软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(11):3058−3071 [doi: 10.13328/j.cnki.jos.005342]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

移动应用安全生态链构建方法*

杨昕雨, 徐国爱

(北京邮电大学 网络空间安全学院,北京 100876)

通讯作者: 徐国爱, E-mail: xga@bupt.edu.cn

摘 要: 移动应用软件安全检测和防护是软件安全领域中的研究热点.传统的安全解决方案是安全厂商将其开发的 APP 安装到用户终端进行保护,但对于安全意识薄弱的普通用户而言,他们不了解安全威胁的严重性和安全管理 APP 的重要性,终端缺少安全威胁的防御能力,需要从威胁发生的源头和传播途径进行保护.从威胁发生的源头、途径和终端出发,实现了基于编程风格的源代码作者溯源追踪、移动应用安全加固及渠道监测、基于深度学习的移动应用安全检测,构建移动应用安全生态链,保障用户个人信息安全.在实际应用环境中验证了所提出方法的有效性,结果显示,该方法能够达到应用全方位安全防护的目的.另外,也对未来的研究方向进行了展望.

关键词: 移动应用安全;源代码作者溯源;安全加固;渠道监测;安全检测

中图法分类号: TP309

中文引用格式: 杨昕雨,徐国爱.移动应用安全生态链构建方法.软件学报,2017,28(11):3058−3071. http://www.jos.org.cn/1000-9825/5342.htm

英文引用格式: Yang XY, Xu GA. Construction method on mobile application security ecological chain. Ruan Jian Xue Bao/ Journal of Software, 2017,28(11):3058−3071 (in Chinese). http://www.jos.org.cn/1000-9825/5342.htm

Construction Method on Mobile Application Security Ecological Chain

YANG Xin-Yu, XU Guo-Ai

(School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Mobile application security detection and protection is an active research topic in the domain of software security. The traditional security solution is to install the APP developed by security vendors on user terminals. However, for the normal users lacking of security awareness, they do not understand the seriousness of security threat and the importance of security management, thus leading to insufficient terminal security defense. It is necessary to take protection from the threat source and transmission route. This paper implements various security measures including source code authorship attribution based on coding style, mobile application security reinforcement and channel monitoring, and mobile application security detection based on deep learning over the view of threat source, transmission route and threatened terminal. A mobile application security ecological chain is also constructed to protect users' personal information security. The paper verifies the effectiveness of proposed method in the practical application environment. The results show that it can achieve the goal of all-around application security protection. Future work in this research area is also discussed.

Key words: mobile application security; authorship attribution of source code; security reinforce; channel monitoring; security detection

随着移动互联网的飞速发展和智能终端的广泛普及,移动应用在满足人们日常工作和生活需要的同时,也面临越来越多的安全威胁.2015 年,国家互联网应急中心(CNCERT/CC)[1]通过自主捕获和厂商交换,共捕获移动互联网恶意程序数量近 148 万个,比 2014 年增长了 55.3%,而且主要针对安卓平台.按恶意行为进行分类,排名前

* 基金项目: 国家自然科学基金(U1536119, 61401038)
Foundation item: National Natural Science Foundation of China (U1536119, 61401038)
本文由复杂环境下的机器学习研究专刊特约编辑张长水教授推荐.
收稿时间: 2017-04-14; 修改时间: 2017-06-16; 采用时间: 2017-08-23

3 位的恶意行为分别是恶意扣费类、流氓行为类和远程控制类,占比分别为 23.6%,22.2%和 15.1%.CNCERT/CC 发现移动互联网恶意程序下载链接达 30 万余条,同比增长 7.2%,涉及的传播源域名 4 万余个、IP 地址近 2 万个,恶意程序传播次数达 8 384 万余次,比 2014 年增长了 9.8%.移动互联网恶意程序数量持续大幅度增长,应用商店安全检测标准参差不齐,智能终端黑色产业链不断发展,国家相关监管措施存在滞后,严重威胁用户财产和信息安全,实现移动应用软件的全方位安全保护,是目前信息安全领域的关键性问题.

传统的移动安全解决方案往往是安全厂商将自主开发的 APP 安装到用户终端进行保护,这种方法所提的保护仅仅是在终端层面,而且对于零日攻击或者高级持续性威胁(APT)攻击缺少安全防御能力.据艾瑞咨询[2]报告,手机病毒主要通过应用商店与第三方市场传播,即移动应用危害传播的“来源”是重灾区,也是移动安全保护需关注的主要领域.而且对于移动安全意识较弱的普通用户,他们不了解安全威胁的严重性和安全管理 APP 的重要性,终端缺少危害的防御能力,更需要从威胁发生的源头和途径层面进行保护.对此,建立一套完整的移动终端安全检测与防御体系,尽可能地覆盖威胁传播的源头、途径、终端各个环节,提供全方位的安全防护至关重要.

本文拟从源码作者溯源追踪、应用加固和渠道监测、应用软件终端安全检测这 3 个方面入手,增强终端安全性、提高途径可靠性、实现源头可追踪,并引入人工智能前沿技术,有效应对越来越复杂的网络攻击和未知威胁,降低现有威胁检测方案的误报率和漏报率,打造安全的移动应用安全生态链.

现有的移动应用软件终端安全检测技术可分为静态检测和动态检测两类[3−6].静态检测是指在不实际执行样本的情况下,对样本特征、行为或者缺陷等进行分析[7−9],比对待测应用特征和已有的恶意代码特征库中的样本相似性.静态检测全面而且速度快,但是需要人工更新恶意代码特征库,难以应对每天约有上万个恶意软件变种生成[10]的节奏,而且无法发现未知威胁.与静态检测相比,动态检测[11−14]是指在真实或者虚拟的测试环境中实际运行样本,分析结果更精确,但是存在路径覆盖的全面性问题.面对越来越复杂的网络攻击时,基于人工智能的检测是未来发展的趋势.在没有任何人工干预下,比如事先标记学习样本是恶意的还是合法的,深度学习的核心引擎不断自行学习升级,无需手工提取恶意代码特征.与传统的机器学习相比,基于深度学习的解决方案在检测首次发现的恶意软件、APT 攻击等方面呈现出突破性的成果[15−18].

移动应用加固现在广泛应用的措施是代码混淆和软件加壳.开发者可以利用代码混淆工具混淆自己的源程序,增加其被逆向的难度[19−21].目前,国内外均有成熟的代码混淆工具,比如 DexGuard、爱加密、梆梆加固.但随着逆向工程的发展,传统的混淆算法已经不再满足要求,或者是混淆算法代价过高而失去了其本身的意义.软件加壳[22]是对待保护的程序进行压缩或加密,但现有加固技术仅覆盖了移动应用保护的部分需求,且很多开发者或者公司迫于技术、成本或者开发周期的压力,不愿意对应用加固投入过多资源.对此,整合现有的移动安全加固技术,实现 APP 自动化加固,防止软件被破解、被非法加入病毒、计费 SDK、广告 SDK 等恶意代码,满足不同开发者的安全保护需求,维护开发者权益,是软件安全加固切实可行的解决方案.与此同时,全面监测 APP 在主流渠道中正版、盗版的使用情况[23−29],帮助开发者了解 APP 的盗版情况,也是现在移动应用安全整体态势分析的关注方向.

