

# IoT 设备程序同源性智能检测技术综述

孔凯薇<sup>1</sup>, 霍冬冬<sup>2,3</sup>, 苏东楠<sup>1</sup>, 徐震<sup>2,3</sup>

<sup>1</sup>北京计算机技术及应用研究所 北京 中国 100039

<sup>2</sup>中国科学院信息工程研究所 北京 中国 100093

<sup>3</sup>中国科学院大学 网络空间安全学院 北京 中国 100049

**摘要** IoT 设备与各行业的深度融合方兴日盛, 使得 IoT 程序快速开发的需求不断增长。开发者习惯于集成第三方库或常用代码。不幸的是, 若这些代码中隐藏着潜在的漏洞, 那他们也会被扩散到不同的程序中, 为其大规模扩散创造了条件。这也是造成近年来 IoT 设备群体性安全事件频发的重要原因之一。为了降低危害, 发现具备相似漏洞的程序并进行相关的处置是一个有效的方法。同源性分析作为挖掘程序间关联关系的重要手段之一, 可高效地实现程序漏洞的智能溯源取证。结合机器学习和深度学习技术, 它表现出解决大规模程序安全性检测的巨大潜力。然而, IoT 设备的软硬件特点仍使得该技术的使用面临挑战。当前已有诸多方案在 IoT 设备程序的同源性智能检测方面取得了进展。因此, 本文将系统性回顾近年来相关技术研究的成果, 将他们分为相似性分析和创作者归属技术。首先, 我们介绍了两种方式的数据来源。接着, 检测过程中涉及的特征选择、特征表示以及相对应的检测方法也被依次介绍。进一步的, 本文不仅比较和总结了方案的特点和局限性, 还对他们在不同类型 IoT 设备程序的适配性进行了对比分析。最后, 文章针对 IoT 程序分析提出了一些研究建议。作者希望本综述可为研究者阐明这些工作的核心技术点, 并为他们在 IoT 设备上的进一步应用提供启发。

**关键词** IoT 设备; IoT 程序同源性; 相似性分析; 创作者归属; 智能检测  
中图分类号 TP309.1 DOI 号 10.19363/J.cnki.cn10-1380/tn.2023.05.08

## Survey of Intelligent Homology Detection Technology for IoT Programs

Kong Kaiwei<sup>1</sup>, Huo Dongdong<sup>2,3</sup>, Su Dongnan<sup>1</sup>, Xu Zhen<sup>2,3</sup>

<sup>1</sup> Beijing Institute of Computer Technology and Applications, Beijing 100039, China

<sup>2</sup> Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

<sup>3</sup> School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

**Abstract** The tendency that IoT devices are deeply integrated into industries is flourishing, which spawns large requirements for developing IoT programs rapidly. Developers are used to integrating third-party libraries or accustomed code. Unfortunately, if there are potential vulnerabilities hidden in these code, they will also be dispersed into different programs, creating conditions for its large-scale proliferation. This is an important reason for the frequent occurrence of mass IoT security incidents in recent years. To mitigate such incidents, finding programs with similar vulnerabilities to impose corresponding security treatments is an effective approach. Towards this issue, the homology analysis technology, as one of the important means to mine the association between programs, can realize the traceability and forensics of vulnerabilities. Combining with recent intelligent methods such as machine learning and deep learning technique, the homology analysis is showing great potential to fulfill the security detection for large-scale IoT programs. However, the hardware and software characteristics of IoT devices still make the use of this technology challenging. To this end, many studies have made progress in homology intelligent detection for IoT programs. In this paper, we provide a systematic review of state-of-the-art proposals, which are classified into similarity analysis and authorship attribution. First, we introduce these two types of work from the perspective of data sources. Then, specific features used in homology analysis, ways of feature representation and relevant detection methods are proposed to describe these proposals. Further, we not only compare characteristics of these studies to summarize their practicalities and limitations, but discuss their portability towards different types of IoT programs. Finally, some research suggestions are put forward. The authors hope that this review can clarify core processes of this technique and provide insights for its further applications on IoT devices.

**Key words** IoT devices; the homology of IoT programs; similarity analysis; authorship attribution; intelligent detection

通讯作者: 霍冬冬, 博士, 高级工程师, Email: huodongdong@iie.ac.cn。

本课题得到国家重点研发计划-制造大数据智能治理方法研究(No. 2018YFB1700403)资助。

收稿日期: 2022-09-10; 修改日期: 2022-12-15; 定稿日期: 2023-03-28

## 1 前言

随着 5G 技术的蓬勃发展, 物联网(Internet of Things, IoT)设备以其轻量化、易使用、广互联的特性, 已经被广泛应用于以智慧医疗、智慧物流、智慧工厂为代表的人类生产生活中的各个方面, 起到了越来越重要的作用<sup>[1-3]</sup>。根据有关机构的预测, 至 2025 年全球 IoT 设备的总量将达到 300 亿台, 并将继续呈爆炸性增长趋势<sup>[4]</sup>。然而, IoT 设备的大规模部署也使得其一旦发生安全问题则容易形成群体性安全事件, 因而引来了越来越多的关注。有研究指出, 仅 2021 年上半年, 针对 IoT 设备的攻击就高达 15 亿次<sup>[5]</sup>, 涵盖敏感信息窃取<sup>[6]</sup>、构筑僵尸网络<sup>[7]</sup>以及远程恶意控制<sup>[8]</sup>等, 为他们的深度扩展应用造成了极大的负面影响。

分析 IoT 设备易于引发群体性安全问题的原因, 一部分在于轻量化的软硬件资源条件<sup>[9]</sup>使他们自身的安全性变得越发脆弱。而另一个不容忽视的事实, 则是这些受到攻击的设备大都具备相似的漏洞。为了缩短开发周期, IoT 设备通常会集成不同种类的第三方库<sup>[10]</sup>, 同一个组织、开发者开发也习惯使用过去自己使用过的模块。若这些复用的程序中存在可以被攻击者利用的漏洞, 那么数以万计的 IoT 设备将面临被攻击的危险。

许多学者都注意到了漏洞重用对 IoT 设备带来的严重安全威胁。例如, Cui 等<sup>[11]</sup>报告了用于升级的许多固件中普遍存在由包含某些第三方库而引入的已知漏洞; 高通公司基带芯片固件漏洞(CVE-2020-11292)则间接使得全球 30% 智能手机的安全性受到威胁<sup>[12]</sup>。攻击同一类漏洞的恶意代码经过加壳、混淆以及架构适配后便可被轻易移植到大量的目标设备上<sup>[13-14]</sup>, 使得引发大规模 IoT 设备攻击的成本变得越来越低, 如 MIRAI 以及其变种<sup>[15]</sup>。因此, 分析目标设备以获取重要程序代码的出处和来源, 将他们与已知漏洞或恶意开发者进行对应, 可指导安全人员采取更为针对性的安全加固措施, 将是提升 IoT 设备整体安全能力的重要手段。

同源性检测是一种对程序代码或二进制文件进行来源分析判定的技术<sup>[16-17]</sup>。通过分析重要逻辑的表现形式以及关键功能的使用方式, 该技术可以为目标程序与待匹配对象建立起关联和映射关系, 并以此确定程序的安全性。为了对不同程序的特征进行准确刻画, 传统分析方案不得不结合大量的人工干预优化, 在面对海量 IoT 设备程序分析时方案的实用性不高。而随着以机器学习、深度学习为代表的

智能检测技术兴起, 使得其不仅支持自动化地从大量程序特征中高效获取专属于某一类程序的表示, 同时在面对被混淆化的程序时仍具备较好的分析稳定性。这些因素都使得基于智能检测技术的同源性分析方案具备更好的实用性。

然而, 由于 IoT 设备架构以及程序的特殊性<sup>[18-19]</sup>, 为了达到足够的检测精度, 该项技术的应用仍面临下述问题。第一, IoT 设备的部署规模庞大, 涉及到的程序软件繁多, 如何从海量设备程序中挖掘出程序的特点以提升同源性分析的效率是需要解决的难点; 第二, IoT 设备运行所需的基础硬件支撑环境尚未形成统一的标准共识, 架构的多样性对分析方案的跨平台适配性提出挑战<sup>[20]</sup>; 第三, IoT 设备程序的获取、标记以及关键信息提取都是同源性分析顺利实施的前提。

虽然已经有研究者对程序同源性分析技术进行了梳理, 然而当前还缺乏针对 IoT 设备程序进行智能检测的系统性总结。孙等<sup>[21]</sup>将同源性分析技术从静态分析和动态分析角度进行了检测能力的对比总结; Haq 等<sup>[22]</sup>面向二进制文件相似性检测技术, 从应用场景、方案使用、方案实现以及其效果等方面进行总结; Kalgutkar 等<sup>[23]</sup>对代码的作者归属技术进行了综述并对其涉及到的典型特征以及归属方案进行了总结。然而, 这些工作都未对 IoT 设备程序同源性分析进行关注。当前在该领域还缺乏结合智能检测的同源性分析技术在 IoT 设备上应用的系统性总结归纳。

因此, 本文将围绕 IoT 固件的同源性智能检测技术, 在分析和总结国内外优秀的研究成果的同时, 对这些研究的效果进行评价。具体而言, 第 2 章将对一些背景知识, 包含 IoT 设备程序分类、同源性检测技术以及本文涉及到的研究成果进行整体介绍; 第 3 章~第 5 章将对 IoT 设备程序同源性智能检测技术成果进行介绍, 包括所使用的相关数据集以及在此基础上面向两类同源性检测目标的检测技术; 第 6 章针对技术现状提出了一些研究建议; 第 7 章对全文进行总结。

## 2 IoT 程序与同源性检测

### 2.1 IoT 设备程序分类

IoT 设备程序的部署方式与 X86 系统不同, 每一个设备通常仅支持部署一个软件包, 该软件包中将集成所有运行逻辑和设备驱动代码。这类程序与通用 PC 等桌面计算机系统应用有以下的区别: 1) IoT 设备程序应用逻辑通常可以直接通过与底层的硬件

进行交互,而通用系统的软件通常需要中间件的支持;2)IoT 设备程序在部署时一般存储在设备的 Flash、ROM 等非易失性存储器当中,并被分配物理地址空间,在执行时将会被处理器的指令寄存器直接寻址,而通用系统的软件需要被映射到相应的虚拟内存空间后才可以被执行;3)IoT 设备程序在编译后会根据系统需求生成涵盖 blob、img 等多种格式的输文件,较通用系统的软件格式更为丰富。

在 IoT 设备程序分类方面,当前已经有研究者<sup>[24]</sup>根据是否搭载操作系统将其进行分类: (1) 裸机系统程序(类别 I), 这种程序不需要搭载操作系统, 通常使用 C 语言进行编程, 可通过直接控制物理内存的方式实现与硬件的交互, 硬件设备的软硬件能力较差; (2) 基于实时操作系统(Real-Time Operation System, RTOS)的程序(类别 II), 这种程序开始配合简单的微内核进行工作。虽然仍以操作物理内存为基础, 但程序已经可以通过 RTOS 提供的硬件抽象层与底层硬件交互, 具备了模块化系统的雏形。此外, 系统在设计上重点关注中断和任务调度能力, 因此可以快速响应需求。典型的 RTOS 有 ROS2<sup>[25]</sup>、uCOS<sup>[26]</sup>

以及 Contiki<sup>[27]</sup>等, 他们常搭载 C/C++ 编写的程序; (3) 基于嵌入式操作系统的程序(类别 III), 基于这种程序的设备软硬件能力较强, 通常搭载 Linux 等系统辅助进行业务处理, 可利用内存管理单元(MMU)以实现物理-虚拟地址的映射, 具备良好的多任务处理以及资源管理能力, 可支持基于多种高级语言(如 Python、Java、C++)的程序运行。

根据实际应用场景中对软件逻辑复杂程度以及处理效率的需求, 设备所搭程序(类别 I/II/III)类型也将会出现差异。同时, 针对不同使用环境所开发的 IoT 设备程序, 其分析的关注点以及工作量也会出现些许差异。例如, 对于类别 I 的程序来说, 其代码交互逻辑相对直接, 所需的代码量较少, 因而同源性分析的工作量和复杂度将在一定程度上得到缓解; 而对于类别 III 程序而言, 软件逻辑的复杂度也会得到显著提升, 诸如类的继承嵌套、各种新编程方法的使用, 都会使得程序的分析工作变得更为繁琐, 同时也加大了同源性判定的挑战。不同类型程序的特点以及其对同源性分析的可能影响如表 1 所示。

表 1 IoT 设备程序分类  
Table 1 IoT Program Classification

程序类型	搭载系统	移植成本	硬件能力	编程语言	代码量	代码逻辑	编写人员
类别 I	无系统	低	低	单一	小	简单	较少
类别 II	RTOS	中	中	较单一	小	复杂	较多
类别 III	嵌入式操作系统	高	高	丰富	大	复杂	很多

2.2 同源性分析技术

2.2.1 IoT 设备程序同源性智能检测流程

概括上来说, 典型的 IoT 设备程序的同源性智能检测技术包含 4 个主要步骤, 如图 1 所示。

**样本程序获取。**在对目标 IoT 程序进行检测前, 研究者需要获取足够数量的训练样本以生成可靠的检测模型。由于并没有权威数据集支撑 IoT 设备同源性检测任务, 因此很多工作在提出检测方法的同时也会对数据集来源进行介绍, 并使用自有方式进行

数据收集和标注。针对不同目标, 样本程序可分为固件和源码两类, 检测任务可分为对前述三类程序的检测, 与之对应的分析方法也各有不同。

**特征选择。**在获取样本的基础上, 需要将原本的程序进行处理, 以数个特征的形式对程序的特点进行表征。根据不同的关注点, 所需要提取可表征程序特点的特征也各不相同, 如面向指令、函数以及逻辑的特征。不仅如此, 软件(固件和源码)和硬件(不同架构)类型的区别也使得对 IoT 程序特征的提取需要考

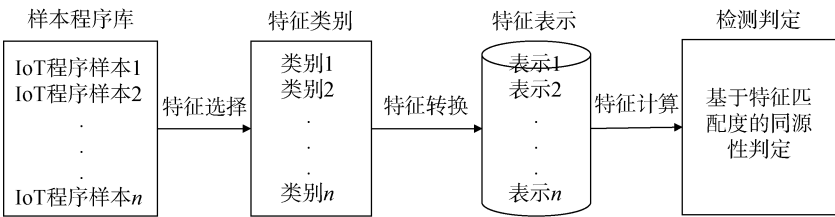


图 1 IoT 设备程序同源性智能分析流程

Figure 1 Process of Intelligent Homology Detection for IoT Programs

虑到精度的问题。例如, 对部署在不同架构 IoT 设备上的同一个固件程序进行特征提取, 其获得的特征有可能出现一定的偏差。

**特征表示。**在获得了程序对应的特征后, 需要将其转换为便于进行数学计算的特征表示。特征表示将会使各离散的特征联系起来, 以生成对训练程序更为抽象的表达。获取特征表示的方法有很多, 典型的如借助统计学方法直接表示或利用深度学习的数据挖掘能力获得深度的数学表示。

**检测判定。**基于所生成的特征表示, 同源性检测任务随即展开。确定两个程序是否同源即确定程序的特征表示是否相近。确定的方法可以基于机器学习模型预测, 也可基于其他特殊的相似度计算方法。

2.2.2 同源性分析模式

同源性分析可从以下两种检测目标进行分类。第一, 从功能性角度来说, 可以将目标程序(以及其内含的漏洞)归类到某一个已知漏洞家族谱系中, 即程序的相似性分析(similarity analysis)。通过将待测程序与已知比对程序(或漏洞片段)进行对比分析, 提取诸如控制流、调用图等特征的方式, 该技术可利用特征匹配的方式对程序是否存在潜在异常进行相似性判断。在匹配过程中, 归一化的距离匹配、镜像相似性分析、特殊字符匹配以及模糊哈希相关性分析等都是常见的方法。近年来, 研究人员倾向于将机器学习技术融入相似性检测的过程中来, 以此挖掘更多可以表征程序信息, 以更高效的完成检测任务。

