(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 110020552 B
(45)授权公告日 2021.02.26

(21)申请号 201910278560.6
(22)申请日 2019.04.09
(65)同一申请的已公布的文献号
申请公布号 CN 110020552 A
(43)申请公布日 2019.07.16
(73)专利权人 中南大学
地址 410083 湖南省长沙市岳麓区麓山南路932号
(72)发明人 王伟平 舒子蔚 宋虹 张士庚
(74)专利代理机构 长沙市融智专利事务所(普通合伙) 43114
代理人 杨萍
(51)Int.Cl.
G06F 21/62(2013.01)

(56)对比文件
CN 106570399 A,2017.04.19
CN 106940773 A,2017.07.11
US 10049222 B1,2018.08.14
郭帆 等.基于ICC的Android恶意程序检测方法.《江西师范大学学报》.2019,第43卷(第2期),

审查员 刘洁

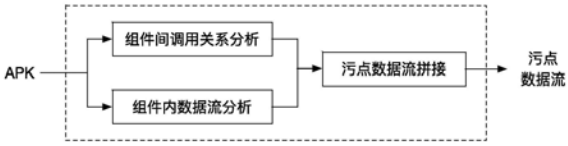
权利要求书2页 说明书9页 附图2页

(54)发明名称

一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法

(57)摘要

本发明提出了一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法,首先提取Android应用的组件间的调用关系与组件内的污点数据流,然后依据污点数据流的Source点与Sink点进行分类,组成污点数据流序列集,最后建立有限自动机模型,判断各污点数据流序列能否进行拼接,对能够进行拼接的污点数据流序列,拼接形成跨组件污点数据流,获得Android应用跨组件污点数据流集。本发明所提出的跨组件污点数据流拼接方法,能得到正确的Android应用跨组件污点数据流,且能减少空间开销。



CN 110020552 B

1.一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法,其特征在于,包括以下步骤:

步骤1:提取Android应用的组件调用参数,确定组件间调用关系;并提取Android应用组件内的污点数据流,存储其相关信息;

步骤2:依据污点数据流的Source点与Sink点情况,对Android应用组件内的污点数据流进行分类;

步骤3:将分类后的污点数据流,按照组件间调用关系,组成多条污点数据流序列,由所有污点数据流序列组成污点数据流序列集;

步骤4:建立污点数据流拼接自动机模型;将上述污点数据流序列集中的各条污点数据流序列分别输入污点数据流拼接自动机模型进行处理,判断能否进行拼接;

步骤5:分别对每条能够拼接的污点数据流序列,将其中的污点数据流按顺序进行拼接,形成跨组件污点数据流;由所有跨组件污点数据流组成跨组件污点数据流集;

所述步骤1中,自定义结构体flow用于存储污点数据流的相关信息,结构体flow包括组件名、Source点、Sink点、读取键集Kg、存放键集Kp、和映射表f;其中组件名表示污点数据流所在组件,Source点表示污点数据流的入点,Sink点表示污点数据流的出点;读取键集Kg表示污点数据流中读取Intent中数据的键的集合;存放键集Kp表示污点数据流中于Intent存放数据的键的集合;映射表f用于表示读取键集Kg和存放键集Kp之间的元素的对应关系;若Kg中的元素g1与Kp中的元素p对应同一个数据,则g1的对应关系为g1:{g1,p},其中{g1,p}为g1映射后的元素集合;若Kg中的元素g2与Kp中的任何元素都没有对应关系,即集合Kp中不存在与g2对应同一个数据的元素,则g2的对应关系为g2:{g2}。

2.根据权利要求1所述的基于有限状态自动机的Android应用跨组件污点数据流拼接方法,其特征在于,所述步骤2中,依据污点数据流的Source点与Sink点情况,将Android应用组件内的污点数据流分为N、S、D和M四种类型,具体如下:

N类型污点数据流的source点为直接获取敏感数据的API,sink点为直接发送敏感数据的API;

S类型污点数据流的source点为直接获取敏感数据的API,sink点为组件调用API;

D类型污点数据流的source点为组件调用API,sink点为直接发送敏感数据的API;

M类型污点数据流的source点为组件调用API,sink点为组件调用API。

3.根据权利要求2所述的基于有限状态自动机的Android应用跨组件污点数据流拼接方法,其特征在于,所述步骤3中,污点数据流序列有两种形式;

1)包含2条污点数据流,表示为$flow_1$,$flow_2$,其中$flow_1$为S类型污点数据流、$flow_2$为D类型污点数据流,且$flow_1$和$flow_2$所在组件满足组件调用关系;

