计算机科学
COMPUTER SCIENCE

基于交叉指纹分析的公共组件库特征提取方法

郭威，武泽慧，吴茜琼，李锡星

引用本文

郭威，武泽慧，吴茜琼，李锡星.基于交叉指纹分析的公共组件库特征提取方法[J]．计算机科学, 2023, 50(1): 373-379.

GUO Wei, WU Zehui, WU Qianqiong, LI Xixing. Feature Extraction Method for Public Component Libraries Based on Cross-fingerprint Analysis [J]. Computer Science, 2023, 50(1): 373-379.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于类间和类内密度的多视角距离度量学习

Multi-view Distance Metric Learning with Inter-class and Intra-class Density

计算机科学, 2022, 49(11A): 211000131-6. https://doi.org/10.11896/jsjkx.211000131

一种基于实时代码装卸载的代码重用攻击防御方法

Defense Method Against Code Reuse Attack Based on Real-time Code Loading and Unloading

计算机科学, 2022, 49(10): 279-284. https://doi.org/10.11896/jsjkx.220500091

基于神经网络的二进制函数相似性检测技术

Neural Network-based Binary Function Similarity Detection

计算机科学, 2021, 48(10): 286-293. https://doi.org/10.11896/jsjkx.200900185

二进制代码相似性检测技术综述

Summary of Binary Code Similarity Detection Techniques

计算机科学, 2021, 48(5): 1-8. https://doi.org/10.11896/jsjkx.200400085

动态数据流分析的在线超限学习算法综述

Survey of Online Sequential Extreme Learning Algorithms for Dynamic Data Stream Analysis

计算机科学, 2019, 46(4): 1-7. https://doi.org/10.11896/j.issn.1002-137X.2019.04.001

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211100121

基于交叉指纹分析的公共组件库特征提取方法

郭 威 武泽慧 吴茜琼 李锡星
信息工程大学网络空间安全学院 郑州 450001
数学工程与先进计算国家重点实验室 郑州 450001
(1037057802@qq.com)

摘 要 软件公共组件库的广泛使用在提升了软件开发效率的同时,也扩大了软件的攻击面。存在于公共组件库中的漏洞会广泛分布在使用了该库文件的软件中,并且由于兼容性、稳定性以及开发延迟等问题,使得该类漏洞的修复难度大,修补周期长。软件成分分析是解决该类问题的重要手段,但是受限于特征选择有效程度不高和公共组件库的精准特征提取困难的问题,成分分析的准确度不高,普遍停留在种类定位水平。文中提出了一种基于交叉指纹分析的公共组件库特征提取方法,基于GitHub平台25000个开源项目构建指纹库,提出利用源码字符串角色分类、导出函数指纹分析、二进制编译指纹分析等方式来提取组件库的交叉指纹,实现了公共组件库的精准定位,开发了原型工具LVRecognizer,对516个真实软件进行了测试和评估,精确率达到94.74%。

关键词:软件成分分析;组件识别;动态链接库;版本识别
中图法分类号 TP311

Feature Extraction Method for Public Component Libraries Based on Cross-fingerprint Analysis

GUO Wei,WU Zehui,WU Qianqiong and LI Xixing
School of Cyberspace Security,University of Information Engineering,Zhengzhou 450001,China
State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract The widespread use of software public component libraries increases the speed of software development while expanding the attack surface of software. Vulnerabilities that exist in public component libraries are widely distributed in software that uses the library files,and the compatibility,stability,and development delays make it difficult to fix such vulnerabilities and the patching period is long. Software component analysis is an important tool to solve such problems,but limited by the problem of ineffective feature selection and difficulties in extracting accurate features from public component libraries,the accuracy of component analysis is not high and generally stays at the level of kind location. In this paper,we propose a public component library feature extraction method based on cross-fingerprint analysis,build a fingerprint library based on 25000 open source projects on GitHub platform,propose source string role classification,export function fingerprint analysis,binary compilation fingerprint analysis,etc. to extract cross-fingerprints of component libraries,realize the accurate localization of public component libraries,develop a prototype tool LVRecognizer,test and evaluate 516 real softwares,and obtain a accuracy rate of 94.74%.

Keywords Software component analysis,Component identification,Dynamically linked library,Version identification

1 背景介绍

公共组件库为开发者提供了各种功能的API,直接调用功能API可以大幅度地降低开发的难度和复杂程度。因此,开发人员经常在软件开发中使用公共组件库。公共组件库在提高开发效率的同时也扩大了软件的受攻击面,带来了一系列安全隐患,如潜在的漏洞等。根据《2020年中国开源生态安全概况分析》描述,近年开源组件生态中的漏洞数量一直呈现上涨趋势。2020年开源组件的漏洞数量为3426,较2019年增加了981个,同比增长40.12%。除了数量的持续上涨,漏洞的危害性也越来越大,例如不当使用开源组件导致的用户隐私泄露问题、组件延迟更新造成的漏洞遗留问题以及组件使用不规范而造成的开源许可证冲突的法律问题等。Windows 10上的编号为CVE-2020-1362的高危漏洞,就是由于WalletService在处理CustomProperty对象的过程中出现了越界读写,导致攻击者可以获得管理员权限,WalletService服务由动态链接库WalletService.dll提供。Grace等[1]确定了100个具有代表性的内置广告库,评估了Android官方应用市场的10万款应用,结果显示即使在一些部署广泛的广告库中,也存在对用户安全和隐私的威胁。这些威胁范围包括

