计算机科学
COMPUTER SCIENCE

混淆应用中的第三方库函数定位

袁江风, 李昊翔, 游伟, 黄建军, 石文昌, 梁彬

引用本文

袁江风, 李昊翔, 游伟, 黄建军, 石文昌, 梁彬混淆应用中的第三方库函数定位[J]. 计算机科学, 2023, 50(7): 293-301.

YUAN Jiangfeng, LI Haoxiang, YOU Wei, HUANG Jianjun, SHI Wenchang, LIANG Bin. Locating Third-party Library Functions in Obfuscated Applications [J]. Computer Science, 2023, 50(7): 293-301.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221100147

混淆应用中的第三方库函数定位

袁江风　李昊翔　游　伟　黄建军　石文昌　梁　彬
中国人民大学信息学院　北京 100872
(202225feng@ruc.edu.cn)

摘　要　第三方库是 Android 应用程序的重要组成部分。在对应用进行基于重打包技术的安全增强或分析时,往往需要定位第三方库中的一些特定函数,此时需要将第三方库源码中的函数映射到目标应用反汇编代码中,以找到其对应的位置。在实际工作中,很多应用经过了代码混淆,这给定位第三方库函数带来了挑战。在经过混淆处理的应用程序反汇编代码中,大部分可供定位的特征被消除,代码也变得晦涩、难以分析。在缺少线索的情况下,从庞大的代码空间中定位到一个特定的函数十分困难。目前对混淆后应用进行的分析仅仅关注识别应用程序中包含了哪些第三方库,而没有更细粒度的函数级别的识别。文中提出了一种在混淆后的应用代码中定位第三方库中特定函数的方法。首先,对应用所用到的混淆器和混淆参数进行识别,从而将第三方库源码处理成与目标应用相同混淆方式的代码,即混淆对齐;在此基础上,通过静态插桩在待定位的函数中引入查找特征,并抽取其混淆后的结构特征来从目标应用中最终识别出待定位的函数位置。实验结果表明,所提方法能以较高的正确率识别出目标应用所使用的混淆工具及混淆参数,且能准确定位流行的混淆闭源应用中感兴趣的第三方库函数。

关键词:Android 应用;重打包;混淆;第三方库;定位
中图法分类号　TP309.2

Locating Third-party Library Functions in Obfuscated Applications

YUAN Jiangfeng,LI Haoxiang,YOU Wei,HUANG Jianjun,SHI Wenchang and LIANG Bin
School of Information,Renmin University of China,Beijing 100872,China

Abstract　Third-party libraries are an important part of Android applications. When enforcing security enhancement or analysis based on application repackaging,it is often necessary to locate specific functions in third-party library. To this end,there is a need to map the functions of the third-party library to the disassembly code of the target application. However,many applications are obfuscated,which brings challenges to locating third-party library functions. In the disassembly code of the obfuscated application,the discriminated fingerprints are often eliminated,hence the code becomes obscure and difficult to analyze. Due to the lack of location fingerprints,it is very difficult to identify a specific function from the huge code space. So far,the existing studies only focus on identifying which third-party libraries are included in the target application rather than locating specific functions. In this paper,a method to locate the third-party functions in obfuscated applications is presented. In the first place,the obfuscator and obfuscation parameters used in the target application are identified. The source code of the third-party library is obfuscated in the same way as done for the target application. The stage is called as obfuscation alignment in this study. On this basis,we introduce some location fingerprints into the target functions with static instrumentation,and extract the structural features to identify the function location from the target application. Experiments show that the proposed method can identify the obfuscation tools and obfuscation parameters with high accuracy,and can accurately locate the third-party library functions for popular obfuscated close-source applications.

Keywords　Android application,Repackaging,Obfuscation,Third-party library,Location

1　引言

在 Android 应用开发过程中,开发人员往往会调用各种开源第三方库,从而快速地实现想要的功能。流行的第三方库已经经过大量的测试和应用。相较于从头开发某项功能,调用第三方库来实现对应功能往往更加稳定。例如,通过调用 Litho,QMUI 等 UI 框架可以帮助开发人员快速地设计实现想要的界面。

重打包是一种重要的 Android 应用分析技术手段。通过重打包,分析人员可以向应用中注入安全代码来实施定制的应用加固或进行特定的安全分析。例如,可以在应用的交互 UI 中注入监听程序,对交互中的一些有害信息或敏感操作

到稿日期:2022-11-17　返修日期:2023-03-27
基金项目:国家自然科学基金(U1836209,62272465,62002361);CCF—华为胡杨林研究创新基金(CCF-HuaweiSE2021002)
This work was supported by the National Natural Science Foundation of China(U1836209,62272465,62002361) and CCF-Huawei Populus Euphratica Innovation Research Funding(CCF-HuaweiSE2021002).
通信作者:梁彬(liangb@ruc.edu.cn)

进行检测和拦截。

在实施基于重打包的安全加固和分析中，往往需要调用或修改应用中的第三方库的某些功能函数。这时就需要在目标应用中对这些感兴趣的第三方库函数进行定位。例如，在对具有聊天功能的应用注入恶意链接检测和拦截功能时，为了对其界面上包含恶意链接的消息进行修改或删除，需要找到实现聊天界面 UI 的第三方库中的对应函数，然后注入程序代码调用这些函数实现检测和拦截功能。

实际工作中，往往可以直接获得第三方库的源代码和相关文档，从而快速地在源代码中找到目标函数，进而利用函数标识等特征在目标应用的代码中进行检索，定位到对应的函数。遗憾的是，很多目标应用进行了混淆处理，在混淆后的代码中进行函数定位面临着诸多技术挑战。

混淆使得代码变得晦涩、难以阅读，令代码分析变得困难。如图 1 所示，混淆会将一些包进行扁平化处理和重构，使第三方库中的类以及应用原本的类被混在同一个包内。这会使得通过包名来区分目标应用中第三方库的代码变得困难。混淆还会隐藏函数名，并对控制流进行修改。如图 2 所示，图 2(a)是开源框架 Litho 对界面消息进行修改的 API 的 Smali 代码；图 2(b)是该 API 在一款办公应用 WorkPlace 中对应的 Smali 形式。两部分代码不仅在标识符的名称上存在差异，而且在控制流上也存在着差异。这导致无法直接通过函数名检索或代码结构匹配来定位第三方库的函数。