2)包含3条以上污点数据流,表示为$flow_1$,…,$flow_i$,…,$flow_n$,其中$1<i<n$,$n\geq 3$,$flow_1$为S类型污点数据流,$flow_2$～$flow_{n-1}$为M类型污点数据流,$flow_n$为D类型污点数据流,且两条相邻的污点数据流$flow_{j-1}$和$flow_j$所在组件满足组件调用关系,其中$2\leq j\leq n$。

4.根据权利要求3所述的基于有限状态自动机的Android应用跨组件污点数据流拼接方法,其特征在于,所述步骤4中,污点数据流拼接自动机模型的状态包括:

$state_0$:自动机初始态,此时自动机内没有污点数据流;

$state_1$:自动机拼接状态;

$state_2$:自动机拼接终止态;

$state_3$:自动机错误终止态;

所述步骤4中,将污点数据流序列集中的每条污点数据流序列分别输入污点数据流拼接自动机模型进行处理,判断各条污点数据流序列能够进行拼接的方法为:

步骤4.1:自动机进入$state_0$态,等待输入污点数据流序列,设置当前污点数据流c为空;

步骤4.2:输入一个污点数据流序列,并读入其中的第一条污点数据流;若读入的污点数据流是S类型污点数据流,则保存当前污点数据流c为读入的S类型污点数据流,自动机状态转换为$state_1$,转步骤4.3;否则,自动机状态转换为$state_3$,转步骤4.5;

步骤4.3:从输入的污点数据流序列中读入下一条污点数据流$flow_i$;

(1)若读入的污点数据流$flow_i$为M类型污点数据流,则将当前污点数据流c的组件名改为$flow_i$的组件名,将当前污点数据流c的Sink点改为$flow_i$的Sink点,将当前污点数据流c的存放键集Kp中与$flow_i$的读取键集Kg中相同的元素,按照$flow_i$的映射表中该元素的对应关系,改为映射后的元素,自动机状态仍为$state_1$,转步骤4.3继续读入下一条污点数据流;

(2)若读入的污点数据流$flow_i$为D类型污点数据流,且当前污点数据流c的存放键集Kp与$flow_i$的读取键集Kg的交集非空,则将当前污点数据流c的组件名改为$flow_i$的组件名、Sink点改为$flow_i$的Sink点、存放键集Kg改为空,自动机状态转换为$state_2$,转步骤4.4,否则,自动机状态转换为$state_3$,转步骤4.5;

(3)若读入的污点数据流为其它类型污点数据流,自动机状态转换为$state_3$;

步骤4.4:输出当前污点数据流序列能够拼接的信息;

步骤4.5:输出当前污点数据流序列不能拼接的信息。

一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法

技术领域

[0001] 本发明涉及移动端安全领域，特别是一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法。

背景技术

[0002] 随着移动互联网技术的发展以及移动端智能设备的普及，移动互联网在生活中逐渐占据了重要的地位。移动端智能设备存储了大量用户隐私信息，随之而来的隐私泄露问题逐渐被人们所关注。

[0003] 污点数据流分析是一种用来获取相关数据沿着程序执行路径流动的程序信息分析技术。污点数据流分析是常见的分析程序中数据传播的一种方法，通过对程序中的敏感数据进行污点标记，利用数据流分析技术跟踪标记的数据在程序中的传播，来检测程序中是否存在安全隐患问题。污点数据流分析在Android应用的安全漏洞检测、污点数据泄露检测等安全方面具有广泛应用。

[0004] 现有Android应用的污点数据流分析按照技术大致可以分为两个方向：动态污点数据流分析，静态污点数据流分析。由于动态污点数据流分析的实现与Android平台的版本有关，不同的设备生产商对自己生产的设备会进行不同程度的定制，Android系统自身也在不断发展，这使得动态分析在兼容性方面的局限性较大。静态污点数据流分析则无需考虑Android环境的问题，具有较好的兼容性。

[0005] Android应用具有跨组件通信的特性，会发生多个组件联合泄露数据等安全问题，因此有必要对应用内跨组件污点数据流进行分析。现有的分析方法首先拼接出整体组件调用图，再在整体组件调用图上进行组件间的污点数据流分析，得到跨组件污点数据流。该方法需要先拼接出整体组件调用图，而整体组件调用图所需的存储空间大。因此，有必要设计一种能够解决该问题的跨组件污点数据流获取方法。

发明内容

[0006] 本发明所解决的技术问题，针对现有技术的不足，提供一种跨组件污点数据流拼接方法，通过组件间的调用关系和组件内的污点数据流获得跨组件污点数据流，能减少空间开销。

