计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200400085

# 二进制代码相似性检测技术综述

方 磊 武泽慧 魏 强
信息工程大学数学工程与先进计算国家重点实验室 郑州 450001
(nanbeiyouzi@qq.com)

**摘 要** 代码相似性检测常用于代码预测、知识产权保护和漏洞搜索等领域,可分为源代码相似性检测和二进制代码相似性检测。软件的源代码通常难以获得,因此针对二进制代码的相似性检测技术能够适用的场景更加广泛,学术界也先后提出了多种检测技术,文中对近年来该领域的研究进行了综述。首先总结代码相似性检测的基本流程和需要解决的难题(如跨编译器、跨编译器优化配置、跨指令架构检测);然后根据关注的代码信息的不同,将当前的二进制代码相似性检测技术分为4类,即基于文本的、基于属性度量的、基于程序逻辑的和基于语义的检测技术,并列举了部分代表性方法和工具(如Karta,discovRE,Genius,Gemini,SAFE等);最后根据发展脉络和最新研究成果,对该领域的发展方向进行了分析和论述。

**关键词**:软件安全;二进制程序;代码相似性检测
**中图法分类号** TP311

## Summary of Binary Code Similarity Detection Techniques

FANG Lei,WU Ze-hui and WEI Qiang
State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

**Abstract** Code similarity detection is commonly used in code prediction,intellectual property protection and vulnerability scan,etc. It includes source code similarity detection and binary code similarity detection. Since the source code is usually difficult to access,binary code similarity detection is more widely applicable,and a variety of detection techniques have been proposed in academia. We review researches of this field in recent years. First,we summarize the basic process of code similarity detection and challenges it faces,which include the cross-compiler,cross-optimization and cross-architecture detecting. Then,in consideration of different code information concerned,we propose to classify current binary code similarity detection techniques into 4 categories,including text-based,attribute-based measurement,program logic-based and semantic-based detection technologies,and list some representative methods and tools,such as Karta,discovRE,Genius,Gemini,SAFE,etc. Finally,according to the development context and the latest researches,we analyze and discuss the development direction of this field.

**Keywords** Software security,Binary program,Code similarity detection

## 1 研究背景及意义

代码相似性的概念源于软件分析技术,目前还没有标准和权威的定义。通常认为,如果一段代码是由另一段代码复制或经过一定规则变换而来的,则认为它们是相似的。代码相似性的研究对象分为源代码和二进制代码。二进制代码的相似性指由同一或相似的源代码编译得到的不同二进制代码是相似的。代码相似性检测通常应用于但不限于代码预测(即根据现有的代码预测可能的代码修改和更新,提供代码模板推荐)、知识产权保护(即发现代码中未经授权的代码的复用和克隆,协助软件所有者保护知识产权或规避潜在的侵权行为)、漏洞搜索(如脆弱代码搜索、软件漏洞追踪和定位)。代码相似性检测常被作为一种辅助分析技术,其由于在提升代码质量和提高代码分析效率方面效果显著,近年来逐渐成为学术界的研究热点。

在安全领域,开源和第三方代码的引入能够有效提高开发效率,同时降低开发成本,因此其在互联网应用程序、大数据与人工智能、工业控制与自动化等领域被广泛应用。但是,开源和第三方代码中包含的漏洞数量也呈现快速增长的趋势,随着开源代码使用的普及,这些存在漏洞的代码也被广泛用于各类项目中,从而导致同源漏洞的引入。美国新思科技公司(Synopsys,Nasdaq:SNPS)在其发布的《2020年开源安

到稿日期:2020-04-20 返修日期:2020-07-30 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发课题(2017YFB0803202);之江实验室"先进工业互联网安全平台"项目(2018FD0ZX01);河南省软科学研究计划项目(192102210128)
This work was supported by the National Key Research and Development Project(2017YFB0803202),Advanced Industrial Internet Security Platform Project(2018FD0ZX01) and Henan Soft Science Research Program Project(192102210128).
通信作者:魏强(prof_weiqiang@163.com)

全和风险分析报告》[1]中指出，在2019年审计的代码库中，99%的代码包含开源组件，其中75%包含了至少一个公开漏洞，该比例明显高于2018年的60%，几乎又回到了2017年的78%；每个代码库平均发现82个漏洞，其中高风险漏洞的百分比在2019年增至49%，高于2018年的40%。例如，2014年4月被曝光的OpenSSL“心脏滴血”漏洞就是因为多数网银、在线支付、电商、门户网站、电子邮件等重要服务商为实现加密通信而使用的OpenSSL开源软件包中存在安全漏洞，所以导致全球三分之二的网站受到了威胁。

通过代码相似性检测技术，安全人员能够对脆弱代码实现快速定位和追踪，从而评估漏洞带来的影响。然而，在实际应用中，出于对软件知识产权保护的考虑和其他原因，软件的源代码常常无法获得，甚至不可用，但是因为可执行的二进制代码是由源代码编译而来的，所以对源代码的修改通常在二进制代码中也能得到一定的体现。因此，针对二进制代码的相似性检测技术更具有通用性和现实意义。

本文第1节从介绍代码相似性检测的应用和近几年代码复用面临的安全形势入手，简要阐述了二进制代码相似性检测的研究背景和意义；第2节简要介绍了二进制代码相似性检测的基本流程，介绍了需要解决的主要难题；第3节根据关注的代码信息的不同，分别对基于文本、基于属性度量、基于程序逻辑和基于语义的检测技术的分类依据进行了详细阐述，然后列举了现有的技术和工具，以便于读者更直观地理解；第4节总结和分析了该领域的技术发展趋势；最后总结全文。