源代码作者溯源追踪现有的方案大多是从攻击样本的相似性、网络流量特征等方面出发,本文借鉴软件作者识别的相关技术[30−37],为源代码作者溯源追踪提供了新思路.软件作者识别是将源代码看作是文本文件,提取作者编程风格特征,在一组候选作者中判定给定代码的真实作者.在许多应用场景下,追溯软件作者源头都具有重要的实际应用价值,比如系统被攻击时,利用得到的病毒或者逻辑炸弹的代码片段追踪可疑攻击者,或者是为软件作者识别法庭审判提供辅助证据,提炼代码重构的证据等.

本文第 1 节对移动应用安全生态链的构建方法进行总述,其框架包括源代码作者溯源、应用加固及渠道监测、应用检测这 3 个环节.第 2 节提出基于编程风格的源代码作者溯源追踪方法,从源头追踪恶意代码传播者.第 3 节阐述移动应用安全加固及渠道监测方法,从途径防范恶意攻击和遏制盗版软件的传播和泛滥.第 4 节介绍基于深度学习的移动应用安全检测方法,从终端保障移动用户的个人隐私安全.第 5 节对所提出的方法进行实验验证和分析比对.第 6 节总结全文,并对未来值得关注的研究方向进行初步探讨.

1 研究框架

移动应用安全检测与防护体系是当前移动安全研究的热点问题,本文在现有的移动应用安全检测的研究基础上,结合人工智能领域的前沿进展,从危害传播的源头、途径和终端入手,搭建移动应用安全检测与防护体系,实现整个移动应用生态链的安全.整体技术研究方案如图 1 所示,包括 3 个部分.

(1) 源代码作者溯源.随着恶意软件新技术的发展和简易开发工具的传播,恶意软件开发成本不断降低,攻击成功率不断提高.恶意开发者将应用上传到第三方应用市场、论坛、网页等,利用社会工程学手段抓住用户心理弱点,吸引用户下载和安装恶意应用.源头危害离终端用户距离最远,但覆盖的用户数最广.本文在源代码可得的前提下,借鉴作者归因现有技术,实现基于编程风格的源代码作者溯源,从源头查找恶意代码的编写人员或者团队.

(2) 应用加固及渠道监测.网上盗版现象猖獗,同款应用在不同或者同一应用商店中出现多种不同作者、不同大小的版本.移动应用被篡改后危害严重,攻击者可能非法篡改应用插入广告进而获利,或者是非法篡改植入木马.此外,刷机现象屡见不鲜,ROM 内置已成为恶意病毒的主要传播渠道.对此,本文进行应用加固防护以及传播渠道监测,从途径遏制盗版软件的传播和泛滥.

(3) 智能终端安全检测.移动应用面临各种各样的安全威胁,攻击者可能利用应用软件安全漏洞,实现 8 类安全威胁,包括恶意扣费、隐私窃取、远程控制、恶意传播、资费消耗、系统破坏、诱骗欺诈和流氓行为.对此,本文在现有移动应用安全动态检测和静态检测技术的基础上,结合深度学习对终端应用实施安全检测,从终端保障用户的个人信息安全.

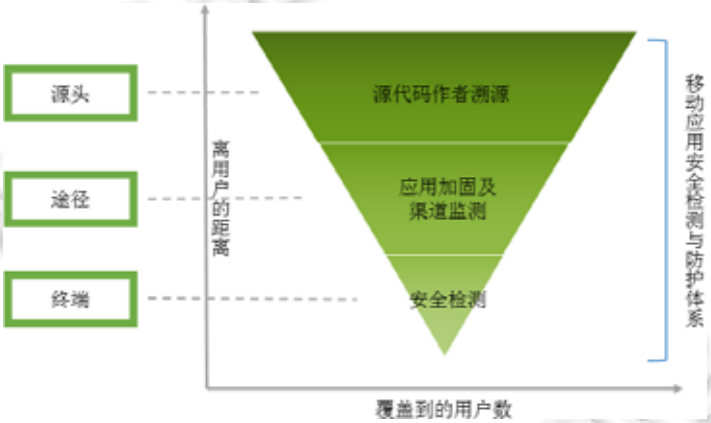

Fig.1 Overall technical solution

图 1 整体技术方案

2 基于编程风格的源代码作者溯源追踪

软件源代码可看作是一类特殊的文本,不同于纯文本文件,虽然编译器限制源代码的表达方式不如纯文本文件自由,但是程序员仍然会在源代码中留下指纹信息.例如,如果程序员过去写了一个排序代码,当再次面对排序问题时,他很可能使用已封装好的代码片段.一个人的编程习惯很难改变,具有一致性,这也是能够从编程风格分析程序作者的主要原因.本文提出了基于编程风格的软件源代码作者溯源方法,如图 2 所示.该方法首先将源代码中体现作者编程风格的特征归纳为程序词法度量标准、程序布局度量标准、程序结构度量标准、程序语法度量标准和程序风格度量标准.针对细化的体现软件作者编程风格的特征,剔除冗余信息后得到特征向量集.然后,特征向量输入到分类器,训练作者识别模型,判断给定代码的可疑作者.该方法包含两个模块,即特征提取模块和分类器模块.

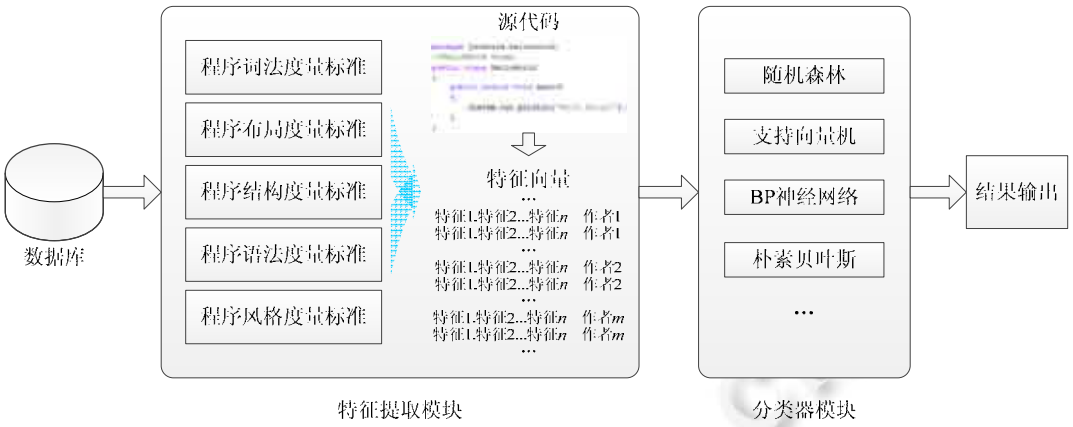

Fig.2 Authorship attribution of source code based on coding style

图 2 基于编程风格的源代码作者溯源方法

2.1 特征提取

特征提取模块主要完成数据清洗和特征提取,其中,数据清洗将网上爬取的原始数据集进行筛选过滤,得到海量由单一作者编写的源代码.特征提取根据自定义的与作者编程风格相关的度量标准,使用正则表达式匹配和抽象语法树(AST)遍历方法提取定义的特征度量标准.