到稿日期:2021-11-11 返修日期:2022-06-21
基金项目:国家重点研发计划(2019QY0501)
This work was supported by the National Key Research and Development Project(2019QY0501).
通信作者:武泽慧(wuzehui2010@foxmail.com)

收集用户信息到允许具有未知来源的第三方代码在托管应用程序中的执行等。

在当前 PC 端的公共组件库识别工作中，较为先进的工具有清华大学的 BCFinder[2] 和 OSLDtector[3] 两个工具。其中，BCFinder 使用源码作为输入，通过提取字符串常量作为特征构建特征库，并以字符串的出现频率为依据进行相似性计算，强调唯一字符串的作用。而 OSLDtector 使用 pkgs.org 中的安装包作为输入，同样提取字符串常量作为匹配依据，通过使用 TF-IDF 算法计算权重，并且通过创建内部克隆森林着重解决内部代码克隆的问题。

两个工具都着重解决公共组件库种类识别的问题，未进一步探索同类库的版本识别问题，但仅仅识别版本是不够的，要想精确地定位公共组件库中的漏洞，还需要确定公共组件库的版本。本文在原有的基础上提出了新的识别特征，并且对字符串常量特征进行了分类，赋予其不同的权重，这些特征有助于区分相近版本之间的组件库；同时，设计了一套更加精确的库版本识别算法，开发了原型工具 LVRecognizer 进行测试实验，实验结果显示 LVRecognizer 的精确率达到了 94.74%，召回率达到了 91.84%。

当前二进制公共组件库版本检测面临着以下两方面的挑战：1）没有找到适用于版本匹配的特征，无法提升库版本识别的精度；2）库识别算法可扩展性差，数据量增大会严重影响检测的效率。

精确的版本识别对于研究漏洞定位和版本更新等非常重要，而识别的精度很大程度依赖于特征数据库的完整程度和所选特征能够识别库之间差异的能力，仅使用字符串频率作为依据可以区分不同类别的库，但不能精确地区分同类别的库，因此需要添加新的特征来扩大库版本之间的差异。

现有的匹配算法通常使用遍历指纹数据库的方法，来计算目标库与数据库中库的相似性，然后输出匹配程度最高的结果，但这也导致了匹配时间会随着数据量的增加而快速增加，可扩展性差，不适合大批量数据的检测。

本文的贡献有如下 3 点：

(1)提出了能够有效识别库版本差异的交叉特征，并对其有效性进行了验证。

(2)设计了一套快速匹配交叉特征的算法，提高了匹配算法的可扩展性。

(3) 设计并实现了二进制组件版本识别的原型工具 LVRecognizer，并通过实验验证了其精确率达到 94.74%，召回率达到 91.84%。

本文第 2 节回顾了二进制组件库识别技术的相关工作；第 3 节介绍了系统的框架以及数据集构建及匹配过程；第 4 节评估了 LVRecognizer 与 BCFinder 在二进制组件库识别方面的性能；最后讨论了系统的局限性以及未来的改进方向。

2 相关工作

根据检测的精度和速度差异，库检测技术主要分为白名单、相似性检测、聚类、基于 hash 检测以及机器学习等。白名单检测技术使用文件名等单一的基础特征，因此检测速度非常快，但是对抗混淆能力很差。AdRisk 使用白名单技术探究了 Android 应用中内嵌的广告库带来的潜在的隐私和安全风险。相似性检测技术相比白名单使用了更加复杂的特征或者特征组，提升了检测精度和抗混淆能力，但检测速度有所下降。为了检测开源许可证冲突的问题，OSSpolice[4] 用常量字符串以及函数质心等特征对 Android 中的库进行了检测，BCFinder 同样使用字符串等特征主要对 PC 端的库进行检测。聚类技术由于不需要先导知识以及速度快等特点，常被用来识别解构后的 Android 模块中是否含有开源库。WuKong[5] 就是使用聚类技术，通过 Android 模块的 API 调用频率对其进行快速分类。LibRadar[6] 在 WuKong 的基础上使用基于哈希的表示法，从库中提取稳定的代码特征并将其转换为轻量级的散列格式，对应用进行多级聚类以识别潜在的候选者。此外，BinPro[7] 使用基于机器学习的方法，测试了不同特征在源代码到二进制识别过程中的重要性，该工具也是第一个源代码和对应二进制的测量工具。