## 2 相关介绍

### 2.1 代码相似性检测流程

Whale[2]于1988年首次提出将代码相似性检测过程分为两个阶段：代码格式转换和相似度确定。后随很多检测方法都参考了该框架，具体的检测过程可被细分为4个部分：代码预处理、中间表示转换、比较单元生成和度量算法匹配[3]（见图1）。代码预处理的目的是剔除部分与代码相似性无关的或影响较小的信息，同时规范和统一输入格式以便于处理，其操作通常包括：统一代码布局、去除无关字符（如空行、注释）、替换名称（如自定义函数名、数据名和通用寄存器名）、程序切片等。中间表示转换的目的是在尽量不改变代码携带信息的前提下，提取感兴趣的信息（如字节流、字符串、可度量属性、控制流和数据流，以及代码的词法、语法和语义等），并将其转换成自动化程序可处理和比较的其他形式，该步骤一般借助于编译器、反编译器、程序分析器或自定义规则等进行转换。常见的中间表示形式有线性结构（字符串或序列）、树形结构（抽象语法树或XML文档等）、图形结构（控制流图或数据流图等）和其他非线性结构（多维特征向量或特征矩阵）。比较单元生成是对代码或其中间表示进行切分或聚合，得到便于相似性度量的最小单元。常见的比较单元有字符串、标识符（Token）序列、特征向量、树的结点及其子树、图的节点及其子图等，也可以是中间表示本身。比较单元还可以分为固定粒度和自由粒度。针对固定粒度，如果粒度太大，那么检测的准确度会降低；如果粒度太小，那么检测的工作量会增加。自由粒度的实现难度较大。因此，在实际操作中需要根据中间表示的形式和任务目标选择合适的检测粒度。度量算法匹配是根据不同形式的比较单元，采用与之相适应的相似性度量算法（如串搜索、向量距离计算、子树和子图的匹配、聚类等），度量检测对象之间的相似性。

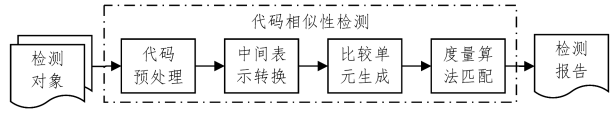


图1 代码相似性的检测过程

Fig. 1 Process of code similaritydetection

程序的源代码和二进制代码在形式上差别较大：源代码由高级语言编写，形式灵活多样，包含的程序信息更丰富，通常具有优秀的跨平台特性；二进制代码（通常指二进制程序的反汇编代码）较相应的源代码不仅损失了部分信息，还增加了编译器优化和指令架构等特性。虽然二者有诸多不同，但是其检测都遵从上述提到的代码相似性检测的架构，依据中间表示的不同，二者的相似性检测技术的分类不尽相同，本文重点关注二进制代码相似性检测技术。

### 2.2 二进制代码相似性检测的难题

二进制代码通常由源代码编译而来，源代码级的复制和修改也会部分体现在相应的二进制代码中。代码克隆（也称代码抄袭）和混淆手段虽然多应用于源代码的复制和修改，但也可以用于可执行二进制程序或组件的篡改或修补。已有较多文献对相关技术和手段进行了总结和归纳[3-6]，因此本文不再赘述。除此之外，同一源代码经过不同的编译器和优化配置，针对不同的硬件平台，编译得到的二进制代码不尽相同，因此二进制代码相似性检测会遇到其特有的难题，且该难题较源代码检测更为复杂。

在检测由同一源代码编译得到的二进制代码时，可能会遇到以下难题。

难题1 跨编译器问题。由于设计目的和采用的算法不同，不同的编译器产生的二进制代码不尽相同。例如，虽然寄存器的选择过程是通过各种启发方式和复杂的代码来传递驱动的，但在某些情况下，即使存在公共约定，编译器也不会完全遵守。这些都导致了寄存器选择的任意性。

难题2 跨编译优化配置问题。同一版本的编译器采用不同的编译优化配置所产生的二进制代码不尽相同。例如，可执行程序的调试版本相比发行版本会增加许多与调试相关的信息和调用。不同的优化等级，其区别在于增加或减少了对堆栈的检查和清理等操作，会导致最终的可执行文件的长度发生显著变化。再比如在x86-64架构下的GCC编译器，只有在-O0不优化的编译条件下，寄存器rbp才具有帧指针的特殊用途。