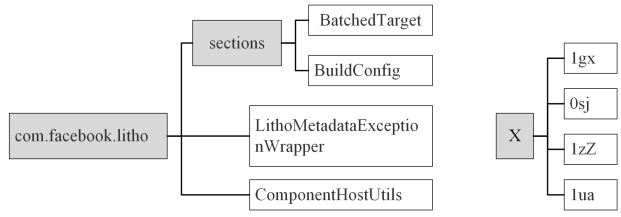

图 1 混淆前后包结构对比

Fig. 1 Comparison of package structures before and after obfuscation

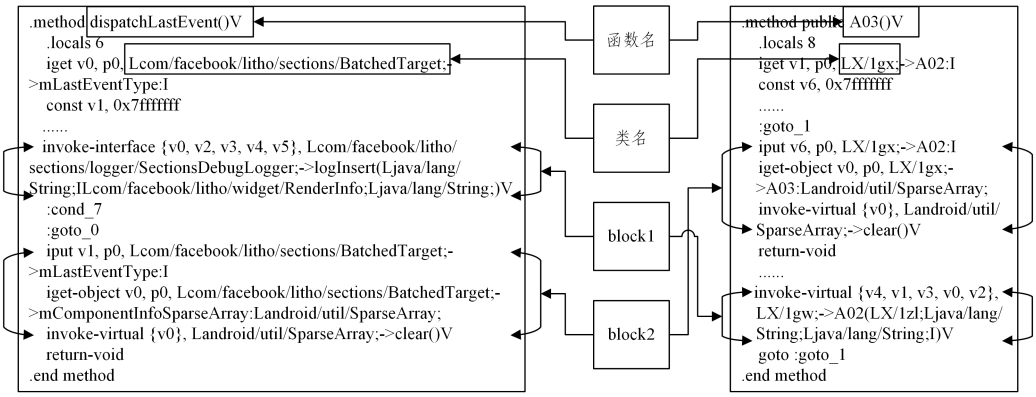

图 2 混淆前后代码对比

Fig. 2 Comparison of code before and after obfuscation

为此，本文提出了一种在混淆后的目标应用中定位第三方库函数的方法。其基本思想是进行混淆对齐(Obfuscation Alignment)，即通过探测目标应用所使用的混淆方式(包括混淆工具和参数)，对第三方库代码进行类似的混淆处理，从而在同等或类似混淆层面进行函数定位。本文方法分为 3 个步骤：1)离线构造了一个测试应用来收集常用混淆工具的特征，根据特征构造分类器来识别目标应用所使用的混淆工具；2)引入模糊测试思想，遍历相应混淆工具的参数来识别目标应用所使用的混淆参数；3)使用识别出的混淆工具和混淆参数对第三方库进行混淆处理，然后提取感兴趣函数的结构特征并与目标应用代码进行函数层面的匹配，找出第三方库感兴趣函数在目标应用中的位置。

本文对常用的混淆工具进行了分析并实现了对混淆工具及混淆参数的识别，其识别目标应用所使用的混淆工具的正确率达到 100%，混淆参数的有效识别率达到 87%。对应用市场中 WorkPlace，Twitter 等流行闭源应用进行感兴趣函数的定位实验结果表明，本文方法能成功定位感兴趣的函数，提高了混淆后第三方库函数定位工作的效率，能有效支持基于重打包的应用安全加固和分析。

本文第 2 节简要阐述了工作背景以及相关研究现状；第 3 节详细介绍了所提方法的工作流程和设计细节；第 4 节对该方法的效果进行实验评估；最后总结全文并展望未来。

2 背景与相关工作

本节第一部分阐述了代码混淆的基本操作和常用的 Android 代码混淆工具，第二部分对现有的相关研究工作进行总结。

2.1 混淆

代码混淆(Code Obfuscation)是将程序代码转换成一种功能上等价但是难于阅读和理解的形式的行为[1-2]。代码混淆可以在源代码[2]中进行，也可以在中间代码[3-5]中进行。混淆操作通常会：1)将代码中的各种元素的名字改写成无意义的名字，使得阅读的人无法根据名字猜测其用途；2)重写代码中的部分逻辑，将其变成功能上等价但更难理解的形式；3)打乱代码的格式。

混淆在 Android 应用开发中应用广泛[1,6-9]。本文对 Android 平台的混淆工具进行了深入的调研和分析，最终挑选了 5 个混淆工具作为代表：ProGuard，R8，Obfuscapk，Allatori 和 Dasho。选择这 5 个混淆工具有以下 4 个原因：1)这 5 个混淆工具是如今最常用的混淆工具，尤其是 ProGuard 和 R8，其分别集成在前一代和目前 Google 官方开发工具中[1,9-13]；2)其他与 Android 应用混淆相关的工作[3,9,12-19]也经常选取这些

混淆工具进行分析；3）其余混淆工具（如 DexProtector），因收费无法进行大范围的测试，只能简单地混淆几个应用作为对比；4）混淆技术本质上是通用的[9,20]，不同工具实现的功能虽然有所不同，但实现的基本逻辑相同，这 5 个混淆工具已经可以覆盖全部混淆功能。经过人工分析，其他的混淆工具的混淆结果会与其中之一相似，并无本质区别。

混淆工具的功能主要分为：标识符重命名、字符串加密、控制流混淆、重新打包类、资源加密等。部分混淆工具还兼具编译器的功能，如优化和代码收缩。其中会影响到应用反汇编代码进而给函数匹配带来困难的功能有：标识符重命名、字符串加密、控制流混淆、优化和代码收缩。上述 5 个混淆工具功能的对比如表 1 所列。

表 1 混淆工具功能对比

Table 1 Comparison of obfuscators' functions

混淆工具	标识符混淆	字符串混淆	控制流混淆	优化	代码收缩
R8	√	×	×	√	√
Proguard	√	×	×	√	√
Allatori	√	√	×	√	√
Dasho	√	√	√	×	×
Obfuscapk	√	√	√	×	×

2.2 相关工作

本小节简单介绍目前第三方库检测的研究成果和重打包技术的使用情况。

关于 Android 第三方库检测的研究有很多，如 LibLoom[15]，LibID[16]，Orlis[17]，LibPecker[21]，LibRadar[22]，LibScout[23]等。Orlis 利用从应用程序的过程间结构和行为（如方法的调用图）派生出的混淆弹性代码特性来检测混淆应用中的第三方库。LibID 用二进制整数规划模型来描述库识别问题，通过对应用程序二进制文件的静态分析，结合第三方库数据库，识别候选应用程序中第三方库的特定版本。LibLoom 将第三方库检测视为一个集合包含问题，利用一种可扩展的两阶段检测方法来检测经过混淆处理的应用中包含哪些第三方库。上述 3 项工作均针对混淆进行了处理，可以较好地抵御包括代码收缩和包修改在内的代码混淆，另外 3 项工作的抗混淆能力相对较弱。LibRecker 的方法依赖包层次结构来进行匹配，但包结构信息会被重打包混淆破坏，该方法不能抵抗该类型的混淆。LibRadar 基于稳定的 API 特性检测库，这些特性在大多数情况下都具有模糊处理能力。LibScout 是业内第一个轻量有效的 APP 第三方库扫描方法，且此方法可以对抗一般的代码混淆。上述第三方库检测方法能够以较高的准确率识别出应用中包含了哪些第三方库，但均没有对第三方库中的特定函数进行识别。