真实、可靠的数据集是软件作者溯源的基础.网络上存在海量的开源代码,我们需要对其加以筛选,构建满足实验要求的数据集.比如,有时候软件开发者为了快速实现系统功能而从其他开源项目中复制粘贴部分代码片段,甚至是完全拷贝整个工程文件,这样的工程文件并不能反映作者的编程风格,所以数据集中不应该包括完全拷贝的工程文件.又如,软件开发者并不一定会对所传的工程文件署名,那么这种无标签的样本对我们也是无意义的,也不应该加入到数据集中.本文设置严格的数据筛选条件,从大规模的开源代码库中过滤源代码,这些源代码都具有唯一的作者标签.虽然我们不能保证单一的源代码文件完全是由一个作者开发的,比如,开发者 A 在开发者 B 的协助下完成某些较难的功能,但我们认为这种噪声是不可避免的,而且经过数据筛选和特征提取步骤,多个源代码文件的特征可以尽可能地减少噪声干扰.

经过数据清洗步骤,我们构建了带有作者标签的源代码库,之后可对其进行特征提取.目前,研究人员提出了各种各样的特征,并分析不同特征对分类准确率的贡献.本文在总结前人工作的基础上,定义了 5 类特征度量标准,包括程序词法、布局、结构、语法和风格特征,多角度、多方面地覆盖作者编程风格.其中,程序词法度量标准主要包括关键词的频率、n-gram 频率等;程序布局度量标准主要包括列表缩进风格、注释风格、条件语句比例、空行比例、距离操作符的平均空格符等;程序结构度量标准主要包括平均每个类的方法数量、接口相对于类的百分比等;程序语法度量标准主要包括抽象语法树的深度、抽象语法树的节点数目、程序复杂度等;程序风格度量标准主要包括平均行长度、命名长度、大写/小写/下划线/美元符命名习惯、for/while/do-while 使用惯例、if/if-else/switch-case 使用惯例等.这 5 类特征中,程序词法、布局和风格特征属于易混淆特征,即由于公司编程规范要求、编译器编译和优化、对抗溯源分析等原因易被开发人员规避;程序结构和语法特征属于抗混淆特征,即体现程序语法和语义的本质属性,反映了程序深层次的作者编程习惯,难以被恶意攻击人员绕过.无论是易混淆特征还是抗混淆特征,研究结果表明,均能有效识别作者.

提取特征采用正则表达式匹配和抽象语法树遍历两种方法.正则表达式处理简单的文本特征,实现简单而高效.本文选择用 Python 语言编写程序,通过输入目标文件的地址,然后可以自动扫描并打开目录下的所有文件夹,然后搜索整个文件夹下以“.java”(源代码是 Java 语言,后缀是“.java”)为后缀的源代码文件,然后逐行扫描每个文件,利用正则表达式匹配这些特征,记录到特征向量.

抽象语法树是源代码的抽象语法结构的树状表示,树上的每个节点都表示源代码中的一种结构,抽象语法

树并不会表示出真实语法出现的每一个细节,比如说,嵌套括号被隐含在树的结构中,并没有以节点的形式呈现.本文采用遍历抽象语法树的方法获取程序结构和语法特征,然后将正则表达式和抽象语法树获取的特征级联,作为最终的特征向量.

2.2 样本分类

机器学习在解决分类问题上已有很多成熟的算法,本文集成多种分类器,包括随机森林、支持向量机、BP神经网络和朴素贝叶斯,并且可进一步扩展.经特征提取后,数据集映射为特征向量,每一维均是数字表示,并且标有作者标签.为了方便机器处理,我们将作者名称按首字母先后排序,序号对应为作者标签.特征向量按比例划分为训练集和测试集.训练集输入分类器进行训练,测试集验证分类结果的准确率.

对于不同的数据集,各分类器各有其适用性.从实际需求出发,本文既提供单一分类器分类的准确率,方便使用者根据自身需要选择合适的分类器,又采用投票方式给出 4 种分类器的最终分类结果,也就是说,以过半的分类器判定的结果作为某一样本的作者,但若是 4 种分类器彼此之间差异较大,则以随机森林判定结果为准.由实际工程经验,随机森林的判定准确率和速度总体上优于其他分类算法.分类算法的选择方式灵活多样,便于辅助专家进行人工判断.

在样本数据有限的情况下,对于 BP 神经网络和朴素贝叶斯,为了防止过拟合的发生,本文采用 10 折交叉验证的方法验证分类算法的有效性.10 折交叉验证是指将数据集随机划分为 10 份,其中一份留作测试集,其余作为训练集,重复 10 次,计算均值记为分类的准确率.随机森林由于随机生成产生决策树的样本、随机选取构建决策树的特征值、树产生过程中选择 N 个最佳方向随机裂变,调整参数可以有效地控制过拟合.支持向量机采用libsvm,该软件包对支持向量机所涉及的参数调节相对较少,还有交叉校验的功能,防止过拟合.

3 移动应用安全加固及渠道监测

移动应用安全加固及渠道监测是为防止移动应用在运营推广过程中被篡改、盗版、二次打包、注入、反编译等破坏,对应用进行加固保护,并根据所提供的官方应用或应用名监测同款应用在不同商店、论坛等多方渠道的篡改、破解情况,为开发者提供有关渠道的最新数据,如图 3 所示.移动应用安全加固及渠道检测实现流程是:服务窗口接受任务,服务调度子系统将任务加入队列中,安全加固模块调用集成的软件加固 SDK 接口库或者是加固工具,提供所需服务;渠道监测从应用软件、下载站以及论坛自动化采集样本,根据相似性比对算法识别原版和盗版应用,生成整体态势分析报告.