此外, 同源性分析工作还可以从写作风格, 即原作者归属分析(authorship analysis)的角度展开。若某一个软件或第三方库被证实存在潜在漏洞, 作为开发该程序的组织机构或个人, 他们以往开发过的软件或借助这些软件功能做二次开发的软件都应被划为重点安全检测对象。不仅如此, 将程序漏洞与其开发的组织团体以及个人进行联系的方式, 也能方便其进行后续的安全事件调查溯源和漏洞修补。在

进行原作者归属判断时, 基于自然语言处理的特征维度会被广泛采用。另外, 社会学相关维度, 诸如利用专家知识并结合软件签名证书、漏洞外在表现形式以及诸多环境因素(如: 模块开发者记录、开发时间、所属组织)也将被考虑在内, 以帮助研究者进行漏洞归属的联合分析定位。

经过这些年的积累和沉淀, 不断有针对程序的同源性智能分析工作被发表出来, 表 2 对本文中涉及到的程序同源性智能分析技术研究进行了总结, 时间跨度在 2014-2022 年。概括来说, 当前绝大多数相似性检测的工作都具备跨平台的性质, 也都在以 ARM 架构为代表的 IoT 设备上进行了实验验证, 不过这些工作并未更进一步的对不同设备类型(即 I、II 和 III)设备的适应情况进行进一步讨论。而对于原作者归属工作来说, X86 平台应用程序因为更易于收集, 成为了众多研究的首选对象。而这些工作在针对 X86 平台程序作为检测验证对象的同时, 也开始关注相关工作在 IoT 设备上的表现。事实上, 这些研究中有一部分采用了通用的研究方法, 因此具备移植到 IoT 设备上的潜力。例如, 有部分研究工作专注于面向 IoT 设备程序的分析, 也取得了良好的效果。

2.3 后续研究安排

在后文的介绍中, 将以章节 2.1 和 2.2 为基础对 IoT 设备程序同源性智能检测技术展开介绍。如图 2 所示, 依照检测的流程, 本文将首先介绍同源性智能检测检测时涉及到的数据来源。在此基础上, 本文分别从程序相似性和创作者归属分析两个角度对同源性技术进行介绍, 包括涉及到的特征描述, 特征的表示以及如何将他们应用到检测任务, 之后将会对这些技术从方案特点比较以及对 I、II、III 型 IoT 设备程序的适配性进行讨论。接着, 将针对文中所述 IoT 设备同源性检测技术提出综合性建议。最后, 作者将会对全文进行总结。

表 2 现有程序同源性智能分析技术总结

Table 2 Recent Homology Analysis Techniques Towards IoT Firmware

同源性类型	检索年限	相关文献
相似性匹配	2014-2022	$\alpha$ Diff <sup>[28]</sup> 、DeepBinDiff <sup>[29]</sup> 、SAFE <sup>[30]</sup> 、Genius <sup>[31]</sup> 、Gemini <sup>[32]</sup> 、VulSeeker <sup>[33]</sup> 、Kimberly Redmond 等人 <sup>[34]</sup> 、InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup> 、Asm2Vec <sup>[37]</sup> 、VulSeeker-Pro <sup>[38]</sup> 、Luo 等 <sup>[39]</sup>
创作者归属	2014-2022	Bayrami 等 <sup>[40]</sup> 、Dauber 等 <sup>[41]</sup> 、Caliskan-Islam 等 <sup>[42]</sup> 、Bogomolov 等 <sup>[43]</sup> 、Abuhamad 等 <sup>[44]</sup> 、CPA <sup>[45]</sup> 、BinEye <sup>[46]</sup> 、Abuhamad 等 <sup>[47]</sup> 、Gonzalez 等 <sup>[48]</sup> 、Kalgutkar 等 <sup>[49]</sup> 、Zhang 等 <sup>[50]</sup> 、Meng 等 <sup>[51]</sup> 、Wisse 等 <sup>[52]</sup> 、Oba2 <sup>[53]</sup>

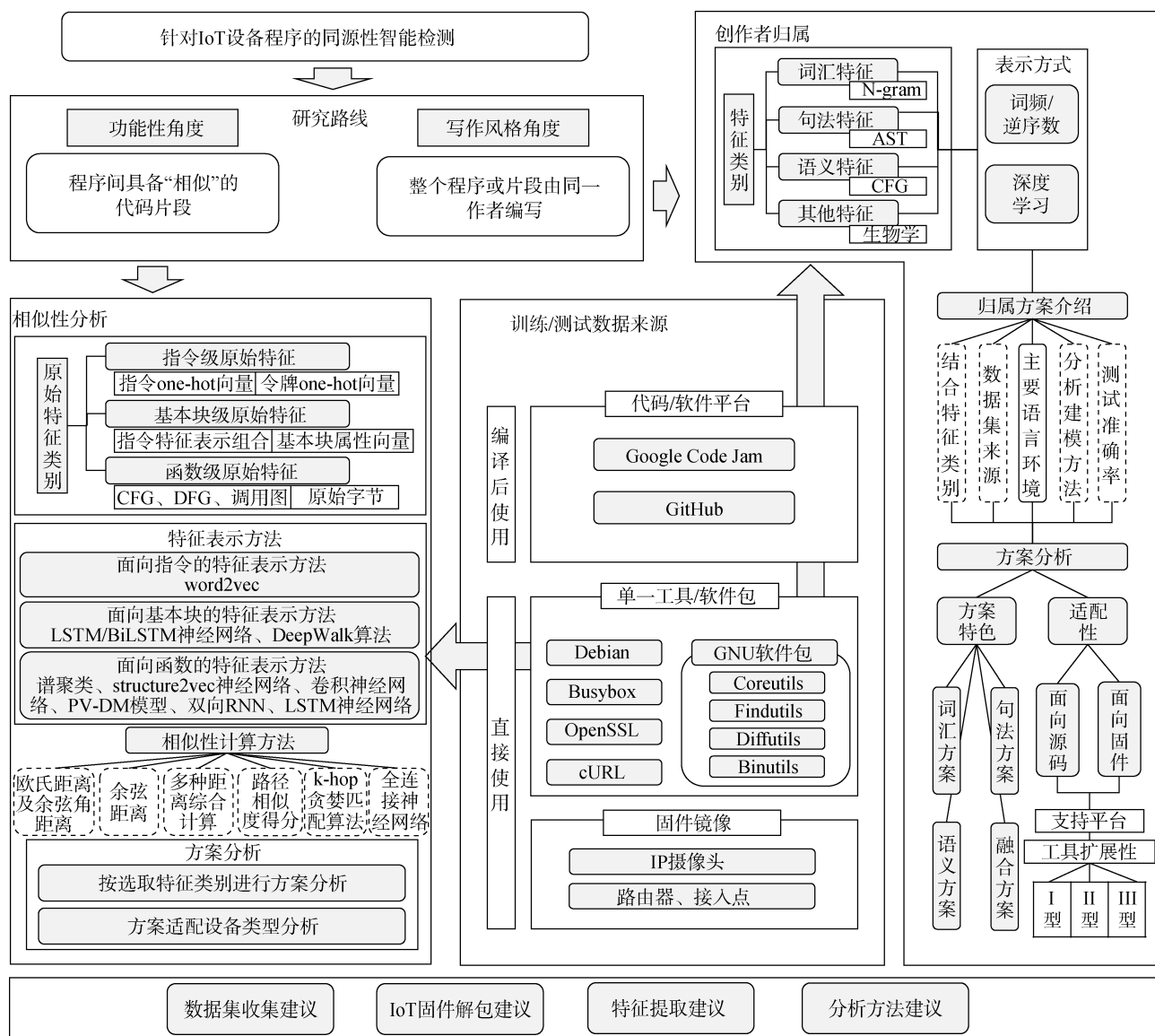


图 2 IoT 设备程序同源性智能检测

Figure 2 Intelligent Homology Detection for IoT Programs

### 3 同源性分析数据来源介绍

研究者需要采集到足够多的训练样本以支撑同源性智能检测技术理念的实施和验证。如表 3 所示,本章对文中介绍文献涉及到的数据来源进行了收集,并将他们以代码/软件包库、单一工具/软件包、固件镜像三大类进行介绍。

#### 3.1 代码/软件平台

一些研究会从代码/软件包库中选取若干源代码或软件包来编译生成 IoT 固件镜像,或直接对这些进行源码级别的特征提取。本文介绍研究的数据来源包括 Google Code Jam 以及 GitHub 平台。其中,谷歌全球编程挑战赛(Google Code Jam, GCJ)是谷歌主持的一项国际性的编程竞赛。该竞赛始于 2003 年,主

要面向解决一些算法类的问题。由于该比赛的解题形式不限制编程语言,并且参赛者需要提交身份信息,因此易于收集到可以用于描述创作者归属的相关源码文件。事实上,很多研究工作都通过采集该网站提供的程序数据形成具备作者信息的程序数据集;GitHub 是一个面向开源及私有软件项目的托管平台,其基于 Git 提供了代码托管服务,包括为开发者或团队提供订阅、代码片段分析等功能,帮助其高效率、高品质地进行代码编写。由于 GitHub 中的程序可以被方便的下载,且数量巨大,例如,  $\alpha\text{Diff}^{[28]}$ 、 $\text{DeepBinDiff}^{[29]}$ 便从中选取不同数量的项目作为训练测试数据源。

#### 3.2 单一工具/软件包

一些研究则选取单一的工具或软件包可执行文

件作为实验数据的来源, 比较典型的有 Busybox、OpenSSL 以及 cURL、Debian 软件包以及 GNU 软件包等。其中, Busybox 是一个集成了三百多个最常用 Linux 命令和工具的软件, 它将许多具有共性的小版本 UNIX 工具结合到一个可执行文件中。由于其占用的软硬件资源比其他常用工具小, 因此 Busybox 常用于嵌入式 Linux 领域, 可以用于构建根文件系统; OpenSSL 提供了一组用于安全通信软件库包, 被广泛应用在互联网的服务端、客户端上, 可确保信息在网络层秘密性传输。OpenSSL 可在 Windows、Linux 平台下使用, 是一套跨平台的软件库包; cURL 是一个利用 URL 语法在命令行下工作的文件传输工具, 它支持文件上传和下载, 是一个综合传输工具, 支

持 FTP、FTPS、HTTP、HTTPS 等多种通信协议; Debian 是目前世界最大的非商业性 Linux 发行版之一, 其对应的软件包中含有文档编辑、电子商务、游戏娱乐、软件开发等软件, 可通过软件包管理器或其他软件包工具进行获得。从 Debian 软件包库中收集软件包进行研究的工作有  $\alpha$ Diff<sup>[28]</sup>和 SAFE<sup>[30]</sup>等; GNU 软件包库提供了支撑 GNU 操作系统运行的各类软件包, 其内部的 Coreutils、Findutils、Diffutils、Binutils 被文中多个文献选取用来进行相关研究。其中, Coreutils 包含 Linux 下的 ls 等常用指令; Findutils 包含一系列用于查找文件的程序; Diffutils 是用来更新 RecyclerView 的工具; Binutils 是 GNU 下的一组二进制工具集。

表 3 本文涉及到的数据来源  
Table 3 Data Resources in This Paper

来源类型	数据集来源	使用文献
代码/软件平台	GCJ	Caliskan-Islam 等 <sup>[42]</sup> 、Abuhamad 等 <sup>[44]</sup> 、CPA <sup>[45]</sup> BinEye <sup>[46]</sup> 、Abuhamad 等 <sup>[47]</sup> 、Oba2 <sup>[53]</sup>
	GitHub	$\alpha$ Diff <sup>[28]</sup> 、DeepBinDiff <sup>[29]</sup> 、Bayrami 等 <sup>[40]</sup> 、Dauber 等 <sup>[41]</sup> 、Abuhamad 等 <sup>[44]</sup> CPA <sup>[45]</sup> 、BinEye <sup>[46]</sup> 、Abuhamad 等 <sup>[47]</sup> 、Kalgutkar 等 <sup>[49]</sup> Zhang 等 <sup>[50]</sup> 、Meng 等 <sup>[51]</sup> 、Wisse 等 <sup>[52]</sup>
	Debian 软件包库	$\alpha$ Diff <sup>[28]</sup> 、SAFE <sup>[30]</sup>
单一工具/ 软件包	Coreutils	DeepBinDiff <sup>[29]</sup> 、SAFE <sup>[30]</sup> 、Genius <sup>[31]</sup> 、VulSeeker <sup>[33]</sup> 、Kimberly Redmond 等人 <sup>[34]</sup> 、 InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup> 、Asm2Vec <sup>[37]</sup> 、VulSeeker-Pro <sup>[38]</sup>
	Findutils	DeepBinDiff <sup>[29]</sup> 、Kimberly Redmond 等人 <sup>[34]</sup> InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup>
	Diffutils	DeepBinDiff <sup>[29]</sup> 、Kimberly Redmond 等人 <sup>[34]</sup> InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup>
	Binutils	SAFE <sup>[30]</sup> 、Kimberly Redmond 等人 <sup>[34]</sup> InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup>
	BusyBox	Genius <sup>[31]</sup> 、VulSeeker <sup>[33]</sup> 、Asm2Vec <sup>[37]</sup>
固件镜像	OpenSSL	DeepBinDiff <sup>[29]</sup> 、SAFE <sup>[30]</sup> 、Genius <sup>[31]</sup> 、Gemini <sup>[32]</sup> 、VulSeeker <sup>[33]</sup> Kimberly Redmond 等人 <sup>[34]</sup> 、InnerEye <sup>[35]</sup> 、SimInspector <sup>[36]</sup> 、Asm2Vec <sup>[37]</sup> 、VulSeeker-Pro <sup>[38]</sup>
	cURL	SAFE <sup>[30]</sup> 、Asm2Vec <sup>[37]</sup> 、VulSeeker-Pro <sup>[38]</sup>
	固件镜像 (来自多个供应商)	Genius <sup>[31]</sup> 、Gemini <sup>[32]</sup> 、VulSeeker <sup>[33]</sup>

3.3 厂商提供的固件镜像

除了上述公共领域获取的数据信息之外, 收集不同厂商打包好的固件镜像被认为是最为行之有效的办法。例如 Genius<sup>[31]</sup>就从 ATT、Verizon、Linksys、D-Link、Seiki、Polycom、TRENDnet 等 26 个供应商处收集到涵盖 IP 摄像头、路由器、访问点以及第三方或开源的固件镜像。在此基础上, Gemini<sup>[32]</sup>、

VulSeeker<sup>[33]</sup>等方案也继续使用了一些固件镜像。在后续的章节中, 将进一步介绍这些数据如何被相关研究所使用来实现 IoT 设备程序的同源性分析。

4 IoT 程序相似性检测技术

相似性检测的主要参考依据是判断两个软件间所能达成的功能是否一致。在应用于漏洞检测的过

程中,若待测软件的片段与某些已知漏洞形成了“相似的功能”,则可以认为完成了漏洞相似性判断。本章将以典型的IoT固件程序分析为目标,对相似性智能检测过程中涉及到的关键技术依次进行介绍。具体来说,4.1节将按不同的二进制代码粒度介绍提取的可用于进行相似性检测的原始特征,4.2节介绍面向不同二进制代码粒度生成特征表示的方法,4.3节介绍特征表示之间相似度的计算方法。4.4节对这些方案的特点进行总结。

#### 4.1 可用于相似性检测的特征

在进行检测前,IoT固件需要被反汇编成二进制代码,这种代码本质上是用来描述特定体系结构的机器代码和数据的字节流。从这个原始输入开始,研究人员使用了许多方法来提取更高层次的信息,将这些信息作为二进制代码的原始特征。为了提升检测能力,一些研究还会提取两种或者三种代码粒度的原始特征。本节将分别对指令、基本块、函数等三种粒度的特征进行介绍。