由于 Android 应用程序构建时的结构特点，反编译 Android 应用很容易，导致重打包 Android 应用的现象普遍存在。目前重打包技术已经得到了广泛的应用。一方面，攻击者可以在目标应用中注入一些恶意程序来达到窃取用户隐私、破解应用、在用户不知情下做出一些恶意操作等目的[24-26]；另一方面，安全研究人员可以通过注入安全代码对应用进行安全加固[27]或者利用重打包技术为 Android 应用的安全研究带来帮助[28]。可以说 Android 应用程序重打包技术已经成为 Android 生态系统中一项无法忽视的技术。

3 方法设计与实现

3.1 概述

为了有效定位混淆后代码中的感兴趣函数，本文方法的基本思路是：通过探测目标应用所使用的混淆方式（包括混淆工具和参数），对第三方库代码进行类似的混淆处理，从而在同等或类似混淆层面（即混淆对齐）进行函数定位。

如图 3 所示，本文方法分为 3 个步骤：识别混淆工具、识别混淆参数和定位函数。之所以将混淆工具的识别和混淆参数的识别分开，是因为如果不先确定要测试的混淆工具，则会因为搜索空间过大而无法在有效的时间内识别出目标应用使用的混淆参数。各步骤所负责的工作如下。

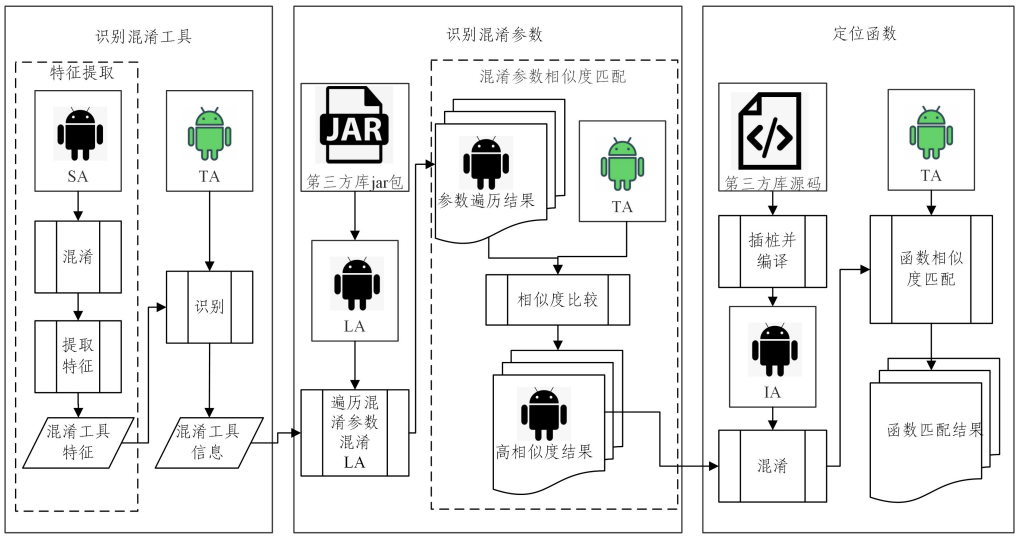

图 3 方案流程

Fig. 3 Workflow of the proposed method

(1)识别混淆工具。我们离线构造了一个包含各种典型代码段的测试应用 SA(Sample Application)。对 SA 使用常见混淆工具进行混淆,通过对比分析获得常见混淆工具在混淆后代码中表现出来的识别特征。根据这些特征来有效识别出目标应用所使用的混淆工具。

(2)识别混淆参数。针对每个目标应用,首先构造一个引用其所包含的第三方库的实验应用 LA(Library-container Application),然后基于模糊测试思想,调用第一步识别出的混淆工具对 LA 进行不同混淆参数的混淆处理,得到一个混淆后的应用集 $S=\{LA_0', LA_1', \cdots, LA_n'\}$。在此集合中,通过类层面的相似度匹配识别出与目标应用 TA 最相似的混淆后应用 LA′,从而识别出 TA 所使用的混淆参数。

(3)定位函数。为了方便在混淆后的第三方库代码中提取感兴趣的函数体,首先要对第三方库源代码进行插桩,在感兴趣的函数中插入一些混淆后仍能识别出的特殊标记;然后,构造一个引用经过插桩的第三方库的实验应用 IA(Instrumented-library-container Application);接着,对 IA 使用识别出的混淆工具和混淆参数进行混淆,使其与目标应用在混淆层面上对齐,获得混淆后的应用 IA′。最后在 IA′中通过插桩特征定位出感兴趣的函数,并将其与 TA 进行函数层面的匹配,从而定位出目标应用中感兴趣的函数的位置。下面将介绍这 3 个步骤的技术细节。

3.2 混淆工具识别

3.2.1 构造测试应用 SA

离线构造一个用于提取混淆工具识别特征的测试应用 SA,其包含各种典型的代码段。我们希望对 SA 的混淆能体现混淆工具的各种混淆功能。通过对比不同混淆工具对 SA 的混淆结果,可以提取出不同混淆工具对相同代码的处理的差异,用于识别混淆工具。

通过查阅相关文献[10,19]以及混淆工具的手册,并进行实际混淆测试,我们收集了常见混淆工具具有的功能及在不同代码上的表现。基于此,我们在 SA 中添加了用于测试各种混淆功能的特征代码段,具体如表 2 所列。在 SA 中,不同类型的特征代码段会组合出现,以确保能触发各种混淆功能。

表 2 混淆功能对应的特征

Table 2 Characteristics of obfuscation function

混淆功能	特征代码段
字符串混淆	不同类型的字符串变量声明
	不同类型的字符串变量引用
控制流混淆	循环语句
	条件跳转语句
	连续的条件跳转语句
	嵌套的条件跳转语句
	嵌套的循环语句
标识符重命名	Native 函数
	系统组件类
	不同类型的函数及变量
优化	条件永远为否的跳转分支
	不同类型的字段声明
	不同类型的函数及变量
	短函数及调用
	相同代码段的分支
	未使用的参数
代码收缩	未使用的参数、变量和函数
	条件永远为否的跳转分支