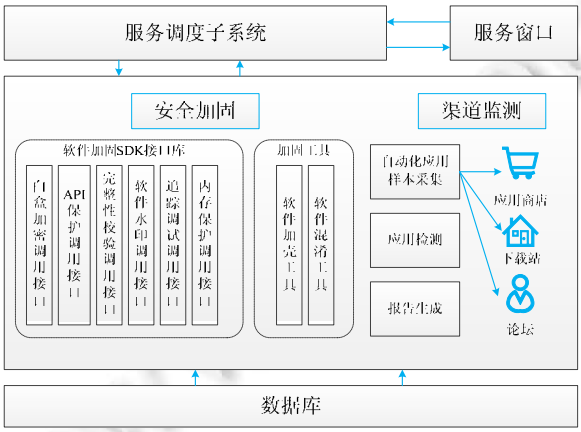

Fig.3 Mobile application protecting and channel monitoring service

图 3 移动应用保护和渠道监测服务

3.1 安全加固

安全加固模块整合现有软件加固 SDK 接口库,包括白盒加密调用接口、API 保护调用接口、完整性校验调用接口、软件水印调用接口、追踪调试调用接口和内存保护调用接口,对软件实施保护.具体功能包括:防逆向保护,即以加密代码的方式阻止反编译,从而防止 APP 被窃取代码;防篡改保护,即通过对 APP 的完整性保护,防止 APP 被篡改;反调试保护,即防止应用被外挂、木马偷窃账号密码、修改交易金额等;存储数据加密保护,即更底层、跨文件格式的数据加密,防止应用数据被窃取.安全加固模块也集合现有成熟的加固工具,包括软件加壳工具和代码混淆工具,方便无编程经验的普通用户快捷操作.

安全加固模块基于多种安全加固手段对软件版权进行保护,但开发者无需修改原程序,无需在开发过程中使用 SDK,并且不改变移动应用软件原有的用户体验,不增加软件运行的负担.安全加固模块服务流程简单,用户上传待保护 APP,经安全性验证判断其是否为恶意应用:若是恶意应用,则不提供保护;若是普通应用,则实现自动化安全加固.然后,用户下载加固包完成整个操作流程.

3.2 渠道监测

渠道监测模块结合自动化应用样本采集、自动化检测、自动化报告生成等机制,同时兼具人工手动研判功能,对发现的仿冒 APP 进行精确研判.其中,

- 自动化应用样本采集中的样本来源于应用商店、下载站和论坛,将要爬取的应用网站的应用列表的首页作为入口 URL 地址,对对应的网页页面进行抓取.抓取至本地后,对页面信息进行分析抽取,从中抽取到的应用信息将依照相应字段存入数据库中,抽取到的 URL 地址信息将作为新的目标 URL 传入系统进行爬取;在进行抓取过程的同时,调度平台将不断查询数据库,如果发现有新的应用信息入库,则调取应用信息的下载链接地址信息,并传入下载模块进行应用的下载,应用将被下载至应用软件库.由于下载过程是多线程的,调度平台中的下载调度模块将负责下载机阵列的调度使用.调度平台的分发调度模块则负责将应用软件库中的应用软件分发至检测模块,以便进行安全检测.
- 自动化检测提供 Android 应用仿冒检测,首先将应用采集样本库中的应用作为检测输入,输入到相应队列上.样本检测模块再从消息队列上获得应用样本信息,从应用 APK 存放路径找到应用 APK,使用 APKTool 工具对 APK 进行解析.然后,通过解析得到的应用包名从样本库中取相同包名的应用样本作为样本集,对其与本地正版应用规则库中相同包名的正版应用进行比对.最后,比对他们的签名中的公钥大素数是否一致:若一致,则判断应用为安全应用;若如不一致,则通过相似度对比队列传送类结构序列化为的 JSON 串,Androguard 功能模块收到消息后调用 Androguard 工具进行详细检测,将两种检测结果整合,生成带有应用对比相似度的完整报告.根据下载量、用户评分、时间戳等信息,也容易界定原版和盗版应用.
- 自动化报告生成针对盗版应用,分析报告中详细列出该应用的 MD5 值、下载渠道、APK 文件大小、APK 中 Activity、Service、Receiver、Permission、Provider 信息、源码分析、res 目录分析、meta-data 信息分析以及 so 库分析.同时,以月为周期,说明本月检测应用详细分析情况,并给出本月与上月对比数据总结统计分析,主要说明本月与上月仿冒应用数量统计及对比本月与上月仿冒应用分布情况.

4 基于深度学习的移动应用安全检测

移动应用安全检测结合静态检测、动态检测和深度检测技术,如图 4 所示.其中,静态检测技术基于特征码匹配,抽象行为特征判定待测样本与恶意样本相似度;动态检测在沙箱环境实际运行待测应用,对其敏感行为进行分析,静态和动态检测结果存在一定的误报率和漏报率;深度检测是基于现有的深度学习相关技术,无需人工干预,提高检测准确率.移动应用安全检测可对指定智能终端、批量移动应用软件进行自动化安全性评估,指出移动应用可能的恶意行为及恶意行为的位置.

