软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.006810]

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

智能合约漏洞检测技术综述*

董伟良[1], 刘 哲[1], 刘 逵[1], 黎 立[2], 葛春鹏[1], 黄志球[1]

[1](南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

[2](蒙纳士大学, 克莱顿 VIC 3800, 澳大利亚)

通信作者: 刘逵, E-mail: kui.liu@nuaa.edu.cn

摘 要: 智能合约作为可信的去中心化应用, 获得了广泛的关注, 但其安全漏洞问题对其可靠性带来了巨大威胁. 为此, 研究者们利用各种前沿技术 (如模糊测试、机器学习、形式化验证等) 研究了多种漏洞检测技术, 并取得了可观的效果. 为了系统性地梳理与分析现有智能合约漏洞检测技术, 搜集截至 2021 年 7 月关于智能合约漏洞检测的 84 篇论文, 根据它们的核心方法进行分类, 从每种技术的实现方法、漏洞类型、实验数据等方面展开分析, 同时对比国内外研究现状在这些方面的差异. 最后, 对现有的智能合约漏洞检测技术进行总结, 探讨面临的挑战, 并展望了未来的研究方向.

关键词: 智能合约; 合约安全; 合约可靠性; 合约质量保障; 漏洞检测; 合约程序分析

中图法分类号: TP311

中文引用格式: 董伟良, 刘哲, 刘逵, 黎立, 葛春鹏, 黄志球. 智能合约漏洞检测技术综述. 软件学报. http://www.jos.org.cn/1000-9825/6810.htm

英文引用格式: Dong WL, Liu Z, Liu K, Li L, Ge CP, Huang ZQ. Survey on Vulnerability Detection Technology of Smart Contracts. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/6810.htm

Survey on Vulnerability Detection Technology of Smart Contracts

DONG Wei-Liang[1], LIU Zhe[1], LIU Kui[1], LI Li[2], GE Chun-Peng[1], HUANG Zhi-Qiu[1]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](Monash University, Clayton VIC 3800, Australia)

Abstract: As the trusted decentralized application, smart contracts attract widespread attention, whereas their security vulnerabilities threaten the reliability. To this end, researchers employ various advanced technologies (such as fuzz testing, machine learning, and formal verification) to study several vulnerability detection technologies and yield sound effects. This study collects 84 related papers by July 2021 to systematically sort out and analyze existing vulnerability detection technologies of smart contracts. First of all, vulnerability detection technologies are categorized according to their core methodologies. These technologies are analyzed from the aspects of implementation methods, vulnerability categories, and experimental data. Additionally, the differences between domestic and international research in these aspects are compared. Finally, after summarizing the existing technologies, the study discusses the challenges of vulnerability detection technologies and potential research directions.

Key words: smart contract; contract security; contract reliability; contract quality assurance; vulnerability detection; contract program analysis

智能合约 (smart contract) 是最早由 Szabo[1]于 1996 年首次引入的概念, 它是一段以计算机指令方式实现的传统合约的自动化处理程序, 旨在以信息化方式传播、验证或执行合约的计算机协议. 智能合约设计的总体目标是

* 基金项目: 科技部重点研发计划 (2021YFB2700503); 国家自然科学基金 (62172214, 62032025, U20A201092, 62071222); 广东省重点研发计划 (2020B0101090002); 江苏省自然科学基金 (BK20210279, BK20200418); 江苏省科技支撑计划 (BE2020106); 数学工程与先进计算国家重点实验室开放基金 (2020A06)

收稿时间: 2021-09-06; 修改时间: 2022-01-12, 2022-03-03, 2022-05-15, 2022-09-06; 采用时间: 2022-10-09; jos 在线出版时间: 2023-05-17

满足合约条件, 最大限度地减少对可信中介的依赖. 近年来, 以以太坊[2]为代表的一批智能合约平台得到迅猛发展, 以太坊作为最流行的智能合约平台, 截止到 2020 年末地址数量已经超过 1 亿, 总市值超过 1 000 亿美元[3]. 除了开启全新的数字化经济商业模式, 智能合约已经在数字身份、物联网、供应链等多个领域得到应用, 同时也支持广泛的去中心化应用程序, 如钱包、预测市场、即时消息、博客、众筹等[4], 且前景广阔.

智能合约虽名曰“智能”, 其本质就是双方资产交易时触发执行的一段代码, 且并不“智能”, 其“智能”之处在于: 智能合约程序不只是一个可以自动执行的计算机程序, 它本身就是一个系统参与者, 对接收到的信息进行回应, 可以接收和储存代币, 也可以向外发送信息和代币[5]. 智能合约由高级编程语言 (如 Solidity[6]) 编写, 然后编译成支持图灵完备指令集的虚拟机 (如 EVM、WASM 等) 字节码, 并发布到支持运行智能合约的系统上[7], 智能合约一旦发布成功, 任何人都无法修改其代码或阻止其执行, 即不可逆性[2]. 智能合约和传统的软件程序一样, 其代码也可能存在漏洞问题[5], 由于智能合约所关联的巨额数字资产和其不可逆性, 其代码中的漏洞容易被攻击, 这将造成不同程度的数字资产损失, 可能会造成灾难性的后果[8]. 2016 年, 以太坊上的 DAO 项目遭到黑客攻击, 攻击者利用项目中存在的重入漏洞窃取了当时市值 5 000 万美元的 360 万个以太币[9]; 2017 年 6 月, 攻击者利用 Parity 签名钱包使用的库合约漏洞, 偷取了当时市值 3 000 万美元的以太币[10]; 同年 11 月, Parity 钱包新版本的漏洞永久性冻结了当时价值 1.5 亿美元的以太币[11]; 2021 年 8 月, 黑客通过漏洞盗取了当时价值 6 000 万美元的加密货币[12].

智能合约的漏洞问题会带来巨大的经济损失, 然而智能合约的不可逆性使得其代码漏洞问题在其发布之后无法被修复, 这对智能合约的可靠性提出了更高的要求. 为此, 智能合约漏洞检测技术应运而生, 其旨在发布智能合约之前, 从智能合约中有效地检测出潜在的代码安全漏洞, 以提高智能合约的可靠性. 近年来, 研究者们探索利用软件测试、程序分析和机器学习等各种前沿技术研究了多种多样的智能合约漏洞检测技术, 并取得了可观的效果. 为了对智能合约漏洞检测技术的研究进展进行系统地归纳总结和分析比较, 本文整理了自 2014 年以太坊发表以来智能合约漏洞检测研究的相关论文发表情况并进行了梳理. 我们采用知网、谷歌学术、ACM Digital Library、IEEE Explore、Springer、Elsevier 与 DBLP 等文献搜索引擎和数据库, 先使用“smart contract”作为检索关键词查找出所有智能合约相关论文, 而后依据“security”“vulnerability”“vulnerable”“reliability”“reliable”等关于可靠性、安全性的关键词通过对文章题目、摘要、引言等信息的检阅, 人工过滤掉了与智能合约漏洞检测的无关论文, 最后从 1 031 篇智能合约论文中收集到了 84 篇智能合约漏洞检测技术的相关论文.

图 1 展示了 2014 年至 2021 年 7 月每年关于智能合约和智能合约漏洞检测的论文发表统计情况. 图中数据表明, 在 2016 年出现 DAO 攻击后, 关于智能合约的研究逐年暴增, 智能合约漏洞检测也成为一大研究热点, 且研究热度在不断增加. 为此, 我们进一步对论文的发表情况进行了统计, 相关论文主要发表在软件工程领域和网络信息安全领域的各类高质量会议和期刊上. 软件工程领域包含会议 ICSE (4 篇)、FSE/ESEC (2 篇)、ASE (4 篇)、ISSTA (3 篇)、PLDI (1 篇)、ISSRE (1 篇)、SANER (2 篇)、EASE (2 篇)、APSEC (1 篇)、ATVA (1 篇)、ICFEM (1 篇)、ICECCS (1 篇), 期刊 TSE (1 篇)、TSC (1 篇)、SPE (1 篇), 共计 23 篇会议论文, 3 篇期刊论文; 网络信息安全领域包含会议 CCS (4 篇)、S&P (2 篇)、USENIX (3 篇)、NDSS (2 篇)、ACSAC (2 篇), 期刊 TIFS (1 篇), 共计 13 篇会议论文, 1 篇期刊论文; 另有 3 篇文章分别发表在人工智能领域的会议 IJCAI、数据挖掘领域的期刊 IPM 和交叉领域的会议 FCS 上. 依据中国计算机学会 (CCF) 推荐的学术会议和期刊分类, A 类会议 24 篇、A 类期刊 2 篇、B 类会议 7 篇、B 类期刊 3 篇、C 类会议 7 篇. 其他未被列入 CCF 推荐期刊/会议文章 41 篇, 其中有 19 篇未经同行评审直接公布在开放电子论文库 arXiv 上, 非 CCF 推荐期刊/会议 (即“other”分类) 文章 22 篇 (包括 Workshop 文章 4 篇).

目前已有关于智能合约漏洞检测技术研究进展的分析与总结工作, 付梦琳等人[13]在 2019 年就对智能合约面临的安全威胁与主流的智能合约漏洞检测手段进行了分析和总结, 此后又有多篇相关综述文章发表[14,15]. 上述工作处于智能合约漏洞检测初步阶段, 主要对 2020 年之前的相关研究进行了调查与介绍, 这些文章主要综述了智能合约的漏洞问题, 对智能合约漏洞检测技术的分析过少, 对智能合约漏洞检测技术中的数据也未进行全面梳理, 尤其没有对国内外的研究现状进行对比分析. 图 1 中的统计数据表明, 智能合约漏洞检测技术的研究在 2019 年与 2020 年得到了迅猛发展, 例如, 采用诸如模糊测试、污点分析、形式化验证等方法的漏洞检测技术不断被提出,

且机器学习也被应用到智能合约漏洞检测中. 因此, 本文旨在对目前已有的智能合约漏洞检测技术进行一个全面的分析与总结, 对已有技术进行系统分类, 并从实现方法、漏洞类型、实验数据等统计信息展开分析, 此外, 基于目前该领域的研究现状, 本文对该领域面临的挑战以及研究的可行思路进行了探讨.

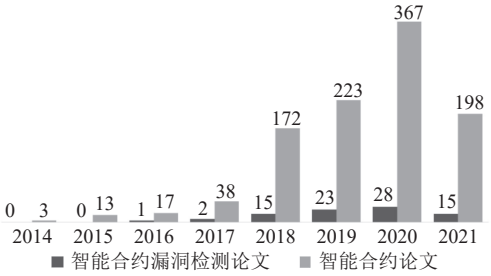

图 1 论文年份分布图

综上所述, 本文的主要贡献如下.

(1) 依据每篇论文作者单位的国别分类, 对国内外智能合约漏洞检测技术的研究进展进行系统的对比分析和介绍.

(2) 根据智能合约漏洞检测的核心技术对已有的研究工作进行了分类.

(3) 总结了该领域的开源工具.

(4) 分析对比了国内外的研究现状, 以促进国内在该领域的研究.

(5) 对当前智能合约漏洞检测技术所面临的挑战进行了系统梳理, 并总结了未来可能的研究方向.

1 相关工作

伴随着智能合约技术的发展及其应用领域的增加, 越来越多的安全漏洞被攻击者发现并利用, 催生了一批关于智能合约生态安全的研究, 目前已有一些关于这批智能合约安全文章的调研与综述, 包括安全漏洞审查[5,16−20]、设计模式[21−24]、验证方法和工具[13−15,25−32]、形式化规范与建模技术[33−35]和智能合约平台[36,37]与开发语言[38,39]等方向[40−46], 本节总结了当前关于智能合约安全分析和漏洞检测的调研与综述.

Atzei 等人[5]和一些最近的工作[16−20]收集并各自分类了多种可能危及智能合约安全的常见漏洞模式; Permenev 等人[47]和 Bernardi 等人[48]定义了安全分析师在合约审计时应该注意的几类属性; 其他研究人员提出了几种智能合约的设计模式[21−24], 是一种旨在针对特定任务的合约开发过程中, 通用且可重用的解决方案.

Bartoletti 等人[36]比较了 6 种平台: 比特币、以太坊、Counterparty、Stellar、Monax 和 Lisk; Seijas 等人[37]讨论了比特币、Nxt 和以太坊等系统采用的语言, 并列出了这些语言的缺陷, 此外, 他们还指出了一些可能有助于扩展智能合约功能并加强其安全性的技术; Parizi 等人[38]和 Harz 等人的最近一份研究[39]则对比现阶段智能合约编程语言之间性能与安全方面的差异, 并简要介绍了一些检测方法.

Durieux 等人[30]在两个数据集上运行并对比了 SmartCheck、Slither、Securify、Oyente、Osiris、Mythril、Manticore、MAIAN、HoneyBadger 这 9 个工具, 两个数据集一个包含 69 个带注释的漏洞合约, 另一个数据集则是 Etherscan 上获取到的 47 518 份合约. Praitheeshan 等人[25]和另外几项研究[26−32]则对智能合约存在的漏洞检测技术进行了调研, 并介绍和对比了部分智能合约漏洞检测技术的各种特性.

同时, 国内也有一些研究[13−15]对智能合约当前漏洞分类、检测技术现状与未来发展进行了调研. 付梦琳等人[13]和倪远东等人[14]分别于 2019 年和 2020 年调研分析了智能合约面临的安全威胁, 并讨论了主流的智能合约漏洞检测手段, 钱鹏等人[15]于 2020 年从形式化验证、符号执行、模糊测试、中间表示和深度学习这 5 类方法综述了智能合约漏洞检测技术的进展, 并对现有方法的可检测漏洞类型、准确率、时间消耗等方面进行了对比分析.

上述关于智能合约漏洞检测技术的研究存在发表年份过早[13,14]、检测技术过少[13,14]、检测技术信息分析不

全面[13−15]等问题, 与上述讨论的综述不同, 我们的调研包含了当前智能合约漏洞检测领域的大多数技术, 并对它们进行了系统性地分类, 同时总结了该领域的开源工具网页链接, 分析了现有技术的实现方法、漏洞类型、实验数据等信息, 并首次对比了国内外技术的研究现状, 最后讨论了智能合约漏洞检测技术当前面临的挑战以及未来可行的研究方向. 我们相信, 这项工作可以帮助研究人员从各方面对现存的漏洞检测技术有一个更为全面的认知.