##### 4.1.1 指令级特征

指令是计算机执行某种操作的基本单位,一条指令(汇编指令)通常由操作码和操作数两部分组成。在处理时,研究者通常将指令或令牌(指令的操作码及操作数)转化为one-hot向量,并以此作为它们的原始特征。为了提取指令级特征,首先需要将所有指令或令牌组成一个词汇表,并将词汇表的大小表示为one-hot向量的维度。例如,假设某个指令或令牌在词汇表中的序号为 $x$ ,则它的one-hot向量的第 $x$ 位为1,其余为0。在实际应用的过程中,SAFE<sup>[30]</sup>、Kimberly Redmond 等人<sup>[34]</sup>、InnerEye<sup>[35]</sup>以及SimInspector<sup>[36]</sup>都将指令转化为one-hot向量来提取指令级原始特征,DeepBinDiff<sup>[29]</sup>和Asm2Vec<sup>[37]</sup>则使用令牌来转化为one-hot向量。

##### 4.1.2 基本块级特征

基本块由程序中顺序执行的指令序列组成,一个基本块只存在一个入口和出口。基本块通常被使用以下方式定义:在程序执行时,从入口语句开始,在出口语句退出,其中不存在跳转与分叉汇合的情况。在有分支、转移语句的情况,将会以此为分界生成不同的基本块。根据收集到的文献,基本块级的原始特征可分为两种:指令的特征表示的组合以及基本块属性向量。

在基于指令特征表示的组合的研究方面,很多研究将基本块内所有指令嵌入后生成的特征表示进行组合作为原始特征。例如,SAFE<sup>[30]</sup>和InnerEye<sup>[35]</sup>中提取指令特征表示组成的序列,DeepBinDiff<sup>[29]</sup>将

基本块内所有令牌的特征表示进行汇总、运算,运算后得到的向量作为原始特征。

在基本块级的属性向量的研究方面,Genius<sup>[31]</sup>、Gemini<sup>[32]</sup>、VulSeeker<sup>[33]</sup>、VulSeeker-Pro<sup>[38]</sup>以及Luo等人<sup>[39]</sup>选取多个基本块属性,并统计在每个基本块中每个属性对应的值,将这些值组合成向量作为基本块级的原始特征。选取的属性包括但不限于字符串常量数量、数字常量数量、各个类型指令的数量以及调用的数量。例如,研究中选取了字符串常量数量、数字常量数量、算术指令数量这3个基本块属性。若某一基本块内有2个字符串常量、3个数字常量、1个算术指令,则向量[2,3,1]便是该基本块的原始特征。

##### 4.1.3 函数级特征

函数是实现某个特定功能的基本单元,可以被其他程序或代码引用。函数可以由多个基本块按照一定的逻辑顺序构成。在收集到的相似性检测相关文献中,我们发现提取的函数级原始特征包括原始字节和图两大类。对于原始字节组成的函数特征, $\alpha$ Diff<sup>[28]</sup>直接使用二进制代码的原始字节,以矩阵的形式作为二进制函数的原始特征。矩阵的大小为 $100 \times 100$ ,若函数小于10000字节,则用0补充,若函数大于10000字节,则舍弃多余的代码。

在以图作为原始特征的研究中,学者们利用工具提取了各类图作为函数级的原始特征,包括控制流图(Control Flow Graph, CFG)、数据流图(Data Flow Graph, DFG)和函数调用图。虽然研究所使用图的类别并不固定,但是他们都不约而同使用了CFG。例如,Genius<sup>[31]</sup>、Gemini<sup>[32]</sup>、VulSeeker-Pro<sup>[38]</sup>以及SimInspector<sup>[36]</sup>选取了CFG这一种图作为原始特征,CFG因带有函数的结构信息,所以经过嵌入可捕获函数结构信息。基于数据流的特点,DFG也可作为原始特征捕获函数行为方面的特征,而VulSeeker<sup>[33]</sup>则结合选取了CFG和DFG两类图进行研究。DeepBinDiff<sup>[29]</sup>添加了函数调用图来丰富CFG,这样可捕获函数结构特征和函数上下文信息。

在后续的研究工作中,很多研究采用了图嵌入技术获取特征表示。图嵌入输入的原始特征为函数级原始特征中的图与带有基本块特征的信息(例如基本块级原始特征、基本块的特征表示)相结合。例如,Genius<sup>[31]</sup>、Gemini<sup>[32]</sup>、VulSeeker-Pro<sup>[38]</sup>、SimInspector<sup>[36]</sup>将CFG与基本块属性向量或基本块特征表示结合,形成了属性控制流图(Attributed Control Flow Graph, ACFG);VulSeeker<sup>[33]</sup>将CFG、DFG结合,形成语义标签控制流图(Labeled Semantic Flow Graph, LSFG),再附加

基本块属性向量组成图嵌入的原始特征; DeepBinDiff<sup>[29]</sup>将 CFG、函数调用图结合, 形成过程间控制流图(Interprocedural Control Flow Graph, ICFG), 再附加指令特征表示组合组成图嵌入的原始特征。

## 4.2 面向不同代码粒度的特征表示方法

在提取了程序对应的原始特征后, 需要使用适当的表示方法对这些原始特征进行处理。通过挖掘众多特征背后的关联关系的方式形成专门的特征表示, 以此支持相似性检测。由于特征表示的精细程度与相似性检测的结果的正相关性质, 因而研究者的主要精力也主要集中在设计优秀的特征表示方案。当前, 特征表示通常通过 embedding 技术实现, 即将从 IoT 程序所提取的特征表示为一个向量, 随后对向量间的差异进行分析。

### 4.2.1 面向指令特征表示的研究工作

IoT 设备程序二进制代码被分解后, 将会形成面向多种架构的汇编语言表示。这些汇编语言的指令之后会被进行一定的处理, 以表示为 one-hot 向量。这些向量被嵌入 Word2vec 模型中进行学习, 利用嵌入后得到的信息来表示指令的语义特征。然而, 仅依靠指令嵌入得到的特征表示是不足以进行 IoT 固件相似性分析的。因此, 这些研究通常将获得指令特征表示作为 IoT 固件相似性分析的其中一个步骤, 后续再对指令特征表示进行运算、处理来分析 IoT 固件之间的相似性。

Word2vec 是一种通过训练学习将词转化为向量的方法。该方法支持两类型模型: skip-gram 模型和 CBOW 模型。skip-gram 模型的输入是一个中心词, 输出是最可能出现在该中心词上下文的词。而 CBOW 模型相反, 它将一个词语的上下文作为输入, 来预测这个词语本身。skip-gram 模型和 CBOW 模型的结构如图 3 所示。

两个模型都由输入层、隐藏层和输出层组成。每个词均可表示为长度为  $V$  的 one-hot 向量( $V$  代表整个词汇表的大小), 词的 one-hot 向量随后会被输入至输入层。隐藏层表示为一个大小为  $V \times N$  的矩阵  $W$  ( $N$  为词嵌入模型后得到的向量的维度)。隐藏层可以把  $V$  维的 one-hot 向量映射成为最终想要得到的  $N$  维词向量( $N$  远远小于  $V$ )。输出层的参数是一个大小为  $N \times V$  的矩阵  $W'$ 。经过隐藏层得到的词向量与  $W'$  进行矩阵计算后重新变为  $V$  维向量。通过 softmax 归一化, 向量的每一维将是词汇表对应的词与输入的词出现在上下文中的概率。在训练过程中, 输出还需要与样本数据进行比较, 利用损失函数进行传播来优化模型。最终, 模型训练完成后将得到了矩阵

$W$ 。  $W$  的每一行代表对应词汇表中这个词嵌入模型后的词向量。

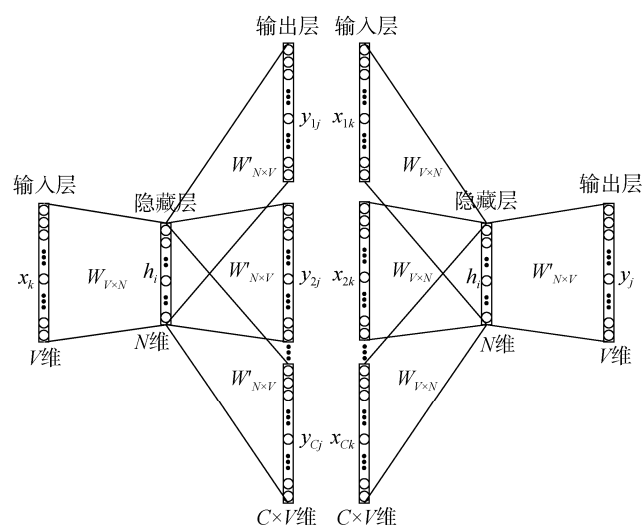


图 3 Skip-gram 模型(左)和 CBOW 模型(右)结构  
Figure 3 Skip-gram (Left) and CBOW (Right) Models

在应用于指令特征表示的研究中, 作者将指令视为词汇, 代码的基本块视为句子, 并将指令转化为向量来表示指令的语义特征。由于指令中经常使用常量、地址偏移、标签和字符串, 因此会带来 out-of-vocabulary 问题, 解决办法是在将指令转换为 one-hot 向量之前, 对指令进行预处理。有研究使用以下规则对训练数据集中的指令进行预处理: (1)常量值用 0 替换, 减号保留; (2)字符串面值被替换为 <STR>; (3)函数名替换为 FOO; (4)其他符号常量替换为 <TAG>。训练完成后, 通过将指令嵌入隐藏层的参数矩阵  $W$ , 得到的向量便可表示指令的语义特征。另外, 在 DeepBinDiff<sup>[29]</sup>中, 作者将指令的操作码和操作数(称为“令牌”)视作单词, 经过训练, 得到每个操作码和操作数的特征表示。总结来说, 按照嵌入的模型类型分类, SAFE<sup>[30]</sup>、InnerEye<sup>[35]</sup>和 SimInspector<sup>[36]</sup>采用的是 skip-gram 模型, Kimberly Redmond 等<sup>[34]</sup>和 DeepBinDiff<sup>[29]</sup>则采用 CBOW 模型。

### 4.2.2 面向基本块特征表示的研究工作

当前研究对基本块特征进行了两种表示方式的探索: 一种是将指令的特征表示组成序列嵌入长短期记忆(Long Short-Term Memory, LSTM)神经网络中; 一种是将 4.1.3 节提到的 ICFG 与指令特征表示组合成原始特征嵌入 DeepWalk 算法中。

#### 4.2.2.1 利用 LSTM 神经网络获得基本块表示

LSTM 神经网络是在循环神经网络(Recurrent Neural Network, RNN)基础上进行的一种改进, 它可以解决 RNN 无法处理长距离依赖的问题。解决该问



题的原理是 LSTM 设计了记忆细胞的概念, 使得 LSTM 具备了选择性记忆的能力, 可以选择记忆重要的信息, 过滤掉噪声信息, 减轻信息传递的负担。

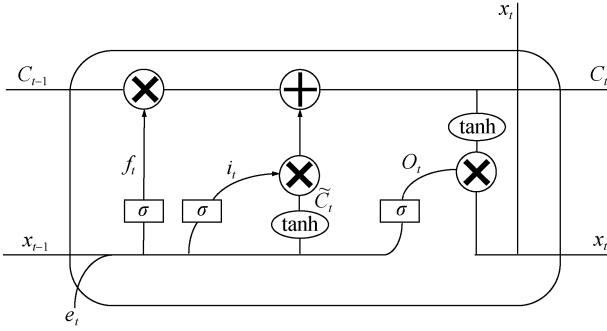


图 4 LSTM 网络链的时间步结构

Figure 4 The Repeated Module in a Standard LSTM

LSTM 基本形式是由多个相同的时间步组成的神经网络链。链的每个时间步的结构如图 4 所示。其中, 输入有三个:  $t$  时刻的输入  $e_t$ 、 $t-1$  时刻的输出  $x_{t-1}$  以及  $t-1$  时刻的记忆细胞状态  $c_{t-1}$ 。输出有两个:  $t$  时刻的输出  $x_t$  和  $t$  时刻的记忆细胞状态  $c_t$ 。每个时间步由遗忘门、更新门和输出门三部分组成。

(1)遗忘门: 遗忘门的目的是通过 Sigmoid 层来决定删除上一段时间细胞状态中的哪些信息, 表示为:

$$f_t = \text{sigmoid}(W_f e_t + U_f x_{t-1} + v_f)$$

(2)更新门: 更新门将输入到时间步中的新信息同样通过 Sigmoid 层来将一些无用的信息过滤掉, 用  $i_t$ 、 $\tilde{c}_t$  表示, 其中:

$$i_t = \text{sigmoid}(W_i e_t + U_i x_{t-1} + v_i)$$

$$\tilde{c}_t = \tanh(W_c e_t + U_c x_{t-1} + v_c)$$

通过 LSTM 时间步的前两个部分, 记忆细胞状态被更新为

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$

它既包含当前时间添加的信息, 也包含一些历史信息。

(3)输出门: 输出门的参数  $o_t$  表示为:

$$o_t = \text{sigmoid}(W_o e_t + U_o x_{t-1} + v_o)$$

因此时间步在  $t$  时刻最后的输出为:

$$x_t = o_t \odot \tanh(c_t)$$

其中,  $W_f$ 、 $W_i$ 、 $W_c$ 、 $W_o$  是权重矩阵,  $v_f$ 、 $v_i$ 、 $v_c$ 、 $v_o$  是偏置向量, 这些参数都是在训练过程中学习得到的。

在获取基本块特征表示的研究工作中, 研究者通常利用 4.2.1 节的方法得到基本块内所有指令的特征表示, 再将它们组成序列, 然后依次输入到

LSTM 神经网络的时间步中。代入模型, 即将指令特征表示序列的第  $t$  个向量输入到  $t$  时刻的时间步中。经过传递, 每个时间步的输出都包含了之前所有输入的指令特征表示的信息。LSTM 神经网络最后一个时间步的输出即可表示整个基本块的特征。

在实际的例子中, InnerEye<sup>[35]</sup>采用的 LSTM 神经网络模型如图 5 上边所示, 指令的特征表示信息不仅在神经网络中从左至右传递, 同时从下至上传递。随着指令特征表示的输入, 经过  $n$  层迭代, 神经网络累计和传递更丰富的语义信息。在最后一个指令特征表示输入并达到最后一层时, LSTM 神经网络提供了整个基本块的语义特征; SimInspector<sup>[36]</sup>采用双向 LSTM(BiLSTM)神经网络, 将指令特征表示的序列分别正向、逆向两次嵌入 LSTM 神经网络, 如图 5 下边所示。在经过  $n$  层迭代后, 方案将两次嵌入最终得到的输出进行拼接, 拼接结果即可表示基本块特征。

#### 4.2.2.2 利用 DeepWalk 获得基本块表示

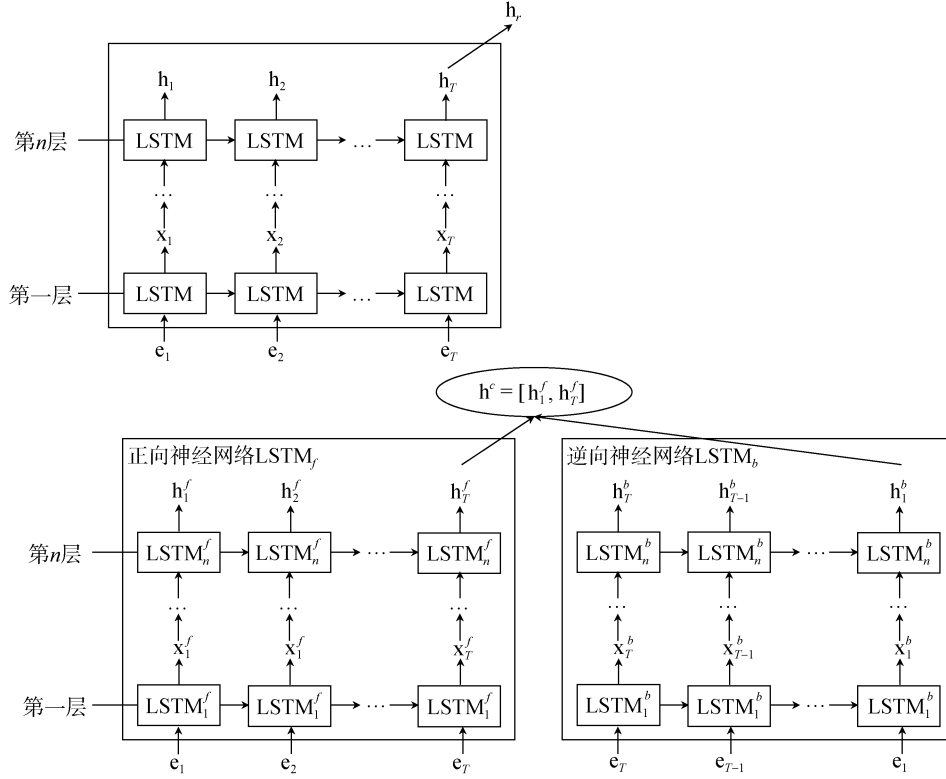
DeepWalk 算法是一种在线图嵌入学习算法, 它可以将随机游走和 Word2vec 两种算法相结合。首先 DeepWalk 利用随机游走算法在图中选取一些连续的顶点, 组成一连串随机游走的序列, 将每个序列视为一个处理单元, 序列里的每个顶点视作单词。随后利用 Word2vec 模型训练出表示顶点的向量。