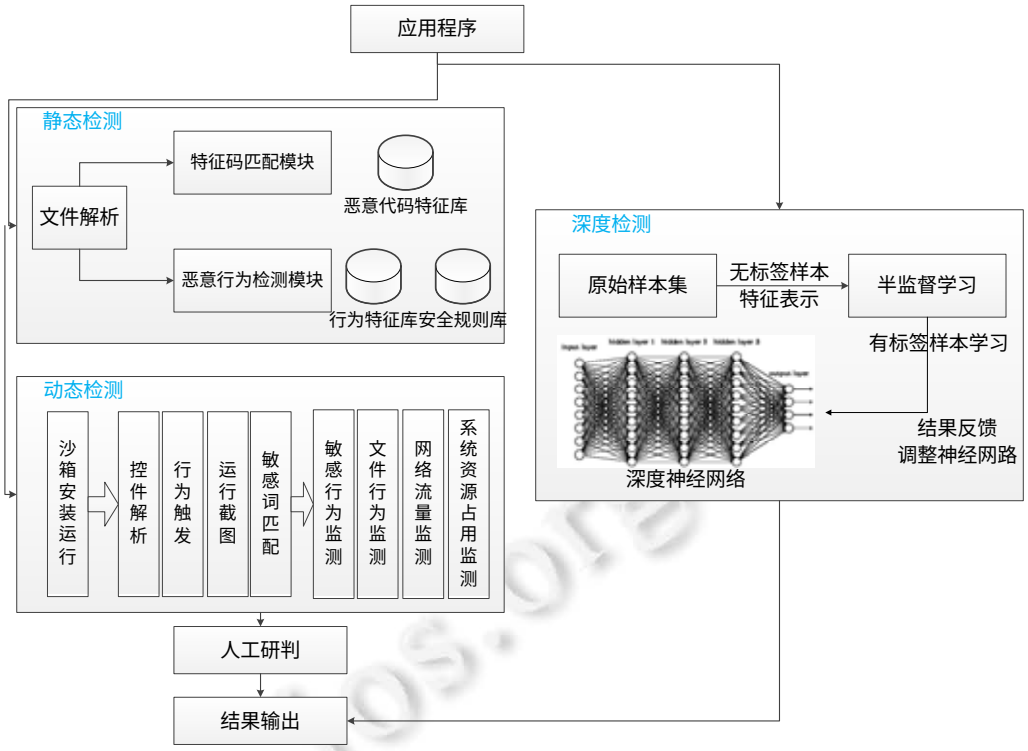

Fig.4 Mobile application security detection technology

图 4 移动应用安全检测技术

4.1 静态检测

静态检测在文件解析后,进行特征码匹配和恶意行为检测.特征匹配主要通过已知恶意程序库中的恶意样本特征对检测样本进行匹配,可以检测出含有已知恶意代码的应用程序.特征匹配具有准确率高的特点,能够检测出大量已知恶意代码.恶意行为检测通过反编译源代码,基于恶意行为特征库和安全规则库对源代码进行分析,除了能够发现已知的恶意应用程序以外,还能通过一系列恶意特征行为发现可疑的恶意样本,供深度分析、研究确认以及特征规则库更新.

静态检测模块主要由代码解析器、代码分析引擎、结果报告器、用户接口和安全规则库这 5 部分组成.

- 代码解析器的目的是负责对源程序进行词法、语法分析,抽象出足够多的信息并转换成中间表示,根据需要生成特定的语法树结构,为后续分析提供便利.为了实现此功能,该模块还可以支持解析项目工程文件,获取工程中全部源代码信息.
- 代码分析引擎的目标是根据规则库,对指定的代码语法树进行分析.同时,根据功能需求,又划分为数据流分析器、控制流分析器、结构分析器、安全分析调度器以及安全分析接口这 5 部分.其中,数据流分析器是在代码解析的基础上提取程序的数据流信息,数据流分析通过遍历抽象语法树,提取出指针变量、内存块、常量、函数结构等数据信息,并根据用户规则对这些信息进行筛选,并向程序分析模块提供接口以读取这些信息;控制流分析器主要是在代码解析的基础上提取程序的控制流信息,控制流分析根据规则,通过遍历 AST 生成对应的程序控制依赖图,并向安全分析调度模块提供接口以读取这些信息;结构分析器的目标是在代码分析引擎提取出的语法树基础上,根据安全规则库提供的代码分析规则,提取程序的主要结构(如入口点信息、主要函数名、函数关系等信息),调度数据流分析器和控制流分析器,完成对规则指定的关键变量的分析,并调用安全分析接口,完成对相关代码的安全性检测;分

析调度器是系统的主要调度模块,它可以根据安全规则库提供的信息调度结构分析器进行安全性分析,并生成报告表,提供接口供安全分析报告器调用;安全分析接口负责调用特定的代码.

- 结果报告器实现的功能是根据代码分析引擎的结果,根据规则库,将发现的与编码规范不符合的代码提交给用户.
- 安全规则库负责为代码分析引擎提供代码分析规则以及对应的检查清单支持,同时,可以提供风险改进策略报告.
- 用户接口模块负责与用户进行交互,一方面可以接受用户扫描代码的请求,另一方面则将扫描分析的结果输出给用户.

4.2 动态检测

静态安全分析技术可以有效地鉴别存在安全威胁的恶意代码,但对于尚未添加到恶意代码库中的新型程序,无法使用静态特征码扫描技术进行检测.静态安全分析技术的时效性相对不足,通常,恶意程序只有预先经过发现并提取特征码和特征行为等步骤后,才可使用静态检测扫描.动态行为检测通过全面、准确、实时的智能手机终端监测,收集恶意行为相关数据,以此进行恶意代码的判断分析,有效地发现未知恶意代码.

动态分析检测是基于恶意行为的分析,是指在受控环境中运行一个应用程序并监测其行为,如利用虚拟机模拟用户执行程序,在程序的运行过程中,通过动态拦截的方式判断程序中是否有可疑行为.为了更好地捕获和分析应用程序的动态行为,各种启发式技术已经被研究运用,比如监控文件的变化、网络活动、进程、线程和系统调用跟踪.为了捕获软件在运行期间的行为特征,软件必须长时间被执行并处于激活状态.

动态分析实现步骤:自动激活应用的行为,同时对手机进行实时的终端监控,在手机运行的整个过程中,从网络流量、手机数据库、手机设备等多方面进行终端监控,检测出恶意代码进行的恶意行为.

4.3 深度检测

神经网络对于解决分类、回归问题效果良好,但目前,大多数学习方法都是浅层结构,在样本和计算单元有限的情况下,对于复杂函数的表示能力具有局限性,在解决复杂分类问题时,其泛化能力会受到制约.深度学习通过学习深层非线性网络结构,逼近复杂函数,实现输入数据的分布式表示,可以用较少的参数表示复杂函数,具有从少数样本中学习数据集本质特征的强大能力.

本文利用大量无标签样本自主学习应用软件特征.该特征能够完全表示应用,涵盖应用的各维度信息,但是并不是所有的信息都能用于恶意应用检测,即存在大量的信息冗余.对此,我们人工标记部分样本,输入到深度神经网络进行有监督学习,识别出对分类有贡献的特征集合,构建恶意应用检测模型,以此作为后续代码检测的依据.更具体地,原始应用数据集经反编译预处理后,对操作码以 one hot 方式编码为后续处理的样本格式.本文先将少量无标签样本输入到自编码器,通过编码和反编码后的输出与原始样本间的差值,反复调整自编码器结构,以此自主学习应用的表示.然后,在大量无标签样本上验证该表达方式的有效性.为了加快处理速度,本文采用多线程、GPU 同时进行实验.然后,人工标记部分样本,即通过逆向工程技术手动判别其恶意性,以之前学习到的应用表达方式将其转化为特征向量,输入到深度神经网络——卷积神经网络(CNN).深度神经网络包括多层隐藏层,每层包含若干个神经元,每个神经元包含两个功能:一个是对输入的加权叠加;另一个是对将叠加的值进行变换(进行规约),并将变换后的值传入下一层.输出层根据得到的多层运算的最后结果进行决策,根据预测值与实际值之间的偏差进行有监督的学习,反复调整神经网络结构和参数,构建恶意应用检测的复杂网络结构.这样就实现了对未知样本的恶意性判别.深度学习对原始数据进行多项与多层运算,计算是非线性的,在人工干预较少的情况下,可自主学习特征.相对于传统的机器学习,在检测首次发现的恶意应用上具有明显优势,弥补了静态检测和动态检测的不足.

