软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007046]

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

智能合约安全漏洞检测研究进展*

崔展齐[1], 杨慧文[1], 陈 翔[2], 王林章[3]

[1](北京信息科技大学 计算机学院, 北京 100101)

[2](南通大学 信息科学技术学院, 江苏 南通 226019)

[3](计算机软件新技术全国重点实验室 (南京大学), 江苏 南京 210023)

通信作者: 王林章, E-mail: lzwang@nju.edu.cn

摘 要: 智能合约是运行在区块链合约层的计算机程序, 能够管理区块链上的加密数字货币和数据, 实现多样化的业务逻辑, 扩展了区块链的应用. 由于智能合约中通常涉及大量资产, 吸引了大量攻击者试图利用其中的安全漏洞获得经济利益. 近年来, 随着多起智能合约安全事件的发生 (例如 TheDAO、Parity 安全事件等), 针对智能合约的安全漏洞检测技术成为国内外研究热点. 提出智能合约安全漏洞检测的研究框架, 分别从漏洞发现与识别、漏洞分析与检测、数据集与评价指标这 3 个方面分析现有检测方法研究进展. 首先, 梳理安全漏洞信息收集的基本流程, 将已知漏洞根据基础特征归纳为 13 种漏洞类型并提出智能合约安全漏洞分类框架; 然后, 按照符号执行、模糊测试、机器学习、形式化验证和静态分析 5 类检测技术对现有研究进行分析, 并讨论各类技术的优势及局限性; 第三, 整理常用的数据集和评价指标; 最后, 对智能合约安全漏洞检测的未来研究方向提出展望.

关键词: 区块链; 智能合约; 安全漏洞; 漏洞检测

中图法分类号: TP311

中文引用格式: 崔展齐, 杨慧文, 陈翔, 王林章. 智能合约安全漏洞检测研究进展. 软件学报. http://www.jos.org.cn/1000-9825/7046.htm

英文引用格式: Cui ZQ, Yang HW, Chen X, Wang LZ. Research Progress of Security Vulnerability Detection of Smart Contracts. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7046.htm

Research Progress of Security Vulnerability Detection of Smart Contracts

CUI Zhan-Qi[1], YANG Hui-Wen[1], CHEN Xiang[2], WANG Lin-Zhang[3]

[1](School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China)

[2](School of Information Science and Technology, Nantong University, Nantong 226019, China)

[3](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Smart contracts are computer programs running in the contract layer of the blockchain, which can be used to manage cryptocurrencies and data on the blockchain, realize diverse business logic, and expand the application of the blockchain. A large number of assets are stored in smart contracts, which attract attackers to steal the assets and obtain economic benefits via security vulnerabilities. In recent years, with the frequent occurrence of smart contract security incidents (such as TheDAO and Parity security incidents), the security vulnerability detection technique for smart contracts has become a hot research topic. This study proposes a research framework for detecting security vulnerabilities of smart contracts and analyzes the research progress of existing vulnerability detection techniques from three aspects: vulnerability discovery and identification, vulnerability analysis and detection, and dataset and evaluation indicators. Firstly, the basic process of collecting security vulnerability information is sorted out, and the security vulnerabilities are classified into 13 types according to their basic characteristics. A classification framework for security vulnerabilities of smart contracts is proposed. Secondly, existing techniques are studied in terms of symbolic execution, fuzzing testing, machine learning, formal verification, and static

* 基金项目: 江苏省前沿引领技术基础研究专项 (BK202002001); 国家自然科学基金 (61702041); 北京信息科技大学“勤信人才”培育计划 (QXTCP C201906)
收稿时间: 2022-04-07; 修改时间: 2023-05-10; 采用时间: 2023-09-01; jos 在线出版时间: 2024-01-03

analysis, and the advantages and limitations of each technique are analyzed. Thirdly, the commonly used datasets and evaluation indicators are summarized. Finally, potential research directions for security vulnerability detection of smart contracts in the future are discussed.

Key words: blockchain; smart contract; security vulnerabilities; vulnerability detection

区块链是以区块作为单位的链式结构, 区块由区块头和区块体 2 部分组成, 前者存储前一个区块的哈希值、当前区块时间戳和块难度等信息, 后者存储交易事务信息[1]. 区块链参与者提交的事务信息保存在彼此相连的区块中, 区块链使用不对称加密和分布式一致性算法保证区块链信息安全和账本一致性[2]. 区块链去中心化、可信任等特性使其在数字加密货币以及金融领域取得了广泛应用, 如比特币[3]及基于区块链的证券交易平台 Chain (http://www.chain.com/) 等.

智能合约是运行在区块链合约层的计算机程序[4], 由合约存储、余额和程序代码组成[5]. 区块链实现了去中心化存储, 智能合约在区块链的基础上实现了去中心化计算[1]. 智能合约能够管理区块链上的资金或数据, 其去中心化、公开透明、不可篡改以及可追溯等特性使智能合约在金融管理、物联网[6]以及医疗[7]等领域取得广泛应用. 目前, 已有很多区块链平台支持通过智能合约操作区块链, 如以太坊[8]、超级账本 (https://www.hyperledger.org/projects) 和 EOS (https://eos.io/) 等. 由于比特币平台仅支持通过脚本操作比特币, 不能实现复杂的业务逻辑, 通常不将比特币脚本视为智能合约.

智能合约能够实现较为复杂的控制逻辑, 实现多样化的业务需求. 然而, 智能合约的灵活性也为其带来了巨大安全风险. 一方面, 由于智能合约的编程体系发展历时较短, 开发者对区块链特性以及智能合约编程语言不够了解和熟悉, 导致其中存在较多漏洞. 另一方面, 智能合约常被用于管理区块链上的资产, 容易吸引攻击者关注, 大量攻击者试图通过潜在漏洞攻击智能合约, 以窃取他人财产. 已经发生过多起智能合约安全事件, 例如, 2016 年 6 月, 攻击者利用 TheDAO 合约中的重入漏洞窃取了价值 6 000 万美元的以太币 (https://www.coindesk.com/learn/2016/06/25/underst anding-the-dao-attack/); 2017 年 7 月, 攻击者利用 Parity 合约中的权限控制漏洞篡夺了合约权限, 窃取了价值 3 000 万美元的以太币 (https://hackingdistributed.com/2017/07/22/deep-dive-par ity-bug/); 2018 年 4 月, 攻击者利用 BEC 合约中的整型溢出漏洞使 BEC 代币价值归零 (https://blog.peckshield.com/2018/04/22/batchOverflow) 等. 据 SlowMist Hacked (https://hacked.slowmist.io/) 统计, 截至 2021 年 12 月, 各区块链平台共计发生了 550 起安全事件, 损失金额达 218 亿美元. 此外, 与传统软件不同, 智能合约具有不可篡改性, 合约部署在区块链平台后难以通过补丁进行漏洞修复[9], 在部署智能合约前对其进行全面的安全检测极为重要.

智能合约中的安全漏洞指其中隐藏的错误, 这些错误可能会导致产生不正确或意外的结果, 或程序以非预期的方式执行. Bartoletti 等人[10]对以太坊平台上的智能合约进行分类统计, 结果表明 811 个智能合约中存在 373 个金融类合约, 且 759 611 次事务执行中 624 046 次与金融交易相关, 即以太坊平台中 46% 的智能合约和 82.2% 事务执行与数字资产的存储或转移相关. 因此智能合约安全漏洞与通常意义下的软件安全漏洞不同, 前者相较于后者能够更直接地处理数字资产, 智能合约中的安全漏洞若被不法分子或攻击者利用, 将直接对智能合约中管理的数字资产造成威胁.

因此, 智能合约的安全漏洞检测得到了研究人员的广泛关注. 例如, Luu 等人[11]通过符号执行检测 Solidity 智能合约中重入、事务顺序依赖、时间戳依赖和未检测 CALL 返回值 4 种安全漏洞; Jiang 等人[12]利用模糊测试技术对 Solidity 智能合约中的贪心、无 gas 发送和时间戳依赖等 7 种安全漏洞进行检测等. 目前, 已有一些文献对智能合约安全漏洞检测问题进行了调研. 付梦琳等人[13]对智能合约安全漏洞挖掘技术进行了调研, 总结了 10 种漏洞类型并对 6 种检测方法进行了比较分析; 倪远东等人[14]对漏洞分类进行了讨论, 将已有漏洞类型划分为 3 个层面, 即高级语言、虚拟机和区块链, 并对 15 种主要安全漏洞进行了分析, 总结了 3 种智能合约安全防御策略; 郑忠斌等人[15]对 14 种安全漏洞进行了分析, 并提供了防范策略; 钱鹏等人[16]介绍了 15 种典型安全漏洞及 5 个典型安全事件, 并分 5 类检测技术类别对 25 种检测方法进行了描述, 总结了各类别检测方法的优势与不足; 涂良琼等人[17]从是否需要运行合约程序的角度将检测工具分为静态检测工具和动态检测工具, 并比较和分析了各检测工具的漏洞检测能力. 胡甜媛等人[18]将智能合约的安全问题总结为合约安全和隐私安全两个方面, 并从设计、实现、测试、

部署与运维等角度对智能合约安全问题的研究现状进行梳理, 提出未来的研究工作应围绕智能合约的全生命周期中每个阶段的安全问题进一步推进.

与已有的研究综述相比, 我们的工作存在如下不同.

一方面, 本文补充了重要的研究进展, 并提出智能合约安全漏洞分类框架. 智能合约安全漏洞检测工作的文献数量每年呈增长趋势. 在最新智能合约安全漏洞检测综述, 即文献[16] 于 2021 年 5 月发表后, 在 IEEE Xplore 上有 23 篇新发表文献, 其中包括权威国际期刊和会议 TSE、ISSTA 等. 本文在已有综述的基础上补充了如 Defect-Checker (TSE)[19]、sFuzz (ICSE)[20]、SmartTest (S&P)[21]和 Harvey (FSE)[22]等检测方法的比较和分析. 同时, 本文在已知的 15 种漏洞模式基础上增加了 26 种漏洞模式, 更加全面、详细地描述了目前已知的漏洞模式, 并归纳出智能合约安全漏洞分类框架, 该框架能够适配目前已知的智能合约安全漏洞, 且为未来的安全漏洞分类提供参考.

另一方面, 本文将研究进展的分析范围扩展到整个安全漏洞检测流程, 相较于已有的研究综述更加全面. 现有综述主要从检测技术角度出发, 描述了已知漏洞模式并对各检测技术进行分析和比较. 而漏洞模式分析是安全检测步骤的前提, 数据集和评价指标是验证方法有效性的数据支撑和评估手段. 本文进一步将研究进展的分析范围扩展到整个安全漏洞检测流程, 补充了漏洞信息收集、漏洞模式识别、数据集获取和评价指标选择等关键步骤的描述和分析, 提出了更为完整的智能合约安全漏洞检测研究框架, 希望能够为未来的研究工作提供帮助.

综上所述, 为了能够系统全面地了解智能合约安全检测领域的科研进展, 我们将 smart contract (智能合约) 组合 bug (错误)、defect (缺陷)、vulnerability (漏洞)、analysis (分析) 和 detect (检测) 等作为关键字, 在 IEEE Xplore、ACM Digital Library、Springer Link Online Library 和 CNKI 等在线文献数据库中搜索截至 2023 年 5 月与智能合约安全漏洞分析及检测相关的文献, 并特别检索了 TSE、ASE、ICSE 以及 CCS 等软件工程和网络与信息安全的权威国际期刊与会议. 同时, 为了避免遗漏, 我们利用雪球文献搜索法[23], 对上述论文所引用的参考文献进行人工检查, 进一步弥补遗漏. 此外, 我们详细分析和整理了已有调研工作中的文献, 补充到我们的文献统计中. 最终, 我们检索到与智能合约安全漏洞检测相关的论文 281 篇. 图 1 为根据发表年份对上述文献进行统计的柱状图, 从图中可以看出与安全漏洞检测相关的文献逐年递增. 其中目前检索到的 2023 年文献较少是由于部分 2023 年度的文献尚未被数据库收录. 图 1 所示的趋势表明针对智能合约安全问题的研究持续升温. 我们将论文的统计和收集结果公布在 GitHub (https://github.com/yagol2020/ SmartContract-Security-Papers) 中, 以供未来研究人员参考.

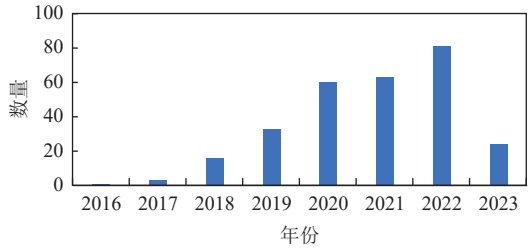

图 1 智能合约安全漏洞检测论文发表情况统计

通过分析论文作者和研究单位, 我们发现目前在智能合约安全漏洞检测领域上最为活跃的研究单位是莫纳什大学/华为公司夏鑫研究团队, 其研究成果发表在 ASE、TSE 等权威国际会议或期刊, 涉及漏洞发现、分类[24]与检测[19,25]等步骤, 形成了较为完整的智能合约安全漏洞检测体系. 此外, 高丽大学 Hakjoo Oh 研究团队[21,26]、北京航空航天大学姜博研究团队[12,27]也在智能合约安全漏洞检测领域取得了较多的成果. 其他研究单位中, 电子科技大学、北京大学、北京交通大学等高校的研究人员也屡有高质量研究成果发表.

本文从 281 篇文献中选取 TSE、ASE 及 ICSE 等软件工程领域国际权威期刊或会议 33 篇, CCS、USENIX Security 等网络与信息安全领域权威期刊或会议 19 篇, IJCAI、IJCNN 等人工智能领域权威会议 2 篇, TKDE 等数据挖掘领域权威期刊 1 篇, 以及 21 篇以上期刊或会议论文中提及的参考文献, 共计筛选出 76 篇高质量论文作为分析对象. 筛选出论文所属的领域、CCF 等级、发表源简称和论文发表数量如表 1 所示.

表 1 本文筛选的 76 篇分析对象的论文发表情况

领域	CCF等级	发表源简称	发表数量
软件工程	A类	ICSE	7
	A类	ASE	9
	A类	TSE	3
	A类	ISSTA	7
	A类	FSE/ESEC	1
	A类	OOPSLA	1
	B类	ICSME	1
	B类	SANER	1
	B类	ISSRE	1
	C类	ATVA	1
	C类	ICFEM	1
网络安全	A类	S&P	3
	A类	CCS	3
	A类	USENIX Security	2
	A类	TIFS	2
	A类	TDSC	2
	B类	ACSAC	2
	B类	NDSS	2
	B类	CSFW	1
	C类	FC	2
人工智能	A类	IJCAI	1
	C类	IJCNN	1
数据挖掘	A类	TKDE	1
其他	—	—	21

本文的主要贡献总结如下.

(1) 梳理并提出了目前智能合约安全漏洞检测方法的研究框架, 将其划分为 3 个阶段, 即漏洞发现与识别、漏洞分析与检测以及数据集与方法评估, 并依次详细介绍和归纳各阶段的通用技术. 本文提出的系统性研究框架可为研究人员提供帮助.

(2) 在已有漏洞分类标准的基础上, 进一步将检测方法中出现的漏洞模式进行总结, 提出智能合约安全漏洞分类框架, 将已知的漏洞模式根据其基础特征归纳为异常处理、权限控制、区块信息依赖等 13 类漏洞类型. 此外, 我们整理了 41 种漏洞模式, 相对于文献[16] 扩展了 26 种.

(3) 从检测方法分类的角度出发, 将检测方法划分为基于符号执行、模糊测试、机器学习、形式化验证和静态分析 5 种类型, 分析各种方法的优势和不足, 并展望了未来可研究的方向.

(4) 整理了智能合约安全漏洞检测领域常用的公开数据集以及评价指标, 为未来对智能合约安全漏洞检测方法的研究提供验证数据集支持以及评价指标参考.

本文第 1 节概要介绍目前安全漏洞检测的研究框架. 第 2 节描述智能合约安全漏洞信息收集的基本流程, 整理归纳了已知安全漏洞模式, 并对漏洞检测情况进行分析. 第 3 节详细介绍目前已有的检测方法, 划分为 5 种类型, 并逐一分析各类检测方法的优势与局限性. 第 4 节统计智能合约安全漏洞检测领域的数据集, 并介绍常用的评价指标. 最后在第 5 节总结全文, 分析现有的检测方法, 并展望未来的研究方向.

1 研究框架

本文将目前智能合约安全漏洞检测方法的研究框架总结如图 2 所示. 现有的检测方法根据其检测流程可分 3

个阶段: 漏洞发现与识别、漏洞分析与检测、数据集与方法评估.

漏洞发现与识别是进行智能合约安全漏洞检测的前提. 首先需要了解漏洞特征, 并针对漏洞特征设计可用于检测漏洞的模式, 才能有效发现智能合约中存在的漏洞. 目前已有 SWC Registry (https://swcregistry.io/) 等平台发布和更新漏洞信息. 此外, 部分研究者收集 StackOverflow 等 Q&A 平台中开发者关于智能合约的讨论信息, 经过整理、分类和归纳后给出常见的漏洞类型. 如 Chen 等人[24]收集了 StackExchange 平台 (https://ethereum.stackexchange.com/) 上 17 128 条讨论数据, 通过关键词筛选和人工检查后, 总结出了 16 种漏洞模式. 在本文的第 3 节将介绍漏洞信息收集的基本流程, 并对已知的漏洞模式整理分类.