2 智能合约漏洞检测方法综述

本节基于收集到的 84 篇智能合约漏洞检测技术论文, 依据其所采用的核心方法, 从已有的智能合约漏洞检测技术中提炼出以下 6 大类: 基于符号执行、基于模糊测试、基于污点分析、基于形式化验证、基于机器学习的漏洞检测技术和其他技术. 如果一种技术使用了多种方法, 我们根据文章中自己强调的核心方法对其进行分类.

2.1 基于符号执行的智能合约漏洞检测

基于符号执行的智能合约漏洞检测技术将合约程序的变量值抽象成符号输入, 运行过程中的所有变量到达目标代码时都会由符号相关的函数组成符号路径, 每个符号路径都有一个路径约束, 通过约束求解器分析相应的路径约束和约束关系, 从而检测出合约中的漏洞问题[49]. 2016 年, Luu 等人[50]提出的 Oyente 是第 1 个智能合约漏洞检测技术, 也是第 1 个提出利用符号执行技术进行智能合约漏洞检测. 该工具首先构造出合约的控制流图, 从控制流图的入口节点开始, 执行符号状态, 并利用 Z3 求解器循环获取要运行的状态, 产生对应的符号路径. 最后根据对 4 种漏洞定义好的一套属性来分析符号状态和符号路径, 从而检测智能合约中的安全漏洞, 并通过求得的约束进行可达性检验降低误报率. 分布式的区块链计算平台需要一定的计量标准来避免拒绝服务攻击和资源持续占用的交易, 以太坊采用 gas 机制作为计量标准, 节点在运行智能合约时, 如果 gas 不足, 智能合约会恢复到运行之前的状态, 该机制同样引进了一些以太坊平台独有的漏洞类型. 2017 年, Chen 等人[51]针对智能合约的高 gas 消耗问题, 在 Oyente 的基础上, 根据预先定义的 7 种导致额外成本的编程模式, 提出智能合约高 gas 消耗检查工具 Gasper.

2018 年, Tsankov 等人[52]提出了一种基于符号抽象的轻量级、可扩展合约分析工具 Securify, 可根据用户定义的安全属性检测合约的行为是否安全. Securify 从字节码开始符号化地分析代码中数据流和控制流依赖等信息, 提取出精确的语义事实, 并使用基于逻辑的编程语言 Datalog 描述语义事实. 之后, Securify 将其与预先定义好的安全属性规则进行检查, 其中安全属性规则分为合规 (compliance) 模式和违反 (violation) 模式, 通过语义事实与两种模式的匹配情况检测合约的安全性.

2018 年, Krupp 等人[53]提出了基于符号执行的漏洞检测与漏洞利用技术 teEther. teEther 先从字节码构造的控制流图中找出到达 4 种敏感指令的关键路径, 然后从控制流图根节点进行符号执行获取到达关键路径的路径约束, 最后, 使用 Z3 约束求解器解决关键路径和状态更改路径的组合约束来产生漏洞利用样例, 并检测出合约中的漏洞. teEther 工具同时实现了漏洞检测和漏洞的自动利用, 但其局限性在于只支持检测单个合约内的易受攻击漏洞, 并需要对这些漏洞预先建立通用定义, 而不能发现调用其他合约产生的漏洞.

2018 年, Nikolić等人[54]借助符号执行分析与依据具体路径值验证执行的方法, 提出了检测痕迹 (trace) 漏洞的 MAIAN. MAIAN 将智能合约划分为贪婪合约 (greedy contract)、浪费合约 (prodigal contract) 和自杀合约 (suicidal contract) 这 3 种类型, 并定义了痕迹特性的活性 (liveness) 与安全性 (safety), 再由符号执行引擎对智能合约字节码的所有路径进行符号分析, 检测是否存在违反痕迹特性活性或安全性的路径. 为了有效地降低漏洞检测结果的假阳性, MAIAN 还会根据求解出的路径条件具体值执行合约, 依据执行结果消除因路径不可达导致的部分假阳性案例.

2019 年, Mossberg 等人[55]实现了一款基于符号执行的动态二进制分析工具 Manticore. Manticore 将合约的执行过程抽象为: 就绪 (ready)、忙碌 (busy)、终止 (terminated) 这 3 个状态, 状态之间可以进行转换. 当符号变量需要被转换为一个或多个具体值时, 就会有一个相应的就绪状态被创建并处理, 在程序退出或发生异常时终止. 其通过符号执行枚举出合约的所有可到达执行状态以及触发这些路径的输出参数, 达到验证关键功能安全性的效果, 标记出整数溢出、未初始化内存等安全问题. Li 等人[56]实现的 Solar 就是基于 Manticore.

蜜罐合约是一种假装将其代币泄露给任意用户 (受害者), 用户向其发送额外代币时用户提供的代币将被困住的智能合约. 2019 年, Torres 等人[57]首次对蜜罐合约进行系统分析, 并提出了基于符号执行和启发式算法的蜜罐合约检测工具 HoneyBadger. 其主要由 3 部分组成: 符号分析、现金流分析和蜜罐分析. 符号分析基于 Oyente 构造出的控制流图, 由符号执行将程序变量的值表示为符号表达式, 并将分析结果传播到现金流分析组件和蜜罐分析组件; 现金流分析用于检测合约是否能够接收和转移资金; 而蜜罐分析则结合启发式算法和符号分析的结果来检测预定义的蜜罐问题. 这 3 步都使用 Z3 求解器来求解公式.

2020 年, He 等人[58]提出了第 1 个用于检测 EOSIO 平台上智能合约漏洞的系统分析框架 EOSAFE. 其先为 WASM 字节码实现一个符号执行引擎, 然后通过应用启发式引导剪枝方法来缓解路径爆炸问题; 其次, 为了分析 EOSIO 智能合约并模拟其外部交互环境, 实现了一个模拟器来模拟关键 EOSIO 库特征的行为; 最后, 提出了一个通用的漏洞检测框架, 并实现了 4 个扫描器, 旨在检测 4 个重要的漏洞, 包括虚假 EOS, 虚假收据, 回滚 (rollback) 和缺失权限检查 (missing permission check), 该框架同时允许安全分析人员将自己的漏洞扫描器实现为插件.

符号执行是目前智能合约漏洞检测的主要方法之一, 其通常先将合约中的变量值符号化, 然后在逐条解释执行程序中指令的过程中, 更新执行状态、搜集路径约束, 以完成程序中所有可执行路径的探索并发现相应的安全问题, 但也存在路径爆炸与约束求解难等问题[49].

2.2 基于模糊测试的智能合约漏洞检测

基于模糊测试的智能合约漏洞检测技术通过随机地构造出非预期的测试输入数据, 对智能合约进行大量测试来发现其中的漏洞. Jiang 等人[59]在 2018 年提出的 ContractFuzzer 是第 1 个采用模糊测试方法的智能合约漏洞检测技术. 其首先对智能合约的应用程序二进制接口 (application binary interface, ABI) 和字节码 (bytecode) 进行静态分析, 提取出 ABI 函数签名和参数数据类型; 同时也对以太坊中收集的智能合约做同样的静态分析; 然后根据分析结果和预定义的测试预言 (test oracle) 生成测试输入; 最后进行测试并分析 EVM 记录的合约执行过程中的指令调用、合约状态等信息来检测合约中的漏洞. 2020 年, Huang 等人[60]利用的类似的技术对 EOSIO 智能合约提出了黑盒模糊测试技术 EOSFuzzer. Ashraf 等人[61]提出的 GasFuzzer 是建立在 ContractFuzzer 的基础上, 专门面向 gas 异常的模糊测试检测.

由于以太坊智能合约在接收以太时会触发独特的回调 (fallback) 机制, 使得存在函数执行结束前再次进入被调函数的可能, 该操作可能导致合约执行危险操作, 此类因以太坊回调机制导致的函数被重复进入的漏洞称为重入漏洞 (re-entry). Liu 等人[62]针对智能合约中的重入漏洞提出了 ReGuard, 利用智能合约的抽象语法树 (abstract syntax tree, AST) 和控制流信息将合约翻译成行为一致的 C++合约, 再依赖成熟的模糊测试引擎生产随机的交易对翻译后的合约进行测试, 并根据有限状态机的机制分析测试过程中的实时运行轨迹来检测重入漏洞.

以太坊网络中每个区块都设定了 gas 上限, 如果交易花费的 gas 超过上限会导致交易失败, 这一问题被称为 out-of-gas 漏洞. Ma 等人[63]基于反馈指导的模糊测试技术, 提出了一种依据加权控制流图生成高 gas 消耗, 进而检测 out-of-gas 漏洞的技术 GasFuzz. GasFuzz 为传统控制流图中包含的每个节点赋予一个 gas 消耗权重, 定义为加权控制流图, 然后 GasFuzz 根据智能合约函数参数生成一个初始变异种子, 在变异策略的指导下从种子生成变异数据, 根据初始变异数据 EVM 运行时的 gas 消耗与遍历路径的信息反馈导向选择出新的种子, 并从新的种子生成新的变异数据, 直到用户设置的持续时间结束或用户想要停止该过程.

2019 年, Kolluri 等人[64]则针对智能合约中的事务顺序 bug (event-ordering bug) 提出了符号执行和模糊测试相结合的检测技术 EthRacer. 其先用符号执行技术分析合约的事件路径, 推理出每个事件的符号路径约束和先行发生关系 (happens-before relation), 并通过消除符号路径约束中的不可达路径减小路径约束的假阳性; 然后将分析的事件路径随机模糊出更多的事件路径组合, 通过测试这些事件路径来检测出事务顺序 bug.

2020 年, Nguyen 等人[65]针对智能合约生成高效测试用例的问题 (即高效地覆盖更多分支), 提出了基于反馈指导的适应性模糊测试技术 sFuzz. sFuzz 先从已有的合约交易信息中初始化初始的测试输入, 然后监控初始测试的执行过程, 并根据反馈信息将 AFL 的适应性函数 (fitness function) 和一个轻量级的多目标搜索策略相结合, 来

优化测试种子的生成, 从而提高测试智能合约的代码覆盖率和效率.

2020 年, Grieco 等人[66]从方便配置和使用、高覆盖率和高检测速度的角度出发, 根据检测合约是否会违背预先定义的用户自定义属性、断言和 gas 预估值, 提出了智能合约模糊测试工具 Echidna. 测试种子的生成与 ContractFuzzer 类似, 不同点在于测试预言是预先定义的用户自定义属性、断言和 gas 预估值.

Wüstholz 等人[67]针对智能合约提出了灰盒模糊测试技术 Harvey. 它属于定向的灰盒模糊测试技术, 先通过程序插桩技术捕捉更多的合约代码结构信息和逻辑覆盖信息, 并计算最优执行到不同程序点的距离, 该距离用于模糊测试时的种子优先度、能量调度和适应性变异, 预测出满足代码覆盖率高的测试种子, 以达到定向测试的目的.

2020 年, Zhang 等人[68]提出了一种采用污点分析方法来优化模糊测试种子生产的智能合约漏洞利用技术 EthPloit. EthPloit 先构造出 Solidity 源代码的控制流图, 依据污点分析得到源代码中变量数据与变量控制流间的依赖关系; 然后基于 SOLC 编译出的源代码字节码和 ABI 产生初步的测试用例; 接下来根据污点分析的依赖关系和上一轮模糊测试产生的反馈结果, 进一步优化模糊测试用例; 最后 EthPloit 运行测试用例并进行痕迹分析 (trace analysis).

尽管模糊测试存在测试用例生成低效、漏洞定位复杂等缺陷, 但因其高效多产的漏洞发掘能力应用广泛, 目前主要研究热点是通过符号执行、污点分析以及新兴的机器学习技术, 获取到更多程序运行时的数据依赖信息, 生成高度结构化的测试用例, 提高模糊测试的路径覆盖率与漏洞发掘的效率[64,68].

2.3 基于污点分析的智能合约漏洞检测

污点分析的一般流程为: 首先识别污点信息在智能合约中的产生点并对其进行标记; 然后按照实际需求和污点传播规则进行前向或后向数据依赖分析, 得到污点的数据依赖和被依赖关系的指令集合; 最终在一些关键的程序点检查关键的操作是否会受到污点信息的影响[69].

最早使用污点分析方法进行智能合约漏洞检测的是 Rodler 等人[70]于 2018 年提出的 Sereum, 一种在扩展的 EVM 客户端保护合约免受重入攻击的技术. Sereum 首先修改了 geth 的字节码解释器, 修改后的解释器会维护一个影子内存以存储与实际数据值分离的污点; 然后污点引擎会检测从存储读取到条件跳转指令中处理条件的任何数据流, 并对于每一次交易的执行, 记录控制流做决策的变量; 最后 Sereum 通过引入一组锁, 禁止进一步更新这些变量, 如果对合约的调用尝试更新这些变量之一, Sereum 会报告重入漏洞并中止交易.

2018 年, Torres 等人[71]提出了一种结合符号执行和污点分析用于检测以太坊智能合约中整数 bug (与整数错误相关的安全漏洞) 的框架 Osiris. Osiris 先基于 Oyente 的符号执行组件构造出合约的控制流图并执行路径信息, 这些信息被传递给污点分析组件和整数 bug 检测组件; 污点分析组件检查符号执行的每条指令是否是定义污点源列表的一部分, 并在堆栈、内存和存储之间引入和传播污点; 最终由整数错误检测组件检查在执行的指令中是否可能存在整数错误.

2020 年, Brent 等人[72]提出了一种利用污点分析的数据无害处理技术 Ethainter, 对智能合约中的数据信息流进行分析. 他们设计了一种用于捕获智能合约信息流语义的小型抽象输入语言: 拥有污点源和汇聚点, 通过抽象语言的各种操作进行变量间的数据传输、加载和存储到持久存储上, 最后根据预定义的信息流规则来捕获复合漏洞.

2020 年, Ashouri[73]则将污点分析和符号测试结合起来, 提出了一种能够识别智能合约漏洞并利用漏洞触发未知错误的技术 Etherolic, 依据污点分析的数据为测试合约中的易受攻击代码段生成触发漏洞的交易来发掘合约中的漏洞.