5 实验验证

5.1 基于编程风格的源代码作者溯源追踪

合适的数据集对软件作者的正确识别至关重要,本文旨在解决真实环境下的作者识别问题,但现阶段并没有公开可用的数据集,所以需要自行爬取开源网站构建数据集.截至 2015 年,GitHub 拥有超过 900 万用户和 2 110 万多个存储库,是世界上最大的代码拥有者.海量带有作者标签的代码,而且标识信息清楚(代码由单一作者提供还是多作者合作、是否为原创代码等),是我们选取 GitHub 作为爬取的开源网站的原因.为保证分类的准确率,需要对爬取的原始数据集进行清洗:首先,所选代码只能由单一作者编写,由多作者合作完成的代码需要剔除,多作者识别问题不在本文讨论范围内;其次,由于分类器采用机器学习算法,每个作者的文件数目不能过少,根据实际工程经验,本文设定每个作者的文件数目不能低于 10 个;然后,第三方库代码并不体现作者编程风格,我们从 Maven 上获取第三方库名单,采用黑名单机制去除工程中的第三方库;最后,Android 框架代码和自动测试用例等,比如 JUnit,并不能体现作者编程风格,也需要删除.目前,针对 Java 语言的源代码作者识别工作与 C/C++等语言相比还不够成熟,本文实现 Java 语言的软件作者溯源追踪.由于 Java 的工程数目比 C/C++少很多,而且为了实现作者识别的目的,我们设定了严格的数据清洗条件,所以大规模的作者被删除,但是目前的数据库规模对于 Java 作者鉴别而言不少于同领域其他工作.截至 2016 年 9 月,我们总共爬取了 100 个作者的存储库.然后依据数据清洗规则,筛选后数据集包含 40 个作者的 3 022 个 Java 文件,作者的 Java 文件数目从 11 个到 712 个不等.代码平均长度为 98.63 行,最少的只有 16 行,最多的有 11 418 行.

基于编程风格的源代码作者溯源追踪实验在 Windows 7 64 位操作系统下完成,系统内存 8G,CPU 3.30GHz;采用 Python 2.7,MATLAB R2014a,Eclipse 4.3.2 版本.实验采用 10 折交叉验证,取平均值后各分类器的准确率和运行时间见表 1.

Table 1 Accuracy and running time of different classifers

表 **1** 分类器准确率和运行时间

分类器	准确率(%)	运行时间(s)
随机森林	79.735	9.679
支持向量机	73.642	201.220
BP 神经网络	83.107	48.200
朴素贝叶斯	49.007	11.974

在给定的数据集下,本文所用的 BP 神经网络包括输入层、隐藏层和输出层,输入层神经元数目等于特征数目,隐藏层根据实际工程经验反复调整确定,输出层神经元数目等于待分类的作者数目.BP 神经网络的准确率优于其他分类器,高达 83.107%.朴素贝叶斯算法由于事先并不知道输入特征数据的确切分布,设置为 kernel 表示核密度平滑估计.朴素贝叶斯的准确率最低,只有 49.007%.随机森林的树的数目设置为 200,其准确率达到了 79.735%.支持向量机训练过程中的惩罚系数为 2,核函数的宽度为 0.01,其准确率达到了 73.642%.神经网络的参数调整对生成的模型的准确率影响较大,本文的参数在前人工作的基础上根据工程经验进行调优.值得一提的是,支持向量机的运行时间明显高于其他分类器,因为本文采用 libsvm 进行参数寻优,该运行时间包含了该过程所耗费时间.总体来说,各分类器的运行时间代价均在合理范围内.本文既提供随机森林、支持向量机、BP 神经网络和朴素贝叶斯各单一分类器的分类结果,方便对于系统分类算法和数据集有深入了解的使用者根据实际需要选择分类器,也集成 4 类分类器,以投票方式给出最终分类结果,使得并不具有专业知识的使用者直观了解分类情况.具体投票方式是过半的分类器(2 个及 2 个以上)的判断结果作为该样本的作者,但倘若 4 个分类器间给出的可能作者差异较大,则以随机森林判定结果为准.之所以选择随机森林作为默认分类器,是因为从分类准确率、运行时间和参数选择等角度综合衡量,该算法优于其他分类器.本文现阶段只是简单集成了一些分类算法,为从源头分析恶意代码的开发者提供了新思路.准确率的提高还需要进一步细化作者编程风格特征、优化神经网络等,是本文后续的研究方向.

5.2 移动应用安全加固及渠道监测

为了实现移动应用安全保护,本文集成一套 SDK 以及软件加壳、代码混淆工具,保障 DEX、so 库、内存和数据等的安全.本文加固实验环境为 Ubuntu 64 位操作系统,JDK 1.8 版本.限于篇幅,本节只列举加壳和加密前后的变化.软件加壳是对 Android DEX 文件进行加壳保护,逆向工具解析 APK 后只看到壳代码,无法真正看到源代码,真正的源代码打包后加密存放于其他位置,有效防止静态调试器对 APK 进行分析破解.在 DEX 整体加壳基础上,DEX 分离加壳是将函数名、函数体进行分离,APK 运行时函数体动态加载到内存,有效防止内存 DUMP 全部源代码,如图 5 所示.

加固前源代码全加载　　加固后源代码动态加载

Fig.5　Source code loading way before and after shelling

图 5　加壳前后源代码加载方式

H5 加固对 HTML,JS 提供加密功能,但不是简单的文件加密,而是内容加密,可以更好地适用 H5 文件.自动实现加解密功能,无需用户参与,并对 H5 文件原功能不会产生影响,无论是在客户端还是服务器本地存储都是密文形式,加密前后效果如图 6 所示.

Fig.6　H5 source code before and after encryption

图 6　H5 源码加密前后图

本文从应用商店、下载站、论坛等爬取移动应用软件,分析 APP 所有路径文件及代码,判断哪些项目或代码被修改过,监测正版软件和盗版软件的版本、渠道、下载源、下载量等信息,在用户配合的情况下,辅助商城实现盗版 APP 的快速下降.本文以某 APP(为隐私起见,已隐去应用名称)的分析结果为例进行说明,正版应用信息包括应用名称、包名和证书.统计版本的正、盗版个数,该 APP 各版本应用共 120 个,其中正版 72 个,盗版 48 个.图 7 是某一款盗版应用(同样隐去应用名称)的渠道,共计 58 个,其中包括豌豆荚、安卓商店、机锋市场等主

流商店,该监测报告可以辅助移动应用商城对盗版应用进行下架清理.