如 4.1.3 节所述, 研究者们利用图嵌入技术进行特征表示提取, 通常会使用图与带有基本块特征的信息相结合作为原始特征来嵌入机器学习算法或神经网络中。例如, DeepBinDiff<sup>[29]</sup>使用 ICFG 进行图嵌入, 它是由调用图与每个函数的控制流图相结合的一种图。嵌入的基本块特征信息为 4.1.2 节提到的令牌特征表示汇总。作者将进行相似性比较函数的 ICFG 进行合并, 然后与基本块特征信息一同输入由 DeepWalk 算法改编的 TADW 算法中。在处理过程中, 基本块被表示为图的一个顶点。方案首先在 ICFG 中进行随机游走, 以此产生由相互连接的基本块组成的序列。随后方案将序列作为句子, 基本块作为单词, 通过套用 Word2vec 模型, 即可生成基本块的特征表示。此时的基本块特征表示不仅包含本身的语义信息, 还包含来自 ICFG 的结构信息, 表示了基本块的语义与结构特征。

#### 4.2.3 面向函数特征表示的研究工作

##### 4.2.3.1 基于图嵌入类神经网络

很多研究将 CFG 与一些其他数据结合, 作为原始特征, 嵌入到神经网络中, 得到了函数的特征表示。例如, Gemini<sup>[32]</sup>、VulSeeker<sup>[33]</sup>、SimInspector<sup>[36]</sup>

图 5 LSTM 神经网络模型图<sup>[35-36]</sup>Figure 5 Structure of LSTM Neural Network<sup>[35-36]</sup>

以及 VulSeeker-Pro<sup>[38]</sup>将由 structure2vec 改编的神经网络进行嵌入; Genius<sup>[31]</sup>通过谱聚类, 为图生成码本, 再利用映射运算, 得到函数的特征表示。

Structure2vec 神经网络是一种注重顶点之间结构关系相似性的图嵌入网络。Gemini<sup>[32]</sup>、SimInspector<sup>[36]</sup>和 VulSeeker-Pro<sup>[38]</sup>都是基于该网络进行了优化改编。structure2vec 网络模型工作原理如图 6 上边所示。研究者们将二进制函数对应的 ACFG 表示为  $g$ ,  $g = \langle V, E \rangle$ , 其中  $V$  和  $E$  分别是  $g$  中的顶点和边的集合, 而  $g$  中的每个顶点  $v \in V$  以二进制函数基本块的特征描述为向量, 记作  $x_v$ 。Structure2vec 神经网络可为每个顶点  $v \in V$  都计算一个多维特征  $\mu_v^{(T)}$ ,  $\mu_v^{(T)}$  是通过特定于顶点  $v$  的特征  $x_v$  根据图拓扑递归聚合得到的。每个顶点处初始化为  $\mu_v^{(0)} = \mathbf{0}$ , 并在每次迭代时更新为  $\mu_v^{(t+1)}$ , 如下式所示:

$$\mu_v^{(t+1)} = \tanh \left( W_1 x_v + \sigma \left( \sum_{v \in N(v)} \mu_v^{(t)} \right) \right), v \in V$$

其中,  $N(v)$  表示图  $g$  中顶点  $v$  的邻居集合。在  $T$  次迭代后终止更新过程, 每个嵌入的顶点  $v$  都将获得一个  $\mu_v^{(T)}$ 。将每一个顶点  $v$  的多维特征  $\mu_v^{(T)}$  进行聚合便生成了二进制代码的图嵌入即特征表示  $\phi(g)$ :

$$\phi(g) = W \left( \sum_{v \in V} \mu_v^{(T)} \right)$$

在 VulSeeker<sup>[33]</sup>中使用 LSFG 嵌入神经网络。由于 LSFG 包含了二进制函数的 CFG 和 DFG, 因此在每次迭代时,  $\mu_v^{(t+1)}$  需要考虑通过控制依赖关系指向  $v$  的顶点集合  $C(v)$  以及通过数据依赖关系指向  $v$  的顶点集合  $D(v)$ , 如图 6 下边所示。而在每次迭代时,  $\mu_v^{(t+1)}$  将根据下面公式进行计算:

$$\mu_v^{(t+1)} = \tanh(W_1 x_v + \sigma_c \left( \sum_{v \in C(v)} \mu_v^{(t)} \right) + \sigma_d \left( \sum_{v \in D(v)} \mu_v^{(t)} \right)), v \in V$$

Genius<sup>[31]</sup>通过谱聚类算法对数据集中二进制代码生成的 ACFG 进行无监督学习, 为 ACFG 生成码本。在实施过程中, 所有 ACFG 被分为若干个集合, 每个集合中包含一个质心节点。质心节点为与所在集合中所有其他 ACFG 的距离最小的 ACFG。所有质心节点的集合构成一个码本。最后, 方案利用 VLAD 编码计算 ACFG 与码本中每一个聚类类别的相似性距离, 并以此组合作为该 ACFG 的特征向量, 也就是二进制函数的特征表示。

#### 4.2.3.2 利用循环类神经网络获得函数嵌入

4.2.2.1 节提到利用 LSTM 神经网络可以获得基本块的嵌入来表示其特征, 有的研究表示同样可以利用该神经网络得到固件镜像的特征表示。例如, Luo 等人<sup>[39]</sup>将函数内基本块的属性向量作为每一个

时刻的输入嵌入 LSTM 网络中。方案强调在不打乱所选基本块序列顺序的情况下,以随机的方式选择固定数量的基本块。经过实验,最终将包括  $t$  时刻在内的前 5 个基本块属性向量作为该时刻的输入,通过神经网络预测出输出  $t+1$  时刻的基本块属性向量。最终,输出隐藏的记忆细胞状态向量作为固件镜像的嵌入,用以表示特征。

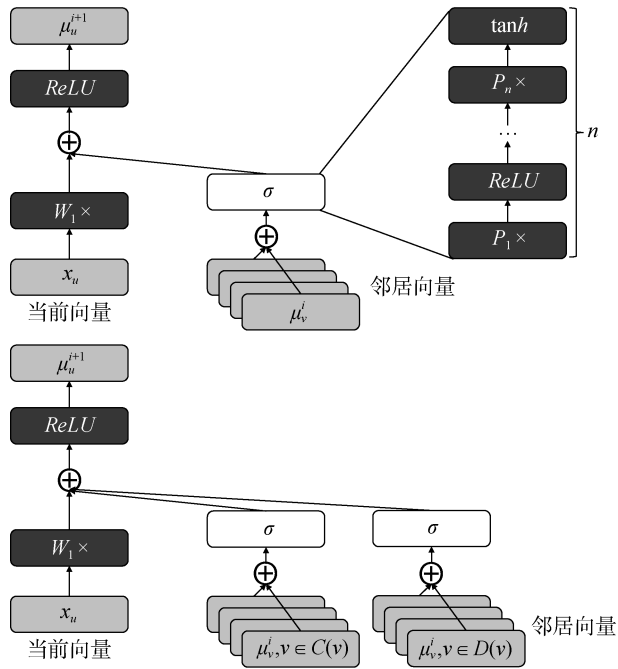


图 6 Structure2vec 神经网络模型<sup>[32-33]</sup>  
Figure 6 The Structure2vec Neural Network Model<sup>[32-33]</sup>

SAFE<sup>[30]</sup>方案借助双向 RNN 来获得函数的特征表示。RNN 是由多个结构相同的循环单元组成的链式神经网络,适合以序列类型的数据作为输入。而双向 RNN 的基本思想是将输入序列向前、向后输入 RNN 中。在实施过程中,SAFE 首先通过 4.2.1 的方法获得函数中所有指令的特征表示,然后组成序列输入双向 RNN 中。每个指令特征向量  $\vec{l}_i$  输入到双向 RNN 后,得到一个大小为  $\mu$  的向量  $\vec{h}_i$ 。至此,所有  $\vec{h}_i$  可组成一个具有函数语义特征的矩阵。这个矩阵经过加权和运算与降维,可得到一个可表示函数特征的向量。

虽然取得了一定的效果,但文献[31-33,38-39]在提取函数/固件镜像特征表示中还存在着一定的缺陷。具体来说,它们都选择了基本块属性向量作为原始特征进行了嵌入。这些基本块属性向量中的属性是手动选择的,存在很大的不稳定性。同一功能的这些属性在不同平台下可能会不同,这可能会导致检

测精度较低。所以另外一些研究将原始的二进制代码或者代码的特征表示等原始数据嵌入模型中,来生成特征表示。

#### 4.2.3.3 利用卷积神经网络获得函数嵌入

在典型方案中,  $\alpha$ Diff<sup>[28]</sup>将二进制函数的原始字节嵌入卷积神经网络(Convolutional Neural Networks, CNN)中,得到了函数的特征表示。作者提出的 CNN 由 8 个卷积层、8 个批处理归一化层、4 个最大池化层和 2 个全连接层组成。整个模型采用线性整流函数 ReLU 作为非线性激活函数。该 CNN 模型以大小为  $100 \times 100 \times 1$  的张量  $T$  作为输入,并输出一个 64 维嵌入向量(即函数特征表示)。

#### 4.2.3.4 利用词嵌入模型获得函数嵌入

在模型优化方面, Asm2Vec<sup>[37]</sup>采用的是 PV-DM 模型,该模型是 Word2vec 模型的拓展,它可以联合学习每个单词和每个段落的向量表示。在每一步, PV-DM 模型执行一个多类预测任务。它将当前段落映射到一个基于段落 ID 的向量,并将上下文中的每个单词映射到一个基于单词 ID 的向量。PV-DM 模型可以平均这些向量,并通过 softmax 分类从词汇表中预测目标词。应用于函数特征表示的提取工作中,将指令的操作码和操作数视为单词,将函数视为段落。当前指令的上下文与函数的映射 ID 即为令牌和函数的 one-hot 向量,将它们作为原始特征输入 PV-DM 模型,当前指令的令牌特征表示为该模型的输出。通过训练,不仅可以学习每个令牌的特征表示,还可以学习函数的特征表示。

#### 4.2.4 利用 Siamese 架构训练模型

很多研究将 Siamese 架构与机器学习模型或神经网络(以下简称“嵌入模型”)结合使用来训练参数。一般地, Siamese 架构包含了两个可在顶部进行连接的嵌入模型。在模型训练的过程中,每个嵌入模型的输入对应一个 IoT 设备程序,并提取原始特征作为其输入,并输出其特征表示。Siamese 架构的最终输出是两个特征表示的余弦距离。此外,两个嵌入模型具有相同的参数集,这使得表示结果具备相同的比照系。整个体系结构如图 7 所示。

为了使相关检测技术具备一定的跨平台性质,有学者们在训练样本的选择上设计方案。例如,他们从数据集中选择一个 IoT 固件,然后提取它在不同平台下的原始特征  $x$  与  $x_1$ 。接着,再从数据集中随机选择另一个 IoT 固件并提取其原始特征  $x_2$ 。然后,将以下两组信息作为训练样本:即标签为 +1 的  $(x, x_1)$  和标签为 -1 的  $(x, x_2)$ 。通过很多对这种形式的信息作为训练样本,使得同一源码在不同平台下编译的 IoT 固件

具有相似的特征表示,从而达到跨平台相似性检测的效果。

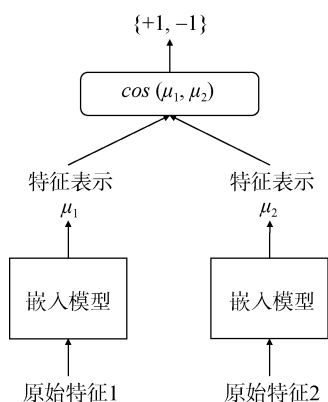


图7 Siamese 体系结构图  
Figure 7 Structure of Siamese

### 4.3 相似性计算方法

在获得特征表示后,便可以利用特征表示计算二进制代码之间的相似性。典型的计算方法如下:

**欧氏距离及余弦角距离:** Genius<sup>[31]</sup>在得到能表示二进制代码的编码特征后,通过 LSH 将编码特征投影到空间中的一个点。两个二进制代码间的相似性则是通过计算欧氏距离及余弦距离这两个经典距离度量得到。

**余弦距离:** 文献[30,32-33,36-37]以及[38]会直接通过利用余弦距离的方式计算二进制代码之间的相似性。在得到二进制代码  $B_1$ 、 $B_2$  的嵌入  $\phi(B_1)$ 、 $\phi(B_2)$  后,通过以下公式获得相似度。

$$\text{sim}(B_1, B_2) = \cos(\phi(B_1), \phi(B_2)) = \frac{\langle \phi(B_1), \phi(B_2) \rangle}{\|\phi(B_1)\| \cdot \|\phi(B_2)\|}$$

作为一个典型的实例, VulSeeker-Pro<sup>[38]</sup>在利用余弦函数进行相似性计算后,选取了相似度最高的  $M$  个函数,利用仿真引擎进行函数模拟,再选取更精确的相似度排在前  $N$  的函数。

**多种距离综合计算:**  $\alpha\text{Diff}$ <sup>[28]</sup>中获取了二进制代码的三种语义特征:函数语义特征(通过 4.2.3.3 节的方法获得)、函数调用语义特征、模块交互语义特征。为了计算两个二进制函数  $I_q$  和  $I_c$  的相似度,首先得到  $I_q$  和  $I_c$  的三种语义特征;然后再分别计算  $I_q$  和  $I_c$  的三种语义特征的距离,得到  $D1(I_q, I_c)$ 、 $D2(I_q, I_c)$  和  $D3(I_q, I_c)$ ;最后将这三个距离进行综合计算,得到的结果表示  $I_q$  和  $I_c$  的相似度。

**路径相似度得分:** InnerEye<sup>[35]</sup>首先利用 4.2.2.1 节的方法获取每个基本块的特征表示;然后, InnerEye 将查询代码组件  $Q$  的 CFG 分解为多条路径。对于来

自  $Q$  的每一条路径, InnerEye 将其与来自目标程序  $T$  的多条路径进行比较并采用以基本块为序列元素的最长公共子序列动态规划算法计算路径相似度得分;最后,通过探索多个路径对, InnerEye 可共同计算一个相似度得分,以表示查询代码组件在目标程序中被重用的可能性。

**k-hop 贪婪匹配算法:** DeepBinDiff<sup>[29]</sup>通过 4.2.2.2 节的方法,将两个被比较的二进制代码的 ICFG 进行合并,得到了图中每个基本块的嵌入。之后其利用 ICFG 上下文信息,根据 k-hop 邻居中已经匹配的基本块特征表示计算出的相似性找到匹配的基本块。

**全连接神经网络:** Luo 等人<sup>[39]</sup>利用两个全连接层计算二进制代码之间的相似性。具体而言,在得到两个二进制代码的特征表示后,他们利用 ReLu 激活函数与最大池化策略进行处理,将特征表示被压平成一个矢量;接着其利用第一个全连接层的输出计算诱导距离,再将该诱导距离送入第二个全连接层,最终输出二进制代码之间的相似性。