[0007] 本发明的技术解决方案如下：

[0008] 一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法，包括以下步骤：

[0009] 步骤1：提取Android应用的组件调用参数，确定组件间调用关系；并提取Android应用组件内的污点数据流，存储其相关信息；

[0010] 步骤2：依据污点数据流的Source点与Sink点情况，对Android应用组件内的污点数据流进行分类；

[0011] 步骤3:将分类后的污点数据流,按照组件间调用关系,组成多条污点数据流序列,由所有污点数据流序列组成污点数据流序列集;

[0012] 步骤4:建立污点数据流拼接自动机模型;将上述污点数据流序列集中的各条污点数据流序列分别输入污点数据流拼接自动机模型进行处理,判断能否进行拼接;

[0013] 步骤5:分别对每条能够拼接的污点数据流序列,将其中的污点数据流按顺序拼接,形成跨组件污点数据流;由所有跨组件污点数据流组成跨组件污点数据流集。

[0014] 进一步地,所述步骤1中,自定义结构体flow用于存储污点数据流的相关信息,结构体flow包括组件名、Source点、Sink点、读取键集Kg、存放键集Kp、和映射表f;其中组件名表示污点数据流所在组件,Source点表示污点数据流的入点,Sink点表示污点数据流的出点;读取键集Kg表示污点数据流中读取Intent中数据的键的集合;存放键集Kp表示污点数据流中于Intent存放数据的键的集合;映射表f用于表示读取键集Kg和存放键集Kp之间的元素的对应关系;若Kg中的元素g1与Kp中的元素p对应同一个数据,则g1的对应关系为g1:{g1,p},其中{g1,p}为g1映射后的元素集合;若Kg中的元素g2与Kp中的任何元素都没有对应关系,即集合Kp中不存在与g2对应同一个数据的元素,则g2的对应关系为g2:{g2}。

[0015] 进一步地,所述步骤2中,依据污点数据流的Source点与Sink点情况,将Android应用组件内的污点数据流分为N、S、D和M四种类型,具体如下:

[0016] N类型污点数据流的source点为直接获取敏感数据的API,sink点为直接发送敏感数据的API;

[0017] S类型污点数据流的source点为直接获取敏感数据的API,sink点为组件调用API;

[0018] D类型污点数据流的source点为组件调用API,sink点为直接发送敏感数据的API;

[0019] M类型污点数据流的source点为组件调用API,sink点为组件调用API。

[0020] 进一步地,所述步骤3中,污点数据流序列有两种形式;

[0021] 1)包含2条污点数据流,表示为$flow_1$,$flow_2$,其中$flow_1$为S类型污点数据流、$flow_2$为D类型污点数据流,且$flow_1$和$flow_2$所在组件满足组件调用关系;

[0022] 2)包含3条以上污点数据流,表示为$flow_1$,...,$flow_i$,...,$flow_n$,其中$1<i<n$,$n\geq3$,$flow_1$为S类型污点数据流,$flow_2$～$flow_{n-1}$为M类型污点数据流,$flow_n$为D类型污点数据流,且两条相邻的污点数据流$flow_{j-1}$和$flow_j$所在组件满足组件调用关系,其中$2\leq j\leq n$。

[0023] 进一步地,所述步骤4中,污点数据流拼接自动机模型的状态包括:

[0024] $state_0$:自动机初始态,此时自动机内没有污点数据流;

[0025] $state_1$:自动机拼接状态;

[0026] $state_2$:自动机拼接终止态;

[0027] $state_3$:自动机错误终止态;

[0028] 所述步骤4中,将污点数据流序列集中的每条污点数据流序列分别输入污点数据流拼接自动机模型进行处理,判断各条污点数据流序列能够进行拼接的方法为:

[0029] 步骤4.1:自动机进入$state_0$态,等待输入污点数据流序列,设置当前污点数据流c为空;

[0030] 步骤4.2:输入一个污点数据流序列,并读入其中的第一条污点数据流;若读入的污点数据流是S类型污点数据流,则保存当前污点数据流c为读入的S类型污点数据流,自动机状态转换为$state_1$,转步骤4.3;否则,自动机状态转换为$state_3$,转步骤4.5;

[0031] 步骤4.3:从输入的污点数据流序列中读入下一条污点数据流$flow_i$;

[0032] (1)若读入的污点数据流$flow_i$为M类型污点数据流,则将当前污点数据流c的组件名改为$flow_i$的组件名,将当前污点数据流c的Sink点改为$flow_i$的Sink点,将当前污点数据流c的存放键集Kp中与$flow_i$的读取键集Kg中相同的元素,按照$flow_i$的映射表中该元素的对应关系,改为映射后的元素,自动机状态仍为$state_1$,转步骤4.3继续读入下一条污点数据流;