现有的工具进行库识别所使用的特征也有差异，常见的是使用字符常量和数值常量等元特征。在 Android 中，由于打包方式的特殊性，文件层级结构[8]、类依赖结构[9] 以及类内部依赖[10] 均可用作库识别特征，同时 Android 中有很多对抗混淆的特征，如 LibDetect[11] 的 5 层抽象。但 PC 端研究程度较弱，因为可利用特征较 Android 平台少，且无法有效对抗平台编译优化等。此外，还有一些底层的特征，例如 LibDX[12] 引入了逻辑特征块[13] 作为匹配特征。

3 交叉指纹提取方法

本文提出了一个基于交叉特征的二进制组件识别的原型工具——LVRecognizer 系统，该系统可以对用户提供的二进制组件库进行精确的种类和版本识别。如图 1 所示，该系统可以分为源代码采集器、指纹提取器、存储模块以及组件识别器 4 个部分。其中，存储模块由代码指纹库和特征指纹库组成，组件识别器分为目标特征提取器和特征匹配器两个部分。下面介绍 LVRecognizer 所使用的交叉指纹提取方法以及相似度匹配算法。

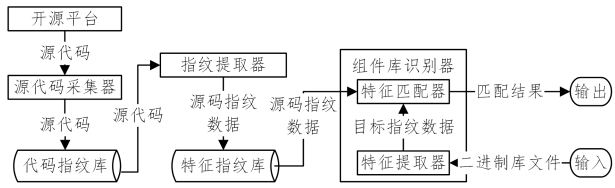


图 1 系统工作流程图

Fig. 1 System workflow diagram

3.1 源数据获取

本文着力于研究 PC 端的公共组件库，以开源动态链接库为例，动态链接库在 Windows 和 Linux 系统中分别以. dll 文件和. so 文件的固定形式存在。首先，实验调研了联想软件应用市场上下载量超过 50 000 的 516 款热门应用，收集了这些应用安装路径中的 41 806 个动态链接库文件，统计了其中各类库的使用频率，并且选择了使用频率最高的 30 种动态链接库作为研究的对象，并且由于 PC 端应用中大部分的动态链接库都是由 C/C++开发的，因此本实验中的测试样例选择 C/C++语言开发的动态链接库。

确定研究对象以后，本文使用 Python 开发了一个多线程源代码采集器，该模块收集 Github，GitLab 或者库官网上的 30 种目标库的所有历史版本信息，并通过官方提供的接口（如 GraphQL API）对库源码进行下载，并将下载的源代码保存在代码指纹数据库中。

LVRecognizer 的识别形式是源代码到二进制，即其目标是从动态链接库的源代码中提取多维特征来构建库的指纹特征库，然后使用该库的指纹特征匹配给定二进制库文件的特征，进而识别给定库的类别和版本。基于此目标，动态链接库特征的选择应该同时考虑源代码形式和二进制形式，以及在编译过程中的变化情况。

3.2　特征的选择

首先是库源码中的特征，源代码包含一个程序所有的母信息，包括常见的代码特征，如基础的常量数据、变量名、函数名、函数参数以及代码行本身等元特征，还有抽象语法树（Abstract Syntax Tree，AST）、控制流图（Control Flow Graph，CFG）、程序依赖图（Program Dependence Graph，PDG）、函数调用图（Function Call Graph，FCG）、程序的代码块以及上下文特性等复合特征。编译生成的二进制代码中会不同程度地保留源代码的特征，并且还会生成一些二进制代码独有的特征，包括寄存器、指令结构以及指令序列等。

$$S(f_1, f_2\cdots)\overset{c\&.o}{\Rightarrow}B(f_1', f_2', \cdots, nf_1, nf_2\cdots)$$

如上文所述，在编译过程中，源代码特征会有不同程度的变异或者磨损，并且可能会产生新的特征，但是这些信息受编译优化、混淆程度以及平台等环境的不同而造成较大差异。上述公式中，fn 指源代码中不同类型的特征，f_n' 指经过磨损或者变异的特征，nfn 指编译后产生的新特征。因此，为了更好地探究 LVRecognizer 的识别形式，需要寻找在编译过程中受影响较小的特征组。

2017 年，Miyani 等[7]使用机器学习探究在优化程度为 o2 且排除内联函数情况下，测试了字符串常量、整型常量、FCG、静态函数、虚拟函数等 12 个特征向量特征的重要性，结果显示权重占比最高的是：FCG(2.929 3)、字符串常量(1.469)以及函数参数(0.929 6)。但是，在混淆和优化过程中常常会修改函数名、变量名以及函数参数名等，然后以 sub_〈函数地址〉、v〈编号〉以及 a〈编号〉的形式加以代替。这些特征无法作为有效特征，FCG 会受到较大的影响，不予考虑，因此，本文指定字符串常量为有效特征。Miyani 等研究的是通用性程序，但是基于动态链接库的特殊性，本文决定把导出函数列表也纳入有效特征的候选列表。本实验通过随机抽样调查 41 806 个真实环境的动态链接库中的，在研究对象范围内的 110 个动态链接库。通过提取二进制动态链接库和相应版本的源代码中的导出函数列表来进行对比，根据导出函数列表的保留情况来判断是否把导出函数列表纳入到有效特征组。