污点分析的核心是追踪系统中不可信数据或敏感数据是否进入污染汇聚点进而对程序造成危害, 是程序鲁棒性检验与漏洞挖掘的关键技术之一, 但其仍存在一定缺陷, 如污染传播过程中内存存储信息的处理, 信息流分析不全面造成的欠污染与过污染等问题, 因而在实际应用中, 其往往会结合符号执行、模糊测试等其他程序分析技术[69].

2.4 基于形式化验证的智能合约漏洞检测

形式化验证技术是一种验证程序是否符合预期设计的属性和规范的技术, 其使用严谨的数学语言和逻辑描述智能合约, 以便对其进行严格的数学推理和验证[74]. 形式化验证技术中, 对智能合约指令和逻辑形式化描述是验

证的前提, Bhargavan 等人[75]于 2016 年的工作最先尝试将以太坊指令形式化描述成具有交互式证明功能的函数编程语言 F*, 再基于此语言进行程序验证. 为了对智能合约指令进行形式化描述, 还有多个研究[76,77]致力于构建一个形式化描述的以太坊虚拟机.

对智能合约的形式化描述只是形式化验证的第 1 步, 一些研究工作已经开始关注在形式化描述的基础上, 对智能合约的安全性展开证明. Grishchenko 等人[78]在用 F*语言形式化描述以太坊指令的基础上, 定义了外部调用完整性、原子性、可变账户独立性和交易环境独立性 4 个通用安全属性并进行了证明.

2017 年, Grossman 等人[79]提出了一种检测有效回调自由合约的技术. 他们先是使用 SMAC 语言形式化描述了智能合约模型与该模型下的合约属性和行为, 然后在模型中定义的通用客户端对有效回调属性进行了分析, 并使用分析结果动态执行合约, 来验证合约中存在的有效回调属性. 而 Wang 等人[80]于 2019 年提出的 NPChecker 主要关注的是其构造的智能合约模型中不确定因素对合约代币转移的影响, 其主要贡献在于通过分析因不可预测的事务调度与外部被调用者的行为而造成的读写风险, 最终实现检测智能合约中的不确定性支付漏洞.

2018 年, Albert 等人[81]基于 Oyente 提出了以太坊字节码分析框架 EthIR, 他们将字节码反编译成了一种基于规则的表现形式 (rule-based representation, RBR), 以便在高层级上推理 EVM 字节码的属性. EthIR 在 Oyente 生成的控制流图上定义了一系列的转换规则, 将 EVM 字节码的高层级控制流图和数据流图重构成了 RMR, 再使用 SACO 分析器[82]推理了字节码的大量属性. 2019 年, Albert 等人又基于 EthIR, 通过推理智能合约上所有公共函数的 gas 使用上限, 实现了检测 out-of-gas 漏洞的 GasTap[83], 同年, Albert 等人提出的 SAFEVM[84], 将 EthIR 生成的 RMR 递归地翻译成带有 SV-COMP 格式验证注释的抽象整数 C 程序, 并使用 3 个 C 语言验证器 CPAchecker[85]、VeryMax[86]和 SeaHorn[87]验证了合约的安全性.

2018 年, Kalra 等人[88]提出了一个验证智能合约正确性 (correctness) 和公平性 (fairness) 的框架 Zeus, Zeus 结合了抽象解释和符号模型检验的方法对智能合约进行自动的形式化验证. 为了准确地编码字节码的执行语义以及正确地推理合约行为, Zeus 先借助污点分析构造出合约正确性与公平性的形式化规范, 再将其抽象解释到低层级的中间表示形式 (intermediate representation, IR), 然后通过静态分析 IR 的特性确定验证谓词需要被断言的点, 最后将插入断言的 IR 与受约束的霍恩子句作为模型验证引擎的输入, 验证合约的正确性与公平性.

2020 年, Permenev 等人[47]提出了第 1 个能验证智能合约函数特性的自动验证器 VerX, 为了解决验证智能合约时间属性的难题, VerX 将智能合约与要验证的时间属性转换成可达性检验属性, 通过分析用户、函数参数、函数的随机组合, 推理合约行为是否符合归纳的可达性检验属性, 并通过符号执行和谓词抽象组成的延迟谓词抽象技术, 扩大可达性检验属性的状态空间, 达到降低漏洞检测假阳性的效果.

2020 年, So 等人[89]通过分析智能合约中的交易不变量, 实现了一个针对算术安全的高准确率与高召回率验证器 VeriSmart. 其方法是先将程序分解成一系列包含验证条件的基本路径, 再检查当前基本路径中的候选不变量是否具有归纳性并且足够证明其查询的安全性, 然后将未证明安全性的不变量集作为反馈生成新的候选不变量集, 基于该反馈机制就形成一个迭代循环, 不断完善不变量集的分析结果, 直到证明程序是安全的或给定的时间预算用完为止.

2020 年, Frank 等人[90]针对现存智能合约漏洞检测技术缺少精确的内存模型与仅部分支持合约间分析的问题, 提出了基于符号执行的有界模型检验技术 EthBMC. EthBMC 针对智能合约分析过程中存在的哈希函数解析、内存操作与合约间调用难题, 构造了以太坊虚拟机的抽象状态机, 再由符号执行引擎搜索该抽象状态机下程序执行的状态空间. 当存在满足状态机停止状态 (halting state) 与实现攻击模型的路径时, 求解出满足该路径条件的具体输入, 并通过在私人以太坊网络上验证该漏洞的存在, 达到降低智能合约漏洞检测结果假阳性的效果.

2020 年, Schneidewind 等人[91]基于以太坊虚拟机字节码语义抽象解释出的霍恩子句, 提出了一个分析框架 HoRSt, 并在该框架上实现了分析工具 eThor. 他们先是基于霍恩子句对字节码进行抽象, 并在抽象解释出的霍恩子句层面形式化描述了智能合约的行为属性, 然后实现了一个基于霍恩子句的分析框架 HoRSt, 并依据霍恩子句的数学规范在 HoRSt 上实现了分析工具 eThor.

形式化验证通常在抽象层次上, 以形式化规范语言描述智能合约的行为和属性, 再运用严格的数学逻辑进行

推理, 以检测智能合约是否符合预期设计的功能要求, 是最严谨的智能合约安全验证技术, 但其自动化程度相对较低, 且无法动态分析, 缺乏了对漏洞检测结果的可达性检验, 会产生较高的误报率[13].

2.5 基于机器学习的智能合约漏洞检测

机器学习使用已有的数据让计算机模拟或实现类似人类的学习行为, 来不断优化计算机程序的性能, 在智能合约漏洞检测领域也有一定程度的应用. 2018 年, Tann 等人[92]提出了一种使用长短期记忆网络的机器学习方法对智能合约弱点进行顺序学习的技术. Tann 等人使用代码向量对智能合约操作码序列进行建模, 当给定一组智能合约操作码数据时, LSTM 模型会学习由嵌入算法初始化后的向量关系. LSTM 模型从 MAIAN 检测的结果中选出相同数量的分别标注为有漏洞合约与无漏洞合约作为训练数据集, 构建好的 LSTM 模型将会使用二元分类的方法将被检测智能合约标记为有漏洞与无漏洞两种类型. 2021 年, Hu 等人[93]同样训练了一个用于检测智能合约漏洞的 LSTM 模型, 不同的是, Hu 等人的数据预处理阶段是人工地从以太坊交易行为中提取出 4 种行为模式与 14 个基础特征, 并设计了一种数据切片技术解决数据集不足的问题.

2018 年, Liu 等人[94]提出了一种结合 N-gram 语言建模和轻量级静态语义标签的语义感知安全审计技术 S-gram. S-gram 的工作方式由模型构建阶段和安全审计阶段两个阶段组成, 在模型构建阶段, S-gram 基于 N-gram 的训练引擎, 使用从合约数据集中解析出的带有语义元数据标记的字符序列, 训练出 S-gram 的语言模型. 而在安全审计阶段, 检测器基于上一阶段训练的 S-gram 语言模型, 扫描审计目标合约的令牌序列以识别不规则的子序列作为候选漏洞. 类似地, 2021 年, Hu 等人[95]则使用 GRU 神经网络模型从智能合约字节码的 N-gram 特征中, 训练了一个检测智能合约是否存在漏洞的分类器.

2019 年, He 等人[96]为了解决模糊测试种子生成的低效难题, 提出了一种使用神经网络在符号执行的数据集上训练模糊测试策略的技术 ILF. ILF 通过使用路径覆盖引导的符号执行引擎提高了程序分支的覆盖率, 大量符号执行的结果将作为神经网络架构的训练数据集, 再由训练后的神经网络架构使用学习到的模糊策略为测试程序生成输入序列, 最后, 对该合约进行模糊测试并生成测试结果.

2020 年, Gao 等人提出了一种基于词嵌入和向量空间的技术 SmartEmbed[97], 可用于智能合约的漏洞检测和合约验证. SmartEmbed 先从 AST 中提取出合约的符号流, 再使用词嵌入算法将符号流映射到固定维度的向量, 通过堆叠单个向量可以得到合约代码的嵌入矩阵, 最后通过相似度阈值的设定提高漏洞检测的准确率. SmartEmbed 会将合约与数据库中的漏洞合约进行相似度比较, 然后依据检测结果生成漏洞检测报告. 与此类似, 2021 年, Huang 等人[98]与 Ashizawa 等人[99]也训练了一个用于漏洞代码相似度检测的机器学习模型, 不同的是, Huang 等人使用的合约向量表示形式是借助图嵌入算法从控制流图中获取, 而 Ashizawa 等人则是通过 PV-DM 模型[100]得到.

2020 年, Zhuang 等人[101]提出了一种使用图神经网络进行智能合约漏洞检测的技术. 主要由图生成阶段、图规范化阶段和消息传播网络 3 个阶段组成: 图生成阶段将智能合约函数公式化为合约图, 并为不同的程序元素分配不同的角色; 而图规范化阶段则通过节点消除算法来规范化不同合约生成的合约图, 以便图神经网络的训练; 消息传播网络阶段时间消息传播网络将按合约图中边的时间顺序依次沿边传递信息, 并使用一个聚合所有节点最终状态的读出函数为整个图计算一个标签, 得到合约漏洞检测结果.

2021 年, Eshghie 等人[102]提出了一个针对重入漏洞的框架 Dynamit, 不同于其他技术, Dynamit 不需要智能合约的代码, 只需要来自以太坊系统中的交易元数据和余额数据, 就可以训练出一个检测合约中重入漏洞的分类器. Dynamit 从交易信息中提取出机器学习模型需要的交易 gas 使用、合约余额差额、平均栈调用深度 3 种特征, 再使用 6 种机器学习算法训练出了 6 种分类器, 最后分析了各种机器学习算法下分类器的假阳性、假阴性、召回率和准确率等数据.

2021 年, Lutz 等人[103]针对当前基于机器学习的检测技术存在的可扩展性能不佳的现状, 提出了通用且可扩展框架 Escort, 其基于深度神经网络实现了多种漏洞类型的检测, 并支持轻量级的迁移学习用于检测未知漏洞. 深度神经网络训练阶段, 已标记漏洞类型的智能合约字节码经过编码层、全连接层、Dropout 层、GRU/LSTM 层等多层神经网络的处理, 最终在分支层实现了多个并行的神经网络层, 每个神经网络层都会训练出一种漏洞类型的

检测器. 迁移学习阶段, 则通过在原来的深度神经网络上添加新的分支层, 从新漏洞类型的样本中训练出新的漏洞检测器. 2021 年, Mi 等人[104]提出了同样基于深度神经网络的智能合约漏洞检测框架 VSCL, 相较 Escort, VSCL 少了一个用于检测未知漏洞的迁移学习阶段.

从 2018 年以来, 越来越多的机器学习算法应用于智能合约的漏洞检测, 与传统漏洞挖掘方法相比, 引入机器学习技术可以缓解传统方法耗费的大量人力与误报率和漏报率较高的问题, 也适应大规模复杂软件系统的应用场景, 但其对数据集和算法的依赖也带来了源码语义建模不足和检测结果可解释性较差等难题[15].

2.6 其他方法

对合约代码抽象语法树、控制流图或自定义中间表示形式的静态分析以及基于代码克隆检测、攻击向量、入侵检测系统等难以归类的方法被分类为其他. 2018 年, Tikhomirov 等人[105]对 Solidity 代码存在的漏洞进行了系统性分类, 并提出了一个可扩展的静态分析工具 SmartCheck, 允许用户使用自定义模板分析合约是否存在漏洞. SmartCheck 根据 Solidity 的语法规则使用 ANLTR[106]生成了源代码的 XML 解析树作为中间表示形式, 通过在 XML 解析树上使用 XPath 查询语句匹配符合模板的漏洞. Lu 等人[107]提出的 NeuCheck 就是在 SmartCheck 上的一种拓展. Quan 等人[108]的 EVulHunter 与 Argañaraz 等人[109]的 OpenBalthazar 与 SmartCheck 方法类似. EvulHunter 基于 EOSIO WASM 字节码的控制流图, 使用预定义的规则针对合约中存在的虚假转账漏洞, 而 OpenBalthazar 则是将预定义的规则应用到 AST 上检测.

2019 年, Nguyen 等人[110]基于以太坊上交易被执行时的异常行为, 提出了检测工具 ABBE. ABBE 为 7 种漏洞类型分别定义了攻击向量, 同时根据攻击向量的执行轨迹定义了每种攻击向量对应的属性, 最后将获取到的 AST、合约地址、交易日志等信息与预定义的攻击向量规则匹配, 通过检测每笔交易中是否存在异常行为, 判断合约是否存在漏洞.

2019 年, Wang 等人[111]将入侵检测系统嵌入到智能合约中, 用来分析上下文标记的非循环路径, 并采用 gas 导向的性能模型进行优化, 实现了第 1 个保护以太坊智能合约的入侵检测系统 ContractGuard. ContractGuard 先从优秀测试用例的执行轨迹中提取出上下文标记的非循环路径作为入侵检测系统的初始安全路径集, 当配置了入侵检测系统的智能合约接收到一笔交易时, 会分析该交易中的循环路径, 并检验它们是否存在于安全路径集中, 如果不存在则由管理员验证该交易是否存在漏洞, 当不存在漏洞时, 再将该交易的非循环路径加入安全路径集中进一步完善入侵检测系统.