[0033] (2)若读入的污点数据流$flow_i$为D类型污点数据流,且当前污点数据流c的存放键集Kp与$flow_i$的读取键集Kg的交集非空,则将当前污点数据流c的组件名改为$flow_i$的组件名、Sink点改为$flow_i$的Sink点、存放键集Kg改为空,自动机状态转换为$state_2$,转步骤4.4,否则,自动机状态转换为$state_3$,转步骤4.5;

[0034] (3)若读入的污点数据流为其它类型污点数据流,自动机状态转换为$state_3$;

[0035] 步骤4.4:输出当前污点数据流序列能够拼接的信息;

[0036] 步骤4.5:输出当前污点数据流序列不能拼接的信息。

[0037] 有益效果:

[0038] 本发明提出了一种跨组件污点数据流拼接方法,采用“分而治之”的思想,先逐个分析每个组件调用图中的污点数据流,得到组件内的污点数据流,然后将组件内污点数据流按组件调用关系和数据关系进行拼接,形成跨组件污点数据流。本发明能得到正确的Android应用跨组件污点数据流,且相比现有的分析方法,能减少空间开销,同时还具有扩展性。

附图说明

[0039] 图1为组件内污点数据流分类示意图;

[0040] 图2为污点数据流序列示意图;

[0041] 图3为污点数据流拼接自动机模型图;

[0042] 图4为跨组件污点数据流拼接方法分析框架图;

[0043] 图5为市场应用内存占用对比图。

具体实施方式

[0044] 以下将结合附图和具体实施例对本发明做进一步详细说明:

[0045] 实施例1:

[0046] 本实施例说明上述方法的具体步骤。

[0047] 图4是本发明一种基于有限状态自动机的Android应用跨组件污点数据流拼接方法分析框架图,具体步骤如下:

[0048] 步骤1:提取Android应用的组件调用参数,确定组件间调用关系,并将组件内污点数据流相关信息存入自定义结构体flow中;

[0049] 步骤1.1:从Android应用中提取各个组件的Intent与Intent Filter参数,通过匹配Intent与Intent Filter,来确定组件间的调用关系。

[0050] 使用现有工具IC3分析出Android应用内组件调用的Intent和Intent Filter的参数,根据Android官方文档,将源组件的Intent与目标组件的Intent Filer匹配,从而确定

组件间的调用关系。

[0051] 步骤1.2:从Android应用中提取组件内的污点数据流,将提取到的污点数据流相关信息存入自定义结构体flow中;

[0052] 自定义的结构体定义为:

```
Struct flow{
        String comp;
        String source;
        String sink;
[0053]
        String[] Kg;
        String[] Kp;
        Map f;
}
```

[0054] 其中,comp表示污点数据流所在组件,source表示污点数据流的入点,sink表示污点数据流的出点。

[0055] 集合Kg表示污点数据流中读取Intent中数据的键的集合,键由污点数据流中的getExtra()中的参数提取而来,提取到的这一类键构成集合Kg。

[0056] 集合Kp表示污点数据流中于Intent存放数据的键的集合,键由污点数据流中的putExtra()中的第一个参数提取而来,提取到的这一类键构成集合Kp。

[0057] 映射表用于表示集合Kg和集合Kp之间的元素的对应关系;若集合Kg中的元素g1与集合Kp中的元素p对应同一个数据,则g1的映射表为g1:{g1,p};若集合Kg中的元素g2与集合Kp中的任何元素都没有对应关系(集合Kp中不存在与g2对应同一个数据的元素),则g2的对应关系为g2:{g2};此时的映射表为{g1:{g1,p},{g2:{g2}}。

[0058] 本实施例中,分析得到应用Firstapp.apk的调用关系如下:

[0059] Firstapp.apk应用有5个组件,分别是com.example.Firstapp.Comp1、com.example.Firstapp.Comp2、com.example.Firstapp.Comp3、com.example.Firstapp.Comp4、com.example.Firstapp.Comp5。其中,

[0060] com.example.Firstapp.Comp1调用com.example.Firstapp.Comp3,com.example.Firstapp.Comp3调用com.example.Firstapp.Comp4,记为com.example.Firstapp.Comp1->com.example.Firstapp.Comp3->com.example.Firstapp.Comp4;

[0061] com.example.Firstapp.Comp2调用com.example.Firstapp.Comp5,记为com.example.Firstapp.Comp2->com.example.Firstapp.Comp5。

[0062] Firstapp.apk中有7条污点数据流,如下:

[0063] flow1{com.example.Firstapp.Comp1,getDeviceId(),d(java.lang.String,java.lang.String),{},{},{}};

[0064] flow2{com.example.Firstapp.Comp1,getDeviceId(),startActivity(android.content.Intent),{},{"str1"},{}};