编译后导出函数保留程度的结果如图 2 所示，导出函数在编译过程中的平均存留程度都达到了 86.93%，因此可以将导出函数作为有效特征使用。最终本文决定使用字符串常量以及导出函数列表作为 LVRecognizer 识别的有效特征。

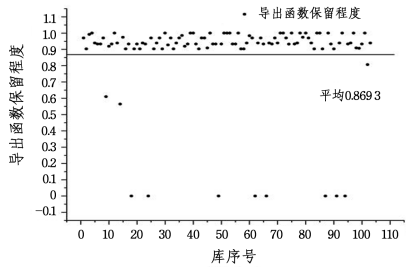


图 2　编译后导出函数保留程度统计图

Fig. 2　Export function retention statistics after compilation

3.3　特征的提取与清洗

由于开源库项目开发团队不同，开发代码风格差异巨大，使用同一方式对所有库进行特征提取显然不可取。因此，本文利用 LVRecognizer 构建指纹提取器，对不同类别的库定制化地提取其特征组。指纹提取主要的功能由代码成分分析和数据清洗两部分功能组成。代码成分分析工具通过对 C/C++源代码中的代码进行静态分析，识别代码中的不同组成成分，提取 LVRecognizer 所需要的特征组，提取出的特征经过数据清洗器进行处理，然后存放到指纹库中。指纹库构建算法伪代码如算法 1 所示。

算法 1　指纹库构建算法

```
输入：(INPUT0，INPUT1)
/* INPUT0 是指定的库名称列表 */
/* INPUT1 是 Github 网址 */
输出：(库指纹数据库)
1. /* VL=FPL=SU=SUK=SUL=DL=DV=[] */
2. /* CF=PF=SF=BF=DSF={}，均为存储指纹的字典 */
3. /* FP，U */
4. For LN in INPUT0
5.     VL ← GetLibVersion(LN)
6.     /* 从 GraphQL API 中获取库的所有版本名称 */
7.     FPL← GetLibFile(VL)
8.     /* 拼接版本文件 URL，并下载库源码到本地 */
9.     For FP in FPL
10.      CF← CommonExtract(FP)
11.      /* 通用性提取该源代码的分类指纹 */
12.      PF← PrivateExtract(FP)
13.      /* 针对不同种类和版本的库，补充提取 */
14.      SF← MergeFingerPrint(CF，PF)
15.      /* 对两次提取的指纹进行融合 */
16.      SaveIntoDatabase(SF，FP)
17.      /* 将源码和指纹都存储到数据库中 */
18.    End
19. End
20. SUK← GetSoftwareURL( INPUT1)
21. /* 获取 Github Stars>1500 项目的 URL */
22. For U in SUK
23.    SU← InstallAndGetSoftwarePath(U)
24.    /* 下载软件并获取安装路径 */
25.    SUL← GetSoftwareDLL(SU)
26.    /* 获取指定安装目录所有的 DLL */
27.    For DL in SUL
```

```
28.     DV← GetDLLVersion(DL)
29.     /* 获取 DLL 的版本 */
30.     BF← ExtractDLLFingerPrint(DL)
31.     /* 提取 DLL 的指纹 */
32.     DSF← ReadDatabase(DV)
33.     /* 从指纹库中提取对应版本的指纹数据 */
34.     SF← MergeFingerPrint(DSF,BF)
35.     SaveIntoDatabase(SF)
36.  End
```

3.3.1 代码成分分析

导出函数的提取方式主要有两种：关键字识别递归提取和特殊格式提取。关键字识别递归是指存在‘__declspec(dllexport)’等导出函数关键字的情况，一般通过宏定义指代，如zlib库中的ZEXPORT。这种情况可以使用类似预处理的方式加载所有可能存在关键字的文件，然后进行宏定义消除，找到库导出列表。特殊格式提取指不存在导出函数关键字，一般出现在库早期的历史版本中，这种情况下导出函数可能存在extern前缀或者存在于配置文件的固定区域，需要构造正则表达式进行提取。对于字符串的提取，选取非注释的且被引号包裹的常量字符作为提取目标，之后对提取的目标进行清洗。

3.3.2 数据清洗

初步提取的字符串常量存在很多冗余，匹配精度低，因此需要对其进行清洗，提高其有效程度。本实验对字符串常量的清洗包括分类和精简两部分。

如图3所示，不同字符串的角色以及使用频率对库版本识别的有效程度不同。字符串的有效程度与其出现频率以及磨损程度呈负相关，与其角色重要程度呈正相关。有效程度最高的版权信息，因为其出现频率低且指向性明确，可以直接判定开源库的种类甚至是版本；调试信息的构成主体是错误或者提示信息，该信息在同类相近版本中相似性很高，但其磨损程度很小。