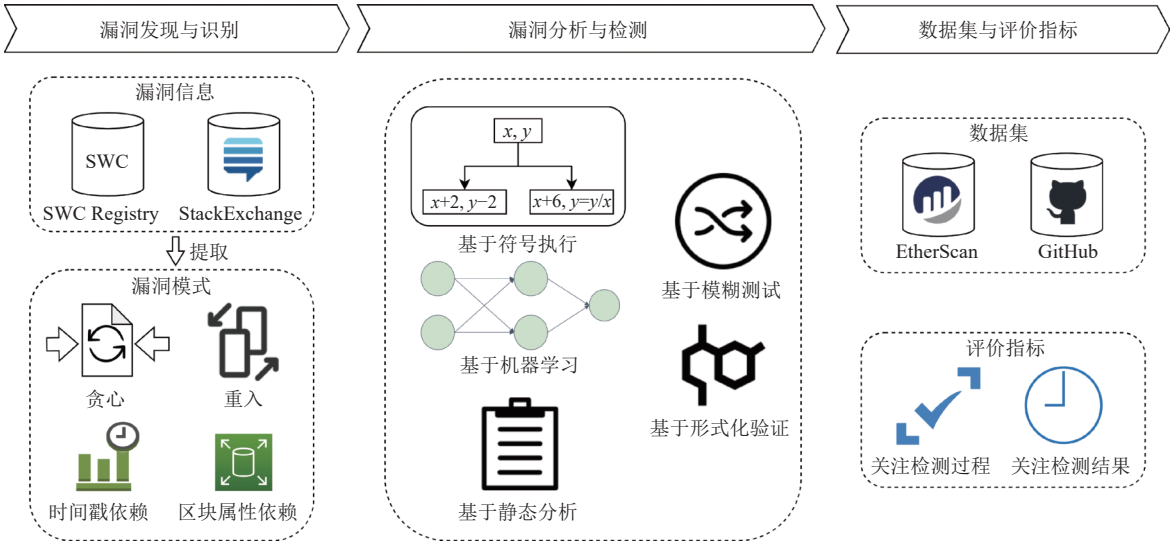

图 2 智能合约安全漏洞检测研究框架

漏洞分析与检测是检测方法的核心步骤. Ren 等人[28]对智能合约测试工具进行了实证研究, 将现有的测试工具分为 4 种类型, 即符号执行、动态模糊测试、深度学习和静态分析. 我们参考该文中的检测方法分类标准, 并补充了基于形式化验证的检测方法. 在本文的第 4 节将逐一介绍各类检测方法的已有工作及特点.

数据集与评价指标可用于验证方法的有效性. 数据集根据有无漏洞标签可分为两种类型. 无漏洞标签的数据可从区块链浏览器 (如 EtherScan (https://etherscan.io/) 等) 获取. 有漏洞标签的数据集可从两种途径获得, 一种是经过人工审查、分析和标注的数据集, 另一种是经过检测工具检测后标注的数据集, 以上两种途径标注的数据集通常可通过开源平台 (如 GitHub 等) 获取. 根据评价的阶段不同, 评估检测方法有效性的评价指标可分为关注检测过程和关注检测结果 2 类. 在本文的第 5 节将详细描述有关数据集和评价指标的有关内容.

2 智能合约安全漏洞发现与识别

漏洞发现与识别是安全漏洞检测的准备步骤. 在对智能合约进行检测前, 首先需要收集漏洞信息, 了解漏洞特征, 并发掘隐含的漏洞模式, 然后才能针对漏洞模式设计有效的检测方法.

本节将描述在安全漏洞检测工作中研究者收集漏洞信息的方法步骤, 以及目前已知的漏洞模式.

2.1 安全漏洞信息收集

漏洞信息与软件测试中测试报告相似, 可通过自然语言或代码示例描述触发漏洞的方法或操作, 以及导致漏洞产生的原理. 在智能合约安全检测领域, 虽然没有系统地对漏洞信息收集这一步骤进行研究, 但已有部分研究者通过文献阅读、问卷调查或专家评审等方式收集漏洞信息, 并对其加以分析, 为识别漏洞模式提供信息支持.

本文将安全漏洞信息收集的基本流程总结如图 3 所示. 首先, 通过开源仓库、博客、Q&A 或讨论平台等互联

网渠道收集原始漏洞数据, 从中提取疑似漏洞的信息; 然后, 通过专家分析或问卷调查等途径对可疑漏洞进行验证, 以获得更为准确的漏洞信息.

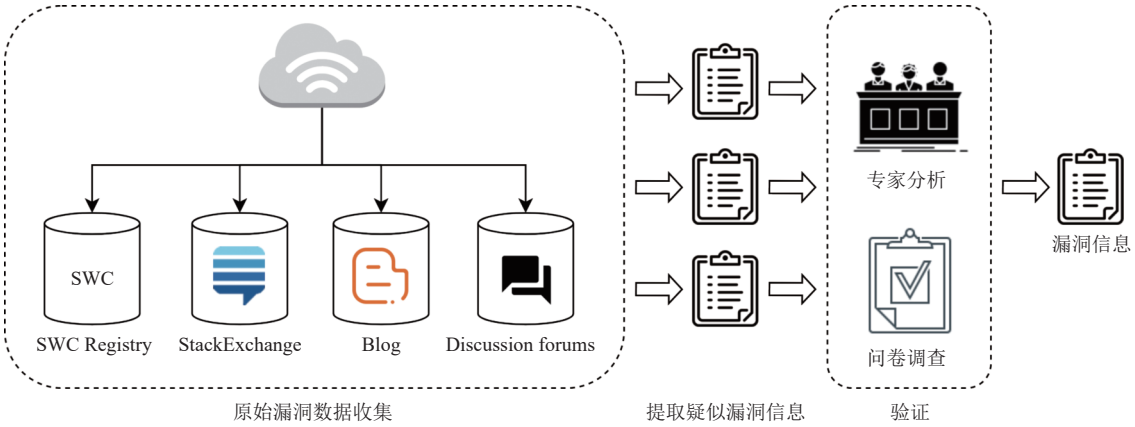

图 3 智能合约安全漏洞信息收集流程图

Atzei 等人[29]通过分析与以太坊相关的学术论文、互联网博客以及论坛, 并结合作者在智能合约上的编程经验, 将 12 种漏洞根据其被触发的环境划分为 3 个类别, 即 Solidity 层、EVM (Ethereum virtual machine, 以太坊虚拟机) 层和区块链层, 并通过漏洞原理、漏洞示例、漏洞相关语法知识对漏洞进行了详细描述. 此外, 对 TheDAO、KotET 及 Multi-player 游戏等 6 个事件或典型合约进行了分析, 阐述了其中隐藏的漏洞.

Chen 等人[24]在 StackExchange 平台中收集与智能合约相关的讨论信息, 如开发人员遇到的问题 (issues)、分享的编程经验及解决方案等. 在收集到的 17 128 条讨论信息中, 进一步通过关键字筛选和人工筛选的方法获得 393 条与智能合约缺陷相关的讨论信息. 在此基础上, 借助卡片分类法人工分析以上信息, 最终总结出 16 种漏洞, 并根据其带来影响的不同分为 5 种类型, 即影响安全、性能、可用性、可维护性和可复用性的漏洞. 此外, Chen 等人[24]还通过问卷调查的方式验证以上总结出的漏洞描述和漏洞分类是否合理, 结果表明大部分智能合约开发者及科研人员认同该分类结果.

Chen[30]观察到 Solidity 智能合约能够通过 self-destruct 指令自我销毁, 部分开发者利用这种方式销毁存在漏洞的智能合约, 经过漏洞修复后部署新合约. 基于以上观察, Chen 从 EtherScan 中收集了 54 739 份源代码, 其中包含 756 份被销毁合约, 将 54 739 份源代码根据创建者聚类分组, 组内按照时间顺序进行排序, 将销毁合约和仍运行的合约组合为一对合约, 利用 AST 解析源代码, 通过替换变量名称等方式删除与语义无关的信息. 然后通过 FastText[31]转换为向量并计算相似度, 当相似度高于 0.6 时则视为这一对合约存在更新关系. 最后根据更新关系分析销毁合约中存在的安全漏洞, 并利用卡片分类方法整理出了权限控制、未检测 CALL 返回值等 4 种漏洞.

此外, Wohrer 等人[32]通过审阅 Solidity 开发文档、互联网博客、论坛以及部分真实智能合约, 从中总结出 6 种漏洞及对应的安全设计模式, 按照安全设计模式编程将避免产生相关漏洞. Delmolino 等人[33]收集高校学生使用 Serpent 语言编写的智能合约, 分析并总结出 6 种 Serpent 智能合约漏洞. Durieux 等人[34]通过 not-so-smart-contracts 仓库、Positive.com 博客等途径收集漏洞信息并构建了智能合约漏洞数据集 (https://github.com/smartbugs/smartbugs), 利用该数据集对 9 种检测工具的性能进行比较与分析. Dias 等人[35]从 Sigma Prime、DASP 以及 SWC Registry 等数据源收集漏洞信息, 从漏洞信息中提取并定义出智能合约安全漏洞分类框架, 将已知漏洞分为智能合约特有漏洞和一般漏洞, 然后构建漏洞数据集 (https://zenodo.org/record/5512155) 对 3 种检测工具的性能进行比较与分析.

除以上研究中提到漏洞信息收集方法外, 还可通过智能合约漏洞信息公开网站获取漏洞信息, 如 SWC Registry 和 SlowMist Hacked 等. 这类网站提供了实时更新的漏洞信息, 如 SWC Registry 提供漏洞描述、漏洞示例以及测试用例等信息; SlowMist Hacked 提供被攻击对象、攻击事件描述及攻击方法等信息. 结合以上描述, 本文

将目前智能合约安全漏洞检测相关文献中出现的漏洞信息数据源总结为表 2. 其中, 文献指基于该数据源提出漏洞定义或分类的文献序号.

表 2 智能合约安全漏洞信息数据源

类型	更新频率	特点	名称	地址	文献
Q&A论坛	较快	涉及问答文本数量较多, 需要对问答文本进行处理, 例如关键字筛选等	reddit/ethereum	https://www.reddit.com/r/ethereum/	[29]
			Ethereum StackExcahnge	https://ethereum.stackexchange.com/	[24]
官方文档	较慢	以太坊官方提供的安全编码提示, 提供示例代码片段	smart-contract-safety	https://eth.wiki/en/howto/smart-contract-safety	[29]
开源仓库	中等	由开源组织或个人维护的数据源, 提供示例代码片段	smart-contract-best-practices	https://github.com/ConsenSys/smart-contract-best-practices	[32]
			not-so-smart-contract	https://github.com/crytic/not-so-smart-contracts	[34]
源代码	较快	仅提供源代码, 需要结合人工和其他手段确认漏洞信息	EtherScan	https://etherscan.io/	[30]
博客	中等	以新闻报道或评论的形式提供安全漏洞信息, 对安全事件的响应较快	Positive博客	https://blog.positive.com/	[34]
			Sigma Prime	https://blog.sigmaprime.io/solidity-security.html	[35]
其他网站	中等	以表格等形式展示漏洞代码片段、测试用例或攻击事件等信息	SWC Registry	https://swcregistry.io/	[34,35]
			DASP	https://dasp.co/	[35]
			SlowMist Hacked	https://hacked.slowmist.io/	无

2.2 安全漏洞模式分类

漏洞信息可以帮助研究者分析漏洞模式, 从而设计有效的检测方法. 本节将描述在智能合约安全漏洞检测领域研究工作中发现的典型漏洞. 本文参考文献[16] 和文献[29] 中的漏洞分类方法, 将层次扩展为高级语言、虚拟机和区块链 3 层. 其中, 高级语言层漏洞指因编程语言的语法特性而产生的漏洞, 虚拟机层漏洞指因虚拟机的特性或限制而产生的漏洞, 区块链层漏洞指由于区块链的特性而产生的漏洞.

进一步地, 本文提出了智能合约安全漏洞分类框架, 将已知漏洞根据其基础特征归纳为变量计算与存储、异常处理和未知函数调用等 13 种漏洞类型, 并在每类漏洞的描述中总结了异同点. 其中, 因缺少与重入漏洞相似的漏洞, 故将其单独分类. 最终, 我们整理出 41 种漏洞模式, 并将漏洞层次、类型、名称和可检测到该类漏洞的文献整理如表 3 所示. 需要注意的是, 部分漏洞暂时无法被现有的工具或方法检测, 例如数据存储位置冲突、缺少提示等漏洞暂无对应检测方法, 对于该部分漏洞, 我们给出了提出该漏洞定义的文献序号, 以供读者查看.

表 3 智能合约安全漏洞分类框架

层次	类型	名称	可检测该类漏洞的文献
高级语言	重入类	重入	[11,12,19,20,22,27,36–53]
	贪心类	严格余额检查	[12,19,20,27,40,42,47,48,51,54–56]
	变量计算与存储类	整型溢出	[20–22,26,37,39,41,42,46–50,57]
		除零/模零	[21,26,47,48,57,58]
		截断错误	[57]
		算术符号错误	[57]
		变量类型推断错误	[47,48,59]
		数据存储位置冲突	暂无自动检测方法, 定义见文献[24]
	异常处理类	未检测CALL返回值	[11,19,20,22,27,40–42,45–49,53,56,60]
		无gas发送	[12,20,27]
		拒绝服务	[19,22,47,48]
		无限循环	[43]

表 3 智能合约安全漏洞分类框架 (续)

层次	类型	名称	可检测该类漏洞的文献
高级语言	未知函数调用类	调用未知回调函数	[45]
		delegatecall函数调用未知函数	[12,20,27,39,42,56,61]
		调用未知library	[47]
	权限控制类	任意发送以太币	[21,40,42,51,54,56,61]
		自杀	[21,42,51,53–56,61]
		函数可见性	[39,48–50]
		任意修改状态变量	[40,61]
		构造函数名称	[39,48,49,53]
	高gas消耗类	不恰当函数或变量类型	[47,48,59]
		高gas循环	[19,47,48,59,62]
		高gas代码模式	[63]
	交互类	无效参数	[40]
		缺少提示	暂无自动检测方法, 定义见文献[24]
		缺少返回值	
		缺少中断	
	影响鲁棒性类	没有正确实现ERC-20接口	[21,47,48]
		使用废弃接口	暂无自动检测方法, 定义见文献[24]
		没有标明编译器版本	[47,48]
		无用参数	暂无自动检测方法, 定义见文献[24]
		地址硬编码	
虚拟机	以太坊地址类	短地址	
		地址错误	[55]
	调用链类	tx.origin	[19,46–48,53,58]
		调用堆栈溢出	[37,44]
区块链	区块信息依赖类	时间戳依赖	[11,12,19,20,27,37,42,43,45–49,58]
		区块属性依赖	[12,19,20,27,42,45,46,48,49]
	事务状态类	不可预知状态	[45]
		保守秘密	暂无自动检测方法, 定义见文献[29]
		事务顺序依赖	[11,27,37,40,42,45]

2.2.1 高级语言层安全漏洞

高级语言指能用于编写智能合约的高级编程语言, 如 Solidity (https://soliditylang.org/)、Vyper (https://github.com/vyperlang/vyper) 等. 这个层次的漏洞与语言特性有关, 例如回调机制导致重入漏洞, 以及由于 send、transfer 和 delegatecall 发生错误时处理方式不同而导致的异常处理漏洞等. 本节将介绍高级语言层的漏洞.

(1) 重入类漏洞

重入类漏洞指在合约未完成当前调用的状态下, 被恶意地再次执行调用函数, 从而导致恶性递归调用. 以太坊中, 当合约 A 向合约 B 转移以太币时, 由于 Solidity 语法机制将自动触发合约 B 的回调函数. 这个中间状态下, 合约 B 可以通过回调函数再次调用合约 A 中的函数, 由于合约 A 中的变量尚未被修改, 因此条件仍然满足, 致使再次转移以太币到 B 合约中. 重入漏洞是 TheDAO 事件中的关键漏洞, 不仅导致了价值 6000 万美元的以太币被窃取, 还致使以太坊产生了硬分支 (https://blog.ethereum.org/2016/07/20/hard-fork-completed/).

(2) 贪心类漏洞

贪心类漏洞指合约能够接收以太币, 但无法将自身的以太币转移出去. 通常含有贪心漏洞的合约缺少转移以太币的指令, 或者转移指令无法到达. 例如转移的条件过于严格 (即严格余额检查漏洞), 或者依赖于其他合约, 通过 delegatecall 转移以太币, 但其他合约存在缺陷被攻击导致无法使用. 贪心漏洞是 Parity 合约存在的漏洞之一,

在 Parity 合约依赖的多签名钱包合约被恶意销毁后, Parity 合约中的以太币无法转移, 导致价值 3 亿美元的以太币被冻结在合约中 (https://www.parity.io/blog/a-postmortem-on-the-par ity-multi-sig-library-self-destruct/).

严格余额检查是导致贪心漏洞的常见原因, 该漏洞指判断条件根据合约余额是否与规定数量相等决定是否执行代码, 而攻击者可绕过以太币发送数量的检查条件强制增加以太币 (如通过 suicide 指令), 导致余额检查条件无法满足, 无法执行发送以太币的代码, 产生含有贪心漏洞的合约.

(3) 变量操作类漏洞

变量操作类漏洞指对变量操作不当而产生的错误. 数值计算相关漏洞是变量操作类漏洞中的一种类型, 指对数值类型变量操作不当而导致计算结果与预期不符, 进而产生非预期行为. 据 CVE 统计 (http://csv.mitre.org/), 截止到 2019 年 3 月 31 日, 以太坊智能合约中 95.7% 的漏洞与数值计算漏洞相关[26]. 变量操作类漏洞包括 6 种, 即整型溢出、除零/模零、截断错误、算术符号、变量类型推断错误和数据存储位置冲突漏洞.

1) 整型溢出漏洞

整型溢出指数值计算的结果超出了变量可表示的范围, 导致计算结果与预期不符. Solidity 为整型开辟固定长度的存储空间, 若将超过最大存储范围的计算结果存储在较小的类型变量中, 将会导致整型溢出. 整型溢出是智能合约中常出现的错误, 例如 BeautyChain 合约的开发者错误地将乘法的计算结果存放到较小范围的变量中, 使得攻击者绕过了数值大小检验, 生成了大量代币, 致使该代币的价值大幅度下降, 对持有者的经济利益造成了损失 (https://tinyurl.com/yd78gpyt).

2) 除零/模零漏洞

除零或模零属于软件中的常见错误, 即在数值计算中位于分母的变量在一定条件下为零, 导致运算错误.

3) 截断漏洞

截断漏洞指将高范围的整型转换为低范围的整型时, 部分数值丢失导致的数值错误. 例如用 uint32 存储 uint 类型的 msg.value, 由于 Solidity 中 uint 默认为 uint256, 因此当 msg.value 大于 $2^{32} - 1$ 时, 会触发截断漏洞, 使得存储的值小于 msg.value, 导致合约错误地记录以太币数量.

4) 算术符号漏洞

算术符号漏洞指将带符号整型转换为不带符号整型时, 生成与预期不符的数值.

5) 变量类型推断错误漏洞

变量类型推断漏洞指在编码过程中不指定变量类型, 而是由编译器根据变量数值进行选择, 编译器将根据变量数值大小选择内存占用最小的变量, 因此在数值超过变量类型最大表示范围时将会出现逻辑错误. 例如“var $i = 0$”语句将导致 i 的变量类型为 uint8, 而 uint8 的最大值为 255, 如果将 i 作为循环控制变量遍历长度超过 255 的数组, 则将导致该循环遍历无法终止.