2019 年, Li 等人提出 MuSC[112], 其实现思想是在 AST 上执行高效精确的变异操作, 再由依据用户定义创建出的测试网络对智能合约进行自动化测试. MuSC 先将智能合约的源代码文件转换为 AST 版本, 并对 AST 上的数据进行突变, 最后由变异 AST 反编译出智能合约, 并再测试网络上对该智能合约进行编译、执行和测试. 同年, Akca 等人[113]提出的 SolAnalyser 也是基于变异 AST 完成智能合约的漏洞检测, SolAnalyser 是先在 AST 上生成相关的断言 (assert) 语句, 再使用由 ABI 和字节码信息生成的输入数据, 对带有断言的智能合约进行测试.

2019 年, Xue 等人[114]使用智能合约的抽象漏洞签名进行代码克隆检测, 提出了一种智能合约漏洞检测技术 Doublade, 并通过应用带有防御机制的规则消除了抽象漏洞签名提取过程中带来的假阳性问题. 为了得到假阳性样本更少的漏洞代码数据集, Doublade 在 Slither、Oyente 和 SmartCheck 的检测结果中添加了各种防御机制, 再经过 AST 噪声处理、代码块分层聚类、代码块共性分析等步骤, 提取出代码块的抽象漏洞签名, 最后, 通过包含抽象漏洞特征的 AST 生成控制流图, 并在控制流图上借助基于最长公共序列和子序列检查的代码匹配算法, 检测合约是否包含漏洞. 2020 年, Ye 等人[115]也提出了一种基于代码克隆检测的智能合约漏洞检测技术, 差别在于, Ye 等人从智能合约的 AST、程序依赖图和中间表示形式中提取出了用于代码克隆检测的 3 种特征, 并综合 3 种特征的匹配结果得出最终的代码相似度.

2019 年, Feist 等人[116]提出了名为 Slither 的框架, 用户可以基于该框架实现程序分析技术, 多种技术[66,68,117]都是基于 Slither 实现. Slither 先从 AST 中恢复合约的继承图、控制流图和表达式列表等重要信息, 再分析出合约代码中存在的变量读写、函数保护、变量数据依赖等信息, 最后使用该框架的合约分析程序可以通过 API 将这些信

息用于不同的用途, Slither 框架自身已经实现了一些漏洞检测、代码优化和代码审计的工具.

2020 年, Chen 等人[118]鉴于当前技术无法在线运行和难以扩展新型漏洞的现况, 提出了一种通用的在线检测框架 SODA, 用于任何支持 EVM 智能合约平台的漏洞检测. 被集成到全节点中的 SODA 从合约层、共识层和数据层分别收集区块、交易和被执行 EVM 指令的原始信息, 并将其抽象成了 11 种结构化信息, 基于该框架的应用程序都可以从队列中获取到原始信息和抽象出的结构化信息. 同时, Chen 等人基于 SODA 框架开发了 8 种检测智能合约漏洞的应用程序, 并在以太坊、Expanse、Wanchain 这 3 个智能合约平台进行了大量实验. 基于类似思想, Wu 等人[119]提出了借助重放以太坊上交易来分析智能合约的框架 EthScope, 基于数据聚合器收集到的信息, 其可以重放以太坊上的任意交易, 同时允许用户使用给定的 API 动态检测合约并分析执行过程中的异常行为.

中间表示形式的静态分析, 以及传统软件分析和安全领域迁移到智能合约漏洞检测领域的入侵检测机制、攻击向量、代码克隆检测等方法, 保障了智能合约生态安全发展的同时, 也带来了诸如运行时内存信息缺失、并发检测不足和未知类型漏洞诊断能力较差等缺陷[105,111,114,115].

3 数据分析与国内外对比

本节依据前文实现方法的分类, 统计并分析了每篇文章中技术的实现方法、漏洞类型、实验数据等信息, 而 Mythril[120]、VaaS[121]等公司开发的技术, 因不存在详细介绍的文章导致大部分数据缺失, 并未在分析范围之内. 同时, 我们还依据每篇论文作者单位的国别分类对国内外智能合约漏洞检测技术进行了比较, 从技术发展趋势、实验漏洞范围以及实验数据分析方面对比分析了国内外在智能合约漏洞检测研究上的差异, 供国内相关研究人员参考, 以期弥补国内在该研究领域的不足之处, 进一步推动国内在智能合约漏洞检测研究领域的研究与发展.

3.1 技术发展趋势

遵循第 2 节对智能合约漏洞检测技术中采用的方法分类, 我们定义了符号执行、模糊测试、污点分析、形式化验证、机器学习以及其他 6 种方法标签, 每篇文章在实现文中技术时可能会采用一种或多种方法, 我们为论文集中的每篇文章都标记上了其涉及的方法标签, 当一种漏洞检测技术使用到多种方法时, 则其涉及的每种方法频数都加一. 为了进一步分析各种方法在智能合约漏洞检测技术发展历史中的应用状况, 我们把论文发表年份作为图表的横坐标, 统计结果如图 2 所示. 一种方法在智能合约漏洞检测技术中可能会以主要方法或辅助方法的形式出现, 如在 ILF[96]中, 主要方法是模糊测试, 而机器学习与符号执行则是辅助方法, 我们又分别统计了每种方法作为主要方法与辅助方法的分布, 结果如图 3 所示.

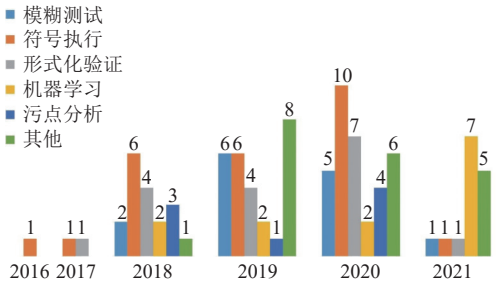

图 2 6 种方法的年份分布图

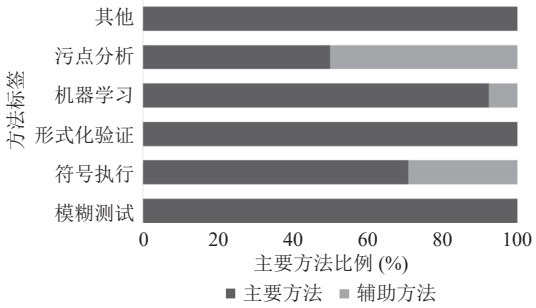

图 3 主要方法与辅助方法的分布图

从图 2 和图 3 可以看出, 自智能合约漏洞检测领域发展以来, 符号执行与形式化验证便作为主流检测方法沿用至今, 同时符号执行因其高效的路径覆盖率, 也可用于辅助生成测试用例、模型状态空间搜索、可达性检验等, 达到提高路径覆盖率、降低假阳性等效果, 符号执行被作为多种检测技术[47,65,73,92,99]的辅助方法. 模糊测试自

2018 年的 ContractFuzzer[59]与 ReGuard[62]发表以来, 也被广泛应用于智能合约漏洞检测领域, 包括检测重入漏洞的 GasFuzz[63]、事务顺序漏洞的 EthRacer[64], 针对 EOSIO 的 EOSFuzzer[60]等, 也逐渐引入了其他辅助方法[64,68,96]来提高模糊测试的效率. 相较于前 3 种技术, 污点分析在智能合约漏洞检测领域的应用则并不普遍, 更多时候作为一种辅助方法[68,80,88,119]来弥补其他技术的缺陷, 同时也有一批将污点分析作为主要方法[70−73]的漏洞检测技术. 机器学习近些年来快速发展, 作为一种新兴的可用于程序分析的方法, 其在智能合约漏洞检测领域也得到广泛应用, 2021 年采用机器学习方法的技术更是急速增长, 多种深度神经网络模型、分类算法等机器学习子方法应用到了智能合约漏洞检测的场景中. 其他分类的方法包含了静态分析, 以及其他传统安全领域与程序分析领域难以归类的程序分析技术, 如入侵检测系统、代码克隆检测、变异测试等, 近些年来其他分类的技术也逐渐增多, 传统安全与程序分析领域的技术也逐步迁移到了智能合约漏洞检测领域. 6 种方法中, 污点分析最常作为辅助方法提升漏洞检测技术性能; 符号执行因其高路径覆盖率的优势, 也常用来弥补其他方法自身缺陷, 达到提高覆盖率和降低假阳性等效果; ILF[96]则另辟蹊径, 利用机器学习辅导模糊测试的策略生成.

我们依据每篇文章作者隶属单位的国别进行国内外研究的划分: 如果一篇论文的所有作者单位都在国内 (包括港澳台地区), 这篇论文的研究工作直接归类为国内研究 (共 18 篇); 如果一篇论文是由国内外的不同研究机构的研究工作者合作完成时, 则依据该文第一作者隶属单位的国别进行划分, 即如果该文第一作者隶属单位在国内, 这篇文章将会归类为国内研究 (共 9 篇[59,62,63,68,107,112,118,119,122]). 需要注意的是, 当一篇研究论文由国内外的研究者共同完成时, 难以十分准确地将该研究工作划分为国内或国外, 一般情况下, 研究论文的第一作者是该研究工作的主要完成者和核心技术完成者, 如果该作者隶属单位在国内, 我们则认为我国掌握了该项研究技术. 同时这 9 篇文章的数据也佐证了这一点, 这 9 篇划分到国内研究的合作文章中国外研究作者仅有一到两位且作者顺序靠后. 其他情况则全部归类为国外研究 (共 57 篇, 其中 3 篇[123−125]是国内外合作研究, 而国内研究者也是仅有一到两位且作者顺序位于末位). 为了展现国内外技术在采用方法方面的差异, 我们依据前述归类方法对国内外技术采用的主要方法进行了分类和对比分析, 如图 4 所示, 图 5 则展示了国内技术采用的主要方法年份分布图. 可以发现, 国内技术更集中在模糊测试、机器学习以及其他方法的应用上, 这 3 种方法相较国外并没有明显差异. 而作为智能合约漏洞检测技术主流方法的符号执行与形式化验证, 国内研究相比国外则呈现不足, 其中 2017 年采用符号执行的 Gasper[51]是基于 2016 年 Luu 等人提出的 Oyente, 2020 年发表的 4 篇[58,122,126,127]采用符号执行方法的文章中, 3 篇是发表在未经同行评审的 arXiv 上, 但也弥补了近些年国内基于符号执行研究的不足, 而基于形式化验证的检测技术, 国内的成都链安科技公司研发了自动化形式验证平台 VaaS[121], 但论文集中国内的研究仍旧空白. 同时以污点分析作为主要方法的智能合约漏洞检测技术, 国内外采用情况都偏低, 国内的 EthPloit[68]、EthScope[119]采用污点分析作为辅助方法, 在辅助方法的应用上与国外没有较大差异.

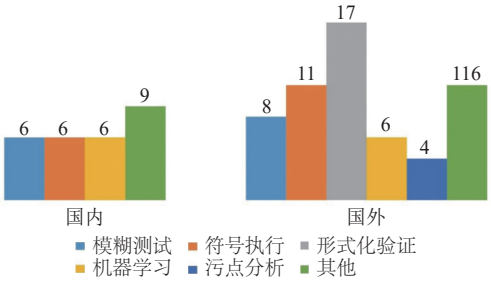

图 4 国内外主要方法分布图

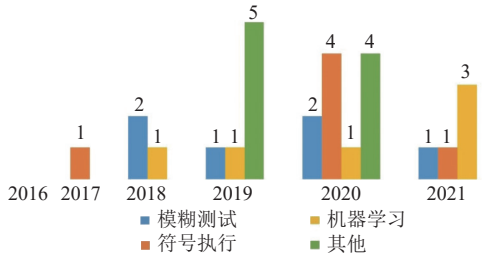

图 5 国内主要方法年份分布图

本节我们统计分析了前文的 6 种方法分类在智能合约漏洞检测领域的年份分布情况, 并参考检测技术采用主要方法的分布数据, 对比了国内外技术在采用方法上的差异. 在智能合约漏洞检测技术采用方法的分布中, 符号执行与形式化验证一直是主流方法, 其中符号执行也常作为辅助方法用于辅助测试用例生成、可达性检验等, 而目前国内技术在这两种方法的应用上并不充分, 其中形式化验证方法在论文集中的研究依旧空白, 工业界则有成都链安科技公司研发的 VaaS; 模糊测试因其漏洞发掘的高效性, 在智能合约漏洞发掘领域也应用广泛, 国内技术在

这一方法的研究进展平稳; 采用新兴的机器学习和传统的软件分析等其他方法的技术近些年来也逐渐迁移到智能合约漏洞检测领域, 国内技术目前在这两种方法的应用上也相对前沿; 而污点分析在智能合约漏洞检测领域的应用较其他 5 种方法并不充分, 但其常作为辅助方法弥补其他方法的不足.

3.2 漏洞涵盖范围

智能合约存在关于整数、算术操作、异常等类似传统应用程序的漏洞, 因其特性也存在与重入、gas、代币等相关的特色漏洞, 我们依据检测技术涵盖的漏洞类型, 将其分为通用漏洞检测技术和专用漏洞检测技术, 其中通用漏洞检测技术指涵盖漏洞类型不止一类的技术, 而专用漏洞检测技术则是仅检测一种漏洞类型的技术, 如检测重入漏洞[128−130]的 ReGuard[62], gas 相关漏洞的 GasTap[83], 算数操作相关漏洞[123,131]的 VeriSmart[89]等.

根据上文对漏洞检测技术的划分, 我们统计了论文集中技术的漏洞涵盖类型分布, 如图 6 所示, 我们将专用漏洞检测技术又分成了两类, 一类是针对智能合约特性的专用漏洞检测技术, 包含 gas 相关漏洞、重入漏洞、代币转移相关漏洞, 另一类是针对传统漏洞类型的其他专用漏洞检测技术, 如整数漏洞、算术操作相关漏洞等. 可以看到, 在当前智能合约漏洞检测技术的涵盖漏洞层面, 通用漏洞检测技术占据多数, 约为专用技术的两倍, 在专用漏洞检测技术中, 因智能合约实现机制与其代币在现实中存在一定经济价值的特性, 检测智能合约特性的专用漏洞检测技术, 相对其他专用漏洞检测技术明显较多. 智能合约面临的严峻安全问题引起了相关研究人员的关注, 因其特性产生的诸如重入、gas 和代币相关漏洞也成为当前智能合约安全研究的重点.