## 4.4 方案分析

### 4.4.1 特点分析

**基于指令级特征的方案。**获取指令特征表示的优点是能够最大程度地保留二进制代码的语义信息,并将指令与指令间的依赖关系提取到特征表示中。但是仅凭获取指令特征表示无法比较程序之间的相似性,很多研究将获取指令特征表示的方法与其他技术结合来进行 IoT 设备程序的同源性检测。例如, Kimberly Redmond 等人<sup>[34]</sup>的方案不仅能够很好的提取指令本身的语义信息,还能容忍跨平台架构下的语法差异。该方案采用联合学习的方法,生成了跨体系结构的指令特征表示,使得不同平台下的同一指令通过模型的训练都能获得等价的语义信息。

**基于函数级特征的方案。**提取函数特征表示的方案可分为图嵌入方案<sup>[31-33,38-39]</sup>和原始字节嵌入方案<sup>[28,37]</sup>。为了实现可扩展性和高精度,很多研究采用了图嵌入技术,从 CFG 中学习函数的特征表示。例如, Genius<sup>[31]</sup>利用无监督学习中的谱聚类算法来为数据集的所有 ACFG 训练一个码本,再通过 VLAD 编码生成函数的特征表示,但是生成码本是一个非常耗时的过程,导致系统开销过大。为了解决相似性检测效率低下的问题,后续的研究使用神经网络将图转化为函数的特征表示。例如, Gemini<sup>[32]</sup>、 VulSeeker<sup>[33]</sup>、 VulSeeker-Pro<sup>[38]</sup>将图嵌入到由 Structure2vec 改编的深度神经网络中,以获得更好的检测性能以及更少的模型训练时间。另外, VulSeeker<sup>[33]</sup>还额外将 DFG 嵌入神经网络一同进行训练,能

够获得更加准确的函数语义信息, VulSeeker-Pro<sup>[38]</sup>在后端集成了仿真引擎, 在更少的系统开销下进一步提高了相似性计算的精度。

然而, 基于图嵌入的函数特征表示获取方案会将基本块属性向量一同进行嵌入, 这些向量信息过分依赖专家的先验知识, 选取的基本块属性稍有变化就会使函数的特征表示发生极大改变, 降低了相似性检测的准确性。 $\alpha$ Diff<sup>[28]</sup>、Asm2Vec<sup>[37]</sup>则使用了原始字节嵌入技术, 前者将代码按字节以矩阵为单位嵌入卷积神经网络中获取函数的特征表示, 后者以二进制代码令牌(操作码和操作数)为单位嵌入 PV-DM 模型获取函数的特征表示。但  $\alpha$ Diff 除了深度学习算法外还利用了其他技术获取函数间、模块间的语义特征来计算相似性, Asm2Vec 则不支持跨平台的相似性检测。

**基于指令+基本块级特征的方案。**对于获取基本块的特征表示, DeepBinDiff<sup>[29]</sup>、InnerEye<sup>[35]</sup>首先获取基本块中所有指令的特征表示, 再将其进行加工后通过机器学习、深度学习得到基本块的特征表示。

提取基本块的特征表示可以获得更细粒度的代码语义信息, 因为在部分代码剽窃、漏洞传播的场景中, 并不是复制了整个函数的代码, 所以提取函数的特征表示来进行相似性检测过于局限。同时, 这类型的方案在获取基本块的特征表示后, 通过其他算法还可以比较函数之间的相似性。基于上面的优势, InnerEye 解决了跨架构下的代码包含检测方案, DeepBinDiff 则比 InnerEye 具有更高的性能, 并能提取更多的代码依赖信息。

**基于指令+函数级特征的方案。**SAFE<sup>[30]</sup>在相似性计算之前提取了两种特征表示, 即指令特征表示、函数特征表示。首先利用 word2vec 模型获取函数中所有指令的特征表示, 再将其进行加工后嵌入双向 RNN 中, 得到函数的特征表示。SAFE 与直接利用图嵌入、原始字节嵌入的方案相比, 其没有将主观的先验知识作为嵌入信息, 有更好的准确性; 同时, 仅利用机器学习、深度学习算法就可以获得能够进行相似性比较的特征表示, 无需通过另外的算法获得其他特征, 可更加高效地完成相似性检测任务。

表 4 相似性检测方案对比  
Table 4 Comparisons of Approaches for Similarity Analysis

文献名称	数据集数据类型	数据集来源	特征表示 粒度	原始特征	特征表示方法	相似度计算 方法	描述	
$\alpha$ Diff <sup>[28]</sup>	固件镜像	GitHub	函数	函数原始 字节	卷积神经网络	多种距离综 合计算	在不受专家经验知识 的干扰下, 从二进制 函数的原始字节中提 取特征表示	
		Debian						
DeepBinDiff <sup>[29]</sup>	固件镜像	GitHub	令牌	令牌 one-hot 向量	word2vec	k-hop 贪婪 匹配算法	提出了一种无监督的 表示学习技术来生成 基本块的特征表示	
		Coreutils						CFG
		Findutils	基本块	调用图	DeepWalk 算法			
		Diffutils		令牌特征表 示汇总				
		OpenSSL						
SAFE <sup>[30]</sup>	固件镜像	Debian	指令	指令 one-hot 向量	word2vec	余弦函数	提出了一种基于注意 力机制的神经网络的 函数嵌入体系结构	
		Coreutils						
		Binutils	函数	指令特征表 示序列	双向 RNN			
		cURL						
		OpenSSL						
Genius <sup>[31]</sup>	汇编函数	其他来源	函数	基本块属性 向量	谱聚类+VLAD 编码	欧氏距离及 余弦 距离	利用基于控制流图的 方法来解决漏洞搜索 技术中的可伸缩问题, 提出了一种适合 IoT 系统的搜索方案	
		Coreutils						
		BusyBox						
	固件镜像	厂商	基本块属性 向量	structure2vec 神经网络	余弦函数			基于神经网络的方法 来生成二进制函数的 特征表示
		OpenSSL						
Gemini <sup>[32]</sup>	汇编函数	OpenSSL	函数	基本块属性 向量	structure2vec 神经网络	余弦函数	基于神经网络的方法 来生成二进制函数的 特征表示	
	ACFG	厂商						
	固件镜像	厂商						

续表

文献名称	数据集数据类型	数据集来源	特征表示 粒度	原始特征	特征表示方法	相似度计算 方法	描述			
VulSeeker <sup>[33]</sup>	汇编函数	Coreutils	函数	CFG	structure2vec 神经网络	余弦函数	通过 LSFG 以及基于语义感知的深度神经网络来生成二进制函数的特征表示			
		BusyBox								
		OpenSSL								
	基本块	Coreutils		DFG						
		BusyBox		基本块属性 向量						
固件镜像	OpenSSL									
	厂商									
Kimberly Redmond 等 <sup>[34]</sup>	基本块	Findutils	指令	指令 one-hot 向量	word2vec	-	构建了统一的跨架构指令嵌入模型, 能够解决跨架构的显著语法差异问题			
		Diffutils								
		Binutils								
		OpenSSL								
		Coreutils								
InnerEye <sup>[35]</sup>	基本块	Findutils	指令	指令 one-hot 向量	word2vec	路径相似度 得分	利用神经机器翻译领域的成功经验解决了二进制代码相似性比较的问题			
		Diffutils								
		Binutils						指令特征表 示序列	LSTM 神经网络	
		OpenSSL								
		Coreutils								
SimInspector <sup>[36]</sup>	基本块	Findutils	指令	指令 one-hot 向量	word2vec		将神经机器翻译与图嵌入相结合, 具有较好的准确性与效率			
		Diffutils								
		Binutils						指令特征表 示序列	BiLSTM 神经 网络	
		OpenSSL								
		Coreutils								
Asm2Vec <sup>[37]</sup>	固件镜像	BusyBox	函数	令牌、函数 one-hot 向量	PV-DM 模型	余弦函数	使用设计的模型为二进制代码构建特征表示向量			
		OpenSSL								
		cURL								
		其他来源								
		Coreutils								
VulSeeker-Pro <sup>[38]</sup>	汇编函数	OpenSSL	函数	CFG	structure2vec 神经网络	余弦函数+ 仿真引擎	在后端集成了语义仿真引擎, 可代替工程师的人工识别工作			
		cURL		基本块属性 向量						
		其他来源								
		固件镜像						其他来源	基本块属性 向量	LSTM 神经网络
				其他来源						
Luo 等人 <sup>[39]</sup>	固件镜像	其他来源	固件 镜像	基本块属性 向量	LSTM 神经网络	全连接神经网络	基于 LSTM 神经网络捕获基本块的一般信息和依赖信息, 使检测过程更加高效			

基于指令+基本块+函数级特征的方案。SimInspector<sup>[36]</sup>是一种特殊的图嵌入方案, 该方案共提取了三种特征表示, 分别是指令特征表示、基本块特征表示、函数特征表示。SimInspector 首先采用与 InnerEye<sup>[35]</sup>相似的方法提取了基本块的特征表示, 然后将这些特征表示代替基本块属性向量与 CFG 嵌入神经网络中获得函数的特征表示。该方案与传统图嵌入方案相比, 因为没有主观先验知识信息作为嵌入, 提高了相似性检测的准确性。同时, 最终获取的函数特征表示携带了代码的结构特征。

4.4.2 适配性分析

通过对收集的文献进行总结, 如表 5 所示, 绝大多数研究支持跨平台的代码相似性分析任务, 其中包括 X86、ARM、MIPS 架构。大多数文献在实验中使用 GCC 或 clang 编译工具对数据集进行编译后, 即可进行特征的提取以及后续工作, 因此只要能够将 IoT 固件通过编译器转变为汇编语言则可以进行相似性检测。GCC 支持多种计算机体系结构(X86、ARM、MIPS 等)芯片, 同时已在多种硬件平台上进行了适配。Clang 是为 C、C++、Objective-C、Objective-C++设计的非通用语法解析器。文献[28]、

[36]和[38]仅使用 GCC 作为编译工具, 预计可支持 III 类设备的相似性检测。文献[31-35]使用 clang 作为编译工具, 预计可支持 I 类、II 类、III 类设备的相似性检测。文献[29]使用 C、C++ 程序作为训练集, 预计也可支持 I 类、II 类、III 类设备的相似性检测。

表 5 相似性技术适配性

Table 5    Applicability of Similarity Analysis Technology					
文献名	支持架构	编译工具	预期可支持 设备类型		
			I	II	III
文献[28]	X86	GCC	/	/	✓
文献[36]	ARM				
文献[29]	-	-	✓	✓	✓
文献[30]					
文献[37]	-	-	/	/	/
文献[39]					
文献[31]	x86	GCC			
文献[32]	ARM				
文献[33]	MIPS	clang	✓	✓	✓
文献[34]	X86	clang			
文献[35]	ARM				
	x86				
文献[38]	ARM	GCC	/	/	✓
	MIPS				

备注: “-”表示文献中未提及

5 程序创作者分析检测技术

程序漏洞的同源性的判定除了可借助其内部结构进行相似性检测, 还可以从作者归属权(authorship attribution)的角度出发, 通过分析目标程序代码是否为同一个原作者的方式确定出他们的同源性。在软件领域, 作者归属最初的应用是在抄袭检测, 早在上世纪 80 年代, Oman 等<sup>[54]</sup>研究人员根据各种代码组件的分析, 开始检查作者的编程风格。在漏洞同源性检测方面, 利用该技术同样可以从创作者角度快速定位其编写的漏洞程序代码(或片段)是否部署在多个待检测软件中, 以此为安全人员从作者习惯角度检查和封堵漏洞提供不同维度的参考。本文接下来将对近几年中主流研究对作者风格特征的划分以及在这些划分基础上所使用的不同处理方法进行介绍, 并将这些工作通过表 10 统一进行梳理。

5.1 特征类别

特征选择是构建待判定代码与作者联系的关键, 作者的编程写作习惯通常会以某种形式被嵌入到他所编写各个程序中去。因此, 将这些写作习惯逐渐的

提取出来, 将有助于找出不同作者的创作风格来完成对不同作者的描述。对于一个程序代码而言, 可以从多种角度进行特征的提取和识别。当前, 许多研究都对所能提取的特征进行了仔细的研究, 结合 Kalgutkar 等人<sup>[23]</sup>的报告, 本节将对文献所涉及的特征类型进行介绍。

5.1.1 词汇特征

词汇特征(Lexical Features)是指直接从源文件或二进制文件中提取的若干符号(token), 通常以特殊的字符串或表示序列组成。引入这种特征的最大原因在于有研究者认为工程师在进行不同程序代码的编写时, 更倾向于(或无意识的)使用过去惯用的词汇来命名函数、变量或是其他组件。

值得注意的是, 词汇特征通常不具备特定的语言含义, 可以认为是一种更为纯粹的符号表示。因此, 这种特征的使用和提取方式都较其他特征简单而直接。例如, 为了从数学角度表征词汇特征, 研究者通常结合计数的方法予以体现。不仅如此, 不与特殊语义挂钩的特性使得针对词汇特征的提取工具不仅可以在跨语言环境的基础上进行使用, 还甚少受到程序形式(如: 固件、源码)的影响。特征提取的易于实现也使得研究者可以将精力集中在同源性分析的数据处理以及算法中, 可大幅提升效率。常见的词汇特征如表 6 所示。

表 6 典型词汇特征

Table 6 Typical Lexical Features		
特征维度	描述	备注
行长度	一行代码的长度	单位: 字符、单词、词组
代码行数	代码的行数	单位: 函数、类、文件、程序
操作码数量	读/写/移动等指令的数量	同上
变量数量	变量的数量	同上
词语	某个单词出现的频率	同上
符号	某个符号出现的频率	同上
特殊符号	符号序列出现的频率	同上
序列		
(N-gram)		
函数/类名	特殊函数/类名	字符属性
标识符	特殊的标记	字符属性
空格	空格的数量	风格/布局
空行	空行的数量	风格/布局

在众多的词汇特征中, 以 N 个符号序列作为特征(N-gram)是一种广泛采用的特征表示形式。结合 Zhang 等人在文献[50]的描述, N-gram 即程序中连续 N 个条目 (Item) 组成的一个序列。

条目可以用以空格分隔的任意字符串表示, 且可以是任意关键字(keywords)、操作符(operators)、语句(statement)、用户定义的标识等等。例如, 设某一程序由  $p_1 p_2 p_3 \dots p_l$  组成, 且以空格分隔, 则经过 N-gram 分隔后的序列如图 8 所示。此外, N-gram 语句分隔方法的步长也可以根据需要进行修改, 以应对不同的研究需求。

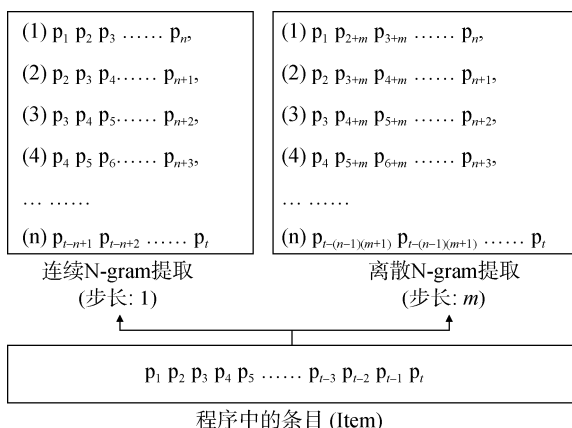


图 8 N-gram 策略示例

Figure 8 Example of N-gram strategy

### 5.1.2 句法特征

句法特征(Syntactic Features)是在词汇特征基础上的一轮语义化改进。这种特征主要用于描述词汇中的各个 token 符号在句子中到底是如何被程序原作者组织和使用的。将这些信息作为特征的原因在于绝大多数的研究者假设工程师都有自己难以舍弃的写作习惯和方式方法, 例如更喜欢使用特殊的判断句式(如: 用 switch-case 替代 if-else)。这种特征的表示同样可以借助于统计方法来映射作者习惯。由于句法特征难以受到函数/变量重命名等方式的影响, 因此从某种程度上来说, 它比词汇特征有着更好的稳定性, 这使得其对于作者归属描绘更加具备可靠性。常见的句法特征如表 7 所示。