6) 数据存储位置冲突漏洞

数据存储位置冲突指在函数中创建 struct、mapping 或 arrays 类型变量, 导致变量指针指向错误位置, 使合约对变量的操作发生错误. 在传统语言 (如 C 或 Java) 中, 函数内创建的变量将动态分配内存 (memory) 空间, 而在 Solidity 中 struct、mapping 和 arrays 这 3 种类型变量存储在存储 (storage) 空间中. 由于 storage 不会动态分配, 因此在函数中创建这 3 种类型变量时, 若未指向已分配的 storage 变量, 则默认指向 0 号 storage 指针 (https://docs. soliditylang.org/en/v0.8.10/types.ht ml#data-location-and-assignment-behaviour), 将使该变量的指针引用错误, 导致错误的变量操作.

(4) 异常处理类漏洞

异常处理类漏洞指因程序没有正确分析函数的返回值或异常而导致的逻辑错误. Solidity 语言中, 不同的函数拥有不同的返回值或异常类型, 如 send、call 和 delegatecall 函数出现错误时返回 false, 而 transfer 出现错误时抛出异常. 当出现异常时, 以太坊会将事务回滚, 状态变量恢复到调用前的数值. 由于 send、call 和 delegatecall 不会抛出异常, 因此如果程序没有考虑函数的返回值, 在调用失败时不进行处理而继续执行代码, 则可能导致代码逻辑错误. 此外, 调用过程中若 gas 数量不足, 将抛出 out-of-gas 异常, 若未进行处理将导致逻辑错误, 甚至合约无法运

行. 异常处理类漏洞包括 4 种, 即未检测 CALL 返回值、无 gas 发送、拒绝服务和无限循环漏洞.

1) 未检测 CALL 返回值漏洞

未检测 CALL 返回值指程序对于 CALL 操作码的返回值缺少检查. CALL 是 send、transfer、call 和 delegatecall 函数的 EVM 操作码, 如果开发者没有检测 CALL 的返回值, 可能导致代码逻辑错误. 此外, 对于嵌套调用链, 如果存在至少一个调用是通过 call 或 delegatecall 进行的, 则异常传递和交易回滚将会在该函数调用处停止并返回 false, 然后继续执行剩余代码, 导致调用者无法获知整个嵌套调用链中的异常. 此漏洞是 KotET 游戏中存在的漏洞, 该游戏曾由于没有正确处理 send 函数的返回值, 导致部分 King 的利益受损 (https://www.kingoftheether.com/postmortem.html).

2) 无 gas 发送漏洞

无 gas 发送指没有携带适量的 gas 向外部合约转移以太币, 触发了 out-of-gas 异常且没有正确处理该异常. 以太坊账户分为两种, 即合约账户以及外部地址[1]. 合约账户运行智能合约, 由合约代码控制; 外部账户能够部署或调用智能合约, 由用户私钥控制. 向外部账户发送以太币仅需 2 300 gas, 而向合约账户发送以太币需要超过 2 300 gas, 因此使用 send 或 transfer 函数向合约账户发送以太币会导致 out-of-gas 异常.

3) 拒绝服务漏洞

拒绝服务指循环中抛出异常而导致整个循环被回滚. 循环体中若执行 transfer 或其他能够抛出异常的函数, 则抛出异常后将导致整个循环所修改的状态被回滚, 最终导致该循环体甚至合约无法正常执行.

4) 无限循环漏洞

无限循环漏洞指函数中的递归或循环没有终止条件, 或终止条件无法满足, 导致 gas 被不断消耗, 最终导致合约无法正常运行. 此外, 由于 Solidity 智能合约中的 fallback 机制, 因此可能导致合约间调用的无限循环. 例如函数 *a* 调用外部合约 *C* 的函数 *b* 时, 由于函数名称或参数错误导致自动调用了合约 *C* 的 fallback 函数, 而 fallback 函数中进一步调用了函数 *a*, 则由此产生了函数 *a* 和 fallback 函数之间的无限循环.

(5) 未知函数调用类漏洞

未知函数调用指在合约调用外部函数时, 可能会调用存在陷阱的函数或存在漏洞的函数, 致使调用合约的代码逻辑出现错误, 甚至会窃取调用合约持有的资产. 根据调用方式的不同, 可将该漏洞分为 3 种, 即调用未知回调函数、delegatecall 函数调用未知函数以及调用未知 library 引起的漏洞.

1) 调用未知回调函数漏洞

当通过 send 或 transfer 函数转移以太币, 或通过 call 函数调用外部函数且外部函数签名或类型无法匹配时, 将自动调用回调函数, 攻击者可以通过回调函数再次调用函数导致重入漏洞, 或编写 gas 消耗较多的回调函数导致 out-of-gas 异常, 引发非预期行为.

2) delegatecall 函数调用未知函数漏洞

调用 delegatecall 函数时, 代码运行的上下文环境处于调用者合约中, 此时 this 指针指向调用者自身, 攻击者如果在 delegatecall 调用的外部函数中编写 send 或 suicide 指令, 可能导致合约资产被恶意转移或合约被销毁.

3) 调用未知 library 漏洞

Solidity 允许使用 using 关键字将其他合约作为 library, 从而将其他合约的函数作为库函数使用, 如果没有关注 library 的安全性, 可能导致将其他合约中的漏洞引入到调用合约.

(6) 权限控制类漏洞

权限控制类漏洞指因程序设计错误而导致攻击者获得合约权限, 致使合约被恶意执行敏感操作. 敏感操作指涉及余额变动或合约控制的操作, 如转移以太币、调用外部函数、修改状态变量或合约销毁等. 开发者可通过修改状态变量来设置合约权限, 当调用者具备相应权限时才能执行敏感操作. 然而, 部分合约中由于缺少设置权限的条件, 或错误地将设置权限的函数可见性声明为 public, 甚至没有针对敏感操作进行权限控制, 导致合约存在安全问题. 权限控制类漏洞可分为 5 种, 即任意发送以太币、自杀、函数可见性、任意修改状态变量和构造函数名称漏洞.

1) 任意发送以太币漏洞

以太币的转移属于智能合约中的敏感操作, 如果任何人均可执行 send 或 transfer 函数, 并且能够控制转移地址参数, 则合约中的以太币可以被发送给任何人.

2) 自杀漏洞

智能合约的自杀也属于智能合约中的敏感操作, 与任意发送以太币漏洞相似, 如果任何人均可执行 suicide 指令, 则可能导致合约被恶意销毁.

3) 函数可见性漏洞

Solidity 默认的函数可见性为 public, 而合约中的关键函数应当声明为 private.

4) 任意修改状态变量漏洞

任意修改状态变量漏洞指修改状态变量的函数能够被攻击者恶意执行, 从而修改了状态变量导致合约运行出现错误. 智能合约中的状态变量通常记录较为重要的信息, 例如合约的拥有者、参与者或某个函数触发的条件等等. 合约的权限可通过设置状态变量实现, 然后通过 require 函数或 if 语句实现对执行敏感操作的调用者的身份验证. 然而若状态变量可被外界任意修改, 将会导致合约的权限被篡夺, 从而引发漏洞.

5) 构造函数名称漏洞

与传统软件相似, 在智能合约中构造函数指创建合约时执行的函数, 构造函数在合约创建完成后不再被执行, 通常用于初始化权限设置, 如设置合约拥有者等. 构造函数应与合约名称相同, 如果不同且未指定可见性则可见性默认为 public, 即任何人均可调用该函数获得合约的权限, 从而导致权限控制漏洞.

权限控制漏洞是典型的智能合约漏洞, 该漏洞曾引发多起安全事件, 如 Parity 事件中, 攻击者通过调用没有被权限保护的初始化函数, 窃取多签名钱包的合约权限, 并调用存在 suicide 指令的函数将多签名钱包销毁, 致使 Parity 合约中的以太币被冻结 (https://www.theregister.com/2017/11/10/parity_280 m_ethereum_wallet_lockdown_hack); Rubixi 合约存在错误的构造函数名称漏洞, 错误地将构造函数的名字命名为“DynamicPyramid”而不是“Rubixi”, 导致任何人均可获得合约权限, 非法获取其中的资产 (https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/).

(7) 高 gas 消耗类漏洞

高 gas 消耗类漏洞指合约代码中使用不恰当的变量类型或代码模式, 导致产生可避免的 gas 消耗, 对合约拥有者和调用者造成经济损失. 该类漏洞可分为 3 种, 即不恰当函数或变量类型、高 gas 循环以及高 gas 代码模式漏洞.

1) 不恰当函数或变量类型漏洞

不恰当函数或变量类型指因使用消耗较多 gas 的函数类型或变量类型而造成的不必要消耗. 为避免此类漏洞, 对于不会被内部调用的函数, 应将其声明为 external 而不是 public[64], 应使用 bytes 代替 byte 等.

2) 高 gas 循环漏洞

在循环体中使用 call 函数时, 由于 call 携带的 gas 数量较多, 如果不规定循环体的大小, 将导致 gas 被大量消耗, 甚至造成 out-of-gas 异常.

3) 高 gas 代码模式漏洞

不恰当的代码模式可能导致消耗不必要的 gas, 例如应将多个 JUMPDEST 可合并为同一个 JUMPDEST 从而减少 gas 消耗[63].

(8) 交互类漏洞

交互类漏洞指代码中缺少对输入、执行过程以及输出的交互或干预. 如缺少对参数的验证、缺少将执行状况反馈给调用者或缺少返回值等. 交互类漏洞可分为 4 种, 即无效参数、缺少提示、缺少返回值和缺少中断漏洞.

1) 无效参数漏洞

无效参数漏洞指缺少对参数有效性的验证, 如数值范围, 或者参数应该在某一个列表中 (例如游戏的参与者列

表) 等.

2) 缺少提示漏洞

缺少提示指合约在执行过程中缺少给予调用者的反馈提示. ABI (application binary interface, 应用二进制接口) 是以太坊框架上的合约交互接口, 能够实现智能合约之间, 或智能合约与区块链外程序, 如 Dapp (decentralized application, 去中心化应用) 之间的交互. 调用过程中 ABI 仅告知函数的输入值和输出值类型, 而不告知函数是否调用成功. 在此情况下, 应通过 event 将调用信息反馈给调用者, 以减少不必要的错误或者 gas 消耗.

3) 缺少返回值漏洞

缺少返回值指合约没有规定返回值, 导致调用者获得的返回值信息与预期不符. 若函数未规定返回值的具体数值, EVM 会为这种函数添加默认数值, 但该数值的含义可能与函数本身返回值的含义不符, 从而导致错误.

4) 缺少中断漏洞

缺少中断指合约没有能够中途停止或销毁的手段, 无法在发现漏洞时中断合约执行从而避免更多损失. 中断函数指能够结束合约生命周期, 停止合约运行的函数. 合约应该拥有至少一个中断函数, 如一个在限定条件下能够调用 suicide 指令的函数, 从而能够在合约受到攻击, 或者发现代码逻辑出现错误时及时停止运行, 减少经济损失[65].

(9) 影响鲁棒性类漏洞

影响鲁棒性指对智能合约的生命周期或健壮性产生影响的编码习惯. 该类漏洞可分为 5 种, 即没有正确实现 ERC-20 接口、使用废弃接口、没有标明编译器版本、无用参数和地址硬编码漏洞.

1) 没有正确实现 ERC-20 接口漏洞

ERC-20 接口是以太坊的技术标准 (https://github.com/ethereum/EIPs/blob/master/EIPS/ei p-20.md), 该接口定义了 9 种函数以及两种 event, 实现了 ERC-20 接口的合约即满足 ERC-20 标准, 便于和其他符合标准的合约进行交互. 没有正确实现 ERC-20 接口指合约没有按照 ERC-20 的标准实现接口. ERC-20 规定了函数的参数类型和返回值的含义, 如果没有正确实现接口的函数或事件, 如函数返回值的含义与标准不同, 则可能产生安全问题.

2) 使用废弃接口漏洞

使用废弃接口指在合约代码中使用旧版本语言支持的接口, 而该接口将在未来版本中被废弃, 使用可能被废弃的接口会降低合约的可复用性. Solidity 版本变更较快, 在其官方文档 (https://docs.soliditylang.org/) 中会提示开发者部分接口在未来版本中将被废弃. 尽管该 API 在当前版本仍可使用, 但使用废弃 API 的智能合约代码将会降低其可复用性, 从而间接增加开发成本.

3) 没有标明编译器版本漏洞

没有标明编译器版本指在合约代码中没有通过 pragma 指令指定编译器版本, 而不同编译器接口的实现可能存在不同, 由此可能会导致 API 调用与预期不符, 甚至编译失败.

4) 无用参数漏洞

合约中多余的参数或者变量将影响代码的可读性.

5) 地址硬编码漏洞

地址硬编码指在部署合约时将账户地址编码在合约代码中, 在该地址处于危险状态时无法修改合约代码中的地址, 对合约的正常运行造成影响.

2.2.2 虚拟机层安全漏洞

虚拟机指运行智能合约的运行环境, 如以太坊平台的智能合约经过编译后运行在 EVM 上, EOS 平台的智能合约经过编译后运行在 WebAssembly 虚拟机 (https://webassembly.org/) 上. 处于这个层次的漏洞通常与虚拟机实现有关, 例如 EVM 对转移以太币的目标地址不会进行检测, 如果目标地址尚未存在账户则创建一个独立账户, 导致丢失以太币漏洞, 或 EVM 堆栈大小限制调用次数小于 1 024, 调用链的深度超过堆栈大小会导致堆栈溢出等.

(1) 以太坊地址类漏洞

以太坊地址类漏洞指因以太坊地址数值错误而导致的地址无法识别或识别错误问题. 以太坊中通过地址识别账户, 如果地址输入错误将无法找到指定账户, 导致转移的金钱被退回甚至丢失. 常见的以太坊地址类漏洞可分

为 2 种, 即短地址漏洞和地址错误漏洞.

1) 短地址漏洞

短地址漏洞指输入的以太坊地址少于 20 字节, 虚拟机读取到非地址数据导致数据解析错误. 以太坊虚拟机在解析时读取 20 字节的数据作为地址, 因此攻击者可以输入少于 20 字节的数据, 使得虚拟机读取非地址字段的数据, 如记录以太币数量的数据, 致使以太币数量被错误解析.

2) 地址错误漏洞

地址错误漏洞指输入的以太坊地址错误或不存在, 虚拟机将自动创建账户, 由于该账户没有代码信息和合约拥有者, 将导致账户中的以太币无法取出, 进而引发贪心漏洞.

(2) 调用链类漏洞

调用链类漏洞指因合约错误使用调用链信息, 或因触发虚拟机对调用链的限制而产生的错误. 该类漏洞可分为两种, 即 tx.origin 漏洞和调用堆栈溢出漏洞.

1) tx.origin 漏洞

tx.origin 漏洞指将 tx.origin 属性作为执行敏感操作的判断条件, 而该属性可被攻击者伪造, 从而绕过检查条件恶意执行敏感操作. tx.origin 是 Solidity 智能合约中的全局变量, 能够获得事务链上首个发起调用的调用者地址, 但由于合约的函数可以被外部调用, 因此攻击者在外部调用合约时, 可以指定事务的发动者, 从而通过权限验证, 使攻击者窃取合约中的资产.

2) 调用堆栈溢出漏洞

调用堆栈溢出指调用链的堆栈深度超过虚拟机限制的最大深度, 导致堆栈溢出而引发异常, 进而产生异常处理漏洞. 该漏洞曾是 GovernMental 游戏存在的漏洞, 该游戏在合约设计阶段未考虑堆栈限制, 导致约 1 100 以太币被冻结 (https://www.reddit.com/r/ethereum/com ments/4ghzhv/governmentals_1100_eth_jackpot_payout_is_stu ck). 自 2016 年 8 月 16 日起, 调用堆栈溢出漏洞已被以太坊通过设置 EVM 调用堆栈的最大可达深度总小于 1 024 来加以解决.

2.2.3 区块链层安全漏洞

区块链维护一个分布式数据账本, 智能合约通过虚拟机与区块链进行交互, 如提交事务信息、获得合约信息或区块链状态等[9]. 智能合约涉及的金钱变动、事务状态修改或合约销毁等操作最终均呈现在区块链中, 处于这个层次的漏洞通常与区块链自身特性有关, 例如矿工修改区块链信息, 包括时间戳、块难度、块号或事务执行顺序等, 可能会导致产生时间戳依赖等漏洞.

(1) 区块信息依赖类漏洞

区块信息依赖指合约执行敏感操作的条件依赖于区块链信息, 而该信息能够被矿工影响或控制, 导致合约的执行与预期设计不符. 区块链能够提供与区块相关的信息, 如合约所在区块的块号、块难度、块哈希值以及时间戳等, 但以太坊允许矿工在一定范围内修改以上信息, 因此如果合约将区块信息作为判断合约执行逻辑的条件, 可能导致恶意攻击者 (即恶意矿工) 操纵合约执行, 使攻击者获利. 该类漏洞可分为两种, 即时间戳依赖漏洞和区块属性依赖漏洞.

1) 时间戳依赖漏洞

时间戳依赖漏洞指合约在执行关键操作 (例如转移货币、决定游戏胜者等) 时, 在判断条件、生成随机数、影响货币价格或交易时间等操作中将区块时间戳作为参数. 矿工在挖掘出新的区块后, 可以在一定范围内决定该区块的时间戳, 具体来讲, 矿工可在 900 s 内修改时间戳, 因此恶意矿工可以根据时间戳控制关键操作的触发条件, 从而对合约进行攻击.

2) 区块属性依赖漏洞

区块属性依赖漏洞与时间戳依赖漏洞相似. 区块属性指合约所在区块的块号、块难度、块哈希值等信息, 由于矿工能够在一定范围内影响区块属性, 因此如果合约将区块属性作为执行关键路径的条件, 或者用于生成随机数, 则可能会被恶意矿工控制.

(2) 事务状态类漏洞

事务状态类漏洞指由于区块链对事务状态的处理与预期不符, 或事务信息被攻击者获得, 进而对合约运行造成影响. 该类漏洞可分为 3 种, 即不可预知状态、保守秘密和事务顺序依赖漏洞.

1) 不可预知状态漏洞

不可预知状态漏洞指区块链短分支上的事务可能会被回滚, 因此合约事务的执行处于未知状态. 区块链中, 当多个矿工同时发现一个新的有效区块时, 区块链会分为多个分支, 一段时间后, 最长的分支会被保留, 而短的分支上的事务会被回滚, 导致合约处于未知状态.