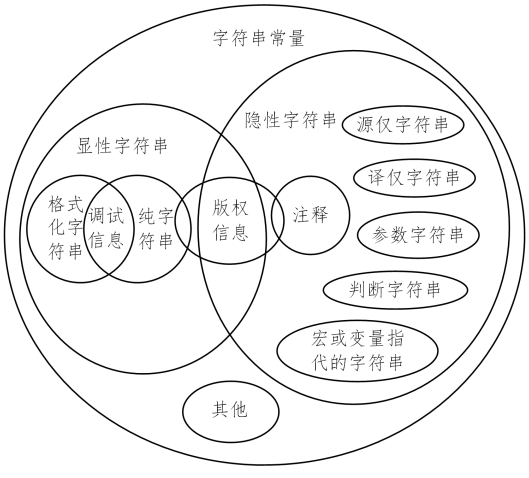


图3 源代码字符串分类图

Fig. 3 Source string classification diagram

本实验中还统计了调试信息在编译后的保留程度，过程同导出函数的统计实验相同。调试信息在编译后保留程度的结果如图4所示，平均保留程度为70.53%，具备相当的有效性，因此纳入特征提取的范畴；还有一些种类的字符串，如注释或者源仅字符串等不参与编译的字符串类型，不具备判断有效性。

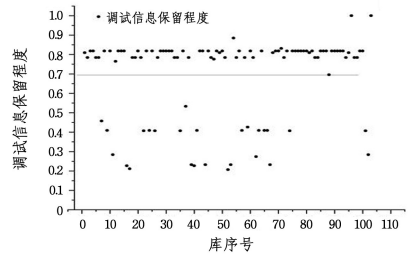


图4 编译后调试信息保留程度统计图

Fig. 4 Debugging information retention statistics after compilation

根据字符串角色的有效性，本文将字符串常量分为版权信息、调试信息以及普通信息3种类别。此外，通过实验调查发现，使用专业反汇编工具IDA对二进制库进行字符串提取后出现了比例不低的函数名。本实验对此做了一个统计，提取源代码中的函数列表以及二进制动态链接库中的字符串列表进行比较，统计其重合度，发现平均占比达到22.52%，结果如图5所示。因此，本文把源代码中提取的函数名归类为字符串信息，最终字符串常量被划分为版权信息、调试信息、函数名以及普通信息4种类别。

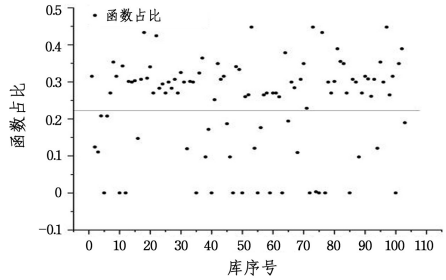


图5 源代码中的函数名在目标库中的字符串占比统计图

Fig. 5 Ratio of function names in source code to strings in target library

为了进一步增加字符串常量的有效程度，本文对分类后的字符串进行了精简，即删除不可能通过编译或者无效程度较高的元素，如源仅字符串等。根据实验对比源代码和二进制代码中的字符串，带有‘--’，‘.h’，‘.c’，‘.cpp’，‘〈filename:〉’的字符串不会通过编译器，带有制表符或者换行符等元素的字符串在二进制代码中会被再次转义或者拆分。最后本文删除了源代码和二进制代码中提取出的长度小于12的字符串。原因如下，本实验统计了数据集中的源代码和二进制代码中不同长度的字符串占比。

字符串长度梯度占比如图6所示，长度较小的字符串均含有较高的占比。

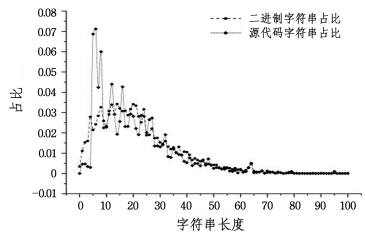


图6 字符串长度梯度占比图

Fig. 6 String length gradient ratio diagram

除此以外，通过实验分析，长度小于 12 的字符串存在较大的冗余，并且容易在不同版本的库之间重复出现，因此有效程度较低。为了验证其有效程度，本实验通过删除不同长度的字符串来测量源代码和二进制代码中的字符串的重合程度。其结果如图 7 所示，在删除长度小于 12 的字符时，代码的重合度达到了最高值，因此本文设定删除短小字符串的值为 12。

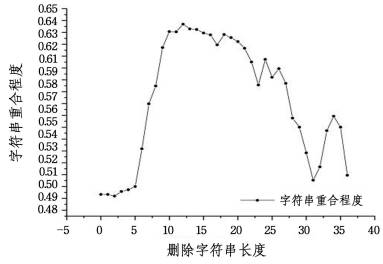


图 7　删除定长字符串后源码和目标库字符串重合度统计图

Fig. 7　Statistics of coincidence degree of source code and target library string after removing fixed length string

3.4　数据存储

该项目中存储模块主要存储两种主要的数据，即库源码及相关信息以及动态链接库特征指纹组。该系统采用“文件系统＋数据库”的方式存储源代码，库指纹使用 MySQL 数据库进行存储，数据结构如图 8 所示。