图 4 给出了一个特征代码段示例。其中,图 4(a)包含了对不同类型字符串的声明、调用和对长度进行混淆测试的代码段;图 4(b)包含了对条件分支、重复代码和简单函数这些优化内容进行测试的代码段;图 4(c)为对 Native 函数的声明和调用,其中还对字符串操作进行组合以触发对私有变量和静态变量进行字符串混淆的功能。

```
public static String static_test = "this is a static String";
private static String Static_test = "this is a private static String";
public String Pub_tes = "this is a public String";
private String toastTip = "toast in MyFragment";

public void methodWithLocalVariable() {
    String logMessage = "log in MyFragment";
    logMessage = logMessage.toLowerCase();
    System.out.println(logMessage + Static_test);
}
public void Test_String_Long(){
    System.out.println("1"+"12"+"123"+"1234"+"12345"+"123456");
    System.out.println("a"+"ab"+"abc"+"abcd"+"abcde"+"abcdef");
    System.out.println("1a"+"12ab"+"123abc"+"1234abcd"+"12345abcde");
    System.out.println("a1"+"ab12"+"abc123"+"abcd1234"+"abcde12345");
}
```

(a)字符串测试

```
public String getToastTip(int test){
    String testString = "";
    if (false)      return "123";
    if (true)       testString  = getToastTip3();
    return testString + getToastTip2();
    }
private String getToastTip2(){return toastTip;}
private String getToastTip3(){return toastTip;}
```

(b)优化测试

```
public static native void methodNative();
public static void methodNotNative() {
    String logMessage = "this is not native method";
    logMessage = logMessage.toLowerCase();
    System.out.println(logMessage);
}
public void methodWithGlobalVariable() {
    nativeUtil.methodNative();
    Toast.makeText(getActivity(), toastTip + Static_test,1000).show();
}
```

(c)标识符重命名测试

图 4 特征代码段

Fig. 4 Characteristic code snippets

3.2.2 提取混淆工具特征

提取的过程为:使用 R8,Proguard,Allatori,Dasho 和 Obfuscapk 这 5 种常见的混淆工具对测试应用 SA 进行混淆,然后对比混淆结果,归纳出它们之间的差异,在此基础上提取出用于识别各个混淆工具的特征。

识别特征分为两类:一类是功能特征,即混淆工具为了实现特定功能而在应用中引入的代码段,开启了某一功能则混淆结果会出现对应的特征;另一类是统计特征,即混淆工具在进行混淆时所引入的可统计性差异,例如,不同混淆工具生成的标识符长度不同,对继承的处理也不相同。统计特征与参数无关,是所有参数下都存在的特征。

提取统计特征时,首先遍历每个混淆工具中对混淆结果影响较大的参数组成的参数空间,对测试应用 SA 进行混淆,参数的挑选会在 3.3.1 小节详细描述。混淆后形成 5 个混淆工具相应的混淆结果集 $S_i(i=1\sim5)$。然后找出每个集合中在不同混淆参数下都存在的统计特征,接着对各个集合的统计特征进行差异分析,找出不同工具间的差异,作为区分不同混淆工具的识别特征。

各混淆工具所具有的特征总结如表 3 所列。

表 3 常见混淆工具特征

Table 3 Characteristics of popular obfuscators

混淆工具	字符串混淆	控制流混淆	随机化	命名	继承
R8	无	无	无	短	父类型
ProGuard	无	无	无	短	子类型
Dasho	解码函数固定	伪随机	格式固定	短	同 R8
Allatori	解码函数固定	随机	无	短	同 ProGuard
Obfuscapk	解码函数固定	伪随机	无	长	不固定

(1)字符串混淆:被混淆的字符串在调用时需要进行还原,每个混淆工具添加的字符串还原函数的解码逻辑是固定的。字符串还原函数可以作为识别对应混淆工具的特征。例如,Obfuscapk 的还原函数命名固定且包含大量解密方法;Allatori 的还原函数中包含连续的字符获取操作。

(2)控制流混淆:混淆工具在进行控制流混淆时会插入随机的跳转或判断分支。这些代码段是伪随机的,不同混淆工具所添加的代码段有各自的特点。例如,Dasho 添加的代码段会先声明一个字符串"0",再将其转换成整型数据并作为后续跳转的判断条件;Obfuscapk 添加的代码段以一个 if 判断开始,且两个分支下的操作都只有一个 goto 跳转语句。

(3)混淆随机化:Dasho 会在每个类的〈clinit〉方法中添加获取随机数的代码段。原本不具有〈clinit〉方法的类会被添加该方法并将代码段插入其中。该代码段格式几乎相同,例如:都会有获取系统时间、声明 wide 类型常量、对 RuntimeException 类型的对象进行初始化等操作。

(4)标识符和跳转标签的命名:对标识符和跳转标签的命名的混淆,主要由混淆工具实现的主体逻辑决定,不受混淆参数的影响。该部分逻辑产生差异的原因是混淆对象不同。R8 和 ProGuard 在源代码层面进行混淆,可以整体混淆,不会出现命名冲突。而 Obfuscapk 在反编译后的 smali 代码层面进行混淆,为了避免重打包时与原来的代码产生命名冲突,混淆后的名称需要与原本的命名形式不同。因此,R8 和 ProGuard 混淆后的标识符一般不超过 5 个字节,而 Obfuscapk 较长,大于 10 个字节。此外 Obfuscapk 混淆后的标识符和跳转标签还根据不同类型具有不同特征,大部分混淆后的类名以"p"开头,字段以"f"开头,方法以"m"开头,跳转标签以"l"开头。

(5)继承关系处理:在 Smali 代码中调用函数需要声明调用对象的类型。经过 R8 混淆的代码在调用父类中定义的函数时,对象被声明为父类。而 ProGuard 中则保持原类型。

3.2.3 特征利用

基于以上特征,设计一棵决策树来进行混淆工具识别。决策树有 6 个叶子节点分别对应 5 个常用混淆工具以及不属于上述 5 个混淆工具的其他混淆工具,设置"其他"类别是为了覆盖本文中未进行详细分析的混淆工具。使用"其他"类别混淆工具时,虽然参数识别精度有所下降,但仍可以成功进行函数定位。识别时经过三层决策,将混淆工具的识别结果最终确定为 6 个叶子节点中的一个。