2) 保守秘密漏洞

保守秘密漏洞指合约的状态变量没有通过加密手段进行处理, 可能被攻击者解析或推断从而影响合约运行. 由于区块链具有公开性, 设置变量的操作会通过事务发布到区块链上, 因此攻击者可以通过事务推断该变量的值. 为了保守变量的秘密, 开发者应该使用合适的加密手段对状态变量进行保密处理, 如使用定时承诺[66]等.

3) 事务顺序依赖漏洞

事务顺序依赖漏洞指合约因事务没有按照预想顺序执行, 导致合约的调用出现错误而引起逻辑错误. 区块链会在每个周期更新自身状态, 一个周期中存在一系列事务, 而这些事务的先后顺序由矿工决定, 因此调用者无法获知事务执行时的合约状态, 如果执行的预期结果依赖于事务顺序, 则可能导致代码逻辑错误.

2.3 安全漏洞检测情况分析

各种检测方法能够检测到的智能合约安全漏洞种类不同, 前文表 3 中列出了可检测相关漏洞的工具或方法对应文献. 结合表 3 以及第 2.2 节中对漏洞的分类与描述, 本节对漏洞检测情况进行分析、比较和评价.

从表 3 中可以看出, 大部分检测工具均支持检测重入漏洞. 重入漏洞是导致 TheDAO 事件的关键漏洞, 造成了 6 000 万美元的损失. 以太坊为了修复这次攻击造成的影响, 造成了以太坊硬分支, 甚至违背了区块链和以太坊宣言, 由此可见重入漏洞的巨大危害. 部分检测方法根据重入漏洞的特征设计了有效分析手段, 如 ReGuard[36]使用重入状态机检测重入漏洞等.

变量操作类漏洞中的数值计算相关漏洞是一类常见漏洞. 由于合约代码中的数值变量通常与加密货币数量直接相关, 因此智能合约相比传统软件对数值数据更加敏感. 相比于其他检测方法仅能检测整型溢出漏洞, 基于符号执行的检测方法 Osiris[57]能够检测更多种类的数值计算漏洞, 如截断漏洞和算术符号错误漏洞等.

贪心漏洞和权限控制类漏洞是造成 Parity 事件的主要漏洞. 攻击者通过权限控制类漏洞将 Parity 合约的子合约杀死, 导致 Parity 合约的金钱无法转移, 引发贪心漏洞, 造成 3 亿美元以太币被冻结. 这两种漏洞也能被大部分检测工具所检测.

异常处理类漏洞和区块信息依赖类漏洞也被大部分检测工具关注. 由于智能合约通常涉及货币转移, 因此 send、transfer 和 delegatecall 这 3 个函数的返回值处理尤为重要. 时间戳或区块属性的信息通常被当作执行敏感操作的条件, 而该类信息可能受外界影响甚至能够被恶意矿工控制, 从而影响合约的运行.

在高级语言层, 目前还缺少针对交互类和影响合约鲁棒性类漏洞的检测方法. 此外, 虚拟机层和区块链层与智能合约的执行环境相关, 短地址、不可预知状态及保守秘密等虚拟机和区块链层的漏洞可能会对智能合约造成较大安全隐患, 但还缺少有效的检测方法. 未来对智能合约安全漏洞检测方法的研究可考虑针对上述类型的漏洞进行分析.

3 智能合约安全漏洞分析与检测

漏洞分析与检测是检测方法的核心步骤. 在漏洞模式识别完成后, 需要通过合适的检测方法收集与漏洞相关的程序信息, 根据程序信息对程序是否存在漏洞进行判断. 部分实证研究[28]将检测方法分为 4 种类型, 即符号执行、动态模糊测试、深度学习和静态分析. 现有智能合约安全漏洞综述[9,14–16]将检测方法分为 5 种类型, 即在上述 4 种类型的基础上添加了形式化验证方法. 本文参考已有的检测方法分类标准, 鉴于 ContractWard[37]等方法采用随机

森林等机器学习方法, 基于深度学习的分类无法覆盖这类文献, 故本文将基于深度学习的分类标准调整为基于机器学习的智能合约安全漏洞检测方法.

为了更清晰地梳理各种类型检测方法的研究进展, 本节以研究问题为主线, 分别提出关于定义、技术分类以及技术分析 3 个研究问题, 具体的研究问题如下.

RQ1: 该类型检测方法在传统软件和智能合约中的定义是什么? 有什么异同?

RQ1 关注各个检测方法的定义. 智能合约运行在区块链这一新兴的运行环境中, 具有与传统软件不同的结构和运行机制, 因此需要将符号执行、模糊测试以及静态分析等应用在传统软件的漏洞检测方法经过改进后再应用到智能合约中才能取得较好的检测效果, 该问题关注应用在智能合约中的检测方法与传统软件中的检测方法的异同. 本文将该问题的回答总结在各个检测方法描述的开头中.

RQ2: 该类型检测方法相关的研究成果可分为哪几种类型?

RQ2 关注各个检测方法的技术分类. 本文根据关注的问题不同, 将 5 种类型的检测方法进一步整理为更细粒度的技术分类, 并按照时间顺序逐条梳理了各个技术分类的定义与研究进展. 本文将该问题的回答总结在各个检测方法的各个小节中.

RQ3: 在 RQ2 的基础上, 这些方法类型之间的优势和局限性是什么?

RQ3 关注各个技术分类的特点、优势与局限性. 在 RQ2 的基础上, 本文进一步对已有的研究进展进行归纳, 提出各个技术分类的特点, 并与其他技术分类进行比较, 从泛用、性能与准确性等角度总结了优势与局限性. 本文将该问题的回答总结在各个检测方法的小结部分中.

3.1 基于符号执行的智能合约安全漏洞检测

符号执行是一种程序分析技术[67], 使用符号化输入代替具体输入, 模拟执行被分析程序, 在分析过程中将程序中的操作转化为相应符号表达式, 利用 SAT/SMT 求解器对路径条件可满足性求解, 可用于分析路径可达性、生成测试数据及检测特定漏洞等.

在智能合约领域, 基于符号执行的检测方法通常以字节码作为输入, 对字节码进行分析或将字节码转换为中间语言再进行分析. 本文根据分析对象的不同, 将目前已有的检测方法分为两种类型, 即基于源代码/字节码的符号执行和基于中间语言的符号执行.

3.1.1 基于源代码/字节码的符号执行

基于字节码的符号执行通过将字节码反编译为操作码, 根据操作码构建 EVM 堆栈并符号化执行程序.

Luu 等人[11]归纳总结了 4 种以太坊智能合约的安全漏洞, 构建了基于符号执行的检测工具 Oyente, Oyente 以字节码作为输入, 逐行分析字节码构建控制流图 (control flow graph, CFG), 利用约束求解器去除不可达路径, 探索可达路径. 在分析阶段, 对 4 种漏洞构建漏洞模型, 并基于漏洞模型对智能合约进行检测, 从而识别存在漏洞的合约. Oyente 的实验结果表明, 在收集到的 19 366 个合约中, 8 833 个合约至少含有一种漏洞, 表明智能合约普遍存在安全问题.

Oyente 具有良好的扩展性, 其提供的符号执行框架为后续研究工作提供了基础. 如 Nikolić等人[54]、Wang 等人[27]以及 Chen 等人[62,63]分别基于 Oyente 构建了 Maian、Artemis、Gasper 以及 GasReducer. 借助 Oyente 收集的路径约束条件, Maian 总结了贪心性、挥霍性和自杀性 3 种以太币流动的特征, 建立漏洞模型并对其进行检测. Artemis 则在符号执行中收集了操作码序列、函数调用和块信息调用等信息, 扩展了贪心、区块属性依赖、无 gas 发送和 delegatecall 函数调用未知调用函数这 4 种漏洞的检测逻辑. Gasper 和 GasReducer 关注 gas 的消耗问题, 前者根据符号分析以及 CFG 判断是否存在任何条件下均不会被执行的死代码 (dead code), 后者能够检测字节码中是否存在高 gas 消耗模式, 并通过 EVM 指令替换的方式将其转换为 gas 消耗更低的字节码.

随着智能合约代码规模的增加, 可能带来路径爆炸以及约束求解耗时较高等问题, 降低了安全漏洞检测效率. 针对这些问题, 研究人员尝试优化符号执行的路径探索策略, 例如利用静态分析等方法减少搜索空间, 或通过并行的方式提高符号执行的效率. 例如 Chang 等人[55]构建了 sCompile, 将金钱相关的路径作为关键路径, 根据安全属

性计算优先级分数, 再利用符号执行判断关键路径的可达性, 实验结果表明 sCompile 平均检测单个合约的时间仅需 5 s, 且误报率较低. So 等人[21]和 Zhang 等人[68]针对事务序列的组合爆炸问题, 分别提出了 SmarTest 和 MPro, SmarTest 从已知存在漏洞的事务序列中学习统计模型, 并基于该模型制导符号执行, 优先执行更可能触发漏洞的事务, MPro 则根据 RAW (read after write, 写入后读取) 函数依赖信息剪枝事务序列的组合数量, 从而达到更高的检测效率. Mossberg 等人[69]构建了动态符号执行工具 Manticore, 以符号化表示事务序列的数据以及以太币数量, 有效地减少约束求解效率较低的问题, 实验结果表明 Manticore 能够达到更高的代码覆盖率. Zheng 等人[70]提出了并行符号执行框架 Park, 当符号执行遇到 JUMP 等指令时, 将创建新进程并复制当前 EVM 状态, 为在多个进程间共享全局变量, Park 将序列化的全局变量存储至共享内存, 当需要读取时再将其重构, 以维护整个符号执行过程中的全局变量符号值, 实验结果表明 Park-Oyente 相较于 Oyente 在检测速度上提高了 6.84 倍.

此外, 研究人员通过挖掘常见安全漏洞的特征, 提出了多种具有针对性的漏洞分析和检测方法. 例如 Torres 等人[57]和 So 等人[26]针对整型溢出、除零或模零等数值相关漏洞, 分别提出了 Osiris 和 VeriSmart 方法, 前者利用污点分析检测数值相关漏洞, 后者求解不变量以验证安全属性. Grossman 等人[71]和 Rodler 等人[38]针对重入漏洞提出了 ECFChecker 和 Sereum, 前者识别事务能否通过非回调的方式实现等效的状态转换以检测漏洞, 后者则用于保护已部署的智能合约, 利用动态污点分析技术监控事务执行时的数据流, 根据重入漏洞模式识别并阻止可疑事务执行, 以避免引发重入漏洞. Liu 等人[72]针对权限控制漏洞构建了 SPCon, 通过挖掘历史事务交易构建权限控制模型, 利用符号执行和污点分析收集信息流信息并对合约进行检查, 以识别与用户权限相关的潜在漏洞. 赵伟等人[39]详细描述了对于重入、整型溢出等 4 种漏洞的定义和约束条件, 基于字节码构建 CFG 并利用约束求解器生成测试用例. Zhou 等人[58]构建了 SASC, 根据源代码生成调用关系图, 通过符号执行和语法分析检测和定位漏洞. Krupp 等人[61]定义了两种漏洞模式, 一种是事务如果涉及关键路径的执行则可能存在漏洞, 另一种是如果合约涉及状态变量改变则可能存在漏洞, 并提出了 TeEther 对其进行安全漏洞检测, 实验结果表明 38 757 个合约中, 815 个合约存在上述两种漏洞. Chen 等人[19]构建了 DefectChecker, 根据 CFG 以及金钱相关调用、循环体和支付函数 3 种特征检测重入等 8 种安全漏洞, 同时利用该工具对智能合约进行了大规模的评估, 结果表明 15.89% 的智能合约含有至少 1 种漏洞, 其中占比最高的类型为未检测 CALL 返回值漏洞.

3.1.2 基于中间表示的符号执行

基于中间表示的符号执行方法在输入与检测步骤之间加入代码转换步骤, 将使用不同语法的代码转换为中间表示形式, 或将漏洞定义以中间表示形式描述. 通过该方法可将代码形式与检测方法分离, 达到更高的可扩展性.

基于中间表示能够实现跨平台的漏洞检测, 例如 Jin 等人[73]构建了 ExGen, 将以太坊和 EOS 平台的代码均通过中间形式进行表示, 其中, 将以太坊平台的 Solidity 智能合约根据 AST 映射到 LLVM 类型中, 将 EOS 平台的 C++智能合约直接转换为 LLVM 的中间表示, 通过对中间表示进行符号执行, 实现不同平台智能合约的漏洞检测.

此外, 漏洞定义也可通过中间表示进行描述. 例如 Tsankov 等人[40]构建了 Securify, 使用特定领域语言作为中间表示, 用以定义漏洞模式, 再通过符号执行提取代码中精确的语义信息, 与漏洞模型进行匹配检测安全漏洞. Feng 等人[41]构建了基于摘要的符号执行工具 SOLAR, 将智能合约转换为中间表示, 添加了符号变量和符号表达式以便于符号执行, 同时利用自定义的查询语言定义漏洞模式, 该语言由 uses、matches 和 where 这 3 个部分组成, 分别用以定义变量类型、语句序列和匹配语句的约束条件.

3.1.3 小 结

表 4 将上述基于符号执行的智能合约安全漏洞检测方法的类型、名称、检测对象以及主要特征进行了总结, 并根据发表年份进行排序. 目前, 现有基于符号执行的检测方法通常以字节码作为输入, 通过字节码 (或反编译后的操作码) 构建 EVM 堆栈, 并根据堆栈符号化执行程序, 在符号化执行过程中收集程序信息, 包括路径条件、变量修改或合约调用等. 部分检测方法如 DefectChecker 会根据程序信息进一步建模为程序特征 (如金钱调用、循环体或外部调用等), 然后根据预先定义的属性或漏洞模式检测是否存在漏洞.

表 4 基于符号执行的智能合约安全漏洞检测方法小结

类型	名称	发表年份	检测对象	主要特征
源代码或字节码	Oyente[11]	2016	字节码	基于字节码构建CFG, 在符号执行过程中收集程序信息检测漏洞
	Gasper[62]	2017	字节码	基于字节码构建CFG, 利用操作码和控制流图判断是否存在高gas消耗模式
	GasReducer[63]	2018	字节码	扩展高gas消耗模式, 通过字节码替换优化代码
	SASC[58]	2018	源代码	基于源代码构建调用关系图, 用两种方式获得操作码, 能够获得漏洞位置
	Maian[54]	2018	字节码	考虑合约连续调用
	TeEther[61]	2018	字节码	定义关键指令和关键漏洞, 解决哈希数值的符号化问题
	Osiris[57]	2018	字节码	关注数值计算相关漏洞
	Sereum[38]	2018	字节码	基于动态污点分析, 对已部署的智能合约进行检测
	ECFChecker[71]	2018	字节码	针对重入漏洞识别ECF属性
	sCompile[55]	2019	字节码	将涉及货币交易的路径视为关键路径, 根据预定义的重要属性对路径排序
	Manticore[69]	2019	源代码	动态符号执行
	MPro[68]	2019	源代码	借助静态分析工具获得RAW函数依赖信息, 通过剪枝缓解函数组合爆炸问题
	VeriSmart[26]	2020	源代码	关注事务执行过程中的不变量, 并基于不变量检测漏洞
	赵伟等人[39]	2020	字节码	基于字节码构建CFG, 利用约束求解器生成测试用例
	Artemis[27]	2020	字节码	扩展了Oyente, 基于字节码序列等特征检测漏洞
	SmartTest[21]	2021	源代码	识别存在漏洞的事务序列
	DefectChecker[19]	2021	字节码	将字节码反编译为操作码, 并构建EVM堆栈, 基于3种特征检测漏洞
	Park[70]	2022	字节码	并行符号执行框架
	SPCon[72]	2022	字节码	关注权限控制漏洞
中间表示	Securify[40]	2018	字节码	基于特定领域语言定义漏洞模式
	SOLAR[41]	2020	字节码	基于摘要进行符号执行, 利用自定义查询语言定义漏洞模式
	ExGen[73]	2022	源代码	基于LLVM进行中间表示, 能够检测跨平台漏洞

基于源代码或字节码的检测方法直接将字节码作为输入, 或将源代码通过编译器转换为字节码后进行分析, 是基于符号执行的检测方法中的常用做法, 相对于基于中间表示的检测方法, 其易于理解且不存在因中间表示不准确而改变代码语义的风险.

基于中间表示的检测方法在基于源代码或字节码的检测方法步骤前加入了转换为中间语言的过程, 或引入特定领域语言作为漏洞模式定义的中间表示, 相对于直接对字节码进行分析, 该类方法提升了可扩展性, 但需要注意转换后的中间表示应与原始代码或漏洞定义的语义保持一致.

3.2 基于模糊测试的智能合约安全漏洞检测

模糊测试是一种通过向目标系统提供非预期输入并监视异常结果来发现软件漏洞的方法. 它使用半有效的数据作为程序输入, 以程序是否出现异常作为标志, 可发现程序中存在的安全漏洞[74]. 在智能合约领域, 相关研究工作主要分为两种类型, 即覆盖制导的模糊测试方法和目标制导的模糊测试方法.

3.2.1 覆盖制导的模糊测试

覆盖制导的模糊测试方法以达到更高的代码覆盖率为目的[75]. 由于智能合约可通过函数访问的方式修改状态变量, 而部分程序状态 (或代码) 需要满足特定的状态变量条件, 即通过特定的函数调用序列才能到达 (如需先执行权限控制函数指定合约拥有者, 才能覆盖到仅允许合约拥有者才能执行的代码). 因此, 测试用例执行序列是覆

盖制导模糊测试方法的研究重点之一.

在智能合约领域, 覆盖制导的模糊测试方法通过模仿学习、动态数据依赖分析等策略生成事务调用序列, 按照该序列对智能合约进行模糊测试, 覆盖更深层的程序状态, 提高代码覆盖率.

现有工作中, Jiang 等人[12]和 Huang 等人[76]分别提出了以太坊和 EOS 平台智能合约的模糊测试框架, 即 ContractFuzzer 和 EOSFuzzer. ContractFuzzer 和 EOSFuzzer 均通过应用程序二进制接口 (abstract binary interface, ABI) 获得函数参数并生成测试用例, 分别通过插装 EVM 或 WebAssembly 虚拟机收集智能合约的执行情况, 再结合漏洞模式检测重入等安全漏洞.