在该系统中，需要存储动态链接库项目的各项信息，为了优化匹配查找的速度，首先需要把待存储的信息分为项目源码信息、常用匹配信息以及后备存储信息 3 种。项目源码信息包括项目的 ID、名称、版本、本地地址、平台地址、项目类别等信息。其中前三项信息与常用匹配信息的前三项对应，常用匹配信息表中字符串 MD5 列表以及字符串类型列表均来自于后备存储信息表中。这种存储结构根据信息的性质和使用频率进行分类，可以在存储大量信息的前提下保持较高的查询速度。

值得注意的是，根据编译过程的损耗程度及其有效程度对字符串类别赋予权重，损耗程度越低，有效程度越高，所占权重就越大。因此，在后备存储信息表中记录所有测试库的字符串，会出现不同种类的库中出现同一字符串且类型不同的情形。为了解决该问题，在构建指纹库时，如果遇到新加入的字符串类型与库中类型不一致的情况，则把低权重字符串修改为高权重字符串的类型。

Lib_id	lib_1
Lib_name	lib_name_1
lib_version	lib_version_1
local_path	.../lib_name_1/lib_version_1
url	https://github.com/.../lib_name_1/lib_version_1/...
etc	...

Lib_id	lib_1
Lib_name	lib_name_1
lib_version	lib_version_1
export_function_list	[ef1,ef2,ef3, ...]
string_md5_list	[sm1,sm2,sm3, ...]
string_type_list	[st1,st2,st3, ...]

string_md5	str_md5_1
string_self	str_1
string_type	str_st_1
string_length	st1_len
string_number	st1_num
etc	...

图 8　数据库结构图

Fig. 8　Database structure diagram

3.5　逆向增强

LVRecognizer 下载了开源平台 Github 上 Start 数量大于 1 500 的项目，然后提取其中的 Windows 版本系列进行下载和安装，并获取其安装路径中的所有动态链接库，使用版本识别器识别动态链接库的版本，然后提取导出函数列表以及字符串数据，从特征指纹库中选取对应版本的库指纹数据与现有的数据进行融合，将融合后的数据存放到特征指纹库中。使用真实环境中的指纹数据对特征指纹进行增强，使其减小编译对指纹特征的影响，扩大版本之间的差异。

3.6　目标文件的特征提取

LVRecognizer 的用户输入是二进制形式的动态链接库文件，该环节使用专业反汇编工具集 IDA 对二进制文件进行分析，提取文件数据段的字符串常量以及导出函数列表，然后对这些特征进行与源代码中同样的精简操作，不同的是直接从二进制文件中提取的字符串常量无法直接做分类，需要与数据库中的分类进行配比。

3.7　相似度匹配

本实验设计了一套匹配算法，用于计算给定二进制库与指纹数据库中的候选库之间的相似度。为了提高 LVRecognizer 匹配的速度、精度以及可扩展性等性能，库匹配过程被分为粗细两种粒度。使用粗粒度特征筛选候选库，然后使用细粒度特征确定精确的匹配结果。

在粗粒度阶段，LVRecognizer 使用导出函数列表作为筛选特征。由于导出函数列表在不同种类的库之间几乎没有重叠，在同类库相近版本之间相差较小，因此使用导出函数列表作为粗粒度阶段的匹配特征，可以快速排除不同种类的库以及同类库的差异较大的版本。匹配规则如下：

$$M(EFPools, EF_{(target)}) = Candidate\_class \tag{1}$$

$$EF\_Similarity = \frac{EF_{(Candidate\_class\_n)} \cap EF_{(target)}}{EF_{(Candidate\_class\_n)}} * 100\% \tag{2}$$

其中，$EFPools$ 指 LVRecognizer 构建的以类为单位的导出函数池，$EF(target)$ 表示目标库中的导出函数列表，$EF(Candidate\_class\_n)$ 表示候选指纹库中第 n 个库的导出函数列表，$EF\_Similarity$ 为目标库中的导出函数列表与指纹库中第 n 个库的导出函数列表的相似度。粗粒度阶段分为两个步骤，第一步通过导出函数池，筛选出目标库的候选库类，第二步对目标库与候选库的导出函数列表进行相似度匹配，对该相似度设定阈值，其匹配结果超过阈值的匹配指定为最终候选库。

在细粒度阶段，导出函数的作用并不明显，因此 LVRecognizer 使用精度更高的字符串特征作为匹配特征，对于 3.3.2 节所述的不同类别的字符串，根据其在确定版本方面的有效程度以及编译过程中的损耗程度，给予其相应的权重，其权重的计算式为：

$$ST\_Weight = ST\_Effective * STC\_Retention \tag{3}$$

其中，$ST\_Effective$ 为字符串类别在判断具体版本时的有效程度，$STC\_Retention$ 为字符串类别在编译过程中的保留

程度，ST_Weight 为字符串类别的权重。

并且计算过程中所使用的字符串为目标库中的字符串与指纹库中的字符串列表的交集，即：

$$String\_Pool = SL_{(database_n)} \cap SL_{(target)} \tag{4}$$

其中，$String\_Pool$ 为重合字符串的集合，因此 $String\_Pool$ 中的字符串的权重之和为：

$$SUM_{(String\_Pool)} = \sum_{i}^{len(String\_pool)} (ST_{Weight(i)} * TF\text{-}IDF(i)) \tag{5}$$