自以太坊将智能合约这一概念实现之后, 又出现了 EOS、Fabric 等一批智能合约平台, 不同平台因底层特性与虚拟机的不同, 检测方法并不统一, 论文集中的技术涵盖了以太坊、EOS、FISCO BCOS 等平台, 其中 Zhuang 等人[101]与 Liu 等人[124]提出的漏洞检测技术不仅支持以太坊, 这两种技术还分别兼容了 VNT Chain 和 FISCO BCOS, SODA[118]则支持任何使用 EVM 的平台, 而 Seraph[127]除了 EVM 外还支持任何使用 WASM 的平台, 我们将这类支持多平台漏洞检测的技术归类在跨平台中.

图 7 展示了各种技术检测平台的数据, 以太坊作为最受欢迎的智能合约平台, 在智能合约漏洞检测领域依然热门, 与此同时, 针对以太坊之外智能合约平台的漏洞检测研究也成为未来研究趋势, EOS、Fabric、Tezos 等平台的漏洞检测吸引了一批研究人员, 针对漏洞检测技术仅支持单一智能合约平台的局限性, 近些年来, SODA、Seraph 等跨平台漏洞检测技术也逐渐成为智能合约漏洞检测领域研究的新方向.

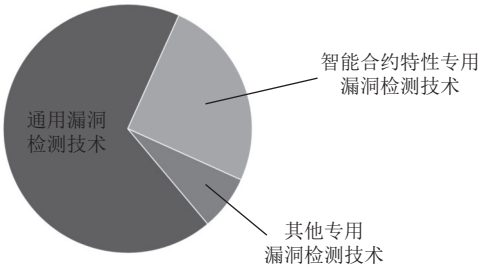

图 6 基于涵盖漏洞类型的技术分类图

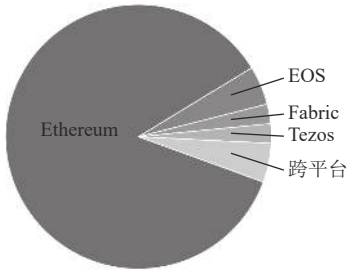

图 7 检测平台的分布图

图 8、图 9 展示了国内外漏洞检测技术在漏洞类型涵盖与检测平台两方面的数据, 可以看到, 国内外技术在漏洞类型分布上整体差距并不大, 通用漏洞检测技术的占比都在 70% 左右, 而在专用漏洞检测技术方面, 国内将研究重心放在了智能合约特性漏洞的研究上, 而国外则同时兼顾了整数操作符等传统漏洞类型. 在智能合约平台的检测层面, 智能合约底层技术的更新迭代与多样化的应用需求, 催生了大量的智能合约平台, 以太坊平台之外的漏洞检测与跨平台漏洞检测技术也都成了新的研究方向, 国内关于以太坊平台智能合约安全问题的研究虽然起步较晚, 但在 EOS、跨平台技术的研究上差异已经较小.

本节我们统计分析了国内外技术在漏洞类型、检测平台两种标准下的分布状况. 在覆盖漏洞类型方面, 我们划分了通用漏洞检测技术以及专用漏洞检测技术, 其中专用漏洞检测技术又包含了智能合约特性与其他两种, 国内外在通用技术和专用技术两方面研究的分布则没有太大差异, 以通用技术的研究为主, 在专用技术中, 智能合约

特性相关的漏洞在国内比传统应用程序漏洞引起了更大的关注; 在检测平台方面, 以太坊平台依然是目前的研究热门, 但其他平台的漏洞检测和跨平台技术的研究也成为当下新的趋势, 国内虽然在以太坊平台的漏洞检测起步较晚, 但近年来在以太坊之外的 EOS、跨平台检测技术的研究方面则相对前沿.

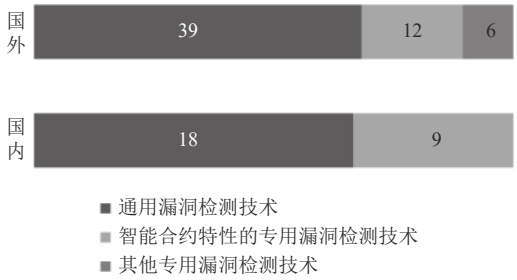

图 8 基于漏洞类型的技术分类分布图

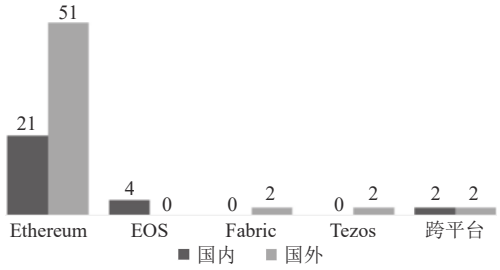

图 9 检测平台的分布图

3.3 实验数据分析

在评判智能合约漏洞检测技术的准确率、误报率、运行效率等性能时, 会使用一定数量的智能合约作为实验数据集, 一些智能合约漏洞检测技术还会存在与其他技术的对比实验, 同时在开/闭源 (开源是指研究人员将其研究工作的智能合约漏洞检测技术工具的源代码或二进制码公开在了相关开源平台上, 如 GitHub; 闭源则是指没有公开任何关于漏洞检测工具的代码信息) 的选择上, 不同技术的选择也不相同. 本节统计分析了论文集中技术的开/闭源状况、实验数据集差异与对比技术的信息, 以及国外技术在这些方面的差异.

论文集中的智能合约漏洞检测技术在源码是否开源的抉择上存在一定程度差异, 一项开源的技术既可以被测试人员用来检测其开发的智能合约是否存在安全隐患, 也可以被其他研究者优化或参与其开发智能合约漏洞检测技术的对比实验, 论文集中还有一些技术源码并未开源, 但提供了使用该技术的网页界面, 智能合约开发人员可以在该页面检测智能合约的安全性能, 我们将这两种情况都归类在开源技术中. 我们收集了论文中提供代码或使用界面的网页链接, 一些在论文中并未找到相关链接的论文, 我们分别以论文名与技术名在 GitHub 中作为关键词查找, 最后统计出共有闭源技术 41 个, 其中国内 20 个, 国外 21 个, 开源技术 43 个, 国内 7 个, 国外 36 个, 其中部分技术[47,72,83,132,133]并未开放源码, 但提供了网页调用技术进行合约的安全性检测, 最终开源技术的网页链接如表 1 所示.

表 1 智能合约漏洞检测开源技术信息汇总

序号	主要文献	网页地址
1	Brent等人[8]	https://github.com/usyd-blockchain/vandal
2	Permenev等人[47]	https://verx.ch/
3	Luu等人[50]	https://github.com/melonproject/oyente
4	Tsankov等人[52]	https://github.com/eth-sri/securify2
5	Krupp等人[53]	https://github.com/nescio007/teether
6	Nikolić等人[54]	https://github.com/ivicanikolicsg/MAIAN
7	Mossberg等人[55]	https://github.com/trailofbits/manticore
8	Torres等人[57]	https://github.com/christoftorres/HoneyBadger
9	Jiang等人[59]	https://github.com/gongbell/ContractFuzzer
10	Huang等人[60]	https://github.com/gongbell/EOSFuzzer
11	Kolluri等人[64]	https://github.com/ashgeek/Ethracer
12	Grieco等人[66]	https://github.com/crytic/echidna
13	Torres等人[71]	https://github.com/christoftorres/Osiris
14	Brent等人[72]	https://contract-library.com/

表 1 智能合约漏洞检测开源技术信息汇总 (续)

序号	主要文献	网页地址
15	Grossman等人[79]	https://github.com/shellygr/ECFChecker
16	Albert等人[81]	https://github.com/costa-group/ethIR
17	Albert等人[83]	https://costa.fdi.ucm.es/gastap/
18	So等人[89]	https://github.com/kupl/VeriSmart-public
19	Frank等人[90]	https://github.com/RUB-SysSec/EthBMC
20	Tann等人[92]	https://github.com/wesleyjtann/Safe-SmartContracts
21	He等人[96]	https://github.com/eth-sri/ilf
22	Gao等人[97]	https://github.com/beyondacm/SmartEmbed
23	Ashizawa等人[99]	https://github.com/fseclab-osaka/eth2vec
24	Tikhomirov等人[105]	https://github.com/smartdec/smartcheck
25	Lu等人[107]	https://github.com/Northeastern-University-Blockchain/NeuCheck
26	Quan等人[108]	https://github.com/EVulHunter/EVulHunter
27	Li等人[112]	https://github.com/belikout/MuSC-Tool-Demo-repo
28	Akca等人[113]	https://github.com/sefaakca/RandomInputGenerator https://github.com/sefaakca/MuContract
29	Feist等人[116]	https://github.com/crytic/slither
30	Zhang等人[117]	https://github.com/QuanZhang-William/M-Pro
31	Chen等人[118]	https://github.com/pandabox-dev/SODA
32	Wang等人[126]	https://github.com/gongbell/wana
33	Liu等人[124]	https://sites.google.com/view/modcon
34	Honig等人[138]	https://github.com/JoranHonig/vertigo
35	Hajdu等人[139]	https://github.com/SRI-CSL/solidity
36	Chen等人[125]	https://github.com/Jiachi-Chen/DefectChecker
37	Chinen等人[132]	https://github.com/wanidon/RA
38	Reis等人[133]	https://gitlab.com/releaselab/fresco
39	Torres等人[140]	https://github.com/christoftorres/Horus
40	Barboni等人[141]	http://pros.unicam.it/sumo/
41	Grech等人[142]	https://github.com/nevillegrech/MadMax
42	Samreen等人[143]	https://smartscan.cs.ryerson.ca:4638/
43	Nishida等人[144]	https://www.fos.kuis.kyoto-u.ac.jp/trylang/Helmholtz

图 10 展示了国内外技术在开/闭源方面的分布现状, 论文集中的漏洞检测技术在开放性上分布的差距不大, 但国内外技术间的差异比较明显, 国内开/闭源技术的在国内技术占比分别为 26%、74%, 而国外开/闭源技术的占比则分别为 63%、37%, 在开源技术方面, 国内技术明显低于该领域的平均水平, 且远低于国外技术开源的百分比, 国内技术更倾向于选择闭源.

智能合约漏洞检测技术是保障智能合约生态安全发展的屏障, 但目前漏洞检测领域并没有统一的实验数据集或标准, 因而论文集中的技术在衡量性能时采用的数据集各不相同, 我们依据数据集来源将其分为案例分析与平台检测两种类型. 因智能合约平台允许任何人部署与接入, 使得其上存在的智能合约良莠不齐, 一些技术的实验仅检测少量优质样例合约并进行人工分析, 我们将其归类为案例分析[134−136], 如 Harvey 就是在 27 个真实世界的智能合约进行的评估, Liu 等人则在微众银行的 CMA 合约与 FSolidM 的 BlindAuction 合约上评估 ModCon[124]. 由于智能合约去中心化的优势, 智能合约字节码可以接入网络直接获得, 而源代码则可以在 Etherscan 上获取, 另一类技术则直接测试部署在智能合约平台的字节码或在 Etherscan 上发布源代码的智能合约集, 再在实验过程中去除掉其中会引起超时、异常等错误的部分合约, 我们则将其归类为平台检测[50−52,137], 如 Oyente[50]将以太坊上截止到 2016 年 5 月 5 日的所有智能合约字节码作为实验数据集. 最终实验数据集的分布如图 11 所示.

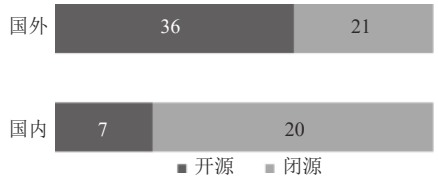

图 10 国内外开/闭源信息分布图

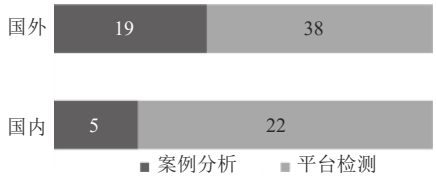

图 11 国内外实验数据分布图

图 11 数据表明, 智能合约漏洞检测技术整体上更倾向于采用平台检测这种方案, 且国内技术在数据集选择上平台检测的采用率更高. 两种分类的实验数据集各有优劣, 由于智能合约平台开放、不可篡改的特性, 使得平台上充斥了大量项目开发前期功能尚不完善的合约或初学者用于相同学习目的的重复合约等冗余实验数据, 案例分析分类使用少量人工筛选出的符合预期测试目的的优质合约[67], 避免了工具在实验时陷入大量冗余数据的测试中, 但因人的精力受限, 其在筛选过程中不可避免地会遗漏一些典型的漏洞案例, 此时通过平台检测的方式则可以有效地涵盖包含各种漏洞类型的合约, 但也存在冗余合约降低工具运行效率的问题[88]. 目前智能合约漏洞检测技术的数据集并没有统一标准, 也是智能合约漏洞检测领域亟待解决的问题之一.

除了数据集的选择, 部分论文[68,89,105]还会对技术性能进行对比实验, 我们统计了论文集中技术是否存在对比实验的数据, 如图 12 所示, 同时, 我们也统计了参与到论文中对比实验的智能合约漏洞检测技术的频数, 如果一篇论文用到了一个或多个技术进行对比实验, 则这些参与对比实验技术的频数都会加一, 最终结果如图 13 所示.