在此基础上, Liu 等人[36]针对重入漏洞构建了 ReGuard, 在模糊测试过程中记录函数调用、返回值、对存储的访问、区块链的参数和分支操作等漏洞相关信息, 通过自动机识别重入漏洞. Wüstholz 等人[22]构建了灰盒模糊测试工具 Harvey, 使用需求驱动的序列模糊技术, 识别需要多个事务执行才能覆盖到的分支, 生成能够覆盖到更深层次程序状态的序列, 从而达到更高的覆盖率. Choi 等人[77]构建了 Smartian, 考虑了事务序列对代码覆盖的影响, 根据定义使用关系生成事务序列, 并将能够覆盖更多数据流的测试用例保留作为种子, 实验结果表明 Smartian 能够在较短时间内覆盖更多合约代码.

优化事务序列的生成或变异策略是提高模糊测试效率的重点之一. Xue 等人[78]在 sFuzz[20]的基础上, 针对跨合约场景构建了模糊测试工具 xFuzz, 将存在依赖关系的智能合约部署在测试环境中, 利用机器学习和适应度评价指标筛选需要执行的函数, 减少事务序列组合数量并提高模糊测试效率. Liu 等人[79]构建了 IR-Fuzz, 根据状态变量的读写情况计算次序优先级分数, 根据排序后的分数生成事务序列, 同时 IR-Fuzz 定义了稀有分支和疑似漏洞分支, 并分配更多的测试资源以提高漏洞检测效率.

此外, 符号执行能够提供高质量的测试用例, 且能够对路径约束条件进行求解. 现有研究工作尝试学习符号执行生成的测试用例, 或利用符号执行解决难以覆盖复杂路径分支的问题, 从而提高模糊测试的效率. 例如 He 等人[56]实现了基于模仿学习的模糊测试工具 ILF (imitation learning based fuzzer), 利用模仿学习框架学习符号执行生成的能够提高覆盖率的测试用例序列, 从而生成新的测试用例序列, 利用这种方法能够覆盖到更深层次的代码状态. Torres 等人[42]和 Chen 等人[80]将模糊测试与符号执行结合, 分别提出了 ConFuzzius 和 WASAI, 利用符号执行求解尚未覆盖的分支路径条件, 将求解的结果用于变异测试用例, 有效地提高了智能合约的代码覆盖率.

3.2.2 目标制导的模糊测试

目标制导的模糊测试方法以覆盖部分分支或程序块为目标, 弥补了模糊测试难以覆盖稀有路径的不足[81]. 在智能合约领域, 基于目标制导的模糊测试方法通常在覆盖制导的方法基础上, 通过语义分析或适应度函数等方式确定目标, 基于目标对种子的分配做出调整, 从而达到更高的测试效率.

例如 Wüstholz 等人[82]提出了目标制导的灰盒模糊测试工具, 在模糊测试过程中根据语义分析获得目标信息, 判断当前路径分支能否覆盖到目标节点, 根据种子能够覆盖到的路径对测试用例的生成策略做出调整. Nguyen 等人[20]设计了轻量级的自适应度策略, 构建了模糊测试工具 sFuzz. sFuzz 在模糊测试中仅监视与漏洞相关的信息, 相对于 ContractFuzzer 更加轻量. 在测试用例选择阶段, sFuzz 设计了目标函数, 用于度量测试用例与条件严格路径之间的距离, 保留与严格路径距离最短的测试用例, 采用 AFL (http://lcamtuf.coredump.cx/afl) 中的交叉和变异方法生成新的测试用例, 与 Oyente 和 ContractFuzzer 相比, 能够有效提高测试效率.

3.2.3 小 结

表 5 将上述基于模糊测试的智能合约安全漏洞检测方法的类型、名称、检测对象以及主要特征进行总结, 并根据发表年份进行排序. 目前, 现有的模糊测试方法通常需要构建模糊测试运行环境 (如以太坊运行环境、EOS 测试网络等), 并在模糊测试过程中收集智能合约执行情况 (如通过 event 反馈、虚拟机插桩等), 部分方法还将根据收集或分析的信息对测试用例生成进行调整 (如 sFuzz[20]、文献[82] 的目标制导方法等), 或结合其他方法 (如符号执行) 达到更高的代码覆盖率或更深层次的程序状态.

覆盖制导的模糊测试方法将研究的重点放在如何生成能够覆盖更深层次程序状态的测试用例序列上, 该类方法首先利用其他分析方法获得初始种子测试序列, 然后通过模仿学习等方式达到覆盖更深层程序状态的目的.

目标制导的模糊测试方法将研究的重点放在如何确定目标, 并生成能够覆盖目标的测试用例上, 该类方法通过收集分析语义信息和路径执行情况对测试用例生成策略进行调整, 能够通过迭代不断地提高代码覆盖率, 从而识别更多的漏洞.

表 5 基于模糊测试的智能合约安全漏洞检测方法小结

类型	名称	发表年份	检测对象	主要特征
覆盖制导	ContractFuzzer[12]	2018	字节码	搭建以太坊智能合约模糊测试框架, 收集EVM执行信息判断是否存在漏洞
	ReGuard[36]	2018	字节码	针对重入漏洞, 通过重入自动机检测漏洞
	IFL[56]	2019	字节码	学习符号执行生成的测试用例序列, 覆盖更深层次程序状态
	Harvey[22]	2020	字节码	分析和识别事务执行顺序, 基于此对测试用例执行顺序进行调整
	EOSFuzzer[76]	2020	字节码	针对EOS平台智能合约, 插装WebAssembly虚拟机收集执行信息, 以此判断是否存在漏洞
	ConFuzzius[42]	2021	字节码	与符号执行结合的混合模糊测试方法
	Smartian[77]	2021	字节码	使用数据流信息优化种子生成和保留策略
	xFuzz[78]	2022	源代码	针对跨合约场景, 利用机器学习技术优化事务序列生成策略
	WASAI[80]	2022	源代码	针对WebAssembly智能合约, 利用符号执行提高覆盖能力
	IR-Fuzz[79]	2023	源代码	计算优先级分数, 定义稀有和疑似漏洞分支, 优化测试资源分配
目标制导	Wüstholz等人[82]	2020	字节码	目标执导的模糊测试方法, 根据语义分析判断目标是否可达, 并以此调整测试用例生成策略
	sFuzz[20]	2020	字节码	设计并引入路径适应度函数, 能够覆盖条件严格的路径

3.3 基于机器学习的智能合约安全漏洞检测

近年来, 机器学习尤其是深度学习技术的发展, 吸引了较多的研究者利用机器学习技术解决软件中的安全问题[83], 例如 Alhuzali 等人[84]利用深度学习识别和生成动态 Web 应用中的漏洞, Melicher 等人[85]利用神经网络对密码安全性进行检测等. 在智能合约领域, 也有研究者利用机器学习方法进行了一系列研究, 如吴雨芯等人[86]利用注意力机制和双向长短记忆神经网络对智能合约进行分类等.

在智能合约安全领域, 研究者采用多种数据表征方式将智能合约转换为合适的数据形式, 并构建相应模型以检测智能合约中的漏洞. 本文参考顾绵雪等人[83]对基于深度学习漏洞挖掘技术的分类标准, 根据在数据表征阶段数据解析的表示结构不同, 将基于机器学习的检测方法划分为 3 类, 即基于图表征的检测方法、基于序列表征的检测方法和基于文本表征的检测方法.

3.3.1 基于图表征的检测方法

基于图表征的检测方法将源代码或字节码转换为控制流图、数据流图等表示形式, 有效地抽象出智能合约代码的控制流逻辑、数据依赖关系等信息. 基于图的表征形式能够反映代码中的语法和语义特征, 从而使机器学习模型达到更优的检测性能.

控制流图和数据流图包含丰富的上下文信息, 可用于获取智能合约的语法语义, 以检测安全漏洞. 例如 Zhuang 等人[43]提出了 DR-GCN 和 TMP, 结合数据流图和控制流图构建合约图, 在图规范化处理后, 构建图卷积神经网络 DR-GCN 和时序消息传播网络 TMP (temporal message propagation network) 用于检测安全漏洞, 实验结果表明 DR-GCN 和 TMP 优于 Oyente[11]等检测方法. 在 DR-GCN 和 TMP 的基础上, Liu 等人[87]进一步结合了图神经网络与专家知识, 定义了时间戳声明等专家模式特征, 与图神经网络获得的合约图特征结合, 以提高模型对漏洞检测的准确性. Zhang 等人[88]提出了 ReVulDL, 在利用图神经网络检测出重入漏洞后, 使用解释分析方法 pos-hoc 获取模型用于判断重入漏洞的重要边, 从而定位重入漏洞所在的位置.

3.3.2 基于序列表征的检测方法

基于序列表征的检测方法对源代码或字节码进行词法分析, 提取变量访问、函数调用序列或语句调用序列等

特征信息, 利用记忆力机制等方法学习提取出的序列信息, 构建长段记忆模型等对未知合约代码进行安全漏洞检测.

现有方法通常先利用嵌入算法将智能合约的代码、字节码或 AST 转换为向量, 再构建序列模型检测安全漏洞. 例如 Tann 等人[89]和 Qian 等人[44]利用代码嵌入算法分别将字节码和源代码转换为向量, 构建递归神经网络模型, 利用长段记忆以及注意力机制学习句法结构、语法语义等上下文关系, 能够检测重入等安全漏洞. Liu 等人[60]提出了 s-gram, 利用静态分析获得状态变量访问依赖和敏感流等语义信息, 将源代码和语义信息结合, 从中获得基于 AST 节点类型的 Token 信息, 并将其转换为 Token 序列, 基于 Token 序列构建 n-gram 语言模型, 以检测未知代码中是否含有漏洞.

3.3.3 基于文本表征的检测方法

基于文本表征的检测方法直接将源代码或字节码作为文本处理, 利用词嵌入算法 (如 Word2Vec、FastText[31]等) 将文本转换为向量, 基于向量构建机器学习模型学习隐藏在代码中的语法语义信息, 用以检测智能合约安全漏洞.

为了使机器学习模型能更好地学习代码结构或漏洞信息, 现有方法通常会在分析阶段进行规范化处理, 或利用切片等技术减少噪音干扰. 例如 Wang 等人[37]构建了 ContractWard, 通过 n-gram 从字节码中得到特征, 基于随机森林、支持向量机等 5 种机器学习算法构建模型, 使用 SMOTE[90]和 SMOTE-Tomek[91]采样方法处理类不平衡问题, 对重入、整型溢出等 6 种漏洞进行检测, 实验结果表明 ContractWard 的召回率达到了 96%. 相似地, Yu 等人[92]构建了 DeeSCVHunter, 通过 FastText 嵌入算法将代码转换为向量, 构建 Bi-GRU 等深度学习模型进行漏洞检测. 不同之处在于, DeeSCVHunter 定义了与漏洞相关的特征函数或变量 (如时间戳依赖漏洞与 block.timestamp 相关), 将代码与漏洞特征匹配, 获得候选语句, 再进一步利用控制流和数据流将与 CS 相关的语句添加到漏洞候选切片 (vulnerability candidate slice, VCS) 中. 在模型训练阶段, DeeSCVHunter 仅将 VCS 转换为向量表示, 减少了代码噪音的干扰.

3.3.4 小 结

表 6 将上述基于机器学习的智能合约安全漏洞检测方法的类型、名称、检测对象以及主要特征进行了总结, 并根据发表年份进行排序. 基于机器学习的智能合约安全漏洞检测方法通过将智能合约代码转化为能够被模型识别的特征, 构建机器学习或深度学习模型学习漏洞模式, 通过模型对未知智能合约进行安全漏洞检测.

表 6 基于机器学习的智能合约安全漏洞检测方法小结

类型	名称	发表年份	检测对象	主要特征
图表征	DR-GCN[43]	2020	源代码	利用图卷积神经网络学习图信息
	TMP[43]	2020	源代码	利用时序信息传播网络学习图信息
	ReVulDL[88]	2022	源代码	利用模型的解释分析方法定位重入漏洞具体位置
	Liu等人[87]	2023	源代码	结合专家知识提供的特征
序列表征	s-gram[60]	2018	源代码	基于静态分析获得语义信息, 构建语言模型预测漏洞
	Tann等人[89]	2019	字节码	利用长短记忆机制学习智能合约中的序列信息
	ReChecker[44]	2019	源代码	利用双向长短记忆和注意力机制, 针对重入漏洞进行检测
文本表征	ContractWard[37]	2021	字节码	通过n-gram从字节码中得到特征, 构建多种机器学习模型对合约进行检测
	DeeSCVHunter[92]	2021	源代码	基于控制流和数据流得到程序切片, 基于切片构建模型

基于图的特征表示方法将智能合约代码转换为控制流图、调用图等, 通过图卷积神经网络或时序信息传播网络等学习图中的信息. 该类方法能够有效地抽象出深层次的代码特征, 但图的构造过程较为复杂, 对于存在复杂逻辑或调用关系的合约, 存在分析阶段耗时较长的问题.

基于序列的特征表示方法对源代码中进行分析, 获得语法、语义或控制流等信息, 关注执行路径、调用序列等特征, 将其作为深度学习模型的输入构建序列模型. 该类方法学习到的序列表征与漏洞模式的关联性较强, 但由于其需要大量训练数据, 故检测效率较慢.

基于文本的特征表示方法利用自然语言处理中的词嵌入算法, 将智能合约转换为词向量, 或通过 n-gram 提取智能合约代码中的特征, 并基于向量特征构建模型进行检测. 该类方法直接将代码文本作为输入, 通常在分析阶段通过切片等方法过滤与漏洞不相关的程序片段, 从而提高模型的学习能力和检测能力. 因此, 如何在分析阶段使模型获得更多的语义信息是该类方法的关键点.

3.4 基于形式化验证的智能合约安全漏洞检测

形式化验证基于形式化规约, 利用数学方法证明程序的安全性. 此类方法通过对问题建立数学模型, 利用数学语言描述整个系统的全部状态, 从而验证程序在各个状态下是否保持安全性和可靠性[93].

王赫彬等人[94]对以太坊智能合约安全形式化验证方法进行了调研, 从对智能合约代码分析的完备性和可应用性角度对已有的形式化验证方法进行了分析比较. 朱健等人[95]从形式化方法、语言、工具和框架的角度出发, 分析了自动化验证、转换一致性等关键问题. 与以上 2 篇文献侧重的方向不同, 本文从安全漏洞检测的角度出发, 分析形式化验证方法在智能合约安全漏洞检测方面的优势和局限性. 本文参考文献[94] 中的分类标准, 将基于形式化验证的安全漏洞检测方法分为 3 种类型, 即定理证明、模型验证和其他形式化验证方法.

3.4.1 基于定理证明的检测方法

定理证明指将智能合约代码定义为可被定理证明器识别并编译的语言, 然后通过定理证明器 (如 Isabelle/HOL[96]、Coq (https://coq.inria.fr/) 等) 和预定义的安全属性对智能合约进行安全检测.

例如 Hirai[97]利用 Lem 语言定义 EVM, 基于 Isabelle/HOL 实现了形式化验证方法. 利用该定义, Hirai 发现了在 gas 计算问题上以太坊黄皮书[64]和 EVM 实现之间的差异, 并向以太坊官方提出了修改意见. Yang 等人[98]开发了基于 Coq 定理证明器的形式化验证工具 FEther. FEther 基于 Solidity 的子集 Lolisa 开发, 保证了智能合约和形式化语言之间的一致性. FEther 包含了一套自动化策略, 将符号执行和高阶逻辑定理证明结合, 能够自动执行和验证智能合约.

3.4.2 基于模型验证的检测方法

模型验证方法将智能合约程序解析为状态或行为并构建程序模型. 在验证阶段从初始状态或行为出发, 若状态转移不符合规约则可能存在安全漏洞. 基于模型验证的智能合约检测方法通常采用有界模型验证, 即验证一定状态转移步数内模型是否违背规约.

Boogie 是一种编程语言的中间表示, 可将源代码转换为更简单、更容易验证的形式. 基于模型验证的现有工作中, 常将 Solidity 源代码转换为 Boogie 后再进行模型验证等步骤. 例如 Wang 等人[99]构建了 VeriSol, 将 Solidity 智能合约转换为 Boogie 形式, 同时将工作流表示为有限状态机, 使用 Corral 对 Boogie 形式的智能合约进行有界模型验证, 用于验证工作流与智能合约代码之间的语义是否一致. 与 VeriSol 相似, Antonino 等人[100]构建了 Solidifier, 将 Solidity 的简化语言 Solid 描述的智能合约转换为 Boogie 形式, 同样使用 Corral 进行有界模型验证, 能够检测用户定义的程序错误.

3.4.3 其他形式化验证方法

除定理证明和模型验证外, 还有其他类型的形式化验证方法.

有研究者构建了针对智能合约的形式化验证框架, 并基于这些框架对智能合约进行分析和安全检测. 例如 Bhargavan 等人[101]构建了智能合约验证框架 F*, 该框架通过 Solidity*和 EVM*这两个模块将源代码和字节码转换为 F*框架可识别的语言, 以检测合约的安全性和功能正确性. Grishchenko 等人[45]在 F*的基础上针对智能合约的安全问题进行了分析, 验证了调用完整性、参数不受矿工控制等安全属性. Hildenbrandt 等人[102]在 K 框架的基础上构建了 KEVM 框架, 该框架可用于分析和验证智能合约程序. KEVM 考虑了 EVM 的异常机制, 构建了支持异常的控制流图, 可用于对 gas 消耗和白皮书中的 EVM 语言规范进行分析.

此外, 还可针对特定属性进行形式化分析, 通过该属性对智能合约进行安全检测. 例如 Permenev 等人[103]提出了 VerX 方法, 将智能合约中的时序属性转换为可达性验证, 并构建了针对 EVM 的符号执行引擎, 结合延迟谓词抽象方法, 在事务执行过程中使用符号执行, 在事务之间使用抽象分析, 以对智能合约的功能属性进行验证. Kalra

等人[46]构建了检测工具 ZEUS, 将智能合约代码翻译为 LLVM 中间语言形式, 利用 LLVM 框架进行抽象解释和符号模型检查. 由于以 LLVM 中间表示作为输入, 因此 ZEUS 能够支持多种语言编写的智能合约.

3.4.4 小 结

表 7 将上述基于形式化验证的智能合约安全漏洞检测方法的类型、名称、检测对象以及主要特征进行总结, 并根据发表年份进行排序. 形式化验证方法通常将智能合约转换为定理证明器、模型验证工具或自定义框架能够识别的输入, 并根据断言或预先定义的安全属性对程序进行验证.