Fig.7 Channel monitoring report

图 7 渠道监测报告

5.3 基于深度学习的移动应用安全检测

为了验证本文提出的移动应用安全检测方法的有效性,本文全量获取中国移动 MM 商城、咪咕游戏、咪咕动漫、咪咕视频中含有的所有手机软件,监测手机软件恶意违规行为;全量获取苹果官方商城中中国移动开发的手机软件,监测应用的缺陷漏洞问题.本文静态及动态检测实验环境为 Windows 7 64 位操作系统,JDK 1.8 版本. Android 模拟器配置:Device “6.0” WVGA;Target Android 4.0-API Level 14;SD card Size 100.测试真机为 SAMSUNG Galaxy nexus,ROM 固件 cyanogenmod CM-11.0.深度检测实验环境为 Ubuntu 64 位操作系统,采用 TensorFlow 框架,为加快处理速度,使用 Nvidia GTX 1080 GPU.

2015 年 12 月 1 日~2016 年 12 月 2 日,共监测中国移动 MM 商城、咪咕游戏、咪咕动漫手机免费应用 53 508 款,监测到的恶意违规应用 139 款及待规范应用 4 210 款;监测收费软件 2 139 款,发现 3 款待规范应用;监测 iOS 应用 57 款,发现 9 款存在漏洞风险的应用.在表 2 所示的恶意违规应用中,其中行为违规应用 137 款,包括隐私窃取类的 119 款,占比为 86.86%;资费消耗类的 11 款,占比为 8.03%;流氓行为类的 7 款,占比为 5.11%.内容违规应用 2 款,属于淫秽色情违规应用.监测到待规范应用 4 210 款,行为涉及向端口号发短信、获取用户地理位置信息和手机号码信息.

Table 2 Statistical result of malicious violation and non-standard applications

表 2 恶意违规和待规范应用统计结果

监测中国移动自有业务应用			总数
恶意违规应用	行为违规	137	139
	内容违规	2	
待规范应用	向端口号发短信	2 050	4 210
	获取用户地理位置信息	488	
	获取用户手机号码信息	1 672	

表 3 显示的是各专项监测结果,发现 2 204 款应用经过了加固处理,并且使用 360 加固的应用居多.有 36 款应用含有广告插件,其中 13 款应用存在恶意资费消耗、流氓软件行为,待进一步规范.仅 1 款应用含有恶意积分墙插件.852 款应用含有自有升级通道,554 款具有下载链接,其中 8 款用过链接下载的应用有待规范的行为.210 款应用含有模糊扣费窗口,但都没有不知情定制的行为.

Table 3 Statistical result of specific monitoring

表 3 监测专项统计结果

监测专项	发现数量	违规/待规范
加固处理应用	2 204	0
含有广告插件应用	36	13
含有积分墙应用	1	1
含有自有升级通道应用	852	8(恶意链接)
含有模糊扣费窗口应用	211	0

6 总结与讨论

智能化安全检测与防护体系研究,是当前移动应用安全研究的热点问题.它不仅在建模和结构设计上有很大的创新空间,其实现过程也有赖于新技术的探索与发明.本文实现基于编程风格的源代码作者溯源追踪、移动应用安全加固及渠道监测、基于深度学习的移动应用安全检测,从危害传播的源头、途径和终端入手,构建移动应用安全生态链,保证用户的移动应用安全.

源代码作者溯源借鉴作者归因技术,提取源代码中体现作者编程风格的特征,利用机器学习进行训练和测试,进而判断给定源代码的作者.在软件源代码可得的情况下,本文从攻击样本的静态特征中提炼攻击者的指纹线索信息,识别攻击样本开发者,从源头狙击恶意代码.

应用加固及渠道监测集成一套安全加固 SDK 和软件加壳、代码混淆工具,为用户提供全方位的应用软件安全保护,防止应用被逆向分析、非法篡改等.并且监测原版和盗版应用对,提供盗版应用的名称、版本和渠道等信息,以便应用商城对盗版应用进行下架清理.

安全检测在现有静态和动态检测技术基础上,结合人工智能的前沿进展,实现基于深度学习的移动应用软件安全检测系统,利用大量无标签样本自主学习应用软件特征,并使用少量有标签样本对深度神经网络进行结构调整和权值微调,降低检测的误报率和漏报率,精准定位恶意代码位置及行为.

本文提出了一套移动应用安全生态链的构建方法,实现移动应用的全生命周期的安全防护,但是溯源分析的分类准确率还可以进一步提高,安全检测的深度神经网络结构和参数还可以优化,并且如何使得其可以抵抗更为复杂的高级持续性威胁,也是未来的研究重点.

References:

[1] CNCERT/CC. 2015 China Internet security report. 2015 (in Chinese). http://10.3.200.202/cache/11/03/cert.org.cn/9ef7d80315e96fbe5617b189a9e2715a/2015annualreport.pdf

[2] IResearch and Safe Manager. 2013 Mobile security report. 2013 (in Chinese). http://www.199it.com/archives/188839.html

[3] Rastogi V, Chen Y, Jiang XX. Catch me if you can: Evaluating Android anti-malware against transformation attacks. IEEE Trans. on Information Forensics and Security, 2014,9(1):99−108. [doi: 10.1109/TIFS.2013.2290431]

[4] Naval S, Laxmi V, Rajarajan M, Gaur MS, Conti M. Employing program semantics for malware detection. IEEE Trans. on Information Forensics and Security, 2015,10(12):2591−2604. [doi: 10.1109/TIFS.2015.2469253]

[5] Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: Semantics-Based detection of Android malware through static analysis. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2014. 576−587. [doi: 10.1145/2635868.2635869]

[6] Das S, Liu Y, Zhang W, Chandramohan M. Semantics-Based online malware detection: Towards efficient real-time protection against malware. IEEE Trans. on Information Forensics and Security, 2016,11(2):289−302. [doi: 10.1109/TIFS.2015.2491300]

[7] Zhou YJ, Jiang XX. Dissecting Android malware: Characterization and evolution. In: Proc. of the 2012 IEEE Symp. on Security and Privacy. 2012. 95−109. [doi: 10.1109/SP.2012.16]

[8] Shan ZY, Wang X. Growing grapes in your computer to defend against malware. IEEE Trans. on Information Forensics and Security, 2014,9(2):196−207. [doi: 10.1109/TIFS.2013.2291066]

[9] Wang W, Wang X, Feng DW, Liu JQ, Han Z, Zhang XL. Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans. on Information Forensics and Security, 2014,9(11):1869−1882. [doi: 10.1109/TIFS.2014.2353996]

[10] Cesare S, Xiang Y, Zhou WL. Control flow-based malware variant detection. IEEE Trans. on Dependable and Secure Computing, 2014,11(4):307−317. [doi: 10.1109/TDSC.2013.40]

[11] Park Y, Reeves DS, Stamp M. Deriving common malware behavior through graph clustering. Computers and Security, 2013,39(Part B):419−430. [doi: 10.1016/j.cose.2013.09.006]