难题3 跨指令架构问题。不同指令架构的二进制代码不同。不同架构的指令集、寄存器、机器字长都有较大差异，从而导致相应的二进制代码差异巨大。

这些二进制代码来自同一源代码，因此具有相同的功能且语义等价，但其汇编指令和语法可能略有差异甚至截然不同。二进制代码相似性检测需要根据不同应用场景来屏蔽噪

声干扰，提取代码中需要关注的信息。

## 3　二进制代码相似性检测技术的分类

本文针对检测技术中关注的信息种类的不同，提出了基于文本、基于属性度量、基于程序逻辑和基于语义的4种检测技术分类。

### 3.1　基于文本的检测

基于文本的检测技术历史最悠久，实现过程最简单，其中某些方法对二进制代码只做简单的预处理。例如，一类方法首先模糊通用寄存器名和内存地址，然后提取指令的操作码和操作数，或者提取字符串以生成标识符序列，最后通过子序列匹配来判断不同代码是否相似，这类方法可以进一步被归纳为基于标识符的检测。Karta[7]作为著名的静态反编译程序IDAPro[8]的二进制程序匹配插件，首先利用唯一的数值常数和字符串等标识符来设置库的锚点函数，然后通过在二进制文件中查找锚点函数来缩小搜索范围，最后利用标识来进一步确定静态编译的开源库的确切版本并匹配函数符号，其也可以在这一步骤使用函数调用列表来匹配一组相似函数。DarunGrim2[9]是基于指纹哈希匹配的二进制文件比较工具，它忽略了汇编代码中的操作数，将程序基本块的指令序列的哈希值作为相似特征，此外它还以程序中符号名称的匹配作为辅助手段。另一类方法甚至不对二进制代码进行反汇编，直接比较程序的二进制字节流是否相近，这类方法可以进一步被归纳为基于字节流的检测。值得注意的是一种名为αDiff[10]的检测方法，该方法使用了当今热门的神经网络技术，利用二维卷积神经网络(Convolutional Neural Network，CNN)[11]对程序的二进制字节进行嵌入(嵌入既可以指神经网络将其他形式的输入转换为向量形式的输出的过程，也可以指神经网络的输出，即高维向量)，并综合函数出入度和调用表的统计特征来构建特征向量，用向量之间的距离来度量程序间的相似性。虽然该方法取得了良好的检测效果，但本文认为，由于其将二进制代码视为线性字节流，其本质上仍属于基于文本的检测。

基于文本的检测因为不考虑程序的语法和语义等信息，所以其原理和实现较其他技术更简单，其时空复杂度也较低。该类技术主要针对未应用复杂混淆手段的代码克隆和复用，该类检测方法由于容易实现对抗检测，因此常作为一种高效的辅助检测手段。

### 3.2　基于属性度量的检测

基于属性度量的检测技术关注程序汇编代码中可度量的属性和特征，提取属性和特征的度量值，构成多维向量(该类技术认为该特征向量能够从不同维度标识代码段)，以特征向量之间的距离近似来度量代码段之间的相似度大小。这类方法需要确定检测的最小粒度(如基本块或函数)，在最小粒度范围内通过对属性和特征的统计计数来获得度量值。度量值可能是但不限于是特定常量和指令等标识、函数的输入和输出等对象的计数。例如，系统discovRE[12]基于跨架构的统计特征缩小相似函数的搜索范围。该方法首先通过人工筛选的特征来构建函数的简单特征向量(这些特征包含算术指令、函数调用、重定向、转移指令、局部变量、基本块、导入函数、所有指令和参数等对象的统计个数)，利用kNN(k-Nearest Neighbors)算法[13]计算函数的相似距离，从而实现预筛选可能相似的函数，降低后续基于函数控制流图(Control Flow Graph，CFG)[14]的最大子图同构匹配方法的计算量。BinDiff[15]是一款二进制文件比较工具，能帮助快速查找汇编代码中的差异性和相似性。BinDiff是基于图形和指纹理论[16-17]开发的IDAPro插件，其函数匹配主要是基于从CFG中得到的NEC值，即基本块数(Nodes)、边数(Edges)、调用数(Calls)。

基于属性度量的检测方法因为不考虑程序的逻辑结构，所以其统计特征易于实现，但其提取的信息十分有限，因此常作为一种辅助方法。其主要问题是，属性特征是人为设计和筛选的，难以保证各属性都处于不同的维度，不同属性之间可能存在拟合问题，因此盲目地增加属性特征的种类，不仅可能无法有效提高检测的准确度，而且会增加计算开销[18]。

### 3.3　基于程序逻辑的检测

基于程序逻辑的检测技术，利用列表、树形或图形等数据结构来记录和描绘程序的数据流或控制流信息(该类技术认为所得的中间表示能够捕获程序中一定的语法和语义信息)，通过匹配相似的序列、节点或边、公共子树或子图来寻找逻辑或功能相似的程序基本块或函数。

程序的数据逻辑在函数内部体现为数据的流向和运算，在函数外部体现为函数的输入和输出。例如，Multi-MH[19]系统通过基本块的输入和输出行为来掌握其语义，利用签名来查找具有类似行为的漏洞代码。二进制搜索引擎Bingo[20]利用从符号表达式中生成的输入和输出样例来匹配语义相似的函数。

程序的控制逻辑在函数间可以用函数调用序列来描述，在函数内部可以用CFG来描绘，在基本块级可以用逻辑树来表示。例如，工具HAWK[21]实现了一种基于系统调用依赖图(System Call Dependence Graph，SCDG)胎记的动态检测方法；Pewny等[22]提出的TEDEM方法利用表达式树的编辑距离来度量基本块级别的相似性；discovRE和Genius[23]是比较著名的基于属性优化CFG的二进制相似性检测系统，其采用了各自经优化的图匹配算法；Esh[24]，BinHunt[25]，iBinHunt[26]等工具在基于CFG的基础上，利用了符号执行来确定基本块或函数的相似性。CFG能够有效获取和展示程序的控制流信息，是程序代码的高度抽象，其作为中间表示既可以用于表示源代码，也可以用于表示汇编代码，但是其跨语言的特性，导致无论是源代码的还是二进制代码的相似性检测技术，大多严重依赖于CFG作为中间表示，尤其是跨架构的二进制代码检测。

基于程序逻辑的检测方法的优点是准确度较高，可以根据不同的任务采用不同的匹配算法和策略，可伸缩性强。其缺点是时空复杂度高：一方面，数据流和CFG的提取过程代价非常昂贵；另一方面，相似性度量所依赖的图匹配算法的时间复杂度缺少多项式解，图匹配算法又多是两两匹配算法，因此在面对大规模查询任务时，计算量随代码库规模成几何倍数式增长。