表 7 基于形式化验证的智能合约安全漏洞检测方法小结

类型	名称	发表年份	检测对象	主要特征
定理证明	Hirai[97]	2017	字节码	基于Isabelle/HOL
	FEther[98]	2019	源代码	基于Coq
模型验证	VeriSol[99]	2019	源代码	验证工作流与合约代码之间的语义一致性
	Solidifier[100]	2020	源代码	针对Solid语言构建合约验证和函数验证两种验证器进行验证
其他方法	F*框架[101]	2016	字节码	将智能合约转换为可用于程序验证的函数式编程语言F*
	Grishchenko等人[45]	2018	字节码	基于F*实现了安全属性自动化验证
	KEVM[102]	2018	字节码	基于K框架实现, 具有可读性强等优势, 能够验证合约的功能性
	ZEUS[46]	2018	源代码	将源代码翻译为LLVM, 针对LLVM进行漏洞检测, 具有良好的可扩展性
	VerX[103]	2020	源代码	验证功能属性

定理证明和模型验证属于形式化验证领域的常用方法, 由于其不依赖于智能合约语言, 仅需提供验证工具能够识别的输入即可完成验证, 具有适用范围广的优点. 然而, 形式化验证存在自动化程度低、人工成本高的不足, 如何构建能够自动分析和验证智能合约安全性的工具是形式化验证未来可探讨的方向之一.

部分研究者根据智能合约或运行环境的特性构建了具有针对性的验证框架, 以更准确地对智能合约程序进行建模, 但这类方法的可扩展性通常较弱, 如 F*、KEVM 和 VerX 等框架仅支持对 Solidity 合约的验证, 未来可探索并尝试构建支持其他智能合约语言的形式化验证框架.

3.5 基于静态分析的智能合约安全漏洞检测

静态分析指在不运行软件的前提下对软件进行分析[104], 基于静态分析的漏洞检测方法通过分析程序的代码、结构或文档等信息挖掘漏洞[105]. 在智能合约领域, 基于静态分析的检测方法通常以源代码作为输入, 通过将源代码转换为中间形式 (例如 XML 等), 然后基于预先定义的漏洞模式行匹配, 或者利用数据流、控制流或逻辑关系分析等方法对智能合约的安全漏洞进行检测. 根据静态分析检测漏洞的判断依据不同, 本文将基于静态分析的智能合约安全漏洞检测方法分为两种类别, 即基于匹配的检测方法和基于语义的检测方法.

3.5.1 基于匹配的检测方法

基于匹配的方法通常首先将智能合约转换为中间表示, 如图、XML 或词向量等形式, 然后将其与预先定义的漏洞模型进行相似性匹配.

模式匹配或关键字匹配是一种简单直接的检测方法. Tikhomirov 等人[47]构建了 SmartCheck, 将 Solidity 源代码转换为基于 XML 的中间表示, 使用 XPath 匹配预先定义的漏洞模式, 从而检测是否存在漏洞. Zhang 等人[48]构建了基于正则表达式匹配的检测工具 SolidityCheck, 对源代码的格式进行预处理, 删除空行、空格以及注释等, 然后针对不同的漏洞类型预定义关键字, 对关键字进行正则表达式匹配.

上述方法在漏洞检测过程中, 由于严格按照漏洞模式或关键字进行匹配, 可能导致较多的误报和漏报. 在此基础上, 利用代码嵌入技术将源代码或字节码转换为向量, 再通过距离度量的方法匹配漏洞片段, 能够提高漏洞检测的性能. Gao 等人[25,106,107]构建了 SmartEmbed, 将源代码根据粒度划分为合约级、函数级和语句级 3 种带有代码结构信息的段落, 利用代码嵌入技术将段落转换为向量, 比较向量之间的距离得出代码之间的相似性, 当合约与漏洞

代码库中的代码片段相似度较高时, 则该合约可能含有特定类型的漏洞. 韩松明等人[49]和 Huang 等人[50]采用程序切片技术减少字节码中噪音对代码嵌入的干扰, 利用图嵌入算法将切片转换为向量, 并利用余弦距离计算向量之间的相似度, 用于字节码匹配和漏洞检查.

3.5.2 基于语义分析的检测方法

基于语义分析的方法收集并分析程序中的语义信息, 如逻辑关系、状态变量操作和敏感流等, 并基于以上分析结果对合约的安全性进行检测.

污点分析是一种常用的数据流分析方法, 其将外部输入的数据作为污点源, 追踪污点源的数据传播轨迹, 以判断程序的安全性. Feist 等人[51]构建了 Slither, 利用污点分析和数据流图获取智能合约代码中的信息, 如变量的读写情况等, Slither 框架可用于漏洞检测、优化消耗资源较多的代码模式以及辅助用户理解代码逻辑. Gao 等人[108]和 Xue 等人[52,109]分别构建了 EasyFlow 和 Clairvoyance, 利用污点分析技术识别疑似漏洞路径, 并通过预定义的路径保护模式进一步过滤, 从而提高漏洞检测的准确性. Ghaleb 等人[110]构建了 eTainter, 将 CALLDATALOAD 等引入外部数据的指令作为污点源, 利用污点分析追踪外部数据的传播, 相较于 MadMax, eTainter 无需专家规则即可检测拒绝服务等安全漏洞.

智能合约的逻辑关系的可用于检测安全漏洞. Brent 等人[53]和 Grech 等人[59]分别构建了 Vandal 和 MadMax. Vandal 利用特定领域语言 Souffle[111]作为逻辑规范, 当需要分析新漏洞时, 可组合已有的规范从而达到较好的扩展性. MadMax 则通过反编译器将 EVM 字节码转换为结构化中间语言, 结合基于逻辑分析的规范生成高级程序模型, 收集和分析数据流、循环和数据结构等信息用于检测相关漏洞.

此外, 控制流图同样包含丰富的智能合约语义信息. Albert 等人[112]构建了 EthIR, 优化了控制流图的生成, 相较于 Oyente 仅保存跳转地址的最后一个值, EthIR 保存所有可能的跳转地址, 从而生成整个合约代码的 CFG. EthIR 分析堆栈变量、本地变量、合约字段和区块链数据, 进一步将 CFG 转换为基于规则的表示形式 (rule-based representation, RBR), RBR 可作为 SACO[113]分析器的输入, 用以推断和分析 gas 消耗等属性. Liao 等人[114]提出了静态分析方法 SmartDagger, 反编译智能合约的字节码, 通过深度学习模型还原丢失的状态变量属性等语义信息, 能够检测重入、时间戳依赖等安全漏洞.

3.5.3 小 结

表 8 将上述基于静态分析的智能合约安全漏洞检测方法的类型、名称、检测对象以及主要特征进行了总结, 并根据发表年份进行排序. 静态分析方法通常以源代码作为输入, 通过对智能合约代码中的语义、语法、代码结构或 Token 序列等信息进行分析, 结合预先定义的漏洞模式对代码进行检测.

表 8 基于静态分析的智能合约安全漏洞检测方法小结

类型	名称	发表年份	检测对象	主要特征
基于匹配	SmartCheck[47]	2019	源代码	基于XPATH匹配
	SolidityCheck[48]	2019	源代码	基于正则表达式匹配, 通过程序插装预防漏洞
	SmartEmbed[25]	2020	源代码	利用词嵌入方法将代码转换为词向量
	DC-Hunter[49]	2020	字节码	通过数值规范化等3种规范方法使源程序切片和字节码切片保持相似
	Huang等人[50]	2021	字节码	通过规范化减少噪音, 利用图嵌入算法将CFG转换为向量
基于语义分析	Vandal[53]	2018	字节码	基于逻辑关系检测漏洞
	MadMax[59]	2018	字节码	关注gas消耗漏洞, 将EVM字节码转换为中间语言
	EthIR[112]	2018	字节码	将CFG转换为中间表示, 基于分析器推断字节码的安全属性
	Slither[51]	2019	字节码	将代码转换为中间语言SlithIR, 结合数据流和污点分析检测漏洞
	EasyFlow[108]	2019	字节码	基于污点分析, 过滤被保护的变量操作降低误报率
	Clairvoyance[109]	2020	源代码	考虑多个智能合约之间的调用, 通过污点分析识别污点路径
	eTainter[110]	2022	字节码	利用污点分析追踪外部数据传播
	SmartDagger[114]	2022	字节码	利用深度学习模型还原字节码中丢失的语义信息

基于匹配的方法将智能合约程序和漏洞模式转换为中间表示, 如 XML、词向量等, 将需要被检测的合约与漏洞模式进行相似性比较. 该类方法可扩展性强, 当发现新的漏洞模式时, 仅需提供漏洞片段即可支持检测新漏洞. 然而, 该类方法缺少对程序语法语义的描述, 具有误报率较高的不足.

基于语义分析的方法收集并分析程序中的数据流、控制流和敏感流等信息, 结合污点分析等方法对合约进行漏洞检测, 相比于基于匹配的检测方法, 具有更高的精确度, 但该类方法需要预先分析并定义漏洞模式, 对检测新漏洞的可扩展性较差.

4 数据集与评价指标

为对智能合约安全漏洞检测方法的有效性进行评估, 一方面需要真实的数据集作为数据支持, 另一方面需要设计合理、有效, 且能准确评估方法特点的评价指标. 从以上两方面出发, 本节总结了在智能合约安全漏洞检测领域中常被使用的数据集和评价指标.

4.1 数据集

数据集可以为后续研究者在构建工具, 或者对方法有效性进行评估时提供数据支撑. 根据有无漏洞标签可将现有智能合约数据集分为两类, 即有漏洞标签的数据集和无漏洞标签的数据集. 前者标记了智能合约是否存在漏洞, 部分数据集还会进一步标记漏洞的具体类型和位置, 此类数据集可用于对检测结果的正确性进行评估; 后者没有标记漏洞是否存在, 可用于对智能合约安全漏洞检测方法的可扩展性或性能等方面进行评价, 如评估检测方法能否应用到包含各类业务需求及代码逻辑的智能合约中, 或评估检测方法的效率等.

有漏洞标签的数据集中, 根据标注的方法不同可分为 2 类. 一种是研究者人工分析智能合约是否存在漏洞, 并根据分析结果对智能合约进行标记. 如 Chen 等人[24]组织专家对 587 个合约进行了人工漏洞检测; Durieux 等人[34]根据 DASP 网站 (https://dasp.co) 的分类标准手动标记了 69 个合约中的漏洞, 该数据集中还包含漏洞的行数等信息. 另一种是研究者利用工具对智能合约进行大规模检测或注入漏洞构建数据集. 如杨慧文等人[115]利用智能合约在线检测平台对 4203 份源代码进行漏洞检测, 构建了包含 37 种代码度量元和 21 种漏洞信息的数据集; Ghaleb 等人[116]构建了 SolidiFI 工具, 将 7 种漏洞注入到无漏洞的合约中, 以此生成存在漏洞的智能合约. 虽然基于工具标注的数据集标签存在标记错误的可能, 但仍可用于评估其他检测方法有效性.

本文整理部分文献中公开的有漏洞标签的数据集, 统计其标记方式、文献、名称、数据集地址以及数据集中标记的漏洞种类数量, 结果如表 9 所示. 其中, VeriSmart-benchmarks 和 prdc-contracts-evaluation 包含多种来源的标签, 如 CVE、ZEUS、SmartBugs 和 Slither 等, 故未对漏洞种类进行统计.

表 9 存在漏洞标签的数据集

标记方式	文献	名称	数据集地址	漏洞种类
人工	[24]	TSE-ContractDefects	https://github.com/Jiachi-Chen/TSE-ContractDefects	20
	[34]	SB Curated	https://github.com/smartbugs/smartbugs/tree/master/dataset	9
	[28]	Smart-Contract-Benchmark-Suites	https://github.com/renardbebe/Smart-Contract-Benchmark-Suites	7
工具	[115]	COOP-SC-Sol-Dataset	https://github.com/yagol2020/COOP-SC-Sol-Dataset	21
	[60]	sgram-artifact	https://github.com/njaliu/sgram-artifact	8
	[116]	SolidiFI	https://github.com/DependableSystemsLab/SolidiFI	7
	[11]	Oyente-benchmarks	https://github.com/oyente/benchmarks	4
	[44]	ReChecker	https://github.com/Messi-Q/ReChecker/tree/master/evaluations/reentrancy	1
	[21,26]	VeriSmart-benchmarks	https://github.com/kupl/VeriSmart-benchmarks	—
	[35]	prdc-contracts-evaluation	https://zenodo.org/record/5512155	—

无漏洞标签的数据集可从 EtherScan、BscScan (https://bscscan.com/) 或 XBlock (http://xblock.pro/) 等区块链浏览器或开源社区获得. 此外, 部分智能合约安全漏洞检测的实证研究工作也将自己收集整理的数据集开源, 供未来

研究者使用, 如 Durieux 等人[34]收集 Google BigQuery 和 EtherScan 的智能合约代码, 将其开源在 GitHub 平台 (https://github.com/smartbugs/smartbugs-wild); Ren 等人[28]同样利用 Google BigQuery 和 EtherScan 收集智能合约代码, 并将去重后的 44 622 份源代码开源在 GitHub 平台 (https://github.com/renardbebe/Sm art-Contract-Benchmark-Suites).

4.2 评价指标

评价指标是衡量检测方法是否有效的重要依据. 在智能合约安全漏洞检测领域, 根据评估的方法阶段不同可分为两种类型, 即关注检测过程的评价指标和关注检测结果的评价指标.

关注检测过程的评价指标度量检测方法在分析和检测阶段的有效性, 例如检测耗时[22,68]、覆盖率[20,56]等. 检测耗时指在检测过程中, 从被测对象输入到检测结果输出的时间, 检测耗时越短, 表明检测方法的效率越高. 覆盖率指在检测和分析过程中, 源代码或字节码被分析或被执行的覆盖情况. 覆盖率越高, 表明检测方法越全面, 越有可能检测出漏洞.

关注检测结果的评价指标度量检测结果是否与真实情况相符, 例如准确率、召回率、*F*1 值[19,92]以及 AUC[37]等. 准确率 (*Accuracy*) 能够直观地评价工具的检测性能, 准确率越高, 表明漏洞的检测结果与真实情况越接近, 准确率的计算公式如公式 (1) 所示:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{1}$$

其中, *TP*、*FP*、*TN* 和 *FN* 如表 10 的混淆矩阵所示.

表 10 智能合约安全漏洞检测结果混淆矩阵

检测结果	真实情况	
	存在漏洞	不存在漏洞
存在漏洞	*TP*	*FP*
不存在漏洞	*FN*	*TN*

TP: 检测结果为存在安全漏洞, 且真实情况也存在安全漏洞的智能合约数量.

FP: 检测结果为存在安全漏洞, 但真实情况不存在安全漏洞的智能合约数量.

FN: 检测结果为不存在安全漏洞, 但真实情况存在安全漏洞的智能合约数量.

TN: 检测结果为不存在安全漏洞, 且真实情况也不存在安全漏洞的智能合约数量.

*F*1 值是综合考虑精准率 (*Precision*) 和召回率 (*Recall*) 的评价指标, 常用于二分类问题 (即智能合约是否存在安全漏洞) 时评价检测方法的有效性. 精确率和召回率的计算公式如公式 (2) 和公式 (3) 所示:

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

*F*1 值的计算公式如公式 (4) 所示, *F*1 值越高, 表明方法在检测智能合约安全漏洞查准和查全的综合能力方面越优秀.

$$F1 = \frac{2\times Precision\times Recall}{Precision+Recall} \tag{4}$$

AUC (area under curve, 曲线下面积) 指 ROC (receiver operating characteristic, 接收者工作特征) 曲线下的面积, ROC 的横坐标为假阳性率 (false positive rate, *FPR*), 纵坐标为真阳性率 (true positive rate, *TPR*), *FPR* 和 *TPR* 的计算公式如公式 (5) 和公式 (6) 所示. AUC 的值越高, 表明安全漏洞的检测结果越与真实情况相符.

$$FPR = \frac{FP}{TN+FP} \tag{5}$$

$$TPR = \frac{TP}{TP+FN} \tag{6}$$

5 总结与展望

随着数字加密货币的快速普及与发展, 区块链技术逐渐成为工业界和学术界的研究热点. 智能合约作为区块链合约层的重要组成部分, 能够持有或转移资产, 实现如投票、众筹或游戏等诸多应用, 其潜在的巨大利益吸引了大量攻击者试图通过漏洞攻击智能合约, 窃取合约中的资产. 如何有效保障智能合约安全是目前软件工程和信息安全领域亟待解决的问题. 本文收集和分析了与智能合约安全漏洞检测相关的文献, 按照高级语言、虚拟机和区块链 3 个层次对已知的 41 种智能合约安全漏洞进行分类, 并将具有相似特征的漏洞进一步归纳为 13 种漏洞类型. 本文还将已有的智能合约安全漏洞检测方法分为符号执行、模糊测试、机器学习、形式化验证和静态分析 5 种类型, 分别介绍了各类型检测方法的原理、基本流程、特点、优势及局限性.

结合本文对智能合约安全检测领域研究进展的分析不难看出, 目前智能合约安全漏洞检测工作主要基于传统软件的漏洞检测方法, 如: 符号执行、模糊测试和静态分析等, 将其进行修改适配后应用于智能合约. 然而, 现阶段各类检测方法仍不够完善, 本文对未来的研究方向提出如下 8 点展望.

(1) 目前的检测工作主要关注影响合约安全的漏洞, 而较少对性能相关漏洞进行检测, 例如可重用性、可扩展性或降低 gas 消耗等. 这类漏洞因不会直接影响智能合约运行或数字财产安全而不被研究者关注, 但性能相关漏洞会缩短智能合约的使用寿命、降低合约的鲁棒性或造成不必要的资源消耗, 同时可能增加出现安全漏洞的风险, 因此未来的研究工作可针对性能相关的漏洞进行检测, 或基于已有经验提出能够增加智能合约健壮性的编码建议等.