首先根据检测目标应用中是否嵌入实现混淆功能的代码段,将结果分为两个分支。含有代码段的目标应用可根据包含的代码段在 Dasho,Allatori 和 Obfuscapk 之间进行区分;没有代码段,即没有使用字符串混淆、控制流混淆等混淆功能时,Dasho 的混淆结果与 R8 基本相同,Allatori 的混淆结果与 ProGuard 相同。此时需要对 R8,ProGuard,Obfuscapk 和"其他"进行区分。首先分析统计特征中的标识符和标签的命名来确定混淆工具是否满足已知特征,不满足的则分类为"其他",R8 和 ProGuard 还需要通过分析继承关系来进一步做区分。决策树的整体流程如图 5 所示。在接下来的混淆参数识别中,对已知的 5 个混淆工具直接使用相似度匹配来识别;对于"其他"类别的混淆工具,虽然无法直接获得其混淆参数,但是可以利用已知的 5 个混淆工具处理出接近的结果,该部分处理的方式在 3.3.2 小节中补充说明。

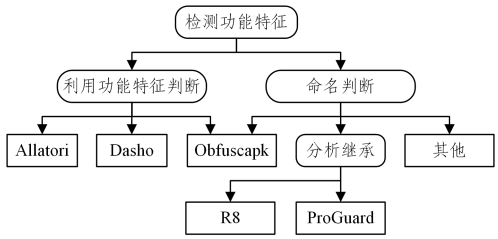

图 5 决策树

Fig. 5 Decision tree

3.3 混淆参数识别

我们采用模糊测试的思想来进行混淆参数的识别,即通过探索混淆参数空间来获得不同参数混淆的结果,再将混淆结果与目标应用进行相似度匹配,以发现与目标应用相近的混淆结果,从而确定目标应用混淆时的参数设置。

3.3.1 参数 Fuzzing

每个混淆工具都会有大量参数,这些参数主要分为:保留、混淆、优化和特定功能这几个类别。由于涉及的参数众多,进行全部覆盖不论在时间上还是存储空间上都是无法实现的,因此,在本文设计的 Fuzzing 策略中,只挑选部分对混淆结果影响较大的参数进行遍历。

首先将可能对混淆结果的代码结构产生影响的参数及其变化值筛选出来,在此基础上,通过实验进一步挑选出能对混淆结果产生有效影响的具体参数。此外,部分参数依赖于特定功能的开启,在进行 Fuzzing 时开启对应功能。

本文探索了常用混淆工具 R8,Proguard,Allatori,Dasho 和 Obfuscapk 的混淆参数。由于篇幅限制,本文只展示 R8 与 Proguard 的混淆参数,如表 4 所列。

Fuzzing 工具将遍历 5 个工具的参数空间,根据所获得的参数组合调用混淆接口对 LA 进行混淆,获得对应的混淆应用集 $Si=\{LA_0',LA_1',\cdots,LA_n\}'$,$(i=1\sim5)$,用于下一步进行相似度匹配。

表 4 ProGuard 和 R8 的混淆参数

Table 4 Obfuscation parameters of Proguard and R8

参数类型	可选变量	作用
shrink	on,off	启用代码收缩
obfuscate	on,off	启用混淆(混淆相关的参数需开启该选项)
overloadaggressively	on,off	混淆时应用多个字段和方法可以获得相同的名称
flattenpackagehierarchy	on,off	进行包结构扁平化
repackageclasses	on,off	进行重打包处理
optimization	on,off	启用优化(优化相关的参数需开启该选项)
mergeinterfacesaggressively	on,off	指定可以合并接口
allowaccessmodification	on,off	优化时允许访问并修改有修饰符的类和类的成员
dontskipnonpubliclibraryclassmembers	on,off	指定不去忽略包可见的库类的成员

3.3.2 相似度匹配

在进行相似度匹配时,基于这样一个假设:应用各个部分的代码使用的混淆工具及其参数保持一致。只要某个混淆后的实验应用 LA_x' 中的部分类与目标应用 TA 中对应的类的相似度较高,那么这个混淆版本对应的参数就与目标应用较为接近。这个假设是合理的,符合现实使用情况。几乎没有开发人员会对不同部分的代码使用不同的混淆工具或参数进行混淆,因为这样不仅会加大混淆时的工作量,还会因为不容易保持一致而出错。因此局部的混淆环境相似度就可以被认为是整体的混淆环境相似度。

采用局部匹配可以有效降低计算开销。假定应用中包含的类的数量为 n,直接在每个类之间进行相似度计算则计算的开销为 $O(n^2)$;引入混淆不变特征等先验知识进行配对后,再进行相似度计算的开销为 $O(n)$。单个 LA′与目标应用 TA 进行匹配的实验中,采用局部匹配计算相似度耗时为 2 s,全局计算耗时为 1 h 53 m 22 s。当使用大量混淆结果与目标应用进行相似度计算时,后者的时间开销是不可接受的。

进行匹配时,对于 S 中的每个混淆结果 LA_x',首先对 LA_x' 和 TA 提取特征,然后将 LA_x' 与 TA 中的类进行配对,最后在这些配对的类之间进行相似度计算。整个过程分为:提取特征、类配对和计算相似度 3 个步骤。

(1)提取特征。每一个类提取的特征包括两个部分:一部分是结构语义信息,包括类名、Native 函数原型和字段类型等标识符;另一部分是操作语义信息,由各语句的操作码构成。每个操作码对应一个字符,则操作语义信息映射成一个字符串,称为“操作串”。其中,结构语义信息用于类配对,操作语义信息用于计算相似度。

(2)类配对。从结构语义信息中选取混淆前后不变的特征,将 LA′中的类与 TA 中的类进行配对。我们利用差分的思想来获取标识符中混淆前后保持不变的部分,差分分为两个阶段:参数差分和应用差分。首先对参数进行差分,利用不同的混淆工具及参数对同一个测试应用(随机开源应用)进行混淆,取不同结果中出现的标识符的交集;然后对应用进行差分,选取不同测试应用重复上述步骤,取每次参数差分结果的并集即为最终混淆前后不变的标识符。

(3)计算相似度。相似度计算在 LA_x' 和 TA 配对的类之间进行。先计算每对类的相似度,再取均值作为 LA_x' 和 TA 的相似度:

$$Similarity=1-\frac{EditDistance}{\max(len1,len2)} \tag{1}$$

其中,$EditDistance$ 表示两个字符串的编辑距离,$len1$ 和 $len2$ 分别表示两个操作串的长度。

采用编辑距离计算相似度是因为编辑距离表示将字符串 A 转变成字符串 B 所需要的最少操作数,两个操作串的编辑距离越小则相似度越高。对应代码层面的表示是将实验应用代码进一步混淆成与目标应用完全相同的代码所需要的最小步骤。编辑距离可以反映两者做出的操作之间的差异,因此适合用于相似度的计算。