上述困难并没有阻止研究人员在该方向上取得突破。针对该问题，一种方法是从中间表示CFG入手，利用其他辅助

检测技术来为CFG引入轻量级且易于度量的属性，从而简化CFG的节点，例如discovRE利用轻量级的统计特征来筛选候选函数，降低图匹配算法的任务量。另一种方法是从图匹配算法入手，采用近似图匹配算法来提高检测效率，在准确度和速度之间进行折中，但该方法受限于图匹配算法的效率，对检测效率的提升有限。近年来，研究人员开始将神经网络技术跨学科地应用到该领域，以解决传统图匹配算法遇到的效率瓶颈问题。其中最具代表性的是Qian等提出的Genius系统，该系统利用机器学习的谱聚类算法[27]来对函数的CFG进行分类并生成码本(Codebook)，再对码本进行编码，将相似函数的搜索问题转化为特征编码的搜索问题，极大地提高了效率并兼顾了结果的准确度。至此二进制代码相似性检测技术再一次实现了突破，并开拓了新的发展方向，相关研究越来越多地关注代码中更高级且更复杂的语义信息。

### 3.4 基于语义的检测

上述研究[3-5]多将中间表示的内容(承载的信息)与其形式(存储的数据结构)合并，将基于图形结构的相似性检测技术归入基于语义的检测中。这种分类方法多应用于源代码相似性检测技术，虽然对二进制代码相似性检测技术的分类具有借鉴意义，但本文认为这样既不能很好地涵盖近年来二进制代码相似性检测领域的最新成果，也不能清晰地反映该领域的发展趋势。本文总结的原因有：1)虽然高级数据结构(例如树形和图形结构)用于描述高级程序信息具有先天优势，但二者不能划等号，因为高级数据结构可以分解或转化为低级数据结构(例如列表或高维向量)，程序语义等高级信息同样可以用低级数据结构来进行描绘；2)“语义”的概念来自自然语言，若将汇编语言的指令看作“单词”，基本块看作“句子”，则可得到相应的二进制代码的“词法”和“语法”概念，从而使函数具有完整的“语义”，但是CFG主要包含程序的控制流信息，该信息仅是函数所包含的丰富语义信息的一部分；3)近几年出现的基于机器翻译或深度神经网络的程序语义相似性检测技术，才是直接关注函数语义信息的检测技术，虽然它们有的仍然使用CFG作为中间表示，但相比传统的基于CFG的检测方法却有了本质的不同，其不同之处在于新方法的检测速度较传统方法提升了几个数量级[28]。有的方法[29]甚至完全摒弃了前述的3类基本检测技术。

本文中的基于语义的检测技术，通过捕获程序汇编代码中的语义信息，来比较函数或组件的语义差异，以实现相似性度量。这类方法通常借鉴自然语言处理(Natural Language Processing，NLP)、图像识别或其他领域的技术，利用深度神经网络来实现程序语义的嵌入，通过对嵌入向量的比较或查询操作来实现大规模任务的处理。其二进制代码的中间表示包括规范化的汇编文本、其他中间语言或CFG，神经网络模型多采用暹罗架构(Siamese Architecture)[30]，利用程序代码的大规模特点构造来训练样本库，相关领域的专家和先验知识可能不是必须的。Xu等[28]于2017年提出了第一种基于神经网络生成的二进制函数CFG嵌入的方法，其在名为Gemini的系统中，利用改进的Structure2vec[31]模型(Structure2vec结构感知模型的灵感来自图形模型推理算法，该算法根据图的拓扑结构递归地聚合得到特定于顶点的特征向量)来构成暹罗架构网络，将函数的CFG嵌入高维向量，通过计算向量的余弦距离来度量函数间的相似性。Ding等[32]提出的Asm2vec模型，是基于改进的PV-DM[33]模型(PV-DM神经网络模型是为文本数据而设计的，基于文档中的标识来学习文档表征)的汇编代码表征学习模型。该模型利用自定义的函数内联和随机漫步机制，将函数的CFG建模为汇编指令的线性序列，以该汇编文本为输入，不需要任何先验知识，学习指令的语义，构建指令嵌入向量，最终得到函数的语义嵌入向量。Asm2vec是第一个将表征学习作为汇编代码构建特征向量的方案，具有优秀的抗混淆和抗编译器优化特性，但其还不能用于跨架构比较。Luca等[29]提出的SAFE网络，先利用NLP的Word2vec[34]模型(Word2vec是谷歌于2013年推出的一款NLP工具，其特点是能将单词向量化，定量度量单词之间的关系[35])来实现汇编语言的指令嵌入，再利用循环神经网络(Recurrent Neural Network，RNN)来捕获指令序列的上下文关系[36]，最终实现函数的嵌入，SAFE摒弃了CFG作为中间表示，汇编代码的语义信息直接由深度神经网络嵌入到高维向量中，这样既省去了会消耗大量时间的CFG提取过程，又避免了人为偏见的引入。SAFE模型利用RNN网络构建了暹罗架构并采用有监督的学习方法来训练模型，虽然其可以由程序自动随机构造正样本对和副样本对，但针对跨架构检测任务的训练，随着系统支持指令架构种类的增加，在理论上，其训练样本库的规模需要根据不同架构的组合成倍地扩大，这就限制了该模型的可扩展性。GeneDiff[37]是第一种使用语义表征模型来学习二进制代码中间语言，从而实现跨架构克隆检测研究的方法。它借助动态分析VEX IR(VEX IR是动态分析框架Valgrind[38-39]的中间语言，是一种二地址形式的中间表示，支持多种指令架构)的中间语言消除了不同指令架构之间的差异，通过改进的PV-DM模型来为函数的VEX IR生成语义以嵌入向量。因为每一条汇编指令会被翻译成多条VEX指令，所以该模型将由同一条汇编指令翻译而来的多条VEX指令组合看作一个单词，将基本块看作句子，将函数看作段落，使用向量间的余弦距离来度量函数间的相似度。