(2) 目前的检测工作主要针对以太坊 Solidity 智能合约, 缺少对其他平台或其他语言编写智能合约的检测工作, 例如 Linux 基金会发起的 Hyperledger 开源区块链项目, 其支持 Go、Java 或 Python 等语言编写的智能合约, 而现有对这些平台的智能合约进行安全漏洞检测的工作较少, 且缺少针对其进行漏洞信息收集和分类的研究工作. Chen 等人[24]的调查也表明, 在 StackExchange 平台中存在上千条关于其他平台智能合约的安全问题讨论. 未来的检测工作可先从漏洞发现与识别出发, 参考现有的流程框架收集相关漏洞信息, 然后构建有效的方法对其他平台的智能合约进行安全漏洞检测. 研究者可从 2 个角度出发构建检测方法, 一种是分析不同平台、不同类型智能合约的特性, 构建有针对性的检测方法; 另一种可以考虑将智能合约代码转换为中间语言或中间表示, 例如 LLVM、AST 或 XML 等, 并针对转换后的中间语言进行安全漏洞检测. 借助中间语言可以实现跨多种智能合约语言的安全检测, 但需要注意将智能合约转换为中间语言过程中保证语义的一致性.

(3) 大部分检测工具将智能合约字节码作为输入. 字节码作为能够在以太坊中直接查看的数据, 通过字节码判断合约的安全性十分必要. 调用者可在使用前通过安全检测工具预先对合约的字节码进行安全检测. 需要注意的是, EVM 执行字节码时可能会修改部分字节码语义, 例如没有返回值的函数会添加默认返回值 0, 导致返回值的含义与调用者预期不同, 从而导致漏洞, 未来以字节码作为检测对象的研究工作需注意到 EVM 运行字节码引发的问题.

(4) 符号执行通过分析操作码构建堆栈, 利用符号化输入模拟执行智能合约程序, 在遍历过程中搜集路径约束, 通过约束求解器去除不可达路径, 具有覆盖率高的优点. 然而, 与 Java 虚拟机指令不同, 以太坊虚拟机指令的跳转目的地址不会直接写出, 需要通过堆栈获取, 这是符号执行过程中分析函数间调用的难点之一. 此外, 由于状态空间和执行路径指数化增长, 符号执行面临着由于约束条件过多或过于复杂而导致的求解失败或检测效率过低等问题. Oyente 可通过设置 Z3 求解时间或调用深度等方式限制检测时间在可接受范围, 但是将导致漏报率上升. 现有的检测方法中, 除 Manticore 外均基于静态符号执行对合约代码进行分析检测, 未来可考虑在符号执行中采用静动态结合的方式, 如通过动态符号执行简化需要求解的约束条件, 以缓解路径爆炸问题. 此外, 未来可尝试结合静态分析技术, 借助静态分析获得可能存在安全问题的代码位置, 将其设置为符号执行过程的重点目标, 从而提高检测效率.

(5) 模糊测试通过构建区块链环境, 部署智能合约程序并生成测试用例执行合约, 具有误报率较低的优点, 并可使用测试用例重现检测到的漏洞, 便于开发和测试人员调试. 然而, 现有的模糊测试工具面临以下问题: 一是对漏洞的建模不够精确导致误报, 未来的模糊测试方法可考虑建立更精准的漏洞模型, 或构建验证器对可能触发漏洞的测试用例进行确认; 二是实证研究结果表明提高测试覆盖率不一定能够提高漏洞检测能力[28]. 未来的模糊测

试工作可考虑如何高效组合测试用例形成调用序列, 以覆盖更深层次的程序状态, 从而更有效地发现漏洞. 此外, 目标制导的模糊测试方法在特定漏洞检测方面更具优势[117], 未来的研究工作可针对智能合约特定的漏洞构建目标制导的模糊测试方法.

(6) 相较于其他方法, 基于机器学习的智能合约漏洞检测方法的优点在于无需构建漏洞模型. 但机器学习, 尤其是深度学习的不足在于缺少可解释性, 其检测结果难以为开发或测试人员提供具体信息. 未来工作可尝试构建可解释性更强的机器学习模型, 或利用解释分析方法分析模型的检测结果, 例如给出可能存在漏洞的代码位置等.

(7) 形式化验证将程序作为输入, 通过构建形式化验证框架对智能合约的状态进行分析, 优点在于能够遍历合约所有可能的程序状态, 漏报率较低. 但部分形式化验证方法基于已有的形式化验证框架, 如定理证明器或模型验证器进行验证, 而将智能合约程序转换为形式化验证框架可识别输入的过程中可能导致语义缺失或修改. 未来的形式化验证方法可尝试构建能够支持直接描述智能合约语义的框架, 或增加一致性验证以保证原始合约与经过转换后的合约之间语义、功能及性能的一致性.

(8) 静态分析通常以源代码作为输入, 通过计算待检测代码与漏洞代码之间的相似度, 或通过语义分析的方法获得程序信息, 判断是否存在漏洞. 静态分析的优点是检测效率较高, 但依赖于漏洞的表示形式以及模型的准确性, 且没有对路径可达性进行分析, 因此具有较高的漏报和误报率. 未来工作可将动态检测方法作为静态分析的补充, 对静态分析结果进行动态确认, 从而降低漏报和误报率.

本文梳理了智能合约安全漏洞检测的研究框架, 整理了在文献中出现的漏洞模式, 并归纳和分析了已有的检测方法, 对智能合约安全检测领域的研究方向提出了展望, 希望能够为未来的研究者提供参考和帮助.

References:

[1] Shao QF, Jin CQ, Zhang Z, Qian WN, Zhou AY. Blockchain: Architecture and research progress. Chinese Journal of Computers, 2018, 41(5): 969–988 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00969]

[2] Zheng ZB, Xie SA, Dai HN, Chen XP, Wang HM. An overview of blockchain technology: Architecture, consensus, and future trends. In: Proc. of the 2017 IEEE Int'l Congress on Big Data (BigData Congress). Honolulu: IEEE, 2017. 557–564. [doi: 10.1109/BigDataCongress.2017.85]

[3] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID3440802_code87814.pdf [doi: 10.2139/ssrn.3440802]

[4] Yuan Y, Wang FY. Blockchain: The state of the art and future trends. Acta Automatica Sinica, 2016, 42(4): 481–494 (in Chinese with English abstract). [doi: 10.16383/j.aas.2016.c160158]

[5] Alharby M, Aldweesh A, van Moorsel A. Blockchain-based smart contracts: A systematic mapping study of academic research (2018). In: Proc. of the 2018 Int'l Conf. on Cloud Computing, Big Data and Blockchain. Fuzhou: IEEE, 2018. 1–6. [doi: 10.1109/ICCBB.2018.8756390]

[6] Christidis K, Devetsikiotis M. Blockchains and smart contracts for the Internet of Things. IEEE Access, 2016, 4: 2292–2303. [doi: 10.1109/ACCESS.2016.2566339]

[7] Azaria A, Ekblaw A, Vieira T, Lippman A. MedRec: Using blockchain for medical data access and permission management. In: Proc. of the 2nd Int'l Conf. on Open and Big Data. Vienna: IEEE, 2016. 25–30. [doi: 10.1109/OBD.2016.11]

[8] Buterin V. A next-generation smart contract and decentralized application platform. 2014. https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

[9] Ouyang LW, Wang S, Yuan Y, Ni XC, Wang FY. Smart contracts: Architecture and research progresses. Acta Automatica Sinica, 2019, 45(3): 445–457 (in Chinese with English abstract). [doi: 10.16383/j.aas.c180586]

[10] Bartoletti M, Pompianu L. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In: Proc. of the 2017 FC Int'l Workshops on Financial Cryptography and Data Security. Sliema: Springer, 2017. 494–509. [doi: 10.1007/978-3-319-70278-0_31]

[11] Luu L, Chu DH, Olickel H, Saxena P, Hobor A. Making smart contracts smarter. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 254–269. [doi: 10.1145/2976749.2978309]

[12] Jiang B, Liu Y, Chan WK. ContractFuzzer: Fuzzing smart contracts for vulnerability detection. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 259–269. [doi: 10.1145/3238147.3238177]

[13] Fu ML, Wu LF, Hong Z, Feng WB. Research on vulnerability mining technique for smart contracts. Journal of Computer Applications, 2019, 39(7): 1959–1966 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2019010082]

[14] Ni YD, Zhang C, Yin TT. A survey of smart contract vulnerability research. Journal of Cyber Security, 2020, 5(3): 78–99 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.07]

[15] Zheng ZB, Wang CD, Cai JH. Analysis of the current status of smart contract security research and detection methods. Information Security and Communications Privacy, 2020(7): 93–105 (in Chinese with English abstract). [doi: 10.3969/j.issn.1009-8054.2020.07.012]

[16] Qian P, Liu ZG, He QM, Huang BT, Tian DZ, Wang X. Smart contract vulnerability detection technique: A survey. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 3059–3085 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6375.htm [doi: 10.13328/j.cnki.jos.006375]

[17] Tu LQ, Sun XB, Zhang JL, Cai J, Li B, Bo LL. Survey of vulnerability detection tools for smart contracts. Computer Science, 2021, 48(11): 79–88 (in Chinese with English abstract). [doi: 10.11896/jsjkx.210600117]

[18] Hu TY, Li ZC, Li BX, Bao QH. Contractual security and privacy security of smart contract: A system mapping study. Chinese Journal of Computers, 2021, 44(12): 2485–2514 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2021.02485]

[19] Chen JC, Xia X, Lo D, Grundy J, Luo XP, Chen T. DefectChecker: Automated smart contract defect detection by analyzing EVM bytecode. IEEE Trans. on Software Engineering, 2022, 48(7): 2189–2207. [doi: 10.1109/TSE.2021.3054928]

[20] Nguyen TD, Pham LH, Sun J, Lin Y, Minh QT. sFuzz: An efficient adaptive fuzzer for Solidity smart contracts. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 778–788. [doi: 10.1145/3377811.3380334]

[21] So S, Hong S, Oh H. SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution. In: Proc. of the 30th USENIX Security Symp. USENIX Association, 2021. 1361–1378.

[22] Wüstholz V, Christakis M. Harvey: A greybox fuzzer for smart contracts. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1398–1409. [doi: 10.1145/3368089.3417064]

[23] Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering—A systematic literature review. Information and Software Technology, 2009, 51(1): 7–15. [doi: 10.1016/j.infsof.2008.09.009]

[24] Chen JC, Xia X, Lo D, Grundy J, Luo XP, Chen T. Defining smart contract defects on Ethereum. IEEE Trans. on Software Engineering, 2022, 48(1): 327–345. [doi: 10.1109/TSE.2020.2989002]

[25] Gao ZP, Jiang LX, Xia X, Lo D, Grundy J. Checking smart contracts with structural code embedding. IEEE Trans. on Software Engineering, 2021, 47(12): 2874–2891. [doi: 10.1109/TSE.2020.2971482]

[26] So S, Lee M, Park J, Lee H, Oh H. VERISMART: A highly precise safety verifier for Ethereum smart contracts. In: Proc. of the 2020 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2020. 1678–1694. [doi: 10.1109/SP40000.2020.00032]

[27] Wang AQ, Wang H, Jiang B, Chan WK. Artemis: An improved smart contract verification tool for vulnerability detection. In: Proc. of the 7th Int'l Conf. on Dependable Systems and Their Applications. Xi'an: IEEE, 2020. 173–181. [doi: 10.1109/DSA51864.2020.00031]

[28] Ren M, Yin ZJ, Ma FC, Xu ZY, Jiang Y, Sun CN, Li HZ, Cai Y. Empirical evaluation of smart contract testing: What is the best choice? In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2021). ACM, 2021. 566–579. [doi: 10.1145/3460319.3464837]

[29] Atzei N, Bartoletti M, Cimoli T. A survey of attacks on Ethereum smart contracts (Sok). In: Proc. of the 6th Int'l Conf. on Principles of Security and Trust. Uppsala: Springer, 2017. 164–186. [doi: 10.1007/978-3-662-54455-6_8]

[30] Chen JC. Finding Ethereum smart contracts security issues by comparing history versions. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2020. 1382–1384. [doi: 10.1145/3324884.3418923]

[31] Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. Trans. of the Association for Computational Linguistics, 2017, 5: 135–146. [doi: 10.1162/tacl_a_00051]

[32] Wohrer M, Zdun U. Smart contracts: Security patterns in the Ethereum ecosystem and Solidity. In: Proc. of the 2018 Int'l Workshop on Blockchain Oriented Software Engineering. Campobasso: IEEE, 2018. 2–8. [doi: 10.1109/IWBOSE.2018.8327565]

[33] Delmolino K, Arnett M, Kosba A, Miller A, Shi E. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In: Proc. of the 2016 Int'l Conf. on Financial Cryptography and Data Security. Christ Church: Springer, 2016. 79–94. [doi: 10.1007/978-3-662-53357-4_6]

[34] Durieux T, Ferreira JF, Abreu R, Cruz P. Empirical review of automated analysis tools on 47 587 Ethereum smart contracts. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 530–541. [doi: 10.1145/3377811.3380364]

[35] Dias B, Ivaki N, Laranjeiro N. An empirical evaluation of the effectiveness of smart contract verification tools. In: Proc. of the 26th IEEE Pacific Rim Int'l Symp. on Dependable Computing. Perth: IEEE, 2021. 17–26. [doi: 10.1109/PRDC53464.2021.00013]

[36] Liu C, Liu H, Cao Z, Chen Z, Chen BD, Roscoe B. ReGuard: Finding reentrancy bugs in smart contracts. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering: Companion. Gothenburg: ACM, 2018. 65–68. [doi: 10.1145/3183440.3183495]

[37] Wang W, Song JJ, Xu GQ, Li YD, Wang H, Su CH. ContractWard: Automated vulnerability detection models for Ethereum smart contracts. IEEE Trans. on Network Science and Engineering, 2021, 8(2): 1133–1144. [doi: 10.1109/TNSE.2020.2968505]

[38] Rodler M, Li WT, Karame GO, Davi L. Sereum: Protecting existing smart contracts against re-entrancy attacks. In: Proc. of the 26th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2019. [doi: 10.14722/ndss.2019.23413]

[39] Zhao W, Zhang WY, Wang JR, Wang HF, Wu CK. Smart contract vulnerability detection scheme based on symbol execution. Journal of Computer Applications, 2020, 40(4): 947–953 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2019111919]

[40] Tsankov P, Dan A, Drachsler-Cohen D, Gervais A, Bünzli F, Vechev M. Securify: Practical security analysis of smart contracts. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 67–82. [doi: 10.1145/3243734.3243780]

[41] Feng Y, Torlak E, Bodik R. Summary-based symbolic evaluation for smart contracts. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2020. 1141–1152. [doi: 10.1145/3324884.3416646]

[42] Torres CF, Iannillo AK, Gervais A, State R. ConFuzzius: A data dependency-aware hybrid fuzzer for smart contracts. In: Proc. of the 2021 IEEE European Symp. on Security and Privacy (EuroS&P). Vienna: IEEE, 2021. 103–119. [doi: 10.1109/EuroSP51992.2021.00018]

[43] Zhuang Y, Liu ZG, Qian P, Liu Q, Wang X, He QM. Smart contract vulnerability detection using graph neural networks. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence (IJCAI). Yokohama: IJCAI.org, 2021. 454.

[44] Qian P, Liu ZG, He QM, Zimmermann R, Wang X. Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access, 2020, 8: 19685–19695. [doi: 10.1109/ACCESS.2020.2969429]

[45] Grishchenko I, Maffei M, Schneidewind C. A semantic framework for the security analysis of Ethereum smart contracts. In: Proc. of the 7th Int'l Conf. on Principles of Security and Trust. Thessaloniki: Springer, 2018. 243–269. [doi: 10.1007/978-3-319-89722-6_10]

[46] Kalra S, Goel S, Dhawan M, Sharma S. ZEUS: Analyzing safety of smart contracts. In: Proc. of the 2018 Network and Distributed System Security Symp. San Diego: The Internet Society, 2018. 1–12. [doi: 10.14722/ndss.2018.23082]

[47] Tikhomirov S, Voskresenskaya E, Ivanitskiy I, Takhaviev R, Marchenko E, Alexandrov Y. SmartCheck: Static analysis of Ethereum smart contracts. In: Proc. of the 1st Int'l Workshop on Emerging Trends in Software Engineering for blockchain. Gothenburg: ACM, 2018. 9–16. [doi: 10.1145/3194113.3194115]

[48] Zhang PC, Xiao F, Luo XP. SolidityCheck: Quickly detecting smart contract problems through regular expressions. arXiv:1911.09425, 2019.

[49] Han SM, Liang B, Huang JJ, Shi WC. DC-Hunter: Detecting dangerous smart contracts via bytecode matching. Journal of Cyber Security, 2020, 5(3): 100–112 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.08]

[50] Huang JJ, Han SM, You W, Shi WC, Liang B, Wu JZ, Wu YJ. Hunting vulnerable smart contracts via graph embedding based bytecode matching. IEEE Trans. on Information Forensics and Security, 2021, 16: 2144–2156. [doi: 10.1109/TIFS.2021.3050051]

[51] Feist J, Grieco G, Groce A. Slither: A static analysis framework for smart contracts. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Montreal: IEEE, 2019. 8–15. [doi: 10.1109/WETSEB.2019.00008]

[52] Ye JM, Ma ML, Lin Y, Sui YL, Xue YX. Clairvoyance: Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-Companion). Seoul: ACM, 2020. 274–275. [doi: 10.1145/3377812.3390908]

[53] Brent L, Jurisevic A, Kong M, Liu E, Gauthier F, Gramoli V, Holz R, Scholz B. Vandal: A scalable security analysis framework for smart contracts. arXiv:1809.03981, 2018.

[54] Nikolić I, Kolluri A, Sergey I, Saxena P, Hobor A. Finding the greedy, prodigal, and suicidal contracts at scale. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 653–663. [doi: 10.1145/3274694.3274743]

[55] Chang JL, Gao B, Xiao H, Sun J, Cai Y, Yang ZJ. sCompile: Critical path identification and analysis for smart contracts. In: Proc. of the 21st Int'l Conf. on Formal Engineering Methods. Shenzhen: Springer, 2019. 286–304. [doi: 10.1007/978-3-030-32409-4_18]

[56] He JX, Balunović M, Ambroladze N, Tsankov P, Vechev M. Learning to fuzz from symbolic execution with application to smart contracts. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 531–548. [doi: 10.1145/3319535.3363230]

[57] Torres CF, Schütte J, State R. Osiris: Hunting for integer bugs in Ethereum smart contracts. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 664–676. [doi: 10.1145/3274694.3274737]

[58] Zhou EC, Hua S, Pi BF, Sun J, Nomura Y, Yamashita K, Kurihara H. Security assurance for smart contract. In: Proc. of the 9th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). Paris: IEEE, 2018. 1–5. [doi: 10.1109/NTMS.2018.8328743]

[59] Grech N, Kong M, Jurisevic A, Brent L, Scholz B, Smaragdakis Y. MadMax: Surviving out-of-gas conditions in Ethereum smart contracts. Proc. of the ACM on Programming Languages, 2018, 2(OOPSLA): 116. [doi: 10.1145/3276486]

[60] Liu H, Liu C, Zhao WQ, Jiang Y, Sun JG. S-gram: Towards semantic-aware security auditing for Ethereum smart contracts. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Montpellier: ACM, 2018. 814–819. [doi: 10.1145/3238147.3240728]

[61] Krupp J, Rossow C. TeEther: Gnawing at Ethereum to automatically exploit smart contracts. In: Proc. of the 27th USENIX Security Symp. Baltimore: USENIX Association, 2018. 1317–1333.

