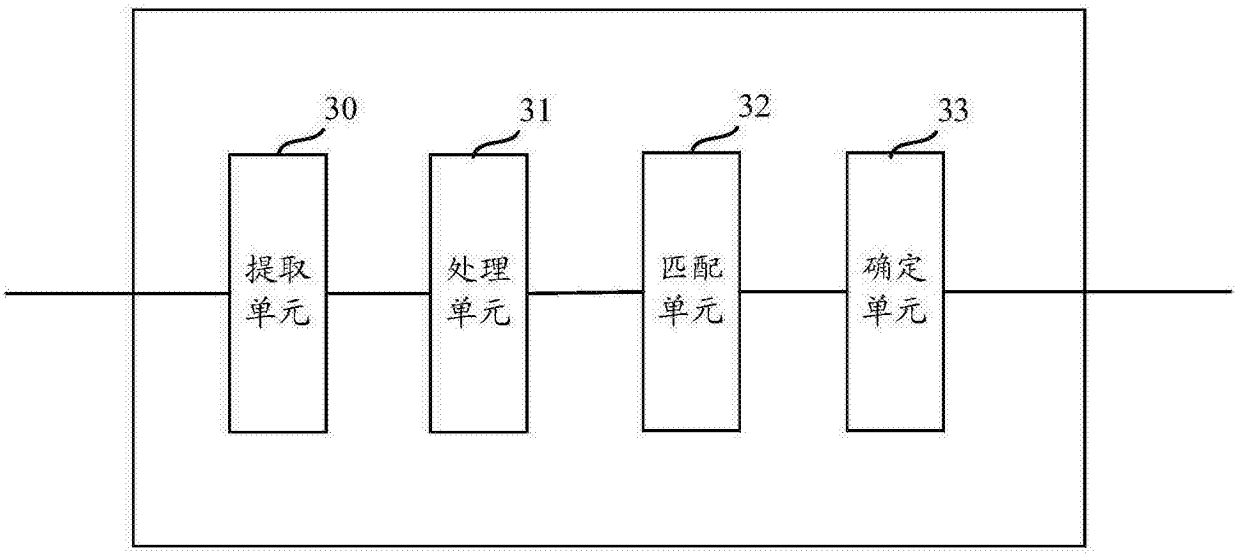


图 3