除了检测速度和准确度的提升，将神经网络应用于相似性检测任务的优势还在于，传统检测方法所采用的匹配算法通常是固定不变的，神经网络可以针对不同任务进行再训练，应用场景更广阔；此外，神经网络不但可以自行学习和选择特征，还可以习得人工方法很难确定的不同特征对相似度影响的权重，从而降低甚至避免人工设计和筛选特征带来的拟合。这类技术在发展过程中也遇到了一些难题：1)跨学科知识的交叉应用给检测技术的突破带来了新思路和启发，同时对研究人员的知识储备和转化能力提出了更高的要求，现有技术所使用的神经网络模型多是其他领域成果改进而来的，因此相似性检测需要更多地针对自身实际的基础进行理论创新；2)神经网络训练需要大量的训练数据，如何针对不同任务构造高质量的训练集仍然充满挑战；3)神经网络的输出是多维度向量，而向量中每一维度所代表的具体含义还无法进行科学的解释，且其结果无法用符号化的表达式或定理来证明。

### 3.5　技术对比

本文对上述分类方法进行了简要梳理和对比,结果如表 1 所列。

在实际应用中,单一检测技术取得的效果十分有限,因此往往采用多种检测技术相结合的方法来适应不同任务的需要。现将相关方法和工具进行梳理和总结,具体如表 2 所列。

表 1　技术分类的对比

Table 1　Comparison of different types of techniques

| 技术分类 | 关注信息 | 中间表示 | 优点 | 缺点 |
|---|---|---|---|---|
| 基于文本 | 汇编文本<br>字节流 | 字符串<br>标识符序列<br>高维向量 | 方法实现简单,检测时空复杂度低 | 不考虑程序的语法和语义等信息;检测准确度较低 |
| 基于属性度量 | 可度量属性 | 度量值;<br>多维向量 | 度量算法简单,方法易于实现,检测速度快 | 不考虑程序逻辑信息;不同属性间可能存在过度拟合问题,准确度难以保证 |
| 基于程序逻辑 | 控制流 | 抽象语法树<br>控制流程图<br>函数调用图 | 准确度高于基于文本和属性度量的检测;能够识别细微语法变化 | 中间表示转换过程复杂;匹配算法复杂度高 |
| | 数据流 | 输入输出表<br>数据依赖图 | | |
| 基于语义 | 程序语义 | 属性控制流图<br>语义嵌入向量 | 能够识别程序语义差异;采用机器学习方式避免了引入人为偏见;能够适应不同任务需求;检测效率高 | 需要大量语义训练样本;语义嵌入向量各维度的意义无法解释 |

表 2　二进制代码相似性检测技术及工具

Table 2　Techniques and tools of binary code similarity detection

| 方法或工具 | 技术类别 | 原理介绍 | 解决的难题 | | | |
|---|---|---|---|---|---|---|
| | | | 跨编译器 | 跨优化 | 跨架构 | 抗混淆 |
| Karta | 基于标识符 | 通过提取二进制函数中的数字常量列表和字符串列表来进行匹配 | — | — | — | • |
| GitZ[40] | 基于文本 | 利用中间表示语言和代码片段统计框架来消除不同编译器、编译优化、架构之间的差异和干扰,使用代码片段的文本哈希来确定相似片段 | • | • | • | — |
| αDiff | 基于字节流<br>基于属性度量 | 利用 DNN 网络从原始字节中提取特征向量,并综合两种统计特征来度量程序间的整体相似性 | • | • | • | — |
| Multi-MH | 基于数据流 | 通过基本块的输入和输出行为来掌握其功能语义,根据 I/O 签名跨架构查找具有类似行为的错误代码 | • | • | • | — |
| Bingo | 基于程序逻辑 | 利用 CFG 分析函数,根据从函数或基本块的符号表达式中生成的输入和输出样例来匹配功能语义相似的函数,可跨架构和跨操作系统匹配 | • | • | • | — |
| TEDEM | 基于控制流 | 利用表达式树的编辑距离来度量基本块级别的相似性 | — | — | — | — |
| Esh | 基于程序逻辑 | 将程序的 CFG 简化为单路径执行的链(Strand),链可以进一步分解为更小的片段,利用 Boogie[41] 程序验证器并根据数据流来检查两个片段是否语义等价,使用片段相似度的统计推理来建立代码的全局相似度 | • | — | — | — |
| BinHunt | 基于控制流 | 利用图同构算法计算函数间的语义差异,并利用符号执行和定理证明来确定基本块之间的功能是否相似 | — | — | — | — |
| iBinHunt | 基于控制流 | 在 BinHunt 的基础上使用深度污染和自动输入生成来发现程序控制流的语义差异 | — | — | — | — |
| HAWK | 基于控制流 | 动态提取程序的 SCDG,提出基于 SCDG 胎记的检测方法 | — | — | — | — |
| TPM[42] | 基于属性度量<br>基于控制流 | 利用设计的函数特征结合调用关系和自定义规则,递归匹配相似的函数对 | — | — | — | • |
| CoP[43] | 基于程序逻辑 | 通过一组表示基本块输入和输出关系的符号公式来建立基本块语义模型,然后以基本块为元素,利用最长公共子序列来对两条路径的语义相似性进行建模 | • | • | — | • |
| Sigma[44] | 基于控制流 | 一种通过匹配语义集成图(Semantic Integrated Graph,SIG)的轨迹来识别二进制代码中克隆函数的技术,SIG 改进和合并了控制流图、寄存器流图和函数调用图,形成了一个新的联合数据结构 | — | — | — | — |
| QSM2015[45] | 基于控制流 | 引入执行依赖图(Execution Dependence Graphs,EDG)来描述二进制代码的行为特征,通过在目标函数中找到类似的 EDG 子图来识别内联库函数 | — | — | — | — |
| Kam1n0[46] | 基于控制流 | 第一个将局部敏感哈希(Locality Sensitive Hashing,LSH)和子图搜索结合的汇编代码克隆搜索引擎 | — | — | — | — |