一种典型的句法特征将通过使用抽象语法树 (Abstract Syntax Tree, AST) 技术进行实现。文献[35]和[36]都对这种类型的特征进行了较为详细的介绍。具体来说, AST 将程序代码表征为了一种树结构。程序中的各个组成成分, 例如数学操作以及变量的定义, 都会作为一个个节点(Node)被映射到树上。

典型的以 fooo 函数构建的 AST 如图 9 所示, 操作符(=、||)、函数调用以及 if-else-return 语句等都会被表征为不同类型的节点进行映射。同时, 这些节点将会根据函数实际的语句用线连接起来, 形成一种相互的关联关系。根据图 9 所示, 使用 AST 对程序

进行表示的方法使得 fooo 函数中的各组件都被映射到了树的不同节点上, 这使得这种树能够更精确地刻画程序与所包含的组件、以及组件与组件之间的关系, 而不会被程序的布局以及注释等词汇特征所影响。

表 7 典型句法特征

Table 7 Typical Syntactic Features

特征维度	描述
函数数量	单位文件中函数数量
关键字数量	某个关键字的数量
结构体的使用	特定结构体使用的数量
输入的使用	函数输入的数量
条件跳转的使用	条件语句的数量
控制结构的使用	指针/结构体/其他方式实现特定的数据赋值
函数调用	特定函数被调用的次数
语句顺序	函数中各语句的出现顺序
分隔符的使用	特殊分隔符出现的次数
函数大小	函数的平均大小(代码量)

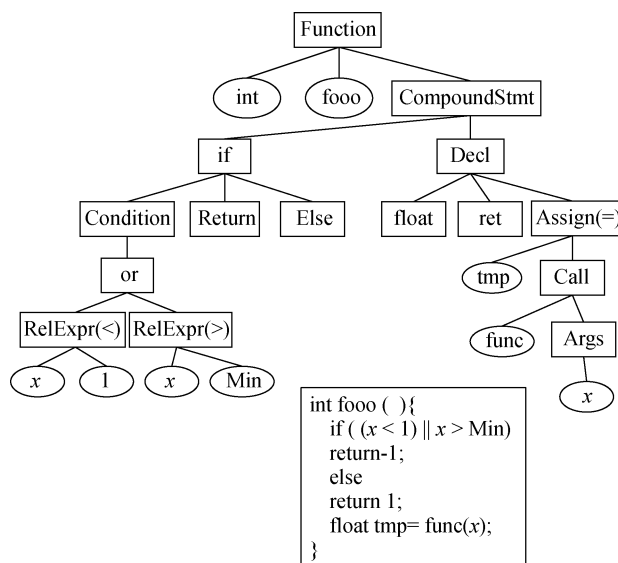


图 9 fooo 函数的 AST 构建示例

Figure 9 Example of AST for Function Fooo

### 5.1.3 语义特征

语义特征(Semantic Features)关注代码逻辑, 旨在从逻辑控制流的角度去诠释程序运行表征的具体含义。这种特征得以使用的原因在于研究者认为作者在进行编程时, 通常会使用相似的思路去解决编程过程中遇到的问题, 这将使得其使用的代码逻辑具备较高的同质性。由于语义特征关注程序执行的运行逻辑, 因此其与待分析程序的语言形式(汇编、C), 函数/变量的名称以及布局等信息都没有关系。



具体来说, 句法特征中的(if-else 与 switch case)理所当然被认为是不同的表述, 但是在语义特征中, 他们都表示一种选择性的语句。语义特征很好的规避了不同编程语言带来的分析挑战, 因为理论上来说作者的编程逻辑在不同语言上也应该保持一定的一致性。表 8 展示了语义特征所关注的典型特征。

表 8 典型语义特征  
Table 8 Typical Semantic Features

特征维度	描述
控制流相关	程序控制流图
数据流相关	程序数据流图
依赖相关	过程依赖

描述语义特征的一种典型实例就是程序的控制流图(Control Flow Graph, CFG), 它主要用于表示在某一个程序执行路径中, 代码、分支语句的执行顺序。控制流图通常以有向图的形式出现, 包含顶点(vertex)和边(edge)两种成分。对于节点来说, 其通常是由某一条语句组成, 语句的粒度可以是一条指令, 一个函数等; 而对于边来说, 其用于表征节点间相互调用的顺序。而数据流图(Data Flow Graph, DFG)与控制流图所包含的组分(顶点、边)相似, 但其表征了某一个数据在不同结构间的传递情况, 是一种表示数据之间依赖关系的非循环图。因此, 数据流图中顶点通常由一个操作码(opcode operation)组成, 而边指以一个操作码  $A$  作为输入被另一个操作码  $B$  使用的过程, 也可以表示被操作码  $A$  所处理的数据  $Data_A$  被操作码  $B$  使用的过程, 记为  $A \rightarrow B$ 。

### 5.1.4 其他特征

除了上述从程序内容中提取的表征作者写作风格与习惯的相关特征外, 程序编写时其他相关的外部因素也同样可以用作判定原作者归属的重要因素。例如, 作者生物特征(Biographical Features)泛指通过作者的一些社会学属性去认定作者身份, 该特征通常可以作为作者归属的一个额外附加的维度进行考虑。引入他们的主要原因在于每一个软件的作者都是社会的一份子, 他可能从属于某一个机构, 一个组织, 或来自于某一个特殊的地区, 有研究者认为这些与出身相关的信息也将会对代码的编写风格产生影响。根据 Bayrami 等人的研究<sup>[40]</sup>, 典型的传记特征有作者所属地区(Region)、性别(Gender)以及编写程序时使用的特殊作者标识(Handle)等。其他的, 还有程序编译时使用的编译器优化程度特征、程序配置文件特征等。

## 5.2 特征属性的常用表示方式

### 5.2.1 TF-IDF 表示

词频-逆文本频率指数(Term Frequency-Inverse Document Frequency, TF-IDF)是一个广为人知的文本数据挖掘方法。该技术基于的主要观点在于某一个/或一组术语在文档中出现的频率越高, 则他们的重要性就越强, 则这些内容就更加具备代表整个文档的资格。在 TF-IDF 中, 通常使用如下公式的方法计算某一个术语的重要性:

$$TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

其中, 在公式的一端,  $t$  表示文档中的关键术语,  $d$  表示需要分析的文档,  $D$  表示语料库中所有的文件。在公式的另一端,  $TF(t, d)$  指词频, 表示某一个给定的关键术语  $t$  在文档  $d$  中出现的频率。

$IDF(t, D)$  指文档逆序数, 由如下公式所定义:

$$IDF(t, D) = \log\left(\frac{|D|}{DF(t, D)}\right) + 1$$

在等式的一端,  $|D|$  表示语料库中文件的总量。另一端,  $DF(t, D)$  则表示语料库中有多少文件实际包含术语  $t$ 。

虽然研究者基于 TF-IDF 可以很容易的计算出特定特征在程序文件中的重要程度, 并可以此从一定程度上表征用户的写作风格, 但如何消除干扰项(如常见的词语、库)对特征表示的影响, 是研究者需要仔细考虑的问题。

### 5.2.2 深度学习表示

基于 TF-IDF 的表示方法事实上可以直接输入机器学习算法中进行训练, 以得到可以用于作者归属分类的模型。然而为了得到较好的分类效果, 这种直接的训练方法通常需要基于复杂且极度消耗人力的特征工程。不仅如此, 应用场景的变化对分类效果的影响所占的比重也极其巨大<sup>[55]</sup>。

作为对比, 利用深度学习模型对特征进行深度表示也已经在多个领域进行了应用<sup>[56-57]</sup>。基于深度神经网络的多层处理结构, 原本众多难以直接表征作者写作风格的离散特征将可能被更为准确的区分。不仅如此, TF-IDF 与深度学习表示也可以进行结合。例如, 文献[44]即使用 LSTM 和 GRU 算法对最初的 TF-IDF 特征进行处理, 以此探索多样的特征表示结果。

## 5.3 基于不同特征属性的作者归属研究

### 5.3.1 基于词汇特征的方法

作为基础且常用的特征, 有关词汇特征的应用研究十分广泛。除了基础的机器学习方法之外, 深度

学习以及多方法融合的技术都得到了一定的尝试,近年来的代表性研究如下。

Abuhamad 等<sup>[44]</sup>面向支持多语言原作者分析,提出融合了 RNN+RF 的大规模程序作者认证技术。该技术首先针对词汇特征进行关键术语提取,接着利用 TF-IDF 方法排序并抽取了前 30 位的重要特征,再将他们输入 RNN 模型后,一种新的表征作者内在身份特征的深度表示即被生成。随后研究者将该深度表示输入了随机森林分类器中,借助该分类器提升大规模程序分类任务的效果。

另一项研究<sup>[47]</sup>关注如何使用 CNN 技术来进行程序原作者的归属分析。具体来说,在区分不同作者所作程序时其利用了两种方式对程序进行表示:一种是基于频率的表示方法,指研究者利用 TF-IDF 技术对重要词汇特征进行筛查,以确定不同程序所表征出的词汇特征的不同;另一种是基于预测的表示方法,研究者首先将目标程序进行分析,以独热编码的形式编码整个程序单词库中单词,并将其输入到对应的工具模型(如: word2vec、Glove 或预先训练的模型)中以生成特殊的 Word embedding 表示方式。最后,研究者通过连接型 CNN(Concatenated CNN Architecture)和堆叠型 CNN(Stacked CNN Architecture)架构模型,介绍了对不同类别的表示信息在不同 CNN 架构下的学习方案,并通过在不同语言环境下比较的方式对预测效果进行评估。

Gonzalez 等人<sup>[48]</sup>和 Kalgutkar 等人<sup>[49]</sup>的工作均面向安卓系统固件的同源性分析技术。他们的研究关注作者在开发 APP 时的各类编程决策,例如开发过程中涉及的特殊类、方法以及字符串的数量。具体来说,该工作首先对 .dex 文件类型的安卓固件进行解析,根据分段信息从代码段的解析结果中提取了多种语法特征(如:类、方法、数据结构以及与数据结构操作相关的操作码)的使用情况。随后作者将这些特征从类、方法的数量,数组定义的统计值,从数组操作码中包含的重要 N-gram 数量以及从数据结构中创建的 N-gram 等 4 个方面进行向量化,以形成作者风格特征。

### 5.3.2 基于句法特征的方法

虽然词汇特征的跨平台性良好,但其对于作者微小改动后适应性不佳。研究者在此基础上进一步开始关注句法特征,这标志着作者归属技术逐渐从简单的关键字、布局分析向语句编写习惯进行转变,相关的研究如下。

Bogomolov 等<sup>[43]</sup>面向多语言(C++、python、java)适配的目标,在提取程序代码的抽象语法树的基础

上,以语法树节点顺序构建了程序路径,提出基于 AST 不同类型节点间连接性的路径表示以及针对该路径的上下文表示,随后该研究进一步将程序路径以出现频率进行表示,以使其更容易的被输入到基于随机森林和神经网络的作者分类模型中来完成认定工作。

Wisse 等<sup>[52]</sup>提供了一个由 JavaScript 编写的相当大小和质量的参考数据集,并对其进行了标注。在此基础上,该文献从句法角度抽取源码的抽象语法树,从中获取了三类信息,分别为:1)代码结构:包含:从中记录节点列表的长度(表征函数声明中定义的参数数量和数组中初始化的元素数量),特定节点的后代节点的数量(反应了该节点的复杂性,例如:函数中传递标识符、对象的多少),N-gram 切分下节点使用频率最高特征;2)表征代码布局,例如:缩进与空格);3)代码风格,即注释、命名约定和数据类型。利用这些特征,本方案尝试对待测程序是否由某一个特定作者或给定范围内的作者所著两个目标进行验证。

### 5.3.3 基于词汇+句法特征的方法

除了前述基于单一类型特征的研究工作,融合这两方面特征进行作者同样也得到了研究者的注意,较为典型的工作如下:

Caliskan-Islam 等<sup>[42]</sup>针对 C++ 程序提出了使用频率(TF)和词频逆序数(TF-IDF)描述 AST 节点的方式进行研究。方案将程序的词汇特征一道输入基于随机森林的作者判定模型中,取得了不俗的检测效果。此外,论文作者还对数十位作者的编程风格进行了数年的追踪,以评估文中所述的基于作者风格的作者归属方法的有效性。作为较早提出使用 AST 句法特征作为作者风格描述分类的研究,该工作具备一定的指导意义。

与多数文献关注于整个程序的原作者归属不同,Dauber 等<sup>[41]</sup>对程序片段的作者归属问题进行了尝试性的研究。具体来说,研究者从句子结构的角度对目标程序进行分析,将程序转变为抽象语法树的表示。在此基础上,作者依照 Caliskan-Islam 等<sup>[42]</sup>的方法统计了目标程序 AST 最大节点深度、AST 节点相互连接的频率以及 58 个 AST 节点类型的平均深度等数据。这些数据随后以若干子树为单位被输入到随机森林算法进行训练建模,以实现对于程序代码所属的不同作者进行区分。

Zhang 等<sup>[50]</sup>同样关注词汇和句法特征所蕴含的作者风格,对文献编程布局特点、编程风格特点、编程结构特点和编程逻辑特点进行了描述,并分别从

他们中提取了可描述作者编程偏好的特征,例如程序中空行的数量占比、IF-ELSE 语句的数量占比, For 循环起始时空格的数量等数据。并结合基于字符串的连续-离散 N-gram 方法产生的序列频率生成表征作者的向量。进一步的,文献在使用支持向量机模型进行作者判定时还提出引入序列最小优化算法对数据进行处理。

5.3.4 基于语义特征的方法

在词汇以及句法分析的基础上,进一步关注作者程序逻辑的语义特征分析也是近来流行的研究方向。与前面的方式相比,程序语义的提取对于作者写作习惯的刻画更具鲁棒性的优势。在进行此类研究时,研究者更倾向于结合词汇以及句法特征以实现对于程序-作者对应关系的更为全面的描述,相关的研究如下。

CPA<sup>[45]</sup>关注语义特征对原作者归属的影响,专门针对与用户有关的函数提取了控制流图 CFG 和数据流图 DFG。在整合这些特征的基础上形成了作者归因图(AAG),以包括作者代码特点(如代码实现细节)、代码结构以及作者编程时的经验(内存分配习惯、特定包的使用习惯或一些特殊的资源库)在内的特征的生成了一系列表征作者代码风格的特征表述。基于这些表述,文献同时使用狄利克雷划分算法(LDA)和 K-means 实现了对已知作者识别分类和对未知作者的聚类,以此完成用户身份签名的构建。不仅如此,CPA 的研究者还使用 Zipr<sup>[58]</sup>和 FLAIR<sup>[59]</sup>工具将目标固件中库相关、编译器相关以及用户相关的函数进行过滤,以尽量消除无用数据对判定结果的影响。

BinEye<sup>[46]</sup>强调采用自动化执行的方式来完成针对程序二进制文件的作者归属任务,基于深度神经网络提出了对应方案。在作者归属模型训练阶段,其采用了三个不同的神经网络,分别为:1)描述语义特征的指令流序列,这可以表述作者解决问题时特殊的编程思路;2)从汇编语言中提取的函数调用,如作者经常使用的一系列 API,这可以认为是一种词汇特征;3)直接将二进制文件转换为灰度图,并基于 Oliva 等的工作<sup>[60]</sup>以识别出该灰度图中有关于镜像执行结构和内容的特征。选用第三种特征的重要原因在于具备相似作者风格的镜像也可能出现相似的图像纹理。而有研究<sup>[61]</sup>指出,这种纹理对作者风格描绘的鲁棒性在一些情况下将优于以 N-gram 为代表的词汇特征表示方法。在经过不同 CNN 网络进行学习后,所获得的作者风格表示将通过 K-means 聚类以及选举的方式得到作者集合,以用来进行后续的程序作