[0065] flow3{com.example.Firstapp.Comp2,getLatitude(),sendBroadcast(android.content.Intent),{},{},{}};

[0066] flow4{com.example.Firstapp.Comp4,getIntent(),write(byte[]),{"str1"},{},{}};

[0067] flow5{com.example.Firstapp.Comp3,getIntent(),d(java.lang.String,java.lang.String),{"str3"},{},{}};

[0068] flow6{com.example.Firstapp.Comp3,getIntent(),startActivity(android.content.Intent),{"str4"},{"str2"},{"str4":{"str4","str2"}}};

[0069] flow7{com.example.Firstapp.Comp5,getIntent(),sendBroadcast(android.content.Intent),{},{},{}};

[0070] 步骤2:依据污点数据流的Source点与Sink点情况,对组件内污点数据流进行分类;

[0071] 使用组件内污点数据流分析工具FlowDroid分析出组件内的污点数据流,以定义好的结构体存储,并依据污点数据流的Source点和Sink点情况,将组件内的污点数据流进行分类,分成N、S、D、M四类。,如图1所示。其中

[0072] source点为直接获取敏感数据的API,sink点为直接发送敏感数据的API的污点数据流为N类型污点数据流,该类型污点数据流中,读取键集Kg为空、存放键集Kp为空、映射关系f为空;

[0073] source点为直接获取敏感数据的API,sink点为组件调用API的污点数据流为S类型污点数据流,该类型污点数据流中,读取键集Kg为空、存放键集Kp非空、映射关系f为空;

[0074] source点为组件调用API,sink点为直接发送敏感数据的API的污点数据流为D类型污点数据流,该类型污点数据流中,读取键集Kg非空、存放键集Kp为空、映射关系f为空;

[0075] source点为组件调用API,sink点为组件调用API的污点数据流为M类型污点数据流,该类型污点数据流中,读取键集Kg、存放键集Kp和映射关系f要么全空,要么全部非空;

[0076] 本实施例中,应用myapp.apk中的组件内污点数据流分类情况为:

[0077] N类型污点数据流有flow1;

[0078] S类型污点数据流有flow2、flow3;

[0079] D类型污点数据流有flow4、flow5;

[0080] M类型污点数据流有flow6、flow7;

[0081] 步骤3:将分类后的污点数据流按组件间调用关系组成污点数据流序列集;

[0082] 污点数据流序列有两种形式;

[0083] 1)包含2条污点数据流,表示为$flow_1$,$flow_2$,其中$flow_1$为S类型污点数据流、$flow_2$为D类型污点数据流,且$flow_1$和$flow_2$所在组件满足组件调用关系;

[0084] 2)包含3条以上污点数据流,表示为$flow_1$,...,$flow_i$..,.,$flow_n$,其中1＜i＜n,n≥3,$flow_1$为S类型污点数据流、$flow_2$～$flow_{n-1}$为M类型污点数据流、$flow_n$为D类型污点数据

流，且两条相邻的污点数据流$flow_{j-1}$和$flow_j$所在组件满足组件调用关系，其中$2 \leq j \leq n$。如图2所示。

[0085] 将分类后的污点数据流，按照组件调用关系，组成污点数据流序列，将污点数据流序列组成污点数据流序列集。

[0086] 本实施例中，在应用Firstapp.apk中由组件调用关系得到两条跨组件污点数据流：

[0087] (1)由com.example.Firstapp.Comp1->com.example.Firstapp.Comp3->com.example.Firstapp.Comp4，可知污点数据流序列为flow2-flow6-flow4；

[0088] (2)com.example.Firstapp.Comp2->com.example.Firstapp.Comp5，得到flow2-flow5；

[0089] 从而，得到污点数据流序列集合为：{flow2-flow6-flow4，flow2-flow5}。

[0090] 步骤4：建立污点数据流拼接自动机模型，将上述污点数据流序列集合中的污点数据流序列依次输入污点数据流拼接自动机进行处理，从而获得跨组件污点数据流集；

[0091] 污点数据流拼接自动机模型包含四个状态，分别是：

[0092] $state_0$：自动机初始态，此时自动机内没有污点数据流；

[0093] $state_1$：自动机内的当前污点数据流为S类型污点数据流时的状态；

[0094] $state_2$：自动机内的当前污点数据流为N类型污点数据流时的状态，为自动机拼接终止态；

[0095] $state_3$：自动机错误终止态。

[0096] 步骤4.1：自动机进入$state_0$态，等待输入污点数据流序列，设置当前污点数据流c为空；

[0097] 步骤4.2：自动机状态为$state_0$时，从输入的污点数据流序列中读入第一条污点数据流。若读入的污点数据流是S类型污点数据流，保存当前污点数据流c为读入的S类型污点数据流，状态转换为$state_1$；否则，自动机转换为$state_3$；