总体的相似度计算方法如下:

$$Similarity=\frac{\sum sim}{Num} \tag{2}$$

其中,sim 表示每个类的相似度,Num 表示匹配上的类的数量。

计算实验应用与目标应用的相似度后,将全部结果按照相似度进行排名即可得出与 TA 混淆最接近的 LA 混淆版本以及其混淆参数。

特别地,对于已知混淆工具的应用,使用遍历参数空间产生的应用集与目标应用进行匹配。对于“其他”类别的应用,识别的思路是利用已知的 5 个混淆工具和特定的参数组合模拟出未知混淆工具和参数的混淆结果。识别时,使用 5 个应用集中全部的实验应用与目标应用进行匹配,根据相似度最高的实验应用确定模拟时使用的混淆工具和参数。

3.4 定位函数

函数的定位在经过混淆对齐处理的实验应用 IA′和目标应用 TA 之间进行。函数混淆前后的映射关系通常不能直接得到,因此需要在感兴趣的函数中插入一些混淆前后保持不变的特征,称为“查找特征”,以便快速找到 IA′中感兴趣的函数。为此,第三方库源代码要先进行插桩再放入实验应用。然后,从混淆后的实验应用 IA′中提取出标记的函数体,与 TA 进行函数匹配定位目标函数。

3.4.1 插桩

插桩时插入的代码必须满足以下要求:在不同混淆参数下都能保证其可用性,方便查找和定位。如前所述,我们发现对于常见的混淆工具,Native 方法在混淆前后通常保持不变,这样可以很方便地构造出便于区分的特征。因此,本文选取调用特殊命名的 Native 方法的代码作为查找特征。

插桩包括两个操作:1)在第三方库源码中额外创建一个包含特殊 Native 方法的类,为了方便区分,这些 Native 方法使用了一些具有标志特征的名字;2)在感兴趣的函数中插入调用上述特殊 Native 函数的代码,即查找特征,这样第三方库代码即使经过混淆和反编译,感兴趣的函数中将依然带有对这些特殊 Native 函数的调用。只需要检索相应的函数调用便可以找到这些被标记的函数。

3.4.2 函数匹配

包含了插桩后第三方库的 IA 需要利用前两步识别出的

混淆工具及参数对进行混淆，使其与 TA 在混淆层面上对齐。经过混淆对齐，IA′中感兴趣的函数在混淆环境上已经与目标应用中对应的函数接近。接下来先通过检索插桩时插入的函数名称，提取出感兴趣的函数及其所属的类，再进行函数匹配，找出目标应用中对应的函数的位置。需要一次找出多个函数的位置时，只需要在插桩时在感兴趣的函数中都插入查找特征，就可以在检索时将感兴趣函数全部提取出来，同时与目标应用进行匹配。为了使匹配的结果更加准确，我们首先寻找相似的类，然后在相似的类中做进一步定位。

类匹配的具体做法是将被标记类与目标应用的每个类进行比较，找出与被标记类相似度较高的类。相似度的计算方式为：1）将被标记的类 C 中的函数 f 的操作串与目标应用类 C' 的每个函数 f' 的操作串进行相似度计算，取其中的最高值作为 f 的相似度，此处使用编辑距离来计算两个操作串的相似度；2）对 C 中所有函数重复上述过程获取全部函数的相似度；3）取相似度最高的前 n 个函数的平均值作为该类的相似度，考虑到目标应用会有代码收缩，因此 n 为 C' 中函数的个数。最后选取目标应用中与被标记类相似度排名较高的类作为下一步进行函数匹配的目标类。

在类匹配的基础上，将 IA′中被标记的函数与目标类中的每个函数进行比较，利用操作串计算相似度。相似度排名越靠前的函数越可能是要在目标应用中定位的第三方库函数。最后，输出相似度排名，然后通过人工分析确定最终的目标函数。

4 实验

本文方案的实际效果可以通过 3 个方面进行评估：识别混淆工具的准确率、识别混淆参数的准确率、函数定位是否能找到现实应用中感兴趣的第三方库函数以及对于“其他”类别的混淆工具是否仍有效。前两项评估混淆对齐处理的准确率；最后一项是第三方库函数定位在实际应用中的效果。接下来将分别介绍这 3 个问题中设计的实验配置和实验结果，并介绍一个重打包时定位应用中混淆后第三方库函数的实际案例。

4.1 混淆工具识别

该部分实验的目的是测试对目标应用使用的混淆工具的识别是否准确。实验的目标应用来源于 LibLoom[17] 的混淆应用数据集。从该数据集包含的 4 个混淆工具类别中分别随机选出 25 个安装包，共 100 个安装包来进行识别。在 LibLoom 数据集中，每个安装包都有标签表明其使用哪个混淆工具。因为 R8 使用 Proguard 的混淆规则，且两者的差别通常在于编译结果上，而在混淆结果上的差异不大，所以该数据集并没有对两者进行区分。实验结果表明，这 100 个安装包均能被正确识别出其使用的混淆工具，如表 5 所列。

表 5 混淆工具识别结果

Table 5 Obfuscator identification results

使用的混淆工具	应用的个数	正确区分的个数	准确率/%
Allatori	25	25	100
Dasho	25	25	100
Obfuscapk	25	25	100
ProGuard/R8	25	25	100
总计	100	100	100

4.2 混淆参数识别

该部分实验的目的是检测方案中对混淆参数识别的准确度。实验的目标应用集合为 4.1 节中的 100 个安装包。数据集中将每种工具混淆后的应用根据混淆形式进行分类，每类使用的混淆参数都有较大差异。上述安装包覆盖了全部类别。

实验中，R8 遍历参数空间产生的不同参数组合的混淆后的实验应用为 348 种，Allatori 为 263 种，Dasho 为 303 种。Obfuscapk 的各个功能是否开启较为明显，例如控制流混淆、字符串混淆和重命名都有较明显的特征，这些特征在混淆工具识别阶段即可被识别出，因此 Obfuscapk 用于匹配的实验应用参数组合较少，为 32 种。

对于目标应用集合中的每一个安装包，我们将其与所用混淆工具对应的实验应用集合中的每一个安装包依次进行相似度匹配，根据实验应用集合中与目标应用混淆参数相同的安装包在相似度排名中的位置来评价实验结果。目标应用的正确混淆参数可以通过其源代码中的混淆配置和 LibLoom 数据集标签获得。