者识别。该方法不需事先得到目标程序的先验知识,可自动实现作者代码风格描述的获取。不仅如此,文献还评估了其在对于多作者共同完成一个镜像时的多作者归属问题的效果。

Meng 等<sup>[51]</sup>面向如何区分单一固件程序中的多作者归因问题,研究了以函数为粒度或以代码组合(称为块级特征)为粒度的鉴别技术。作者倾向于证明基本块的作者归属鉴别效果优于函数粒度,并通过实验区回答了以下两个问题:1)块级特征可以归为单个作者的程度如何 2)块级特征是否包含足够的归属信息。针对第一个问题, Meng 等使用 git-author<sup>[62]</sup>工具计算每行代码的作者贡献百分比向量;针对第二个问题,本研究增加了包含词汇特征(指令)、句法(上下文)以及语义特征(控制流、数据流)辅助作者的归属,因此取得了一定效果,抽取的特征如表 9 所示。

表 9 文献[51]所述特征  
Table 9 Features in Manuscript [51]

代码属性	块级特征
指令	指令前缀,操作数大小和寻址模式,常量值
控制流	CFG 边缘类型,是否某个块抛出或捕获异常
数据流	块进入/退出时活跃寄存器数量,使用/定义的寄存器数量,块的栈高度,变量的依赖
上下文	嵌套循环的深度,循环大小,CFG 函数的深度

Alrabae 等<sup>[53]</sup>认为仅仅从基于控制图和若干高排序 N-gram 文本角度进行作者归属分析虽然达到了一定的效果,但事实上这种方法难以准确表述作者的风格习惯,这是因为分析的结果中可能混杂着由编译器产生的唯一 ID 或随机但唯一的函数名,这些与作者风格无关。基于这一观点,文献提出了基于多层结构的 OBA2 方案。具体而言,OBA2 首先解析了程序中可能调用到的库函数,对其重要字节序列、函数大小、关键字字符串、数值等进行基于双重哈希的签名,为这些共享库函数生成不同的标识,并减少其对后面分析工作的影响;其次,OBA2 基于自定义的翻译库将固件中特定的汇编组合的数量表征作者的语法风格(如: mov-cmp-jnz-xor-jmp 组合使用的次数就相当于 if 指令的使用次数),翻译的过程还同时考虑了直接匹配(Exact matching)和非直接匹配(Inexact matching)的情况;同时,OBA2 通过观察程序执行时寄存器的操作情况获得了寄存器流图(Register Flow Graph , RFG),其认为这种表征同样会反应出作者的代码能力和偏好。

5.3.5 基于其他特征的方法

也有研究者从其他特征角度对作者归属工作做出了有益的尝试, 例如: Bayrami 等<sup>[40]</sup>结合了文献[50]与[63]的工作, 在将从程序源码中提取的典型词汇特征指标, 如: Type-token ratio、Carroll’s Corrected TTR 等称为基于内容的特征, 提出融合若干生物学维度(如: 用户登录名、地区、性别, 基于非内容的特征)的方式训练分类模型。该方案的归属分类效果在三种机器学习方法(贝叶斯、支持向量机以及随机森林)上进行了评估。

5.4 方案分析

5.4.1 特点分析

**基于词汇特征的方案。**由于仅需要关注程序中特殊的符号, 因此只要程序分析工具支持特定的 IoT 设备平台, 特征即有机会被正确提取, 这使得基于词汇特征的方法<sup>[44,47-48]</sup>具备更好的跨平台性质。然而, 作为一种基础的特征集, 使用词汇特征进行原作者分析将会面临诸多挑战。具体来说, 同一个程序其词汇的布局(如: 代码行数、长度)就有可能受到不同编程 IDE、代码查看器的影响, 使得这部分特征所表述的信息难以稳定; 此外, 函数/变量重命名, 重要语句的换位、混淆以及冗余添加等简单手段造成的影响也难以忽视。

**基于句法特征的方案。**句法方案<sup>[43,52]</sup>关注作者

对词汇进行使用和处理的方式, 并把其作为判定依据。这种方式可以屏蔽部分简单修改手段, 具备更好的分析效果。然而, 句法特征由于需要识别符号的组织和使用方法, 而这些特征通常与编程语言相关。因此, 针对不同语言的特征提取工具通常难以直接混用, 这使得该技术在面对使用不同程序语言的不同架构 IoT 设备上遇到挑战。不仅如此, 若想隐藏作者身份, 编程人员仍然可以通过适当改变编程习惯(如: 重新排序函数内语句, 变换常用的 switch-case 语句结构, 常用类的合并, 添加冗余代码)等方式实现。可以预见, 选择更为相关的句法特征以尽可能抵消前述不利因素, 将仍然是后续研究的重点。

**基于语义特征的方案。**语义特征方案<sup>[45-46,51,53]</sup>

关注作者在编写程序时的运行逻辑, 而通常人的思维定式习惯是潜意识流露的, 因此理论上将比其他两类方案具备更高的抗干扰性质。但是 IoT 设备程序语义特征的截取涉及到动态分析技术, 通常需要对 IoT 设备程序进行基于模拟运行的数据捕捉。根据文献[12], 当前模拟方案可分为基于 Jtag 等调试用剑、全系统仿真、Hardware-in-the-loop 以及固件托管等方案。这些方案仍然存在需要大量人工参与、难以全系统接口模拟、可扩展性差以及仅能针对具备 HAL 层的 IoT 固件进行分析的缺点。工具能力的限制为语义分析方案在多类型 IoT 设备上应用提出了难题。

表 10 创作者归属方案对比

Table 10 Comparisons of Approaches for Authorship Attribution

文献名称	数据集来源	特征类别	主要语言环境	建模方法	分析目标	准确率	特点描述
文献[40]	Codeforces GitHub	社会学 词汇特征	C++	RF	单一作者	76.47%	将社会学维度融入模型训练
				NB		55.37%	
				SVM		66.37%	
文献[41]	GitHub	词汇特征 句法特征	C++	RF	主要作者 次要作者	70%	推断出某一个子树所属的次要作者
文献[42]	GCJ	词汇特征 句法特征	C++	RF	单一作者	94%	提供了不同特征类别下的具体维度, 并提炼了训练数据集
				RF		94.8%	
				NN		79.3%	
文献[43]	文献[64]	句法特征	Python	RF	单一作者	95.9%	增加了以抽象语法树中节点顺序构建基于程序路径的维度
				NN		72.3%	
				RF		98.5%	
				NN		97.9%	
				C		94.8%	
文献[44]	GCJ GitHub	词汇特征	Java	RNN+RF	单一作者	97.24%	利用 RNN 技术解决多语言环境下的词汇特征提取
			Python			96.2%	
			C++			92.3%	
			Binary			95.74%	

续表

文献名称	数据集来源	特征类别	主要语言环境	建模方法	分析目标	准确率	特点描述
文献[45]	GitHub GCJ	词汇特征 句法特征 语义特征	固件	LDA	多作者	94%	利用 LDA 算法从作者代码风格方面的维度进行比较
文献[46]	GitHub GCJ	词汇特征 语义特征	固件	CNN	多作者	90%	生成多种 CNN 网络进行作者聚类分析
文献[47]	GitHub GCJ	词汇特征	C++ Java Python	CNN	单一作者	96.2% 95.8% 94.6%	将词汇特征输入多种 CNN 架构中进行结果比较分析
文献[48]	其他来源	词汇特征	固件	RF	单一作者	97.5%	基于作者开发风格将未知 APP 安卓固件进行作者归属分析, 提供了实验数据集
文献[49]	GitHub 其他来源	词汇特征	固件	SVM	单一作者	98% (正常) 96% (恶意) 71% (混淆)	从多种渠道采集了三种固件数据集, 并分别进行原作者分析评估
文献[50]	GitHub 其他来源	词汇特征 句法特征	Java	SVM	单一作者	99.08%	融合多种类别特征以及字符串 N-gram 方法的原作者分析
文献[51]	GitHub 其他来源	词汇特征 句法特征 语义特征	固件	SVM	多作者	52%	从代码块中提取了多个语义角度特征来辅助找出每个块的作者
文献[52]	GitHub	句法特征	Java	SVM	单一作者	91%	针对 JavaScript 提取语法树, 并从中获得描绘写作风格、结构以及布局的特征
文献[53]	GCJ	句法特征 语义特征	固件	SVM	单一作者	81%	将编译器生成的无关作者本身编码风格的部分去掉, 并从汇编序列以及寄存器使用角度获取作者编码风格特征

备注: RF: 随机森林、NB: 贝叶斯、SVM: 支持向量机、NN: 神经网络

**基于多种方式融合的方案。**从理论上来说, 融合多种维度特征的方案将有助于对作者从多角度进行刻画, 因此将有助于提升检测效果。然而, 这无疑将对特征的提取工作量以及工具能力提出挑战。例如作者社会学维度信息只能在特定情况下获取; 分析工具需要支持多平台、多检测模式(动态、静态)以及多软件结构(裸机系统、实时操作系统)等。此外, 由前所述, 部分特征(如词汇特征中的代码行数、长度)易受到环境影响, 在融合时也需要综合考虑, 不能盲目将他们加入, 甚至可能对模型训练的结果造成负面影响。

**5.4.2 适配性分析**

大多针对创作者分析技术的研究其分析对象的具体应用平台是隶属于 IoT 设备还是 X86 平台并不明确。这一部分的原因在于当前很多工作是基于源码进行分析, 并没有对可适配平台进行讨论, 而已有的针对固件镜像的工作也大多以 X86 平台作为先决的实验环境。当前影响面向程序原作者的检测方案效果的重要因素在于所选取的程序特征是否能尽可能准确地描绘出作者的写作风格。因此, 本节将从

特征提取的角度对相关技术是否能应用到 IoT 设备固件上进行分析。

从面向分析对象的角度来说, 原作者分析检测方法可以分成针对源码以及针对固件两方面内容。对于面向源码的分析工作而言, 可以认为只要 IoT 程序可以由该种语言编写, 那么相关的工作就存在可以被移植到 IoT 设备平台上的可行性。根据不同类型设备系统通常使用的编写语言, 前述所介绍工作预期所能支持的设备类型如表 11 所示。大多数基于源码的作者归属工作(即: 文献[40]、[41]、[42]、[43]、[44]、[47]、[50]和[52])都可以支持 II(C、C++)和 III(Python、C++、Java)类设备, 而讨论过 I 类设备所支持 C 语言的研究较少。

对于镜像文件而言, 其反汇编后的汇编语言将于不同的架构相关, 因此不能将相应的方法直接移植到 IoT 设备当中去。例如, 文献[44]利用 Obfuscator-LLVM<sup>[66]</sup>将其研究的源码程序编译成镜像文件, 并在此基础上进行作者归属研究, 由于 Obfuscator-LLVM 支持包含 X86、ARM 在内的多种系统架构, 因此该工作具备潜在的可扩展性; 文献[45]更是将方案直接在基于 ARM、X86、MIPS 等架构下的固件进行作者归

表 11 基于源码方案的适配性  
Table 11 Applicability of Source-code Oriented Schemes

文献名	支持语言	预期可支持设备类型		
		I	II	III
文献[40]	C++	/	✓	✓
文献[41]				
文献[42]				
文献[43]	C++	/	✓	✓
	Python			
	Java			
文献[44]	C	✓	✓	✓
	C++			
	Java			
文献[47]	Python	/	✓	✓
	C++			
	Java			
文献[50]	Java	/	/	✓
文献[52]				

属效果的评估; 文献[46]利用 IDA-Pro<sup>[67]</sup>对目标镜像进行反汇编, 并使用 Nucleus<sup>[68]</sup>获取编译器难以获得的控制流信息, 当前 Nucleus 仅支持 X86 与 x64 下的 C 与 C++固件信息提取; 文献[48]和[49]均针对安卓程序镜像进行分析, 因此其工作直接针对于 III 类 IoT 设备; 文献[51]同样使用 IDA-Pro 对利用 GCC 进行编译的镜像进行反汇编, 同时使用 Dyninst<sup>[69]</sup>获取代码。Dyninst 当前支持 ARM 架构, 因此该工作的可移植性也值得期待; 最后, 文献[53]面向 C++固件反汇编后的作者归属问题设计了方案, 其工作对 II 类和 III 类 IoT 系统固件的分析将有启发性作用。根据不同固件架构, 他们所能支持的设备归类分析如表 12 所示。

表 12 面向固件二进制方案的适配性  
Table 12 Applicability of Binary Oriented Schemes

文献名	文中支持固件架构	预期可支持设备类型		
		I	II	III
文献[44]	X86	✓	✓	✓
文献[45]	多类型	✓	✓	✓
文献[46]	X86、X64	/	/	/
文献[48]	安卓	/	/	✓
文献[49]				
文献[51]	X86	✓	✓	✓
文献[53]	X86	/	✓	✓

从以上的分析可以得知, 通过一些对应性的修改(例如测试数据集以及编译解析环境), 大多数的作者归属研究都具备移植到 IoT 设备上进行研究的条

件。不仅如此, 有部分文献(如: 文献[45])直接声称其具备良好的跨平台特性, 可见将这些工作向 IoT 平台进行移植均有着一定的技术基础。

## 6 讨论与建议

由于 IoT 设备特殊的软硬件特性, 当前大多数研究工作仍难以涵盖多类型的 IoT 设备。本节将面向构建通用的 IoT 程序同源性技术为目标, 对检测过程中所涉及到的关键技术能力展开讨论。

**(1) 分析数据的收集。**对于固件的相似性检测技术来说, 大多软件包(如 busybox、OpenSSL 以及 cURL)本身就具备支持 IoT 设备的版本, 唯一需要注意的是他们通常需要操作系统支持, 即面向 II 型和 III 型设备程序。GNU 工具集面向 X86 平台, 对于 IoT 设备而言, 搭载嵌入式 Linux 的 III 类 IoT 设备固件也可以使用。对于固件作者归属分析任务, 目前专门针对不同类别 IoT 设备的数据集还较少, 部分工作(如文献[45])也是将原适用于 X86 平台的 C/C++程序利用对应编译器进行编译, 利用所获得的程序进行进一步分析。事实上, 虽然当前大的代码平台(如: CGJ、GitHub)存在着大量的嵌入式程序, 但其绝对总量还是较少。

为了收集到足够数量的数据集, 满足同一个作者所著的不同 IoT 程序的条件, 来支持面向不同特征维度的模型训练, 还需要大量的收集和标记工作。从工作路线上来说, 研究者首先需要从典型的漏洞数据集(如: NVD、CNVD、CNNVD 等)中获取不同类别漏洞的信息, 包括原程序的提供者、程序来源以及漏洞描述。随后以此为基础, 通过一定的手段(如代码平台、供应商)对应寻找相关代码并进行标注。当标注量达到一定规模后, 才可以用于实际的检测。

**(2) 固件程序的解包。**为了分析 IoT 固件文件, 首先需要做的事情就是对他们进行反汇编。当前已经有诸如 IDA-Pro、Dyninst 等工具可以实现。然而, 为了应对 IoT 设备多架构分析, 需要反编译工具具备良好的检测精度和广度。不仅如此, 反汇编代码的可读性也将随着不同架构模式而变化, 如何将针对源码的创作者分析研究中取得的成果应用、移植到固件环境中去, 需要进一步研究分析。在此基础上, 从汇编角度分离、区分出 IoT 程序中常用的库函数、系统调用以及用户编写的程序, 以进一步提升创作者分析的精度, 也需要重点关注。

**(3) 特征提取。**不论从固件角度分析还是从源码角度分析, 同源性分析的最终目标还是提取适合于描述作者写作风格的特征。对于作者归属技术而言,

词汇特征因为仅是对标号进行一些排列组合,因而相对而言易于提取。而句法以及语义特征的情况下,都应该基于不同的架构来分析词句之间的关联关系,这些关系的挖掘需要设计专门适用于对应 IoT 设备架构的软件方案来实现。特征提取方案可以分为动态以及静态分析。静态分析相对来说实现的过程较为简单,但是其往往仅针对词汇特征较为有效。而动态分析方案因受 IoT 程序编程语言、生成固件形式以及设备类型的影响,使得相关特征(例如:语义特征)的提取难以覆盖全面。因此,针对上述挑战进行相关数据提取工具研发,将有利于提升语法以及语义特征提取的准确性。