其中，TF-IDF(i)为 Zhang 等在 OSLDetector[3] 工具中所使用的 TF-IDF 公式的改版，即其作用域范围缩小为由候选库组成的库字符串的集合。最终的相似度为交集字符串的权重与指纹库中的库字符串的权重的比值，其计算式如下：

$$Lib\_Similarity = \frac{SUM_{String\_Pool}}{SUM_{database\_n}} \times 100\% \tag{6}$$

通过计算目标库与候选库中所有库的特征的相似度 $Lib\_Similarity$，把相似度结果最高的候选库作为结果输出。

4 评估

本节设计了实验，评估了库检测工具 LVRecognizer，并将实验结果以及其各方面表现与 BCFinder 进行了比较。

本实验的环境配置为 Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz 1.61GHz，16GB Memory 的私人电脑。实验特征指纹库数据源是由从 Github 平台下载的 libpng，libzip 等 30 种动态链接库的所有历史版本组成，测试集的数据源是联想应用商店中下载量超过 50000 的软件中的 516 款，收集软件中的动态链接库列表，统计其中出现频率最高的 5 类动态链接库作为测试对象。

4.1 LVRecognizer 精确性评估

本实验使用 LVRecognizer 工具针对所有的测试对象对动态链接库的种类和版本进行测试。测试过程如下：通过导出函数列表判断动态链接库的种类，然后使用动态链接库的分类字符串和函数列表进行精确的版本识别，实验结果如表 1 所列。

表 1 LVRecognizer 测试

Table 1 LVRecognizer test

（单位：%）

指标库名	libexpat	libpng	zlib	libssh2	libzip
TP	25.96	11.54	43.27	4.81	0.96
TN	0.96	0.96	1.92	0	0.96
FP	0	0	0.96	0	0
FN	2.88	0.96	2.88	0.96	0

表 1 列出了使用测试集对 LVRecognizer 系统的测试结果，其中 TP 和 TN 指判断正确的情况，FP 和 FN 指判断错误的情况，TP 和 FP 指判断结果与数据源结果统一的情况，TN 和 FN 指判断结果与数据源相否的情况。由于 libpng-1.6.38 以及 zlib-1.2.7 等版本在官网和 Github 上没有相应版本，判断的结果一定是错误的，因此使用 TN 指标来记录。同时也存在一些部分混淆的库，由于缺失严重，故认为其不能正确判定，但是其结果是正确的结果，用 FP 指代。

表 2 列出了 LVRecognizer 和 BCFinder 的对比结果，其中 BCFinder 的值是描述该工具识别动态链接库种类的测试结果，而 LVRecognizer 的值是该工具同时识别动态链接库的种类和版本的结果，即只有正确识别具体版本才能统计正确，因为 BCFinder 没有识别库版本的数据，因此此实验使用 LVRecognizer 的细粒度实验结果来与 BCFinder 的粗粒度实验结果做对比。从表中的数值对比可得，LVRecognizer 在精确率和召回率的表现方面均有较好的表现。LVRecognizer 的版本识别精确率为 94.74%，相比 BCFinder 的库种类识别精确率 91.4%提高了 3.34%；而召回率则突破了 90%，达到了 91.84%。因此由上述描述可知，LVRecognizer 使用的特征选择方法以及相似度识别算法可以有效提升动态链接库的版本识别的精度。

表 2 工具性能对比

Table 2 Tool performance comparison

（单位：%）

指标工具	BCFinder	LVRecognizer
精确率	91.4	94.74
召回率	88.9	91.84

4.2 LVRecognizer 测试速度评估

该实验设计了一个对比实验的测试环境，对比两个工具在数据量不断增长的情况下的匹配速度。测试集共有 41806 个测试数据，但只有 5279 个动态链接库包含于数据源中，不能很好地体现数据的匹配速度，因此该实验根据数据源中动态链接库的实际统计情况进行数据模拟。

统计源代码中特征组的比例，字符串的平均数量为 1483，导出函数为 195，其中版权信息为 2‰，调试信息为 35.27%，函数名为 42.87%，普通字符串为 21.51%。按照字符串 50%、导出函数 70%的聚拢程度，构建库类字符串池和导出函数池，设定每个库有 50 个历史版本，并在 1492 个库类中进行测试。并且本文根据 BCFinder 论文中阐述的主要原理，实现了 BCFinder 的模拟模型，图 9 中 BCFinder 的数据来源于 BCFinder 的模拟模型。该实验设定了数据库中逐步增加数据量的实验组，并在其中匹配相同的目标数据，记录实验结果。其结果如图 9 所示。