[0098] 步骤4.3：自动机状态为$state_1$时，从输入的污点数据流序列中读入下一条污点数据流。

[0099] (1)若读入的污点数据流是M类型污点数据流，则将当前污点数据流c的组件名改为读入M类型污点数据流的组件名，Sink点改为读入M类型污点数据流的Sink点、存放键集Kp中与M类型数据的读取键集相同的元素改为映射后的元素，状态自转为$state_1$；

[0100] (2)若读入的污点数据流为D类型污点数据流，假如自动机当前污点数据流c的集合Kp与读入D类型污点数据流集合Kg的交集非空，则将当前污点数据流c的组件名改为读入D类型污点数据流的组件名、Sink点改为读入D类型污点数据流的Sink点、存放键集Kp改为空，状态转换为$state_2$；否则，状态转换为$state_3$；

[0101] (3)若读入的污点数据流为其它类型污点数据流，自动机转换为$state_3$；

[0102] 步骤4.4：若自动机状态为$state_2$，输出当前污点数据流序列能够拼接的信息，且当前污点数据流c中存储有拼接形成的跨组件污点数据流的source点和sink点，自动机运行结束；

[0103] 步骤4.5：若自动机状态为$state_3$，输出当前污点数据流序列不能拼接的信息，自动机运行结束；

[0104] 重复步骤4.1至步骤4.5,直到污点数据流序列集处理完成。

[0105] 将应用Firstapp.apk的污点数据流序列集为{flow2-flow6-flow4,flow2-flow5},依次输入污点数据流拼接自动机。

[0106] 1、自动机进入$state_0$态,等待输入污点数据流序列flow2-flow6-flow4,设置当前污点数据流c为空;

[0107] 2、自动机状态为$state_0$,输入污点数据流序列flow2-flow6-flow4,从中读出第一条污点数据流flow2。因为flow2为S类型污点数据流,则将自动机状态转换为$state_1$,保存当前污点数据流c为

[0108] {com.example.Firstapp.Comp1,getDeviceId(),startActivity(android.content.Intent),{},{"str1"},{}};

[0109] 3、自动机状态为$state_1$,从输入的污点数据流序列flow2-flow6-flow4中读入下一条污点数据流flow6。因为flow6为M类型污点数据流,将当前污点数据流c改为{com.example.Firstapp.Comp3,getDeviceId(),startActivity(android.content.Intent),{},{"str1"},{}},状态自转为$state_1$;

[0110] 4、自动机状态为$state_1$,从输入的污点数据流序列flow2-flow6-flow4中读入下一条污点数据流flow4。因为flow4是D类型污点数据流,且c.Kp与flow4.Kg的交集为{"str1"},将当前污点数据流c改为{com.

[0111] example.Firstapp.Comp4,getDeviceId(),write(byte[]),{},{},{}},状态转换为$state_2$;

[0112] 5、输出当前污点数据流

[0113] c{com.example.Firstapp.Comp3,getDeviceId(),write(byte[]),{},{},{}},自动机结束对序列flow2-flow6-flow4处理。

[0114] 6、自动机进入$state_0$态,等待输入污点数据流序列flow2-flow5,设置当前污点数据流c为空;

[0115] 7、自动机状态为$state_0$,输入污点数据流序列flow2-flow5,读入一条污点数据流flow2。flow2是S类型污点数据流,将自动机状态转换为$state_1$,保存当前污点数据流c为{com.example.

[0116] Firstapp.Comp1,getDeviceId(),startActivity(android.content.Intent),{},{"str1"},{}};

[0117] 8、自动机状态为$state_1$,输入污点数据流序列flow2-flow5中读入一条污点数据流flow5。flow5是D类型污点数据流,c.Kp与flow5.Kg的交集为空,自动机转换为$state_3$;

[0118] 9、自动机状态为$state_3$,输出当前污点数据流序列不能拼接的信息。

[0119] 最终获得Firstapp.apk中的跨组件污点数据流集为{{com.example.Firstapp.Comp3,getDeviceId(),write(byte[]),{},{},{}}}。

[0120] 实施例2:

[0121] 本实施例说明上述方法的正确性。

[0122] 测试环境:CPU为Intel Core i7-7700处理器,内存8GB,搭载Ubuntu16.04TLS的64位操作系统上进行实验。

[0123] 测试样例:16个来自开源样例集DroidBench2.0中的跨组件通信应用。其中

DroidBench2.0是公开的一个涵盖多种污点数据流测试的Android应用样例集。

[0124] 测试对照工具:IccTA。IccTA是近年提出的,针对Android应用的跨组件污点数据流隐私检测常用工具。

[0125] 首先对这16个样例进行人工分析,得到它们真实的污点数据流情况。然后使用IccTA和本发明对这16个样例进行测试。测试数据如表1所示。

[0126] 表1在DroidBench上的测试结果(单位:条)

[0127]

应用名	真实情况	本发明	IccTA
ActivityCommunication1	1	1	1
ActivityCommunication2	1	2	2
ActivityCommunication3	1	0	0
ActivityCommunication4	1	2	2
ActivityCommunication5	1	1	1
ActivityCommunication6	1	0	0
ActivityCommunication7	1	1	1
ActivityCommunication8	1	2	2
ComponentNotInManifest1	0	0	0
EventOrdering1	1	1	1
IntentSink1	0	0	0
IntentSource1	0	0	0
ServiceCommunication1	1	0	0
SharePreferenTes1	1	1	1
Singletons1	0	0	0
UnresolvableIntent1	2	2	2

[0128]

[0129] 从表中可以发现,本发明的测试结果与IccTA一致,能得到跨组件污点数据流的概率约为76.9%。但与IccTA一样,由于采用的Intent提取工具的不精确导致测试中存在一定的误报与漏报。

[0130] 实施例3：

[0131] 本实施例说明上述方法的有益效果。

[0132] 使用2016年安软应用市场随机下载的100个市场应用与IccTA进行对比，内存对比结果如图5。图中横坐标表示应用编号，纵坐标表示每个应用在测试时占用的内存空间大小。

[0133] 由图可以看出，采用本发明测试应用时占用的内存空间比IccTA小。通过分析100个应用的内存空间大小，采用本发明测试应用时所占用的内存空间比IccTA内存平均减少约64.47％。