**(4) 分析方法。**除了体量较小的 I 型 IoT 固件可能独立完成外,其他类型固件的编写通常都不可能仅有一个作者完成,这使得 IoT 固件的细粒度作者区分就显得更有意义。而当前大多数的同源性分析问题仍假设某一个程序的编写者仅由单一作者组成,仅有少量的文献对一个程序从属于多作者进行了讨论。若要深入研究,则需要更为精确的标注数据集以及特征的支撑,例如需要将程序片段映射到某一种漏洞或某一个作者。相比于前者,对于后者的分析难度较大,需要程序特征表示更为明确。因此,为了适配 IoT 设备固件的运行特点,在研究的过程中还需重点考虑不同类型固件的编写逻辑和应用场景,以从合适的粒度提取、计算能恰当描述不同作者的特征。

## 7 总结

由于 IoT 程序软件并没有固定的开发规范,使得生态圈中充斥着各种未经授权的程序作者。他们所创作的软件的安全性和鲁棒性都难以保证。对程序同源性进行检测,可以从一定程度上约束开发者行为,减少恶意软件、库的传播事件。同时,其对程序产权纠纷(如:抄袭)也能提供一定的帮助。本文针对 IoT 设备程序同源性的智能分析技术展开综述,将其从功能性和作者风格两个角度进行区分,介绍了对应的相似性检测和创作者归属技术,并对相关工作从数据获得、特征选取、模型训练、准确率以及主要贡献等多方面进行描述;随后总结了这些方案对于 IoT 设备的有效性、局限性以及适配性。最后,文章对同源性检测过程中的关键技术给出建议。作者希望通过本文的综述,为研究者阐明相关工作的核心技术点,并为后续应用研究提供参考。

## 参考文献

- [1] Dang L M, Min K, Han D, et al. A Survey on Internet of Things and Cloud Computing for Healthcare[J]. *Electronics*, 2019, 8(7): 768.
- [2] Song Y X, Yu F R, Zhou L, et al. Applications of the Internet of Things (IoT) in Smart Logistics: A Comprehensive Survey[J]. *IEEE Internet of Things Journal*, 2021, 8(6): 4250-4274.
- [3] Alkhabbas F, Spalazzese R, Cerioli M, et al. On the Deployment of IoT Systems: An Industrial Survey[C]. *2020 IEEE International Conference on Software Architecture Companion*, 2020: 17-24.
- [4] IoT and non-IoT connections worldwide 2010-2025. <https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>. Mar. 2021.
- [5] IoT Security and the Internet of Forgotten Things. <https://securityintelligence.com/articles/iot-security-internet-forgotten-thing/>. Mar. 2022.
- [6] CVE-2021-39237. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-39237>. Aug. 2021.
- [7] CVE-2021-35395. <https://nvd.nist.gov/vuln/detail/CVE-2021-35395>. Aug. 2021.
- [8] CVE-2021-28372. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-28372>. Mar. 2021.
- [9] Yu Y C, Chen Z N, Gan S T, et al. Research on the Technologies of Security Analysis Technologies on the Embedded Device Firmware[J]. *Chinese Journal of Computers*, 2021, 44(5): 859-881. (于颖超, 陈左宁, 甘水滔, 等. 嵌入式设备固件安全分析技术研究[J]. *计算机学报*, 2021, 44(5): 859-881.)
- [10] Li D, Yin Q, Lin J, et al. Firmware Vulnerability Detection in Embedded Device Based on Homology Analysis[J]. *Computer Engineering*, 2017, 43(1): 72-78. (李登, 尹青, 林键, 等. 基于同源性分析的嵌入式设备固件漏洞检测[J]. *计算机工程*, 2017, 43(1): 72-78.)
- [11] Cui A, Costello M, Stolfo S. When firmware modifications attack: A case study of embedded exploitation[J]. *Network and Distributed System Security Symposium*, 2013.
- [12] Pierluigi Paganini. Qualcomm bug impacts about 30% of all smartphones. <https://securityaffairs.co/wordpress/117620/security/qualcomm-bus-cve-2020-11292.html>
- [13] Song W N, Peng G J, Fu J M, et al. Research on Malicious Code Evolution and Traceability Technology[J]. *Journal of Software*, 2019, 30(8): 2229-2267. (宋文纳, 彭国军, 傅建明, 等. 恶意代码演化与溯源技术研究[J]. *软件学报*, 2019, 30(8): 2229-2267.)
- [14] Li D, Yin Q, Lin J, et al. Firmware Vulnerability Detection in Embedded Device Based on Homology Analysis[J]. *Computer Engineering*, 2017, 43(1): 72-78. (李登, 尹青, 林键, 等. 基于同源性分析的嵌入式设备固件漏洞检测[J]. *计算机工程*, 2017, 43(1): 72-78.)
- [15] Antonakakis M, April T, Bailey M, et al. Understanding the Mirai Botnet[C]. *The 26th USENIX Conference on Security Symposium*, 2017: 1093-1110.
- [16] Wu P. *Research on homology determination technology of polymorphic software code*[D]. Chengdu: Sichuan University, 2021.

- (吴鹏. 多形态软件代码同源判定技术研究[D]. 成都: 四川大学, 2021.)
- [17] Ran L J, Lu L P, Lin H, et al. An Experimental Study of Four Methods for Homology Analysis of Firmware Vulnerability[C]. *2017 International Conference on Dependable Systems and Their Applications*, 2018: 42-50.
- [18] Zhang (L/Y). *Research and implementation of firmware vulnerability analysis system for Internet of Things*[D]. Beijing: Beijing University of Posts and Telecommunications, 2020.  
(张乐. 物联网固件脆弱性分析系统的研究与实现[D]. 北京: 北京邮电大学, 2020.)
- [19] Zhang C, Situ L Y, Wang L Z. Research Progress of Firmware Security Defect Detection in Internet of Things[J]. *Journal of Cyber Security*, 2021, 6(3): 141-158.  
(张弛, 司徒凌云, 王林章. 物联网固件安全缺陷检测研究进展[J]. *信息安全学报*, 2021, 6(3): 141-158.)
- [20] Sun H, Tong Y, Zhao J, et al. DVul-WLG: Graph Embedding Network Based on Code Similarity for Cross-Architecture Firmware Vulnerability Detection[C]. *International Conference on Information Security*, 2021: 320-337.
- [21] Sun T, Wang B, Xin M, et al. Research on homology analysis technology of malicious code[J]. *The Journal of New Industrialization*, 2021, 11(10): 114-117.  
(孙甜甜, 王宝会, 辛敏健. 恶意代码同源性分析技术研究[J]. *新型工业化*, 2021, 11(10): 114-117.)
- [22] Haq I U, Caballero J. A Survey of Binary Code Similarity[J]. *ACM Computing Surveys*, 2022, 54(3): 1-38.
- [23] Kalgutkar V, Kaur R, Gonzalez H, et al. Code Authorship Attribution: Methods and Challenges[J]. *ACM Computing Surveys*, 2020, 52(1): 1-36.
- [24] Muench M, Stijohann J, Kargl F, et al. What You Corrupt is not what You Crash: Challenges in Fuzzing Embedded Devices[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.
- [25] Wind-River. <https://github.com/Wind-River/vxworks7-ros2-build>. Jul. 2020.
- [26] Micrium. <https://github.com/weston-embedded/uC-OS3>. May 2021.
- [27] Contiki-ng. <https://github.com/contiki-ng/contiki-ng>. May 2022.
- [28] Liu B C, Huo W, Zhang C, et al. ADiff: Cross-Version Binary Code Similarity Detection with DNN[C]. *The 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018: 667-678.
- [29] Duan Y, Li X, Wang J H, et al. DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing[C]. *Proceedings 2020 Network and Distributed System Security Symposium*, 2020.
- [30] Massarelli L, Luna G A D, Petroni F, et al. Safe: Self-attentive function embeddings for binary similarity[C]. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2019: 309-329.
- [31] Feng Q, Zhou R D, Xu C C, et al. Scalable Graph-Based Bug Search for Firmware Images[C]. *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 480-491.
- [32] Xu X J, Liu C, Feng Q, et al. Neural Network-Based Graph Embedding for Cross-Platform Binary Code Similarity Detection[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 363-376.
- [33] Gao J, Yang X, Fu Y, et al. VulSeeker: A Semantic Learning Based Vulnerability Seeker for Cross-Platform Binary[C]. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, 2020: 896-899.
- [34] Redmond K, Luo L N, Zeng Q. A Cross-Architecture Instruction Embedding Model for Natural Language Processing-Inspired Binary Code Analysis[EB/OL]. 2018: arXiv: 1812.09652. <https://arxiv.org/abs/1812.09652>
- [35] Zuo F, Li X P, Young P, et al. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs[EB/OL]. 2018: arXiv: 1808.04706. <https://arxiv.org/abs/1808.04706>
- [36] Zhu X D, Jiang L H, Chen Z. Cross-Platform Binary Code Similarity Detection Based on NMT and Graph Embedding[J]. *Mathematical Biosciences and Engineering*, 2021, 18(4): 4528-4551.
- [37] 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*, 2019: 472-489.
- [38] Gao J, Yang X, Fu Y, et al. VulSeeker-Pro: Enhanced Semantic Learning Based Binary Vulnerability Seeker with Emulation[C]. *The 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018: 803-808.
- [39] Luo Z P, Hou T, Zhou X R, et al. Binary Code Similarity Detection through LSTM and Siamese Neural Network[J]. *ICST Transactions on Security and Safety*, 2021, 8(29): 170956.
- [40] Bayrami P, Rice J E. Code Authorship Attribution Using Content-Based and Non-Content-Based Features[C]. *2021 IEEE Canadian Conference on Electrical and Computer Engineering*, 2021: 1-6.
- [41] Edwin D, Robert E, Gregory S, et al. Supervised Authorship Segmentation of Open Source Code Projects[J]. *Proceedings on Privacy Enhancing Technologies*, 2021, 2021(4): 464-479.
- [42] Caliskan-Islam A, Harang R, Liu A, et al. De-Anonymizing Programmers via Code Stylometry[C]. *The 24th USENIX Conference on Security Symposium*, 2015: 255-270.
- [43] Bogomolov E, Kovalenko V, Rebryk Y, et al. Authorship Attribution of Source Code: A Language-Agnostic Approach and Applicability in Software Engineering[EB/OL]. 2020: arXiv: 2001.11593. <https://arxiv.org/abs/2001.11593>
- [44] Abuhamad M, Abuhmed T, Mohaisen D, et al. Large-Scale and Robust Code Authorship Identification with Deep Feature Learning[J]. *ACM Transactions on Privacy and Security*, 2021, 24(4): 1-35.
- [45] Alrabae S, Debbabi M, Wang L Y. CPA: Accurate Cross-Platform Binary Authorship Characterization Using LDA[J]. *IEEE Transactions on Information Forensics and Security*, 2020, 15: 3051-3066.
- [46] Alrabae S, Karbab E M B, Wang L, et al. BinEye: towards efficient binary authorship characterization using deep learning[C].



- European Symposium on Research in Computer Security*, 2019: 47-67.
- [47] Abuhamad M, Rhim J S, AbuHmed T, et al. Code Authorship Identification Using Convolutional Neural Networks[J]. *Future Generation Computer Systems*, 2019, 95: 104-115.
- [48] Gonzalez H, Stakhanova N, Ghorbani A A. Authorship Attribution of Android Apps[C]. *The Eighth ACM Conference on Data and Application Security and Privacy*, 2018: 277-286.
- [49] Kalgutkar V, Stakhanova N, Cook P, et al. Android Authorship Attribution through String Analysis[C]. *The 13th International Conference on Availability, Reliability and Security*, 2018: 1-10.
- [50] Zhang C, Wang S, Wu J, et al. Authorship identification of source codes[C]. *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, 2017: 282-296.
- [51] Meng X Z. Fine-Grained Binary Code Authorship Identification[C]. *The 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016: 1097-1099.
- [52] Wisse W, Veenman C. Scripting DNA: Identifying the JavaScript Programmer[J]. *Digital Investigation*, 2015, 15: 61-71.
- [53] Alrabaei S, Saleem N, Preda S, et al. OBA2: An Onion Approach to Binary Code Authorship Attribution[J]. *Digital Investigation*, 2014, 11: S94-S103.
- [54] Oman P W, Cook C R. Programming Style Authorship Analysis[C]. *The 17th conference on ACM Annual Computer Science Conference*, 1989: 320-326.
- [55] Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives[J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35(8): 1798-1828.
- [56] Boulanger-Lewandowski N, Bengio Y, Vincent P. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription[C]. *The 29th International Conference on Machine Learning*, 2012: 1881-1888.
- [57] Bordes A, Glorot X, Weston J, et al. Joint learning of words and meaning representations for open-text semantic parsing[C]. *Artificial intelligence and statistics*, 2012: 127-135.
- [58] Hawkins W H, Hiser J D, Co M, et al. Zipr: Efficient Static Binary Rewriting for Security[C]. *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2017: 559-566.
- [59] Rafee M M H. Computer program categorization with machine learning[M]. University of Lethbridge (Canada), 2017.
- [60] Oliva A, Torralba A. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope[J]. *International Journal of Computer Vision*, 2001, 42(3): 145-175.
- [61] Kirat D, Nataraj L, Vigna G, et al. SigMal: A Static Signal Processing Based Malware Triage[C]. *The 29th Annual Computer Security Applications Conference*, 2013: 89-98.
- [62] Meng X Z, Miller B P, Williams W R, et al. Mining Software Repositories for Accurate Authorship[C]. *2013 IEEE International Conference on Software Maintenance*, 2013: 250-259.
- [63] Naz F, Rice J E. Sociolinguistics and programming[C]. *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2015: 74-79.
- [64] Alsulami B, Dauber E, Harang R, et al. Source code authorship attribution using long short-term memory based networks[C]. *European Symposium on Research in Computer Security*, 2017: 65-82.
- [65] Yang X Y, Xu G A, Li Q, et al. Authorship Attribution of Source Code by Using back Propagation Neural Network Based on Particle Swarm Optimization[J]. *PLoS ONE*, 2017, 12(11): e0187204.
- [66] Junod P, Rinaldini J, Wehrli J, et al. Obfuscator-LLVM — Software Protection for the Masses[C]. *2015 IEEE/ACM 1st International Workshop on Software Protection*, 2015: 3-9.
- [67] IDA-Pro. <https://www.hex-rays.com/ida-pro/>. 2022.
- [68] Andriesse D, Slowinska A, Bos H. Compiler-Agnostic Function Detection in Binaries[C]. *2017 IEEE European Symposium on Security and Privacy*, 2017: 177-189.
- [69] Dyninst, The Dyninst project. <https://github.com/dyninst/dyninst>. 2022.



孔凯薇 于 2013 年在北京电子科技学院电子信息工程专业获得学士学位。现任北京计算机技术及应用研究所工程师。研究领域为信息安全、软件研发。Email: kongkw1991@163.com



霍冬冬 于 2021 年在中国科学院大学网络空间安全专业获得博士学位。现任中国科学院信息工程研究所第五研究室高级工程师。研究领域为区块链与 IoT 设备安全、数据安全。Email: huodongdong@iie.ac.cn



苏东楠 于 2018 年毕业于北京理工大学, 设计学硕士学位。现任北京计算机技术及应用研究所副主管设计师, 工程师职称。研究领域为信息系统集成、软件开发。Email: sudongnan@sina.cn



徐震 于 2005 年在中国科学院软件所计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所第五研究室主任, 正高级工程师, 博士生导师。研究领域为云计算安全、可信计算、移动互联网安全、系统安全。Email: xuzhen@iie.ac.cn