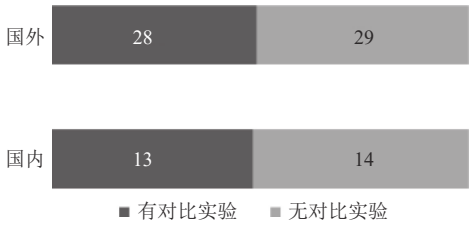

图 12 国内外对比实验分布图

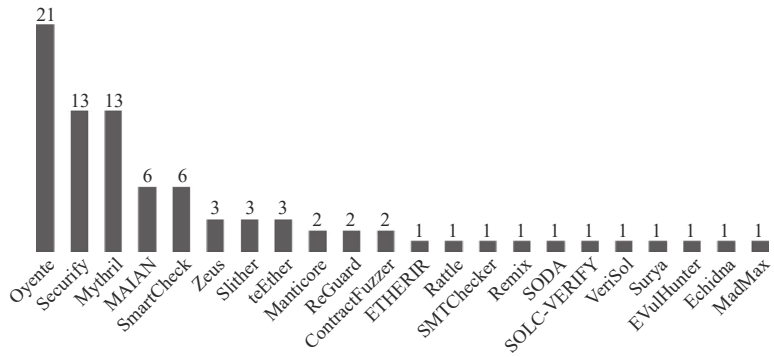

图 13 参与对比实验的技术分布图

可以看到, 论文集中有无对比实验的技术数量相近, 且国内外技术在有无对比实验方面并无较大差异, 同时参与对比实验的技术大多数仍是论文集中的技术, 大部分不在论文集中的社区技术 (Surya、Remix、VeriSol、Rattle、SMTChecker) 频数只有一次, 仅有社区技术 Mythril 虽不在论文集中, 但其参与对比试验的频数高达 13. 以第 2 节划分的 6 种方法分类为依据, 参与对比实验的技术主要以采用符号执行 (Mythril、teEther 等)、其他 (SmartCheck、Slither) 方法的技术为主, 采用模糊测试、机器学习等方法的技术则很少被用到其他工具的对比实验中. 由于目前国内研究智能合约在符号执行方面的不足以及国内论文集中工具开源率偏低等原因, 使得对比实验中常用到的技

术以国外为主, 国内仅有 4 篇论文[59,62,108,118]在 6 篇论文的对比实验中被引用过一次或两次.

本节分析了论文集中技术的工具开/闭源、实验数据集、对比实验等数据的分布, 并总结了国内外研究间的异同. 目前智能合约漏洞检测领域有近一半技术选择工具开源, 而国内研究在工具开源的比例则远低于国外; 我们又将论文实验中采用的数据集分为了案例分析和平台检测两种分类, 案例分析中的数据集是人工挑选出的优质智能合约, 但其数据集相对较小在一些典型案例上可能存在疏漏, 而平台检测则是对平台上的智能合约进行全面性的安全检测, 但其上存在大量不成熟与冗余的合约, 一定程度上降低了平台检测的效率; 同时, 对比实验在论文中广泛存在, 我们统计了参与对比实验技术的频数, 发现符号执行和其他分类标准下的技术被广泛应用在对比实验中, 由于国内符号执行的研究不足、智能合约安全研究起步晚、工具开源比例低等原因, 使得国内技术鲜有参与到其他论文的对比实验中, 仅有 4 项技术在 6 篇论文的对比实验中被分别使用到了一次或两次.

3.4 国内研究的不足之处与改进方向

第 3.1–3.3 节从技术发展趋势、漏洞涵盖范围与实验数据分析 3 个方向分析了当前智能合约漏洞检测研究工作的数据, 并基于此对比了国内外研究工作的异同, 其中国内外研究工作在采用方法、工具开/闭源和对比实验方面存在较多差异, 基于此我们总结了国内研究工作的不足之处与改进方向, 望后续研究人员的研究在促进国内该领域研究发展的同时更有方向性, 弥补国内研究工作的空白.

国内研究工作在漏洞类型涵盖范围与数据集检测方案的抉择上较国外没有太大差异, 其不足之处主要在于采用方法分布不均衡以及技术开放较少, 从而导致国内研究工作在其他技术的对比实验中较为罕见. 国内研究工作采用的方法更集中在模糊测试、机器学习与其他 3 类, 而作为智能合约漏洞检测技术主流方法的符号执行与形式化验证, 国内研究的采用情况则不太理想, 仅有 2017 年的符号执行技术 Gasper[51]与 2020 年发表的 4 篇[58,122,126,127]符号执行文章, 其中 Gasper 是基于 2016 年 Luu 等人提出的 Oyente, 而 2020 年发表文章的中有 3 篇都是发表在未经同行评审的 arXiv 上, 在采用形式化验证方法的检测技术方面, 国内的成都链安科技公司研发了自动化形式验证平台 VaaS[121], 但论文集中并不存在国内的相关研究; 同时在工具开源方面, 国内研究工作开源相对较少, 更倾向于闭源, 该状况并不利于该领域的进一步发展; 与此同时, 符号执行和其他分类标准下的技术被广泛应用在对比实验中, 鉴于国内符号执行的应用不足与技术开放比例低的现状, 国内技术较少参与到其他研究工作的对比实验中, 仅有 4 项技术在 6 篇论文的对比实验中被分别使用到了一次或两次.

目前学术界颇为关注的形式化验证和符号执行安全有效但难度很高, 另外国内智能合约漏洞检测研究起步较晚, 这共同造成了形式化验证和符号执行在国内智能合约漏洞检测技术中的应用相对较少, 希望国内符号执行与形式化验证领域的研究人员将相关研究工作与智能合约漏洞检测结合, 以推动国内该领域研究的全面发展. 同时也希望国内研究人员能够公开相关研究技术的工具代码 (具有保密协议的例外), 一项开源的技术不仅可以协助其他研究人员更快地了解该领域, 还对他们的工作有一定的指导启发意义, 并且在技术开源的过程中, 研究者也可以基于社区的反馈情况对开源技术进行进一步的优化. 此后, 随着国内技术采用方法的均衡发展与开放状况的逐渐改善, 国内技术在对比实验中的出现频率会有一定程度的提升.

4 讨论与展望

4.1 现有研究的不足

如前所述, 近些年来涌现了一批智能合约漏洞检测技术, 能够较为准确地检测智能合约上存在的各种漏洞, 降低了智能合约中潜在的安全隐患, 但其还处于发展前期, 存在一定程度的缺陷, 面临着以下待解决的问题.

(1) 多样化的智能合约平台. 随着智能合约底层技术的更新迭代与其在不同应用领域的多样需求, 继以太坊之后, 涌现了 Fabric、EOSIO 等智能合约平台, 不同平台在虚拟机实现、底层特性、智能合约机制等方面的差异, 给智能合约的漏洞检测带来了新的难题, 而当前智能合约漏洞检测技术则过于集中在以太坊一种平台. 因此, 以太坊之外平台与跨平台的漏洞检测也需要进一步的研究.

(2) 多样化的智能合约编程语言. 现实世界中的编程语言种类繁多且更迭迅速, 目前仅支持编写智能合约的高

级编程语言就有几十种 (如 Solidity、Vyper、Michelson), 不同编程语言在语法语义规则和编译成字节码过程中都存在差异. 智能合约漏洞检测技术如何适配这些高级编程语言特性, 也是一项挑战性的研究.

(3) 漏洞检测方法的局限性. 目前大多数技术依赖的模糊测试、符号执行、形式化验证等漏洞检测方法自身存在一定的局限性. 模糊测试在测试数据生成的过程中会存在路径覆盖率低、测试数据效率低下等缺点; 而符号执行在得到高路径覆盖率的同时, 又引入了路径爆炸与约束求解难等问题; 形式化验证虽然基于严谨的数学推导, 但其自动化程度较低, 且无法动态分析, 缺乏了对漏洞检测结果的可达性检验, 会产生较高的误报率; 污点传播过程中内存存储信息的处理和信息流分析不全面造成的过污染与欠污染也是现阶段面临的难点; 同时, 机器学习对数据集处理与算法选择上的依赖, 造成了源码语义建模不足的难题, 另一方面, 神经网络等算法的黑箱性也导致大多数情况下检测结果的可解释性较差; 中间表示形式的静态分析以及传统软件分析和安全领域迁移而来方法也带来了诸如运行时内存信息缺失、并发检测不足和未知类型漏洞诊断能力较差等缺陷. 因此, 克服现有技术所依赖漏洞检测方法的局限性是亟待解决的关键问题.

(4) 漏洞检测结果中存在的高漏报率和误报率. 由于智能合约漏洞类型的繁多与漏洞特征的复杂性, 以及所采用方法自身的缺陷, 大多数技术在检测漏洞时存在着忽略某种漏洞类型复杂特性部分特征的可能, 仍然存在不同程度的误报和漏报. 降低检测结果的漏报率与误报率也是当前技术的关键挑战.

(5) 较低的漏洞类型覆盖率. 由于智能合约的不断发展与升级, 存在着复杂且数量繁多的漏洞类型, 大部分漏洞检测技术都是提供了一种检测方法并应用在了适应该方法的漏洞类型中, 因而并不一定涵盖智能合约中存在的所有漏洞. 因此, 现阶段有完善技术对漏洞类型更全面检测的需求.

(6) 较低的漏洞利用率[145]. 智能合约通过区块链上的交易来进行交互, 漏洞利用主要关注如何生成恶意的交易数据, 以利用智能合约中的漏洞来完成特定的攻击. 智能合约中的漏洞分类众多, 产生原因也各不相同, 一次成功的攻击往往需要多个漏洞的相互配合, 并可能涉及多个合约之间的相互调用, 多方因素共同造成了当前检测结果中漏洞利用率低下的现状. 检测出更易被利用的高危漏洞或自动化漏洞利用, 也是现阶段需要进一步深入的研究方向.

(7) 较长的漏洞审计时间. 智能合约潜在安全隐患的及时审计也是保障智能合约安全发展的关键要素, 现有技术的高效检测与完全自动化是智能合约开发周期中的重要一环, 而当前技术依然存在较长的漏洞检测时间与人工参与审计的问题. 在面临数目和规模与日俱增的智能合约时, 现阶段低效的审计效率和繁琐的人工审计工作量也是保障智能合约安全的亟需突破的难点.

4.2 未来研究方向与改进思路

基于智能合约漏洞检测技术现阶段的发展与挑战, 本文总结出了以下未来研究方向与改进思路.

(1) 完善智能合约的安全开发流程. 为了减轻智能合约部署之后的漏洞检测负担, 同时为了缓解智能合约平台与高级编程语言多样性带来的问题, 未来研究工作应该首先关注智能合约开发过程中的安全保障工作, 研究如何在智能合约的设计和实现阶段, 针对不同的合约功能需求、编程语言、编译执行环境、区块链平台, 定制合约开发标准, 规范开发流程, 智能合约开发阶段的标准化可以降低部署后合约上存在的漏洞数量, 能一定程度上减小智能合约漏洞检测的负担, 提高智能合约漏洞审计效率.

(2) 改进现有漏洞检测方法. 现有检测方法自身都存在一定程度的局限性, 如模糊测试的低路径覆盖率、符号执行的路径爆炸与污点分析过程中的过污染和欠污染等难题. 我们就它们未来的研究方向与改进思路进行了讨论与分析.

• 模糊测试的关键步骤是测试用例的生成, 传统的模糊测试在生成测试用例时, 往往盲目变异正常测试用例中的某一部分去生成测试用例, 导致测试用例规模庞大, 执行效率并不理想. 因此, 未来需要进一步结合智能合约在区块链、虚拟机、高级语言等不同层面的特性, 采用如遗传、模拟退火、多目标优化等算法, 制定出优秀的测试用例生成策略, 另一个研究方向是通过符号执行、污点分析以及新兴的机器学习技术, 弥补模糊测试过程中语义理解的缺失, 生成能够触发研究人员所关注逻辑的结构化测试用例, 提高模糊测试的路径覆盖率与漏洞发掘的效率.

• 制约符号执行效果的最主要挑战是执行路径爆炸难题, 未来可行的方法是结合现有智能合约的审计经验与

已曝漏洞的分析结果, 寻找合约中易产生漏洞的高危指令, 如 SUICIDE、CALL、DELETECALL 等, 将涉及这些指令的路径定义为重点路径, 只符号执行标记的重点路径并进行漏洞验证, 达到剪枝冗余路径的效果, 有效地缩减符号执行过程中的路径空间, 缓解执行过程中的路径爆炸问题. 与此同时, 符号执行因其高效的路径覆盖率, 能推断出到达特定程序状态的条件, 并约束求解出测试者关注逻辑的输入, 可作为其他漏洞检测方法的辅助方法, 基于符号执行的结果可用于模糊测试的测试输入的优化、形式化验证的模型状态空间搜索与漏洞检测结果的可达性检验等, 达到提高路径覆盖率、降低漏洞检测结果假阳性等效果.

• 形式化验证方法通过数学推演来验证智能合约系统的安全性, 现有的研究工作大多数自动化程度不高, 且检测出来的漏洞不一定存在可达路径. 未来的研究方向应为检测不同漏洞目标的合约定制对应的验证规范描述, 突破验证成本昂贵、大规模合约无法适应等技术限制, 并将应用范围从验证一般合约的功能和安全属性等, 逐步扩展到存在复杂业务逻辑等高阶性质证明的商业场景中. 在定制形式化验证规范阶段, 可借助污点分析与静态分析等方法的源码语义信息, 构造出描述合约正确性与公平性的形式化规范, 而在推理合约行为是否符合归纳的形式化规约时, 则可通过符号执行技术扩大可达性检验属性的状态空间, 达到降低形式化验证成本与漏洞检测假阳性的效果.

• 污点分析通过追踪程序中由污点源引入的数据能否不经无害化处理而直接传播到汇聚点, 来检测智能合约中是否存在漏洞. 污点数据识别的精度与传播策略的优劣, 引发了污点分析过程中的过污染与欠污染问题, 成为污点分析主要的性能瓶颈, 严重影响了智能合约检测的精度与效率. 同时, 动态污点分析过程中也存在精度与效率之间不可调和的矛盾. 基于智能合约性质与已知漏洞特征, 权衡好智能合约执行效率和污点分析精度, 设计更科学的污点识别方法的同时并定制严谨的污点传播策略, 如追踪区块数、时间戳等矿工可以操纵的信息、可操作代币转移的合约函数行为等, 可以尽可能地降低污点传播过程中的过污染与欠污染的出现频率. 除此之外, 污点分析过程中捕获的智能合约信息流中的语义, 对形式化规范的构造、模糊测试策略的制定以及符号执行关键路径的提取, 都能够提供一定程度的深层语义信息, 用于提高智能合约漏洞检测的效果.