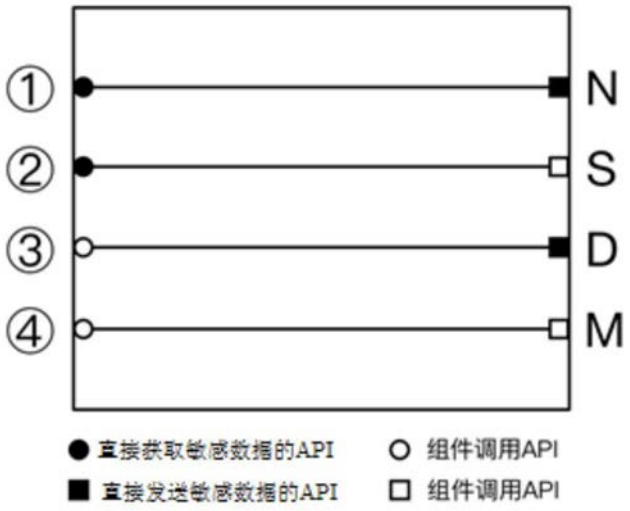

①
②
③
④
N
S
D
M
直接获取敏感数据的API
组件调用API
直接发送敏感数据的API
组件调用API

图1

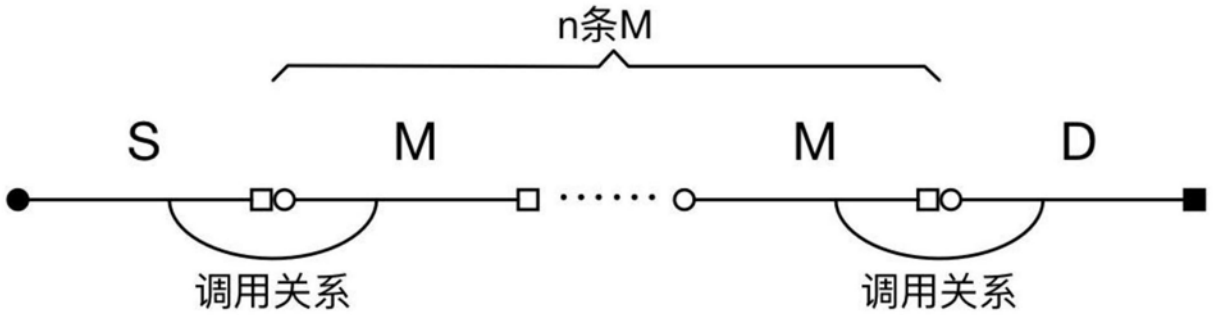

n条M
S
M
M
D
调用关系
调用关系

图2

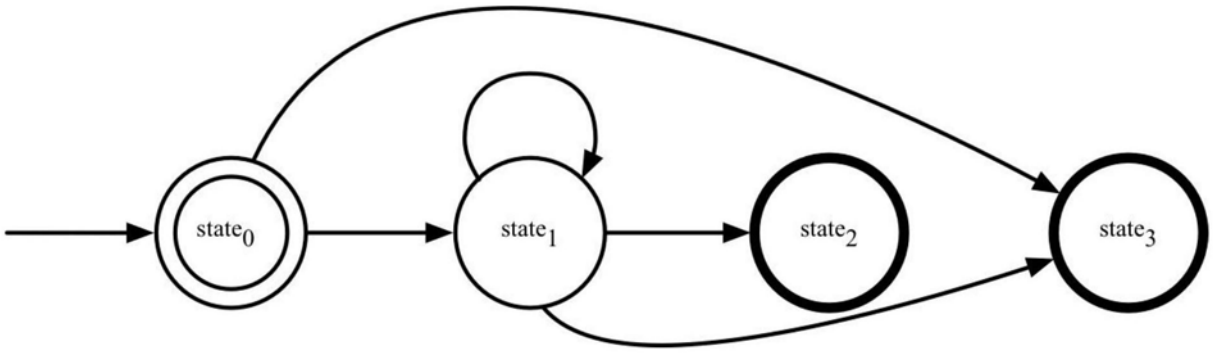

state0
state1
state2
state3

图3

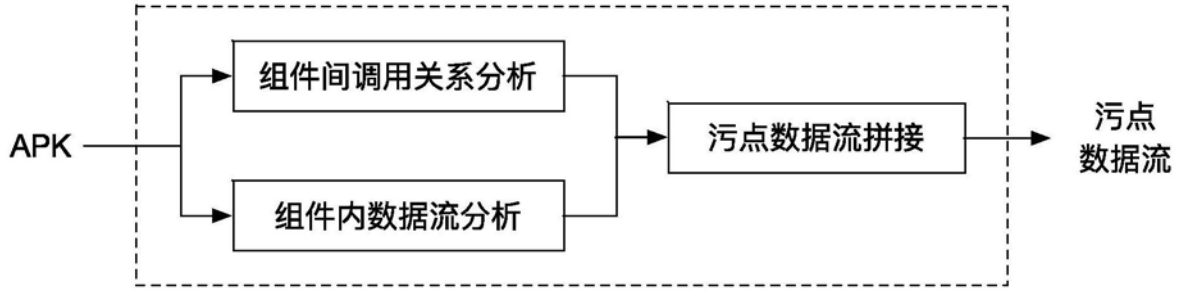

APK
组件间调用关系分析
组件内数据流分析
污点数据流拼接
污点
数据流

图4

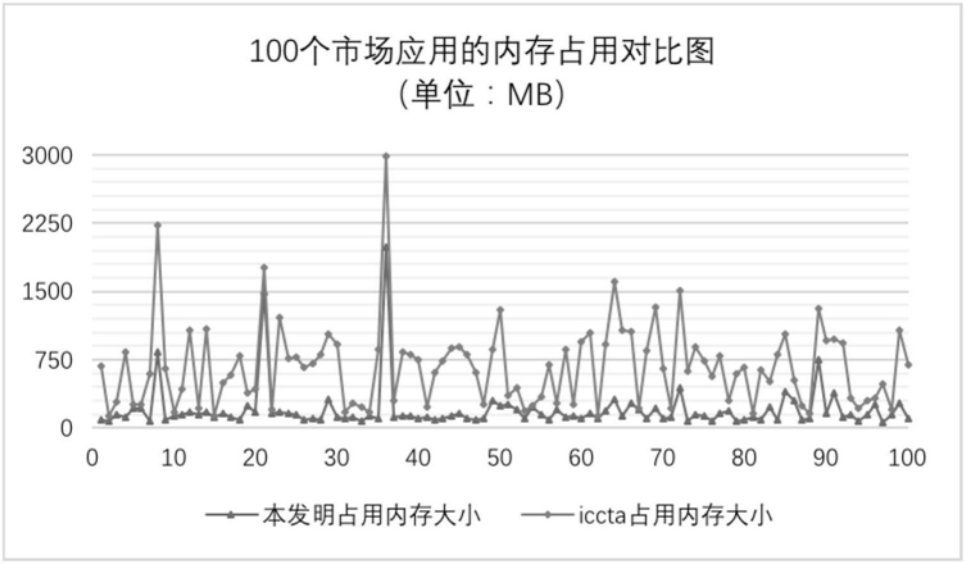

100个市场应用的内存占用对比图
(单位：MB)
3000
2250
1500
750
0
0
10
20
30
40
50
60
70
80
90
100
本发明占用内存大小
iccta占用内存大小

图5