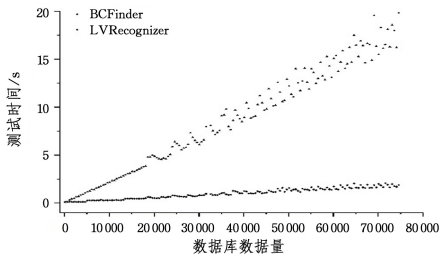


图 9 匹配速度对比图

Fig. 9 Matching speed comparison diagram

图 9 为模拟指纹数据库的数据量从 100 增长到 74600，增长单位为 500 时，BCFinder 和 LVRecognizer 检测一个目标动态链接库的时间变化图。位于上方的是 BCFinder 的时间图，下方的是 LVRecognizer 的散点图。通过对比可知，随着数据量的增长，BCFinder 和 LVRecognizer 的检测时间都会增加，但是 LVRecognizer 的增长率明显小于 BCFinder 的增长率。因此，可以看出 LVRecognizer

表现出了更好的可扩展性。

结束语　本文设计并实现了一款二进制动态链接库检测工具 LVRecognizer，用于检测 PC 端中二进制库的种类和版本，调研了联想应用商店中下载量超过 50000 的热门应用，并分析了其中所使用的 412806 个动态链接库，研究了其中使用频率最高的 30 种库作为测试样例，对工具的性能进行评估，实验结果表明 LVRecognizer 的精确率达到了 94.74%；并且提出了一套有效的、用于源代码到二进制形式检测的多维特征组，通过粗细两种粒度的匹配算法，同时保证了库检测的精确性和可扩展性，着重提高了对小版本识别的性能。

虽然 LVRecognizer 提高了二进制库版本检测的可扩展性、精确度等性能，但是仍然存在一些问题需要在下一步工作中解决：1）库种类的完整性不足，现有的数据集是根据实用性进行逆向构建数据集，但是随着检测范围的增大，需要库种类更加完整的数据库，这就需要收集更多的开源库；2）LVRecognizer 的特征来源于开源库源码中的.c，.h 以及.cpp 等类型的文件，对于版本之间的改动不在此类型文件的情况，例如译仅字符串（即仅在编译后出现的字符串，如源码中的字符串组合变形后生成的新字符串），本系统并不能很好地识别；3）对于混淆程度较高的二进制库版本，如混淆所有导出函数及字符串或者加密处理的库，并不能很好地检测。这些问题都需要在下一步工作中解决。

参 考 文 献

[1] GRACE M C，ZHOU W，JIANG X，et al. Unsafe exposure analysis of mobile in-app advertisements[C]// Proceedings of the fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks. 2012：101-112.

[2] TANG W，CHEN D，LUO P. Bcfinder：A lightweight and platform-independent tool to find third-party components in binaries [C]// 2018 25th Asia-Pacific Software Engineering Conference (APSEC). 2018：288-297.

[3] ZHANG D，LUO P，TANG W，et al. OSLDetector：identifying open-source libraries through binary analysis[C]// 2020 35th IEEE/ACM International Conference on Automated Software Engineering(ASE). 2020：1312-1315.

[4] DUAN R，BIJLANI A，XU M，et al. Identifying open-source license violation and 1-day security risk at large scale[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017：2169-2185.

[5] WANG H，GUO Y，MA Z，et al. Wukong：A scalable and accurate two-phase approach to android app clone detection[C]// Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015：71-82.

[6] 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.

[7] MIYANI D，ZHEN H，DAVID L. Binpro：A tool for binary source code provenance[J]. arXiv：1711.00830，2018.

[8] SOH C，TAN H B，ARNATOVICH Y L，et al. Libsift：automated detection of third-party libraries in android applications [C]// 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). 2016：41-48.

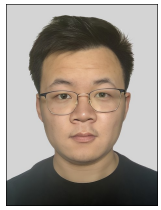
[9] ZHANG Z，DIAO W，HU C，et al. An empirical study of potentially malicious third-party libraries in Android apps[C]// Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks. 2020：144-154.

[10] 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). 2018：141-152.

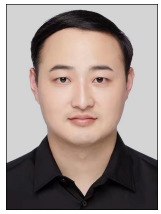
[11] GLANZ L，AMANN S，EICHBERG M，et al. CodeMatch：obfuscation won't conceal your repackaged app[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017：638-648.

[12] TANG W，LUO P，FU J，et al. LibDX：A Cross-Platform and Accurate System to Detect Third-Party Libraries in Binary Code [C]// 2020 IEEE 27th International Conference on Software Analysis，Evolution and Reengineering (SANER). 2020：104-115.

[13] FANG L，WU Z H，WEI Q. Summary of Binary Code Similarity Detection Techniques[J]. Computer Science，2021，48(5)：1-8.



GUO Wei，born in 1996，postgraduate. His main research interests include cyberspace security and software engineering.



WU Zehui，born in 1988，Ph. D. His main research interests include software security analysis and vulnerability analysis of cloud platform.

（责任编辑：喻藜）