（续表）

| 方法或工具 | 技术类别 | 原理介绍 | 解决的难题 | | | |
|---|---|---|---|---|---|---|
| | | | 跨编译器 | 跨优化 | 跨架构 | 抗混淆 |
| BinDiff | 基于属性度量<br>基于控制流 | 忽略程序二进制指令，从 CFG 中得到 NEC 值并生成签名，跨架构寻找匹配签名的函数，也可进一步匹配 CFG 的边 | • | • | — | — |
| CACompare[47] | 基于程序逻辑 | 首先从函数的 CFG 中提取动态执行所需的参数和 switch 跳转目标，然后将不同架构的汇编函数转换为统一的中间表示并模拟执行，从而提取函数的语义签名，最后查找相似的签名 | • | • | • | — |
| BinSign[48] | 基于属性度量<br>基于控制流 | 一种基于代码指纹的汇编函数搜索方案，其代码指纹由从汇编代码中提取的一组特征和从拆分的 CFG 提取的路径特征生成 | — | • | — | • |
| BinShape[49] | 基于属性度量<br>基于控制流 | 一个用于识别二进制代码中标准库函数的系统，其基于 CFGs、指令级特征、统计特征和函数调用图等异构特征为每个库函数构建一种结构独特的签名 | • | • | — | • |
| BinSim[50] | 基于程序逻辑 | 通过增强的动态切片和符号执行来识别两个执行轨迹之间的细粒度语义相似性或差异 | — | — | — | • |
| discovRE | 基于属性度量<br>基于控制流 | 利用 kNN 算法并根据统计特征过滤出候选函数，以降低基于函数 CFG 图同构的计算量 | • | • | • | — |
| Genius | 基于属性度量<br>基于控制流 | 利用多种统计特征来构建函数的属性控制流图（ACFG），利用光谱聚类算法对 ACFG 生成码本，再对码本进行编码，以便于快速搜索，实现相似函数的分类和搜索 | • | • | • | — |
| Gemini | 基于属性度量<br>基于控制流 | 利用改进的 Structure2Vec 模型将函数的 ACFG 嵌入高维向量，以向量之间的距离来度量函数间的相似性 | — | • | • | — |
| SMCSA[51] | 基于属性度量<br>基于控制流 | 分两阶段进行检测，第一阶段利用神经网络将函数 CFG 嵌入向量以实现快速搜索，第二阶段利用函数局部调用图（Local Call Graph，LCG）来分析提高漏洞检测的准确度 | — | — | • | — |
| Fossil[52] | 基于文本<br>基于控制流 | 一个在恶意软件中识别开源代码的系统，它通过操作码频率来捕获函数的语法，通过从控制流图中提取节点间交互来捕获函数的语义，通过计算重要操作码的分布来捕获函数的行为，最后使用贝叶斯网络模型（Bayesian Network Model）综合 3 个组件的结果 | — | — | — | • |
| BinDNN[53] | 基于语义 | 利用了 CNN，LSTM 和 DNN 这 3 种类型的神经网络模型，将模型进行分层，学习函数汇编代码并判断函数是否相似 | — | — | — | • |
| Asm2Vec | 基于语义 | 将基于 PV-DM 模型的函数语义嵌入模型，用于单一架构函数相似性度量 | • | • | — | • |
| SAFE | 基于语义 | 将基于 Word2vec 模型的函数语义嵌入模型，可实现跨架构函数相似性度量 | • | • | • | — |
| INNEREYE[54] | 基于控制流<br>基于语义 | 分别从基本块、CFG 路径和组件这 3 个层次来研究组件间的相似性，其基本块的语义嵌入过程是利用 Word2vec 和 LSTM[55] 模型实现的 | — | • | • | — |
| GeneDiff | 基于语义 | 第一个将二进制代码转换为中间表示再使用语义表征模型来实现跨架构克隆检测的方法 | • | • | • | — |
| OrderMatters[56] | 基于控制流<br>基于语义 | 首先利用改进的 BERT[57] 模型将函数 CFG 节点转换为语义嵌入向量，然后利用改进的 MPNN[58] 模型来获得 CFG 的语义和结构以嵌入向量，接着利用 CNN 模型获得 CFG 节点的顺序以嵌入向量，最后将后两者的综合作为最终的函数嵌入向量，用于函数间的相似性度量 | • | • | • | — |

注：• 代表已解决，— 代表未解决

## 4 总结

从近几年二进制代码相似性检测的研究成果可以看出其近年来的发展趋势。最初的研究仅提取代码中的低级特征，这些方法通常作为快速的辅助检测方法。后来，随着对代码信息研究的深入，一些方法开始借助于高级数据结构和程序分析技术来提取代码的高级特征和信息，提高了检测的准确度。但随着跨编译器、跨操作系统、跨指令架构和海量数据查询等任务需求的提出，如何提高检测的通用性和效率成为更突出的问题。传统相似性检测方法由于多采用图形结构的中间表示，受限于图匹配算法的效率，只能选择在准确度和速度之间做平衡，对相似性度量算法效率的提升逐渐陷入瓶颈。近几年，越来越多的研究借助于深度神经网络技术突破了这一瓶颈。神经网络的优势在于不仅可以自动学习和筛选影响相似性的特征，还可以确定不同特征对相似性影响的大小，该方法的关键在于如何处理原始代码和选择哪种中间表示更利于神经网络的训练和学习，以及哪种神经网络模型更适合程序相似度模型的构建。综上，未来神经网络技术的应用仍将是该领域研究的热点之一，利用不同学科技术的交叉应用并结合传统程序分析手段的检测方法将成为发展的新方向。