[12] Chandramohan M, Tan HBK, Briand LC, Padmanabhuni BM. A scalable approach for malware detection through bounded feature space behavior modeling. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2013. 312−322. [doi: 10.1109/ASE.2013.6693090]

[13] Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Traon YL, Octeau D, McDaniel P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: Proc. of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2014. 259−269. [doi: 10.1145/2594291.2594299]

[14] Alazab M. Profiling and classifying the behavior of malicious codes. Journal of Systems and Software, 2015,100:91−102. [doi: 10.1016/j.jss.2014.10.031]

[15] Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features. In: Proc. of the 2015 10th Int'l Conf. on Malicious and Unwanted Software. 2015. 11−20. [doi: 10.1109/MALWARE.2015.7413680]

[16] David OE, Netanyahu NS. Deepsign: Deep learning for automatic malware signature generation and classification. In: Proc. of the 2015 Int'l Joint Conf. on Neural Networks. 2015. 1−8. [doi: 10.1109/IJCNN.2015.7280815]

[17] Richardson F, Reynolds D, Dehak N. Deep neural network approaches to speaker and language recognition. IEEE Signal Processing Letters, 2015,22(10):1671−1675. [doi: 10.1109/LSP.2015.2420092]

[18] Huo X, Li M, Zhou ZH. Learning unified features from natural and programming languages for locating buggy source code. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. 2016. 1606−1612.

[19] Balachandran V, Emmanuel S. Software protection with obfuscation and encryption. In: Proc. of the Int'l Conf. on Information Security Practice and Experience. 2013. 309−320. [doi: 10.1007/978-3-642-38033-4_22]

[20] Cheng R, Zhang FG. Obfuscation for multi user encryption and its application in cloud computing. Concurrency and Computation: Practice and Experience, 2015,27(8):2170−2190. [doi: 10.1002/cpe.3399]

[21] He YX, Chen Y, Wu W, Chen N, Xu C, Liu JB, Su W. A program flow-sensitive self-modifying code obfuscation method. Computer Engineering and Science, 2012,34(1):79−85 (in Chinese with English abstract).

[22] Dong H. Research on the detection and protection technology of mobile applications [Ph.D. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2014 (in Chinese with English abstract).

[23] Lim HI, Park H, Choi S, Han T. A method for detecting the theft of Java programs through analysis of the control flow information. Information and Software Technology, 2009,51(9):1338−1350. [doi: 10.1016/j.infsof.2009.04.011]

[24] Stojanović S, Radivojević Z, Cvetanović M. Approach for estimating similarity between procedures in differently compiled binaries. Information and Software Technology, 2015,58:259−271. [doi: 10.1016/j.infsof.2014.06.012]

[25] Sæbjørnsen A, Willcock J, Panas T, Quinlan D, Su ZD. Detecting code clones in binary executables. In: Proc. of the 18th Int'l Symp. on Software Testing and Analysis. 2009. 117−128. [doi: 10.1145/1572272.1572287]

[26] Tamada H, Nakamura M, Monden A, Matsumoto K. Java birthmarks—Detecting the software theft—. IEICE Trans. on Information and Systems, 2005,88(9):2148−2158. [doi: 10.1093/ietisy/e88-d.9.2148]

[27] Desnos A. Android: Static analysis using similarity distance. In: Proc. of the 45th Hawaii Int'l Conf. on System Sciences. 2012. 5394−5403. [doi: 10.1109/HICSS.2012.114]

[28] Tian ZZ, Zheng QH, Liu T, Fan M, Zhuang E, Yang ZJ. Software plagiarism detection with birthmarks based on dynamic key instruction sequences. IEEE Trans. on Software Engineering, 2015,41(12):1217−1235. [doi: 10.1109/TSE.2015.2454508]

[29] Crussell J, Gibler C, Chen H. Andarwin: Scalable detection of android application clones based on semantics. IEEE Trans. on Mobile Computing, 2015,14(10):2007−2019. [doi: 10.1109/TMC.2014.2381212]

[30] Caliskan-Islam A, Harang R, Liu A, Narayanan A, Voss C, Yamaguchi F, Greenstadt R. De-Anonymizing programmers via code stylometry. In: Proc. of the 24th USENIX Conf. on Security Symp. 2015. 255−270.

[31] Burrows S, Tahaghoghi SMM. Source code authorship attribution using *n*-grams. In: Proc. of the 12th Australasian Document Computing Symp. Melbourne: RMIT University, 2007. 32−39.

[32] Gray AR, Sallis PJ, MacDonell SG. Software forensics: Extending authorship analysis techniques to computer programs. Information Science Discussion Papers Series No.97/14, 1997.

[33] Neme A, Pulido JRG, Muñoz A, Hernandez S, Dey T. Stylistics analysis and authorship attribution algorithms based on self-organizing maps. Neurocomputing, 2015,147:147−159. [doi: 10.1016/j.neucom.2014.03.064]

[34] Burrows S, Uitdenbogerd AL, Turpin A. Comparing techniques for authorship attribution of source code. Software: Practice and Experience, 2014,44(1):1−32. [doi: 10.1002/spe.2146]

[35] Wisse W, Veenman C. Scripting DNA: Identifying the Javascript programmer. Digital Investigation, 2015,15:61−71. [doi: 10.1016/j.diin.2015.09.001]

[36] Ding HB, Samadzadeh MH. Extraction of Java program fingerprints for software authorship identification. Journal of Systems and Software, 2004,72(1):49−57. [doi: 10.1016/S0164-1212(03)00049-9]

[37] Alrabaee S, Saleem N, Preda S, Wang LY, Debbabi M. OBA2: An onion approach to binary code authorship attribution. Digital Investigation, 2014,11:S94−S103. [doi: 10.1016/j.diin.2014.03.012]

附中文参考文献:

[1] 国家计算机网络应急技术处理协调中心.2015 年中国互联网网络安全报告.2015. http://10.3.200.202/cache/11/03/cert.org.cn/9ef7 d80315e96fbe5617b189a9e2715a/2015annualreport.pdf

[2] 艾瑞咨询&安全管家.2013 年移动安全报告.2013. http://www.199it.com/archives/188839.html

[21] 何炎祥,陈勇,吴伟,陈念,徐超,刘健博,苏雯.基于程序流敏感的自修改代码混淆方法.计算机工程与科学,2012,34(1):79−85.

[22] 董航.移动应用程序检测与防护技术研究[博士学位论文].北京:北京邮电大学,2014.

杨昕雨(1991－),女,吉林长岭人,博士生,主要研究领域为软件安全.

徐国爱(1972－),男,博士,教授,博士生导师,主要研究领域为软件安全,信息安全管理,密码学.