• 现有的基于机器学习的智能合约漏洞检测技术大多是通过数据集训练出的漏洞检测模型, 来给出最终的漏洞检测结果, 是典型的黑盒测试流程. 由于机器学习模型固有的“黑箱性”, 其内部检测漏洞的具体工作状态和处理过程是不透明的, 因此缺乏对漏洞检测结果的合理解释, 使得其检测结果无法令人信服. 传统检测工具中定义的规则与提取出的语义信息是分析合约漏洞的利器, 未来的机器学习模型应考虑融合传统检测方法中漏洞相关的规则及语义信息, 有利于提高模型输出结果的可解释性和漏洞检测的准确率. 同时, 机器学习方法使用已有的数据让计算机模拟或实现类似人类的学习行为, 可用于改善现阶段低效的审计效率并减轻繁琐的人工审计工作量.

• 基于表示形式的静态分析方法通常将定义的漏洞规则, 应用到智能合约源码或字节码转换的中间表示形式来检测漏洞. 未来的研究方向应当提高这类检测方法的扩展性和适应性, 专注于中间表现形式的通用性上. 此外, 将静态分析与动态执行的结合也能够显著提高漏洞检测的效果. 而传统软件分析和安全领域迁移而来的方法应更多结合智能合约在区块链、虚拟机、高级编程语言等层面的特性, 最大程度降低自身方法的局限性, 适应到智能合约漏洞检测的背景中.

(3) 构建智能合约的标准实验数据集. 当前关于智能合约的实验数据集尚未有标准的案例集, 本文调研的技术实验部分也都是采用了不同的数据集, 在检测结果上未能反映出技术漏洞检测的各方面性能与不同技术间的差异, 构建出统一规范的、涵盖漏洞类型与智能合约平台全面的实验数据集, 除了可以验证智能合约漏洞检测技术的多方面 (漏报率、误报率和漏洞利用率) 性能与比较不同技术间的差异, 也可以给智能合约的安全开发和机器学习的模型训练提供参考, 更好地推动该领域的研究.

(4) 制定统一的漏洞检测技术性能评估标准. 根据目前已有的漏洞检测技术以及以往智能合约漏洞的审计经验, 综合考虑漏洞检测结果的漏洞率、误报率、漏洞利用情况、检测时间、涵盖漏洞类型、支持平台等因素, 最终制定统一的智能合约漏洞检测技术性能评价标准, 有利于已有技术的性能评价与对比分析, 并为新的智能合约漏洞检测技术的研发与改进提供参考指导.

5 总 结

智能合约是目前最有前景的技术之一, 提供了丰富、安全、可信的去中心化应用场景, 契合了数字身份、物联网、供应链等领域的现实意义, 但与之而来的安全事故严重阻碍了它的发展, 智能合约漏洞检测技术已成为新的研究热点. 本文梳理了研究者们提出的一系列智能合约漏洞检测技术, 然后将已有技术归纳为基于模糊测试、基于符号执行、基于污点分析、基于形式化验证、基于机器学习和其他方法, 并介绍了目前这 6 种分类下的智能合约漏洞检测技术, 再统计分析了现有技术的采用方法、涵盖漏洞类型、检测平台、开/闭源信息、实验数据、对比技术信息, 同时对比评估了国内外研究现状的差异, 并总结了国内研究的不足之处与潜在的改进方向, 最后对智能合约漏洞检测技术研究现状的不足之处与未来研究方向和改进思路进行了总结.

References:

[1] Szabo N. Smart contracts: Building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought, 1996, 18(2): 28.

[2] Wood G. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014, 151(2014): 1–32.

[3] Etherscan. The Ethereum blockchain explorer. 2023. https://etherscan.io

[4] Bogner A, Chanson M, Meeuw A. A decentralised sharing APP running a smart contract on the Ethereum blockchain. In: Proc. of the 6th Int'l Conf. on the Internet of Things. Stuttgart: ACM, 2016. 177–178. [doi: 10.1145/2991561.2998465]

[5] 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]

[6] Solidity Team. Solidity. 2023. https://docs.soliditylang.org

[7] Antonopoulos AM, Wood G. Mastering Ethereum: Building Smart Contracts and Dapps. Sebastopol: O'Reilly Media, 2018.

[8] 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.

[9] Siegel D. Understanding the DAO attack. 2023. https://www.coindesk.com/understanding-dao-hack-journalists

[10] BlockCAT. On the Parity multi-sig wallet attack. 2017. https://medium.com/blockcat/on-the-parity-multi-sig-wallet-attack-83fb5e7f4b8c

[11] Pretrov S. Another Parity wallet hack explained. 2017. https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c

[12] Wikipedia. Poly network exploit. 2023. https://en.wikipedia.org/wiki/Poly_Network_exploit

[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] 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]

[16] Groce A, Feist J, Grieco G, Colburn M. What are the actual flaws in important smart contracts (and how can we find them)? In: Proc. of the 24th Int'l Conf. on Financial Cryptography and Data Security. Kota Kinabalu: Springer, 2020. 634–653. [doi: 10.1007/978-3-030-51280-4_34]

[17] Dika A, Nowostawski M. Security vulnerabilities in Ethereum smart contracts. In: Proc. of the 2018 IEEE Int'l Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). Halifax: IEEE, 2018. 955–962. [doi: 10.1109/Cybermatics_2018.2018.00182]

[18] Sayeed S, Marco-Gisbert H, Caira T. Smart contract: Attacks and protections. IEEE Access, 2020, 8: 24416–24427. [doi: 10.1109/ACCESS.2020.2970495]

[19] Yamashita K, Nomura Y, Zhou EC, Pi BF, Jun S. Potential risks of hyperledger fabric smart contracts. In: Proc. of the 2019 IEEE Int'l Workshop on Blockchain Oriented Software Engineering. Hangzhou: IEEE, 2019. 1–10. [doi: 10.1109/IWBOSE.2019.8666486]

[20] Tang XY, Zhou K, Cheng JR, Li H, Yuan YM. The vulnerabilities in smart contracts: A survey. In: Proc. of the 7th Int'l Conf. on Artificial Intelligence and Security. Dublin: Springer, 2021. 177–190. [doi: 10.1007/978-3-030-78621-2_14]

[21] Wöhrer M, Zdun U. Design patterns for smart contracts in the Ethereum ecosystem. In: Proc. of the 2018 IEEE Int'l Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data. Halifax: IEEE, 2018. 1513–1520. [doi: 10.1109/Cybermatics_2018.2018.00255]

[22] 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]

[23] Worley CR, Skjellum A. Opportunities, challenges, and future extensions for smart-contract design patterns. In: Proc. of the 2018 Int'l Conf. on Business Information Systems. Berlin: Springer, 2018. 264–276. [doi: 10.1007/978-3-030-04849-5_24]

[24] Xu XW, Pautasso C, Zhu LM, Lu QH, Weber I. A pattern collection for blockchain-based applications. In: Proc. of the 23rd European Conf. on Pattern Languages of Programs. Irsee: ACM, 2018: 3. [doi: 10.1145/3282308.3282312]

[25] Praitheeshan P, Pan L, Yu JS, Liu J, Doss R. Security analysis methods on Ethereum smart contract vulnerabilities: A survey. arXiv:1908.08605, 2020.

[26] Di Angelo M, Salzer G. A survey of tools for analyzing Ethereum smart contracts. In: Proc. of the 2019 IEEE Int'l Conf. on Decentralized Applications and Infrastructures. Newark: IEEE, 2019. 69–78. [doi: 10.1109/DAPPCON.2019.00018]

[27] Chen HS, Pendleton M, Njilla L, Xu SH. A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses. ACM Computing Surveys, 2021, 53(3): 67. [doi: 10.1145/3391195]

[28] Chen T, Li ZH, Zhang YF, Luo XP, Wang T, Hu T, Xiao XZ, Wang D, Huang J, Zhang XS. A large-scale empirical study on control flow identification of smart contracts. In: Proc. of the 2019 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Porto de Galinhas: IEEE, 2019. 1–11. [doi: 10.1109/ESEM.2019.8870156]

[29] Hu B, Zhang ZY, Liu JW, Liu YZ, Yin JY, Lu RX, Lin XD. A comprehensive survey on smart contract construction and execution: Paradigms, tools, and systems. Patterns, 2021, 2(2): 100179. [doi: 10.1016/J.PATTER.2020.100179]

[30] 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: IEEE, 2020. 530–541.

[31] Feng XT, Wang Q, Zhu XG, Wen S. Bug searching in smart contract. arXiv:1905.00799, 2019.

[32] Murray Y, Anisi DA. Survey of formal verification methods for smart contracts on blockchain. In: Proc. of the 10th IFIP Int'l Conf. on New Technologies, Mobility and Security. Canary Islands: IEEE, 2019. 1–6. [doi: 10.1109/NTMS.2019.8763832]

[33] Bartoletti M, Zunino R. Formal models of bitcoin contracts: A survey. Frontiers in Blockchain, 2019, 2: 8. [doi: 10.3389/fbloc.2019.00008]

[34] Ladleif J, Weske M. A unifying model of legal smart contracts. In: Proc. of the 38th Int'l Conf. on Conceptual Modeling. Salvador: Springer, 2019. 323–337. [doi: 10.1007/978-3-030-33223-5_27]

[35] Singh A, Parizi RM, Zhang Q, Choo KKR, Dehghantanha A. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. Computers & Security, 2020, 88: 101654. [doi: 10.1016/j.cose.2019.101654]

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

[37] Seijas PL, Thompson SJ, Mcadams D. Scripting smart contracts for distributed ledger technology. 2016. https://eprint.iacr.org/2016/1156.pdf [doi: 10.13140/RG.2.2.20839.85920]

[38] Parizi RM, Amritraj, Dehghantanha A. Smart contract programming languages on blockchains: An empirical evaluation of usability and security. In: Proc. of the 1st Int'l Conf. on Blockchain. Seattle: Springer, 2018. 75–91. [doi: 10.1007/978-3-319-94478-4_6]

[39] Harz D, Knottenbelt W. Towards safer smart contracts: A survey of languages and verification methods. arXiv:1809.09805, 2018.

[40] Wang S, Ouyang LW, Yuan Y, Ni XC, Han X, Wang FY. Blockchain-enabled smart contracts: Architecture, applications, and future trends. IEEE Trans. on Systems, Man, and Cybernetics: Systems, 2019, 49(11): 2266–2277. [doi: 10.1109/TSMC.2019.2895123]

[41] Wei XT, Lu C, Ozcan FR, Chen T, Wang BY, Wu D, Tang Q. A behavior-aware profiling of smart contracts. In: Proc. of the 15th EAI Int'l Conf. on Security and Privacy in Communication Networks. Orlando: Springer, 2019. 245–258. [doi: 10.1007/978-3-030-37231-6_13]

[42] Arias EJG. Towards principled compilation of Ethereum smart contracts (SoK). In: Proc. of the 10th IFIP Int'l Conf. on New Technologies, Mobility and Security. Canary Islands: IEEE, 2019. 1–5. [doi: 10.1109/NTMS.2019.8763856]

[43] Miller A, Cai ZC, Jha S. Smart contracts and opportunities for formal methods. In: Proc. of the 8th Int'l Symp. on Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice. Limassol: Springer, 2018. 280–299. [doi: 10.1007/978-3-030-03427-6_22]

[44] Zheng ZB, Xie SA, Dai HN, Chen WL, Chen XP, Weng J, Imran M. An overview on smart contracts: Challenges, advances and platforms. Future Generation Computer Systems, 2020, 105: 475–491. [doi: 10.1016/j.future.2019.12.019]

[45] Zou WQ, Lo D, Kochhar PS, Le XBD, Xia X, Feng Y, Chen ZY, Xu BW. Smart contract development: Challenges and opportunities. IEEE Trans. on Software Engineering, 2021, 47(10): 2084–2106. [doi: 10.1109/TSE.2019.2942301]

[46] Rouhani S, Deters R. Security, performance, and applications of smart contracts: A systematic survey. IEEE Access, 2019, 7:

50759–50779. [doi: 10.1109/ACCESS.2019.2911031]

[47] 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. 2020. 1661–1677.

[48] Bernardi T, Dor N, Fedotov A, Grossman S, Immerman N, Jackson D, Nutz A, Oppenheim L, Pistiner O, Rinetzky N, Sagiv M, Taube M, Toman JA, Wilcox JR. WIP: Finding bugs automatically in smart contracts with parameterized invariants. 2020. https://www.certora.com/wp-content/themes/Certora/assets/pdf/sbc2020.pdf

[49] Coward PD. Symbolic execution systems—A review. Software Engineering Journal, 1988, 3(6): 229–239. [doi: 10.1049/sej.1988.0029]

[50] Luu L, Chu DH, Olickel H, 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]

[51] 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. Klagenfurt: IEEE, 2017. 442–446. [doi: 10.1109/SANER.2017.7884650]

[52] Tsankov P, Dan A, Drachsler-Cohen D, Gervais A, Buenzli 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]

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

[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] 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. San Diego: IEEE, 2019. 1186–1189. [doi: 10.1109/ASE.2019.00133]

[56] Li A, Long F. Detecting standard violation errors in smart contracts. arXiv:1812.07702, 2019.

[57] Torres CF, Steichen M, State R. The art of the scam: Demystifying honeypots in Ethereum smart contracts. In: Proc. of the 28th USENIX Security Symp. Santa Clara: USENIX Association, 2019. 1591–1607.

[58] He NY, Zhang RY, Wu L, Wang HY, Luo XP, Guo Y, Yu T, Jiang XX. Security analysis of EOSIO smart contracts. arXiv:2003.06568, 2020.

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

[60] Huang YH, Jiang B, Chan WK. EOSFuzzer: Fuzzing EOSIO smart contracts for vulnerability detection. arXiv:2007.14903, 2020.

[61] Ashraf I, Ma XX, Jiang B, Chan WK. GasFuzzer: Fuzzing Ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities. IEEE Access, 2020, 8: 99552–99564. [doi: 10.1109/ACCESS.2020.2995183]

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

[63] Ma FC, Fu Y, Ren M, Sun WT, Liu Z, Jiang Y, Sun J, Sun JG. GasFuzz: Generating high gas consumption inputs to avoid out-of-gas vulnerability. arXiv:1910.02945, 2021.