**结束语** 二进制代码相似性检测在软件知识产权保护、代码脆弱性检查等领域发挥着重要作用。本文通过梳理已有的二进制代码相似性检测技术和方法，提出了基于代码信息的分类方法，即基于文本、基于属性度量、基于程序逻辑和基

于语义的检测技术，比较了各类检测技术之间的优劣。同时，梳理出了检测技术的发展脉络，并得出基于神经网络和多种技术相结合的程序语义相似性检测技术将成为未来该领域的研究热点。

## 参 考 文 献


[1] Synopsys, Inc. 2020 Open Source Security and Risk Analysis Report[EB/OL]. (2020-06-08)[2020-07-08]. https://www.synopsys.com/software-integrity/resources/analyst-reports/2020-open-source-security-risk-analysis.html.

[2] WHALE G. Plague: Plagiarism Detection Using Program Structure[R]. Dept. of Computer Science Technical Report 8805. University of NSW, Kensington, Australasian, 1988.

[3] XIONG H, YAN H H, GUO T, et al. Code Similarity Detection: A Surve[J]. Computer Scienc, 2010, 37(8): 9-14.

[4] ZHANG D, LUO P. Survey of Code Similarity Detection Methods and Tools[J/OL]. Computer Science. [2020-03-02]. http://kns.cnki.net/kcms/detail/50.1075.TP.20200115.1646.004.html.

[5] CAO Y Z, JIN M Z, LIU C. Overview on Clones Detection[J]. Computer Engineering & Science, 2006(S2): 9-13.

[6] XU H Y, LEI Z Z, LI D. Survey of Code Obfuscation[J]. Computer & Digital Engineering, 2007, 35(10): 4-7.

[7] Eyal Itkin. Karta: Matching Open Sources in Binaries[EB/OL]. (2019-03-21)[2020-03-04]. https://research.checkpoint.com/2019/karta-matching-open-sources-in-binaries/.

[8] Hex-Rays. About IDA[EB/OL]. (2020-03-29)[2020-03-29]. https://www.hex-rays.com/products/ida/.

[9] OHJ. DarunGrim: A Patch Analysis and Binary Diffing Tool [EB/OL]. (2020-06-18)[2020-07-10]. http://www.darungrim.org/.

[10] LIU B, HUO W, ZHANG C, et al. αdiff: cross-version binary code similarity detection with dnn[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018: 667-678.

[11] KRIZHEVSKY A, SUTSKEVER I, HINTON G E, et al. ImageNet Classification with Deep Convolutional Neural Networks [C]// Advances in Neural Information Processing Systems. 2012: 1097-1105.

[12] ESCHWEILER S, YAKDAN K, GERHARDS-PADILLA E. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code[C/OL]// The Network and Distributed System Security Symposium (NDSS 2016). 2016. http://dx.doi.org/10.14722/ndss.2016.23185.

[13] MUJA M, LOWE D G. Fast approximate nearest neighbors with automatic algorithm configuration[C]// International Conference on Computer Vision Theory and Applications. 2009: 331-340.

[14] ALLEN F E. Control flow analysis[J]. ACM Sigplan Notices, 1970, 5(7): 1-19.

[15] Zynamics. BinDiff Home[EB/OL]. (2020-05-05)[2020-07-11]. https://www.zynamics.com/bindiff.html.

[16] FLAKE H. Structural comparison of executable objects[C]// Detection of Intrusions and Malware & Vulnerability Assessment. 2004: 161-173.

[17] DULLIEN T, ROLLES R. Graph-based comparison of executable objects [J]. Symposium Sur la Sécurité Des Technologies De L'information Et Des Communications, 2005, 5(1): 3.

[18] MARIMONT R B, SHAPIRO M B. Nearest Neighbour Searches and the Curse of Dimensionality[J]. IMA Journal of Applied Mathematics, 1979, 24(1): 59-70.

[19] PEWNY J, GARMANY B, GAWLIK R, et al. Cross-architecture bug search in binary executables[C]// 2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 709-724.

[20] CHANDRAMOHAN M, XUE Y, XU Z, et al. Bingo: Cross-architecture cross-os binary search[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016: 678-689.

[21] WANG X, JHI Y C, ZHU S, et al. Behavior based software theft detection[C]// ACM Conference on Computer and Communications Security (CCS 2009). Chicago, Illinois, USA, DBLP, 2009: 280-290.

[22] PEWNY J, SCHUSTER F, BERNHARD L, et al. Leveraging semantic signatures for bug search in binary programs[C]// Proceedings of the 30th Annual Computer Security Applications Conference. 2014: 406-415.

[23] QIAN F, ZHOU R, XU C, et al. Scalable Graph-based Bug Search for Firmware Images[C]// Acm Sigsac Conference on Computer & Communications Security. 2016: 480-491.

[24] DAVID Y, PARTUSH N, YAHAV E. Statistical similarity of binaries[J]. ACM SIGPLAN Notices, 2016, 51(6): 266-280.

[25] GAO D, REITER M K, SONG D. Binhunt: Automatically finding semantic differences in binary programs[C]// International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 2008: 238-255.