我们称一个目标应用的混淆参数能被“Top n 正确识别”，即实验应用集合中与该目标应用使用相同混淆参数的安装包相似度排名在前 n 位。实验结果如表 6 所列，其中第 4－6 列展示了能被“Top n 正确识别”的目标应用数量，n 分别取值 1，2，5。结果以排名前五的安装包是否有正确参数作为识别成功的判断条件。

表 6 R8 混淆参数识别结果

Table 6 Parameters identification results of R8

混淆工具	参数类别	数量	Top n 正确识别的目标应用数量			正确率/%
			1	2	5	
Proguard/R8	Basic	9	8	9	9	92
	Flatten	8	5	8	8	
	Repackage	8	1	4	6	
Allatori	Strong	12	3	3	3	64
	Weak	13	10	11	13	
Dasho	Flatten	13	8	12	12	96
	Repackage	12	8	12	12	
Obfuscapk	All	8	6	8	8	96
	Others	17	15	16	16	
总计		100	64	83	87	87

该部分实验中，以相似度排名前五的安装包中存在目标应用的参数组合为判断条件，则参数识别的成功率为 87%，其中正确参数的安装包与目标应用相似度最高的正确率为 64%。在 Allatori 的实验中，对控制流混淆和字符串混淆相关参数的识别准确率较高，但重打包的两种模式下代码逻辑的差异较小，因此识别准确率较差。Obfuscapk 的各个参数对于混淆结果的代码结构影响较大，不同参数之间的差别较大，更容易区分。

4.3 函数定位

本文选择了实际应用市场中的一些应用作为目标应用，来测试能否正确定位到要找的第三方库函数以及能否成功处理“其他”类别的混淆工具。选取的目标应用有 WorkPlace，Twitter 和 YouTube。用于匹配的第三方库包括 Litho，Constraintlayout，Swiperefreshlayout，Collection 和 OkHttp3。

函数匹配的结果中，我们取相似度最高的前 10 个函数

进一步进行人工分析以确定结果，然后进行重打包实验来检验结果。重打包实验通过修改应用代码来分析对应功能。例如，在 Litho 的聊天界面，消息增加、删除和修改功能对应的函数的开头插入返回语句使函数直接结束。进行过该修改并重打包后的 WorkPlace 应用对应的功能失效，则表示定位出的函数确实是对应第三方库中感兴趣的函数。

对于每个第三方库，选取的感兴趣函数都是与其实现功能相关的函数。例如，在实现 UI 的 Litho 中我们选取了与消息增加、删除、修改等操作相关的函数；在实现网络通讯的 Okhttp 中我们选取了构建数据包以及发送、接收相关的函数。由于 Litho 的实现相对复杂、代码量较大，因此我们在其中选择了 10 个感兴趣函数；其他第三方库则根据自身以及所处应用的代码量选择了较少数量的感兴趣函数。

对于目标应用中存在的感兴趣函数，本文方法均可有效地定位，结果如表 7 所列。在定位实验中我们发现个别函数有在第三方库代码中存在，但是在目标应用中没有定位到的情况。经过人工分析，这部分代码在目标应用中并不存在，这说明了目标应用在混淆时使用了代码收缩，而我们对该应用进行混淆参数识别时也正确地识别出了该参数。因为我们要定位的是目标应用调用的第三方库函数，所以该类函数不计入定位实验的结果中。

表 7 函数定位实验结果

Table 7 Results of function location

实验应用	实验第三方库	是否混淆	成功定位	定位总数	成功率/%
WorkPlace	Litho	是	10	10	100
	Okhttp	是	5	5	100
Twitter	Constraintlayout	是	5	5	100
	Swiperefreshlayout	是	5	5	100
	Collection	是	5	5	100
YouTube	Collection	是	4	4	100
	Swiperefreshlayout	是	2	2	100
	Constraintlayout	是	4	4	100
总计			40	40	100

同时，为了测试该方法对未覆盖混淆工具的效果，我们设置了测试“其他”混淆工具的实验。该部分实验的目的是测试当目标应用使用的混淆工具不在本文详细分析的 5 个混淆工具之中，即为“其他”类别时，函数定位的情况。

我们使用 Android 版的 DexProtector 对上述应用的安装包进行混淆，然后使用对“其他”类别混淆工具的处理方式对这些混淆后的应用进行参数识别。识别结果中显示都没有进行控制流和字符串混淆，以及与 ProGuard 的默认混淆参数接近。使用识别出的混淆参数对第三方库进行混淆后，分别与对应的目标应用进行函数定位实验。

因为使用 DexProtector 处理过的安装包无法正常运行，所以不能通过运行效果来检查定位是否正确，只能通过人工分析来确定定位到的函数是否是目标函数。经过与第三方库源代码以及前一部分实验中定位到的函数进行对比，我们发现在之前成功定位的函数在本次实验中均可以定位成功。

4.4 案例研究：WorkPlace 重打包

在对一款国外办公软件 Workplace 进行安全功能增强时，为了添加对聊天界面的恶意链接的检测和删除功能，需要对其进行重打包。我们要找到 Workplace 中对聊天界面上的消息进行修改和删除的函数，然后在注入的代码中调用这些函数来实现修改和删除包含恶意链接的消息。

我们了解到该应用使用开源的框架 Litho 来实现聊天界面的 UI 以及各种相关功能，通过分析 Litho 的源代码以及相关文档，我们找到了框架中实现消息修改和删除功能对应的函数，但是 WorkPlace 的代码进行了混淆处理，无法直接定位到第三方库函数在应用代码中对应的位置。

首先对 WorkPlace 使用的混淆工具进行识别：WorkPlace 并没有字符串混淆、控制流混淆等混淆功能的特征；其命名方式不符合 Obfuscapk 的命名特征；对象类型声明为父类的调用的占比超过 50%；因此识别出混淆工具为 R8。然后得出其混淆参数与“proguard-android-optimize”这一默认配置最相似。接着对 Litho 源代码进行插桩并放入实验应用中进行混淆。最后，将实验应用和 WorkPlace 的安装包反编译成 Smali 代码并进行函数匹配。定位结果中，目标函数均能正确定位，其中对消息进行分类处理的函数 *dispatchLastEvent* 在混淆前后的对比如图 2 所示。