[62] Chen T, Li XQ, Luo XP, Zhang XS. Under-optimized smart contracts devour your money. In: Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Klagenfurt: IEEE, 2017. 442–446. [doi: 10.1109/SANER.2017.7884650]

[63] Chen T, Li ZH, Zhou H, Chen JC, Luo XP, Li XQ, Zhang XS. Towards saving money in using smart contracts. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Gothenburg: ACM, 2018. 81–84. [doi: 10.1145/3183399.3183420]

[64] Wood G. Ethereum: A secure decentralised generalised transaction ledger. 2014. https://ethereum.github.io/yellowpaper/paper.pdf

[65] Marino B, Juels A. Setting standards for altering and undoing smart contracts. In: Proc. of the 10th Int'l Symp. on Rule Technologies. Research, Tools, and Applications. Stony Brook: Springer, 2016. 151–166. [doi: 10.1007/978-3-319-42019-6_10]

[66] Andrychowicz M, Dziembowski S, Malinowski D, Mazurek L. Secure multiparty computations on bitcoin. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. Berkeley: IEEE, 2014. 443–458. [doi: 10.1109/SP.2014.35]

[67] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Liang B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 80–109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[68] Zhang W, Banescu S, Pasos L, Stewart S, Ganesh V. MPro: Combining static and symbolic analysis for scalable testing of smart contract. In: Proc. of the 30th IEEE Int'l Symp. on Software Reliability Engineering. Berlin: IEEE, 2019. 456–462. [doi: 10.1109/ISSRE.2019.00052]

[69] Mossberg M, Manzano F, Hennenfent E, Groce A, Grieco G, Feist J, Brunson T, Dinaburg A. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1186–1189. [doi: 10.1109/ASE.2019.00133]

[70] Zheng PL, Zheng ZB, Luo XP. Park: Accelerating smart contract vulnerability detection via parallel-fork symbolic execution. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 740–751. [doi: 10.1145/3533767.3534395]

[71] Grossman S, Abraham I, Golan-Gueta G, Michalevsky Y, Rinetzky N, Sagiv M, Zohar Y. Online detection of effectively callback free objects with applications to smart contracts. Proc. of the ACM on Programming Languages, 2018, 2(POPL): 48. [doi: 10.1145/3158136]

[72] Liu Y, Li Y, Lin SW, Artho C. Finding permission bugs in smart contracts with role mining. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 716–727. [doi: 10.1145/3533767.3534372]

[73] Jin L, Cao YZ, Chen Y, Zhang D, Campanoni S. ExGen: Cross-platform, automated exploit generation for smart contract vulnerabilities. IEEE Trans. on Dependable and Secure Computing, 2023, 20(1): 650–664. [doi: 10.1109/TDSC.2022.3141396]

[74] Li HH, Qi J, Liu F, Yang FN. The research progress of fuzz testing technology. SCIENTIA SINICA Informationis, 2014, 44(10): 1305–1322 (in Chinese with English abstract). [doi: 10.1360/N112014-00137]

[75] Manès VJM, Han H, Han C, Cha SK, Egele M, Schwartz EJ, Woo M. The art, science, and engineering of fuzzing: A survey. IEEE Trans. on Software Engineering, 2021, 47(11): 2312–2331. [doi: 10.1109/TSE.2019.2946563]

[76] Huang YH, Jiang B, Chan WK. EOSFuzzer: Fuzzing EOSIO smart contracts for vulnerability detection. In: Proc. of the 12th Asia-Pacific Symp. on Internetware. Singapore: ACM, 2020. 99–109. [doi: 10.1145/3457913.3457920]

[77] Choi J, Kim D, Kim S, Grieco G, Groce A, Cha SK. Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 227–239. [doi: 10.1109/ASE51524.2021.9678888]

[78] Xue YX, Ye JM, Zhang W, Sun J, Ma L, Wang HJ, Zhao JJ. xFuzz: Machine learning guided cross-contract fuzzing. IEEE Trans. on Dependable and Secure Computing, 2022: 1–14. [doi: 10.1109/TDSC.2022.3182373]

[79] Liu ZG, Qian P, Yang JX, Liu LF, Xu XJ, He QM, Zhang XS. Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting. IEEE Trans. on Information Forensics and Security, 2023, 18: 1237–1251. [doi: 10.1109/TIFS.2023.3237370]

[80] Chen WM, Sun ZH, Wang HY, Luo XP, Cai HP, Wu L. WASAI: Uncovering vulnerabilities in Wasm smart contracts. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 703–715. [doi: 10.1145/3533767.3534218]

[81] Lemieux C, Sen K. Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 475–485. [doi: 10.1145/3238147.3238176]

[82] Wüstholz V, Christakis M. Targeted greybox fuzzing with static lookahead analysis. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Seoul: ACM, 2020. 789–800. [doi: 10.1145/3377811.3380388]

[83] Gu MX, Sun HY, Han D, Yang S, Cao WY, Guo Z, Cao CJ, Wang WJ, Zhang YQ. Software security vulnerability mining based on deep learning. Journal of Computer Research and Development, 2021, 58(10): 2140–2162 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2021.20210620]

[84] Alhuzali A, Gjomemo R, Eshete B, Venkatakrishnan VN. NAVEX: Precise and scalable exploit generation for dynamic Web applications. In: Proc. of the 27th USENIX Conf. on Security Symp. Baltimore: USENIX Association, 2018. 377–392.

[85] Melicher W, Ur B, Segreti SM, Komanduri S, Bauer L, Christin N, Cranor LF. Fast, lean, and accurate: Modeling password guessability using neural networks. In: Proc. of the 25th USENIX Conf. on Security Symp. Austin: USENIX Association, 2016. 175–191.

[86] Wu YX, Cai T, Zhang DB. Automatic smart contract classification model based on hierarchical attention mechanism and bidirectional long short-term memory neural network. Journal of Computer Application, 2020, 40(4): 978–984 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2019081327]

[87] Liu ZG, Qian P, Wang XY, Zhuang Y, Qiu L, Wang X. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(2): 1296–1310. [doi: 10.1109/TKDE.2021.3095196]

[88] Zhang Z, Lei Y, Yan M, Yu Y, Chen JC, Wang SW, Mao XG. Reentrancy vulnerability detection and localization: A deep learning based two-phase approach. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 83. [doi: 10.1145/3551349.3560428]

[89] Tann WJW, Han XJ, Gupta SS, Ong YS. Towards safer smart contracts: A sequence learning approach to detecting security threats. arXiv:1811.06632, 2018.

[90] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002, 16(1): 321–357.

[91] Batista GEAPA, Prati RC, Monard MC. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter, 2004, 6(1): 20–29. [doi: 10.1145/1007730.1007735]

[92] Yu XX, Zhao HY, Hou BT, Ying ZH, Wu B. DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection. In: Proc. of the 2021 Int'l Joint Conf. on Neural Networks (IJCNN). Shenzhen: IEEE, 2021. 1–8. [doi: 10.1109/IJCNN52387.2021.9534324]

[93] You J, Li JQ, Xia S. A survey on formal methods using in software development. In: Proc. of the 2012 IET Int'l Conf. on Information Science and Control Engineering (ICISCE 2012). Shenzhen: IET, 2012. 1–4. [doi: 10.1049/cp.2012.2353]

[94] Wang HB, Zheng CY, Huang S, Sun JL, Ding YX. Review: Secure formal verification methods for Ethereum smart contracts. Computer Technology and Development, 2021, 31(9): 104–111 (in Chinese with English abstract). [doi: 10.3969/j.issn.1673-629X.2021.09.018]

[95] Zhu J, Hu K, Zhang BJ. Review on formal verification of smart contract. Acta Electronica Sinica, 2021, 49(4): 792–804 (in Chinese with English abstract). [doi: 10.12263/DZXB.20200723]

[96] Nipkow T, Wenzel M, Paulson LC. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Berlin: Springer, 2002. [doi: 10.1007/3-540-45949-9]

[97] Hirai Y. Defining the Ethereum virtual machine for interactive theorem provers. In: Proc. of the 2017 Int'l Conf. on Financial Cryptography and Data Security. Sliema: Springer, 2017. 520–535. [doi: 10.1007/978-3-319-70278-0_33]

[98] Yang Z, Lei H. FEther: An extensible definitional interpreter for smart-contract verifications in Coq. IEEE Access, 2019, 7: 37770–37791. [doi: 10.1109/ACCESS.2019.2905428]

[99] Wang YP, Lahiri SK, Chen S, Pan R, Dillig I, Born C, Naseer I. Formal specification and verification of smart contracts for azure blockchain. arXiv:1812.08829, 2018.

[100] Antonino P, Roscoe AW. Formalising and verifying smart contracts with Solidifier: A bounded model checker for Solidity. arXiv:2002.02710, 2020.

[101] Bhargavan K, Delignat-Lavaud A, Fournet C, Gollamudi A, Gonthier G, Kobeissi N, Kulatova N, Rastogi A, Sibut-Pinote T, Swamy N, Zanella-Béguelin S. Formal verification of smart contracts: Short paper. In: Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security. Vienna: ACM, 2016. 91–96. [doi: 10.1145/2993600.2993611]

[102] Hildenbrandt E, Saxena M, Rodrigues N, Zhu XR, Daian P, Guth D, Moore B, Park D, Zhang Y, Stefanescu A, Rosu G. KEVM: A complete formal semantics of the Ethereum virtual machine. In: Proc. of the 31st IEEE Computer Security Foundations Symp. (CSF). Oxford: IEEE, 2018. 204–217. [doi: 10.1109/CSF.2018.00022]

[103] Permenev A, Dimitrov D, Tsankov P, Drachsler-Cohen D, Vechev M. VerX: Safety verification of smart contracts. In: Proc. of the 2020 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2020. 1661–1677. [doi: 10.1109/SP40000.2020.00024]

[104] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009, 32(9): 1697–1710 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.01697]

[105] Liu BC, Shi L, Cai ZH, Li M. Software vulnerability discovery techniques: A survey. In: Proc. of the 4th Int'l Conf. on Multimedia Information Networking and Security. Nanjing: IEEE, 2012. 152–156. [doi: 10.1109/MINES.2012.202]

[106] Gao ZP, Jayasundara V, Jiang LX, Xia X, Lo D, Grundy J. SmartEmbed: A tool for clone and bug detection in smart contracts through structural code embedding. In: Proc. of the 2019 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Cleveland: IEEE, 2019. 394–397. [doi: 10.1109/ICSME.2019.00067]

[107] Gao ZP. When deep learning meets smart contracts. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). ACM, 2020. 1400–1402. [doi: 10.1145/3324884.3418918]

[108] Gao JB, Liu H, Liu C, Li QS, Guan Z, Chen Z. EasyFlow: Keep Ethereum away from overflow. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-Companion). Montreal: IEEE, 2019. 23–26. [doi: 10.1109/ICSE-Companion.2019.00029]

[109] Xue YX, Ma ML, Lin Y, Sui YL, Ye JM, Peng TY. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: ACM, 2020. 1029–1040. [doi: 10.1145/3324884.3416553]

[110] Ghaleb A, Rubin J, Pattabiraman K. eTainter: Detecting gas-related vulnerabilities in smart contracts. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 728–739. [doi: 10.1145/3533767.3534378]

[111] Jordan H, Scholz B, Subotić P. SOUFFLÉ: On synthesis of program analyzers. In: Proc. of the 28th Int'l Conf. on Computer Aided Verification. Toronto: Springer, 2016. 422–430. [doi: 10.1007/978-3-319-41540-6_23]

[112] Albert E, Gordillo P, Livshits B, Rubio A, Sergey I. EthIR: A framework for high-level analysis of Ethereum bytecode. In: Proc. of the 16th Int'l Symp. on Automated Technology for Verification and Analysis. Los Angeles: Springer, 2018. 513–520. [doi: 10.1007/978-3-030-01090-4_30]

[113] Albert E, Arenas P, Flores-Montoya A, Genaim S, Gómez-Zamalloa M, Martin-Martin E, Puebla G, Román-Díez G. SACO: Static analyzer for concurrent objects. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Grenoble: Springer, 2014. 562–567. [doi: 10.1007/978-3-642-54862-8_46]

[114] Liao ZQ, Zheng ZB, Chen X, Nan YH. SmartDagger: A bytecode-based static analysis approach for detecting cross-contract vulnerability. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 752–764. [doi: 10.1145/3533767.3534222]

[115] Yang HW, Cui ZQ, Chen X, Jia MH, Zheng LW, Liu JB. Defect prediction for Solidity smart contracts based on software measurement. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1587–1611 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6550.htm [doi: 10.13328/j.cnki.jos.006550]

[116] Ghaleb A, Pattabiraman K. How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2020). ACM, 2020. 415–427. [doi: 10.1145/3395363.3397385]

[117] Wang PF, Zhou X, Lu K, Yue T, Liu YY. Sok: The progress, challenges, and perspectives of directed greybox fuzzing. SSRN, 2022. [doi: 10.2139/ssrn.4129684]

附中文参考文献:

[1] 邵奇峰, 金澈清, 张召, 钱卫宁, 周傲英. 区块链技术: 架构及进展. 计算机学报, 2018, 41(5): 969–988. [doi: 10.11897/SP.J.1016.2018.00969]

[4] 袁勇, 王飞跃. 区块链技术发展现状与展望. 自动化学报, 2016, 42(4): 481–494. [doi: 10.16383/j.aas.2016.c160158]

[9] 欧阳丽炜, 王帅, 袁勇, 倪晓春, 王飞跃. 智能合约: 架构及进展. 自动化学报, 2019, 45(3): 445–457. [doi: 10.16383/j.aas.c180586]

[13] 付梦琳, 吴礼发, 洪征, 冯文博. 智能合约安全漏洞挖掘技术研究. 计算机应用, 2019, 39(7): 1959–1966. [doi: 10.11772/j.issn.1001-9081.2019010082]

[14] 倪远东, 张超, 殷婷婷. 智能合约安全漏洞研究综述. 信息安全学报, 2020, 5(3): 78–99. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.07]
[15] 郑忠斌, 王朝栋, 蔡佳浩. 智能合约的安全研究现状与检测方法分析综述. 信息安全与通信保密, 2020(7): 93–105. [doi: 10.3969/j.issn.1009-8054.2020.07.012]
[16] 钱鹏, 刘振广, 何钦铭, 黄步添, 田端正, 王勋. 智能合约安全漏洞检测技术研究综述. 软件学报, 2022, 33(8): 3059–3085. http://www.jos.org.cn/1000-9825/6375.htm [doi: 10.13328/j.cnki.jos.006375]
[17] 涂良琼, 孙小兵, 张佳乐, 蔡杰, 李斌, 薄莉莉. 智能合约漏洞检测工具研究综述. 计算机科学, 2021, 48(11): 79–88. [doi: 10.11896/jsjkx.210600117]
[18] 胡甜媛, 李泽成, 李必信, 包骐豪. 智能合约的合约安全和隐私安全研究综述. 计算机学报, 2021, 44(12): 2485–2514. [doi: 10.11897/SP.J.1016.2021.02485]
[39] 赵伟, 张问银, 王九如, 王海峰, 武传坤. 基于符号执行的智能合约漏洞检测方案. 计算机应用, 2020, 40(4): 947–953. [doi: 10.11772/j.issn.1001-9081.2019111919]
[49] 韩松明, 梁彬, 黄建军, 石文昌. DC-Hunter: 一种基于字节码匹配的危险智能合约检测方案. 信息安全学报, 2020, 5(3): 100–112. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.08]
[67] 张健, 张超, 玄跻峰, 熊英飞, 王千祥, 梁彬, 李炼, 窦文生, 陈振邦, 陈立前, 蔡彦. 程序分析研究进展. 软件学报, 2019, 30(1): 80–109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]
[74] 李红辉, 齐佳, 刘峰, 杨芳南. 模糊测试技术研究. 中国科学: 信息科学, 2014, 44(10): 1305–1322. [doi: 10.1360/N112014-00137]
[83] 顾绵雪, 孙鸿宇, 韩丹, 杨粟, 曹婉莹, 郭祯, 曹春杰, 王文杰, 张玉清. 基于深度学习的软件安全漏洞挖掘. 计算机研究与发展, 2021, 58(10): 2140–2162. [doi: 10.7544/issn1000-1239.2021.20210620]
[86] 吴雨芯, 蔡婷, 张大斌. 基于层级注意力机制与双向长短期记忆神经网络的智能合约自动分类模型. 计算机应用, 2020, 40(4): 978–984. [doi: 10.11772/j.issn.1001-9081.2019081327]
[94] 王赫彬, 郑长友, 黄松, 孙金磊, 丁一先. 以太坊智能合约安全形式化验证方法研究进展. 计算机技术与发展, 2021, 31(9): 104–111. [doi: 10.3969/j.issn.1673-629X.2021.09.018]
[95] 朱健, 胡凯, 张伯钧. 智能合约的形式化验证方法研究综述. 电子学报, 2021, 49(4): 792–804. [doi: 10.12263/DZXB.20200723]
[104] 梅宏, 王千祥, 张路, 王戟. 软件分析技术进展. 计算机学报, 2009, 32(9): 1697–1710. [doi: 10.3724/SP.J.1016.2009.01697]
[115] 杨慧文, 崔展齐, 陈翔, 贾明华, 郑丽伟, 刘建宾. 基于软件度量的 Solidity 智能合约缺陷预测方法. 软件学报, 2022, 33(5): 1587–1611. http://www.jos.org.cn/1000-9825/6550.htm [doi: 10.13328/j.cnki.jos.006550]

崔展齐(1984－), 男, 博士, 教授, CCF 高级会员, 主要研究领域为软件分析与软件测试技术.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件缺陷预测, 软件缺陷定位, 组合测试.

杨慧文(1997－), 男, 硕士生, 主要研究领域为智能合约测试技术.

王林章(1973－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为软件工程, 软件安全.