[26] MING J, PAN M, GAO D. iBinHunt: Binary hunting with interprocedural control flow[C]// International Conference on Information Security and Cryptology. Springer, Berlin, Heidelberg, 2012: 92-109.

[27] NG A Y, JORDAN M I, WEISS Y, et al. On Spectral Clustering: Analysis and an algorithm[C]// Advances in Neural Information Processing Systems. 2002: 849-856.

[28] XU X, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.

[29] MASSARELLI L, DI LUNA G A, PETRONI F, et al. Safe: Self-attentive function embeddings for binary similarity[C]// International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2019: 309-329.

[30] BROMLEY J, GUYON I, LECUN Y, et al. Signature verification using a "siamese" time delay neural network[C]// Advances in Neural Information Processing Systems. 1994: 737-744.

[31] DAI H, DAI B, SONG L. Discriminative embeddings of latent variable models for structured data[C]// International Conference on Machine Learning. 2016: 2702-2711.

[32] DING S H H, FUNG B C M, CHARLAND P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]// 2019

IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 472-489.
[33] LE Q, MIKOLOV T. Distributed representations of sentences and documents[C]// International Conference on Machine Learning. 2014: 1188-1196.
[34] Google. Tool for computing continuous distributed representations of words[EB/OL]. (2013-07-30)[2020-03-07]. https://code.google.com/archive/p/word2vec/.
[35] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed representations of words and phrases and their compositionality[C]// Advances in Neural Information Processing Systems. 2013: 3111-3119.
[36] LIN Z, FENG M, SANTOS C N, et al. A structured self-attentive sentence embedding[J]. arXiv: 1703.03130, 2017.
[37] LUO Z, WANG B, TANG Y, et al. Semantic-Based Representation Binary Clone Detection for Cross-Architectures in the Internet of Things[J]. Applied Sciences, 2019, 9(16): 3283.
[38] Valgrind. Valgrind Home [EB/OL]. (2020-07-13) [2020-07-13]. https://www.valgrind.org/.
[39] NETHERCOTE N, SEWARD J. Valgrind: a framework for heavyweight dynamic binary instrumentation[C]// Programming Language Design and Implementation, 2007, 42(6): 89-100.
[40] DAVID Y, PARTUSH N, YAHAV E. Similarity of binaries through re-optimization[C]// Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017: 79-94.
[41] BARNETT M, CHANG B Y E, DELINE R, et al. Boogie: A modular reusable verifier for object-oriented programs[C]// International Symposium on Formal Methods for Components and Objects. Springer, Berlin, Heidelberg, 2005: 364-387.
[42] XIAO Y, CAO S, CAO Z, et al. Matching Similar Functions in Different Versions of a Malware[C]// 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE, 2016: 252-259.
[43] LUO L, MING J, WU D, et al. Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection[J]. IEEE Transactions on Software Engineering, 2017(12): 1-1.
[44] ALRABAEE S, SHIRANI P, WANG L, et al. SIGMA: A Semantic Integrated Graph Matching Approach for Identifying Reused Functions in Binary Code[J]. Digital Investigation: The Internatnional Journal of Digital Forensics & Incident Response, 2015, 12(1): 61-71.
[45] QIU J, SU X, MA P. Library functions identification in binary code by using graph isomorphism testings[C]// 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015: 261-270.
[46] DING S H H, FUNG B C M, CHARLAND P. Kam1n0: Mapreduce-based assembly clone search for reverse engineering[C]// Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016: 461-470.
[47] HU Y, ZHANG Y, LI J, et al. Binary code clone detection across architectures and compiling configurations[C]// 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). IEEE, 2017: 88-98.
[48] NOUH L, RAHIMIAN A, MOUHEB D, et al. Binsign: fingerprinting binary functions to support automated analysis of code executables[C]// IFIP International Conference on ICT Systems Security and Privacy Protection. Springer, Cham, 2017: 341-355.
[49] SHIRANI P, WANG L, DEBBABI M. BinShape: Scalable and robust binary library function identification using function shape[C]// International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2017: 301-324.
[50] MING J, XU D, JIANG Y, et al. Binsim: Trace-based semantic binary diffing via system call sliced segment equivalence checking[C]// 26th USENIX Security Symposium. 2017: 253-270.
[51] WANG Y, SHEN J, LIN J, et al. Staged method of code similarity analysis for firmware vulnerability detection[J]. IEEE Access, 2019(7): 14171-14185.
[52] ALRABAEE S, SHIRANI P, WANG L, et al. Fossil: a resilient and efficient system for identifying foss functions in malware binaries[J]. ACM Transactions on Privacy and Security (TOPS), 2018, 21(2): 1-34.
[53] LAGEMAN N, KILMER E D, WALLS R J, et al. BinDNN: Resilient Function Matching Using Deep Learning[C]// International Conference on Security and Privacy in Communication Systems. Springer, Cham, 2016: 517-537.
[54] ZUO F, LI X, YOUNG P, et al. Neural machine translation inspired binary code similarity comparison beyond function pairs[J]. arXiv: 1808.04706, 2018.
[55] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.
[56] YU Z, CAO R, TANG Q, et al. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection[C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2020: 1145-1152.
[57] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv: 1810.04805, 2018.
[58] GILMER J, SCHOENHOLZ S S, RILEY P, et al. Neural Message Passing for Quantum Chemistry[C]// Proceedings of the 34th International Conference on Machine Learning (ICML'17). 2017: 1263-1272.

**FANG Lei**, born in 1989, postgraduate, assistant engineer. His main research interests include security of network information and so on.

**WEI Qiang**, born in 1979, Ph.D, professor, Ph.D supervisor. His main research interests include security of network information and so on.