结束语 本文提出并实现了一种在经过混淆的目标应用中定位第三方库函数的方法，核心思想是通过识别目标应用的混淆方式（混淆工具和混淆参数），对第三方库进行混淆对齐，从而可以在同等或类似的混淆层面实施函数定位。本文方法对混淆工具和混淆参数的识别正确率分别能达到 100% 和 87%，对实际闭源应用的函数定位成功率达 100%。实验表明，本文方法能有效支持基于重打包的应用安全加固和分析。在未来，我们拟针对定制化的专用付费混淆工具，进一步扩展提升方法的混淆方式识别能力，使得本文方法能够用于定制化的混淆应用。同时，考虑到可能会出现需要对混淆应用中的一处混淆代码定位其对应的第三方库函数的情况，该情况为本文讨论的问题的反方向工作，初步设想的解决方案是将应用中混淆代码的操作码与第三方库的全部函数的操作码进行相似度比较，通过相似度确定该函数在混淆之前对应第三方库中的哪一个函数。

参 考 文 献

[1] DONG S, LI M, DIAO W, et al. Understanding Android obfuscation techniques: A large-scale investigation in the wild[C]// International Conference on Security and Privacy in Communication Systems. Cham: Springer, 2018: 172-192.

[2] YOU G, KIM G, CHO S, et al. A Comparative Study on Optimization, Obfuscation, and Deobfuscation tools in Android[J]. Journal of Internet Services and Information Security, 2021, 11(1): 2-15.

[3] AONZO S, GEORGIU G C, VERDERAME L, et al. Obfuscapk: An open-source black-box obfuscation tool for Android apps[J]. SoftwareX, 2020, 11: 100403.

[4] BALACHANDRAN V, TAN D J J, THING V L L. Control flow obfuscation for Android applications[J]. Computers & Security, 2016, 61: 72-93.

[5] KOVACGEVA A. Efficient code obfuscation for Android[C]// International Conference on Advances in Information Technolo-

gy. Cham:Springer,2013:104-119.
[6] GUO R,LIU Q,ZHANG M,et al. A Survey of Obfuscation and Deobfuscation Techniques in Android Code Protection[C]// 2022 7th IEEE International Conference on Data Science in Cyberspace(DSC). IEEE,2022:40-47.
[7] YOU G,KIM G,PARK J,et al. Reversing obfuscated control flow structures in android apps using redex optimizer[C]// The 9th International Conference on Smart Media and Applications. 2020:272-276.
[8] YOU G,KIM G,HAN S,et al. Deoptfuscator:Defeating Advanced Control-flow Obfuscation Using Android Runtime (ART)[J]. IEEE Access,2022,10:61426-61440.
[9] WERMKE D,HUAMAN N,ACAR Y,et al. A large scale investigation of obfuscation use in google play[C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018:222-235.
[10] ZHANG X,BREITINGER F,LUECHINGER E,et al. Android application forensics:A survey of obfuscation,obfuscation detection and deobfuscation techniques and their impact on investigations[J]. Forensic Science International:Digital Investigation,2021,39:301285.
[11] MAIORCA D,ARIU D,CORONA I,et al. Stealth attacks:An extended insight into the obfuscation effects on android malware [J]. Computers & Security,2015,51:16-31.
[12] WANG Y,ROUNTEV A. Who changed you? Obfuscator identification for Android[C]// 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems(MOBILESoft). IEEE,2017:154-164.
[13] BICHSEL B,RAYCHEV V,TSANKOV P,et al. Statistical deobfuscation of android applications[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016:343-355.
[14] MIRZAEI O,DE FUENTES J M,TAPIADOR J,et al. AndrODet:An adaptive Android obfuscation detector[J]. Future Generation Computer Systems,2019,90:240-261.
[15] HUANG J,XUE B,JIANG J,et al. Scalably Detecting Third-Party Android Libraries With Two-Stage Bloom Filtering[J]. IEEE Transactions on Software Engineering,2023,49(4):2272-2284.
[16] ZHANG J,BERESFORD A R,KOLLMANN S A. Libid:reliable identification of obfuscated third-party android libraries [C]// Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2019:55-65.
[17] WANG Y,WU H,ZHANG H,et al. Orlis:Obfuscation-resilient library detection for Android[C]// 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems(MOBILESoft). IEEE,2018:13-23.
[18] ELSERSY W F,FEIZOLLAH A,ANUAR N B. The rise of obfuscated Android malware and impacts on detection methods [J]. PeerJ Computer Science,2022,8:e907.
[19] GRAUX P,LALANDE J F,TONG V V T. Obfuscated android application development[C]// Proceedings of the Third Central European Cybersecurity Conference. 2019:1-6.
[20] BAUMANN R,PROTSENKO M,MULLER T. Anti-proguard:Towards automated deobfuscation of android apps[C]// Proceedings of the 4th Workshop on Security in Highly Connected IT Systems. 2017:7-12.
[21] ZHANG Y,DAI J,ZHANG X,et al. Detecting third-party libraries in android applications with high precision and recall [C]// 2018 IEEE 25th International Conference on Software Analysis,Evolution and Reengineering(SANER). IEEE,2018:141-152.
[22] MA Z,WANG H,GUO Y,et al. Libradar:fast and accurate detection of third-party libraries in android apps[C]// Proceedings of the 38th International Conference on Software Engineering Companion. 2016:653-656.
[23] BACKES M,BUGUEL S,DERR E. Reliable third-party library detection in android and its security applications[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016:356-367.
[24] JUNG J H,KIM J Y,LEE H C,et al. Repackaging attack on Android banking applications and its countermeasures[J]. Wireless Personal Communications,2013,73(4):1421-1437.
[25] LEE Y,WOO S,LEE J,et al. Enhanced Android app-repackaging attack on in-vehicle network[J]. Wireless Communications and Mobile Computing,2019,2019:1-13.
[26] MA H,LI S,GAO D,et al. Active warden attack:On the(in) effectiveness of Android app repackage-proofing[J]. IEEE Transactions on Dependable and Secure Computing,2021,19(5):3508-3520.
[27] LI Y X,LIN B G. Design of application security policy reinforcement system based on Android repackaging[J]. Netinfo Security,2014(1):5.
[28] SALEM A,PAULUS F F,PRETSCHNER A. Repackman:A tool for automatic repackaging of android apps[C]// Proceedings of the 1st International Workshop on Advances in Mobile App Analysis. 2018:25-28.

YUAN Jiangfeng, born in 1998, postgraduate, is a member of China Computer Federation. His main research interests include mobile security and program analysis.

LIANG Bin, born in 1973, Ph.D, professor, is a member of China Computer Federation. His main research interests include software analysis, AI security and mobile security.

(责任编辑:何杨)