[64] Kolluri A, Nikolic I, Sergey I, Hobor A, Saxena P. Exploiting the laws of order in smart contracts. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 363–373. [doi: 10.1145/3293882.3330560]

[65] 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]

[66] Grieco G, Song W, Cygan A, Feist J, Groce A. Echidna: Effective, usable, and fast fuzzing for smart contracts. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 557–560. [doi: 10.1145/3395363.3404366]

[67] 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]

[68] Zhang QZ, Wang YZ, Li JR, Ma SQ. EthPloit: From fuzzing to efficient exploit generation against smart contracts. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. London: IEEE, 2020. 116–126. [doi: 10.1109/SANER48275.2020.9054822]

[69] Schwartz EJ, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: Proc. of the 2010 IEEE Symp. on Security and Privacy. Oakland: IEEE, 2010. 317–331. [doi: 10.1109/SP.2010.26]

[70] Rodler M, Li WT, Karame GO, Davi L. Sereum: Protecting existing smart contracts against re-entrancy attacks. arXiv:1812.05934, 2018.

[71] 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]

[72] Brent L, Grech N, Lagouvardos S, Scholz B, Smaragdakis Y. Ethainter: A smart contract security analyzer for composite vulnerabilities. In: Proc. of the 41st ACM SIGPLAN Conf. on Programming Language Design and Implementation. London: ACM, 2020. 454–469. [doi: 10.1145/3385412.3385990]

[73] Ashouri M. Etherolic: A practical security analyzer for smart contracts. In: Proc. of the 35th Annual ACM Symp. on Applied Computing. Brno: ACM, 2020. 353–356. [doi: 10.1145/3341105.3374226]

[74] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33–61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[75] 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]

[76] 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]

[77] 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. Oxford: IEEE, 2018. 204–217. [doi: 10.1109/CSF.2018.00022]

[78] 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]

[79] 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, 2017, 2: 48. [doi: 10.1145/3158136]

[80] Wang S, Zhang CY, Su ZD. Detecting nondeterministic payment bugs in Ethereum smart contracts. Proc. of the ACM on Programming Languages, 2019, 3: Article No. : 189. [doi: 10.1145/3360615]

[81] 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]

[82] 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]

[83] Albert E, Gordillo P, Rubio A, Sergey I. Running on fumes: Preventing out-of-gas vulnerabilities in Ethereum smart contracts using static resource analysis. In: Proc. of the 13th Int'l Conf. on Verification and Evaluation of Computer and Communication Systems. Porto: Springer, 2019. 63–78. [doi: 10.1007/978-3-030-35092-5_5]

[84] Albert E, Correas J, Gordillo P, Román-Díez G, Rubio A. SAFEVM: A safety verifier for Ethereum smart contracts. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 386–389. [doi: 10.1145/3293882.3338999]

[85] Beyer D, Keremoglu ME. CPAchecker: A tool for configurable software verification. In: Proc. of the 23rd Int'l Conf. on Computer Aided Verification. Snowbird: Springer, 2011. 184–190. [doi: 10.1007/978-3-642-22110-1_16]

[86] Brockschmidt M, Larra D, Oliveras A, Rodrıguez-Carbonell E, Rubio A. Compositional safety verification with Max-SMT. In: Proc. of the 2015 Formal Methods in Computer-aided Design. Austin: IEEE, 2015. 33–40. [doi: 10.1109/FMCAD.2015.7542250]

[87] Gurfinkel A, Kahsai T, Komuravelli A, Navas JA. The SeaHorn verification framework. In: Proc. of the 27th Int'l Conf. on Computer Aided Verification. San Francisco: Springer, 2015. 343–361. [doi: 10.1007/978-3-319-21690-4_20]

[88] 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, 2018. 1–12. [doi: 10.14722/ndss.2018.23082]

[89] 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]

[90] Frank J, Aschermann C, Holz T. EthBMC: A bounded model checker for smart contracts. In: Proc. of the 29th USENIX Security Symp. Berkeley: USENIX Association, 2020. 2757–2774.

[91] Schneidewind C, Grishchenko I, Scherer M, Maffei M. eThor: Practical and provably sound static analysis of Ethereum smart contracts. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2020. 621–640. [doi: 10.1145/3372297.

3417250]

[92] Tann WJW, Han XJ, Gupta SS, Ong YS. Towards safer smart contracts: A sequence learning approach to detecting security threats. arXiv:1811.06632, 2019.

[93] Hu T, Liu XL, Chen T, Zhang XS, Huang XM, Niu WN, Lu JZ, Zhou K, Liu Y. Transaction-based classification and detection approach for Ethereum smart contract. Information Processing and Management, 2021, 58(2): 102462. [doi: 10.1016/j.ipm.2020.102462]

[94] 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. Montpellier: ACM, 2018. 814–819. [doi: 10.1145/3238147.3240728]

[95] Hu HW, Xu YD. SCSGuard: Deep SCAM detection for Ethereum smart contracts. arXiv:2105.10426, 2021.

[96] 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]

[97] 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]

[98] 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]

[99] Ashizawa N, Yanai N, Cruz JP, Okamura S. Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts. In: Proc. of the 3rd ACM Int'l Symp. on Blockchain and Secure Critical Infrastructure. Hong Kong: ACM, 2021. 47–59. [doi: 10.1145/3457337.3457841]

[100] Le Q, Mikolov T. Distributed representations of sentences and documents. In: Proc. of the 31st Int'l Conf. on Machine Learning. Beijing: JMLR.org, 2014. 1188–1196.

[101] Zhuang Y, Liu ZG, Qian P, Liu Q, Wang X, He QM. Smart contract vulnerability detection using graph neural network. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence. Yokohama: IJCAI, 2021. 3283–3290.

[102] Eshghie M, Artho C, Gurov D. Dynamic vulnerability detection on smart contracts using machine learning. In: Proc. of the 2021 Evaluation and Assessment in Software Engineering. Trondheim: ACM, 2021. 305–312. [doi: 10.1145/3463274.3463348]

[103] Lutz O, Chen HL, Fereidooni H, Sendner C, Dmitrienko A, Sadeghi AR, Koushanfar F. Escort: Ethereum smart contracts vulnerability detection using deep neural network and transfer learning. arXiv:2103.12607, 2021.

[104] Mi F, Wang ZY, Zhao C, Guo JH, Ahmed F, Khan L. VSCL: Automating vulnerability detection in smart contracts with deep learning. In: Proc. of the 2021 IEEE Int'l Conf. on Blockchain and Cryptocurrency. Sydney: IEEE, 2021. 1–9. [doi: 10.1109/ICBC51069.2021.9461050]

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

[106] Terence P. ANLTR. 2023. https://www.antlr.org/

[107] Lu N, Wang B, Zhang YX, Shi WB, Esposito C. NeuCheck: A more practical Ethereum smart contract security analysis tool. Software: Practice and Experience, 2021, 51(10): 2065–2084. [doi: 10.1002/spe.2745]

[108] Quan LJ, Wu L, Wang HY. EVulHunter: Detecting fake transfer vulnerabilities for EOSIO's smart contracts at webassembly-level. arXiv:1906.10362, 2019.

[109] Argañaraz MC, Berón MM, Varanda Pereira MJ, Henriques PR. Detection of vulnerabilities in smart contracts specifications in Ethereum platforms. In: Proc. of the 9th Symp. on Languages, Applications and Technologies. Dagstuhl, 2020. 1–16. [doi: 10.4230/OASIcs.SLATE.2020.2]

[110] Nguyen QB, Nguyen AQ, Nguyen VH, Nguyen-Le T, Nguyen-An K. Detect abnormal behaviours in Ethereum smart contracts using attack vectors. In: Proc. of the 6th Int'l Conf. on Future Data and Security Engineering. Nha Trang City: Springer, 2019. 485–505. [doi: 10.1007/978-3-030-35653-8_32]

[111] Wang XM, He JH Xie ZJ, Zhao GS, Cheung SC. ContractGuard: Defend Ethereum smart contracts with embedded intrusion detection. IEEE Trans. on Services Computing, 2020, 13(2): 314–328. [doi: 10.1109/TSC.2019.2949561]

[112] Li ZX, Wu HR, Xu JH, Wang XY, Zhang LM, Chen ZY. MuSC: A tool for mutation testing of Ethereum smart contract. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 1198–1201. [doi: 10.1109/ASE.2019.00136]

[113] Akca S, Rajan A, Peng C. SolAnalyser: A framework for analysing and testing smart contracts. In: Proc. of the 2019 26th Asia-Pacific Software Engineering Conf. Putrajaya: IEEE, 2019. 482–489. [doi: 10.1109/APSEC48747.2019.00071]

[114] Xue YX, Ye JM, Ma ML, Ma L, Li Y, Wang HJ, Lin Y, Peng TY, Liu Y. Doublade: Unknown vulnerability detection in smart contracts via abstract signature matching and refined detection rules. arXiv:1912.04466, 2022.

[115] Ye JM, Ma ML, Peng TY, Xue YX. A software analysis based vulnerability detection system for smart contracts. In: Proc. of the 2020 Integrating Research and Practice in Software Engineering. Switzerland: Springer, 2020. 69–81. [doi: 10.1007/978-3-030-26574-8_6]

[116] 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. Montreal: IEEE, 2019. 8–15. [doi: 10.1109/WETSEB.2019.00008]

[117] 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]

[118] Chen T, Cao R, Li T, Luo XP, Gu GF, Zhang YF, Liao Z, Zhu H, Chen G, He ZY, Tang YX, Lin XD, Zhang XS. SODA: A generic online detection framework for smart contracts. In: Proc. of the 2020 Network and Distributed Systems Security Symp. San Diego, 2020. 1–12.

[119] Wu L, Wu SW, Zhou YJ, Li RH, Wang Z, Luo XP, Wang C, Ren K. EthScope: A transaction-centric security analytics framework to detect malicious smart contracts on Ethereum. arXiv:2005.08278, 2020.

[120] ConsenSys. Mythril. 2023. https://github.com/ConsenSys/mythril

[121] VaaS. 2023. https://www.lianantech.com/

[122] Ji R, He NY, Wu L, Wang HY, Bai GD, Guo Y. DEPOSafe: Demystifying the fake deposit vulnerability in Ethereum smart contracts. In: Proc. of the 25th Int'l Conf. on Engineering of Complex Computer Systems. Singapore: IEEE, 2020. 125–134. [doi: 10.1109/ICECCS51672.2020.00022]

[123] 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]

[124] Liu Y, Li Y, Lin SW, Yan Q. ModCon: A model-based testing platform for smart contracts. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Virtual Event: ACM, 2020. 1601–1605. [doi: 10.1145/3368089.3417939]

[125] Chen JC, Xia X, Lo D, Grundy J, Luo XP, Chen T. DefectChecker: Automated smart contract defect detection by analyzing EVM bytecode. arXiv:2009.02663, 2021.

[126] Wang D, Jiang B, Chan WK. WANA: Symbolic execution of WASM bytecode for cross-platform smart contract vulnerability detection. arXiv:2007.15510, 2020.

[127] Yang ZQ, Liu H, Li Y, Zheng HX, Wang L, Chen BD. Seraph: Enabling cross-platform security analysis for EVM and WASM smart contracts. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Seoul: ACM, 2020. 21–24. [doi: 10.1145/3377812.3382157]

[128] 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. Seoul: IEEE, 2020. 274–275.

[129] Alkhalifah A, Ng A, Watters PA, Kayes ASM. A mechanism to detect and prevent Ethereum blockchain smart contract reentrancy attacks. Frontiers in Computer Science, 2021, 3: 598780. [doi: 10.3389/fcomp.2021.598780]

[130] Samreen NF, Alalfi MH. Reentrancy vulnerability identification in Ethereum smart contracts. In: Proc. of the 2020 IEEE Int'l Workshop on Blockchain Oriented Software Engineering. London: IEEE, 2020. 22–29. [doi: 10.1109/IWBOSE50093.2020.9050260]

[131] Chen WM, Li XR, Sui YT, He NY, Wang HY, Wu L, Luo XP. SADPonzi: Detecting and characterizing ponzi schemes in Ethereum smart contracts. Proc. of the ACM on Measurement and Analysis of Computing Systems, 2021, 5(2): 26. [doi: 10.1145/3460093]

[132] Chinen Y, Yanai N, Cruz JP, Okamura S. RA: Hunting for re-entrancy attacks in Ethereum smart contracts via static analysis. In: Proc. of the 2020 IEEE Int'l Conf. on Blockchain. Rhodes: IEEE, 2020. 327–336. [doi: 10.1109/Blockchain50366.2020.00048]

[133] Reis JS, Crocker P, de Sousa SM. Tezla, an intermediate representation for static analysis of Michelson smart contracts. arXiv:2005.11839, 2020.

[134] Alqahtani S, He XC, Gamble R, Mauricio P. Formal verification of functional requirements for smart contract compositions in supply chain management systems. In: Proc. of the 53rd Hawaii Int'l Conf. on System Sciences. Grand Wailea, 2020. 5278–5287.

[135] Antonino P, Roscoe A W. Formalising and verifying smart contracts with solidifier: A bounded model checker for Solidity. arXiv:2002.02710, 2020.

[136] Ding MJ, Li PR, Li SS, Zhang H. HFContractFuzzer: Fuzzing hyperledger fabric smart contracts for vulnerability detection. In: Proc. of the 2021 Evaluation and Assessment in Software Engineering. Trondheim: ACM, 2021. 321–328. [doi: 10.1145/3463274.3463351]

[137] Song JJ, He HW, Lv Z, Su CH, Xu GQ, Wang W. An efficient vulnerability detection model for Ethereum smart contracts. In: Proc. of the 13th Int'l Conf. on Network and System Security. Sapporo: Springer, 2019. 433–442. [doi: 10.1007/978-3-030-36938-5_26]

[138] Honig JJ, Everts MH, Huisman M. Practical mutation testing for smart contracts. In: Proc. of the 2019 Int'l Workshop on Data Privacy Management, Cryptocurrencies and Blockchain Technology. Luxembourg: Springer, 2019. 289–303. [doi: 10.1007/978-3-030-31500-9_19]

[139] Hajdu Á, Jovanović D. SOLC-VERIFY: A modular verifier for Solidity smart contracts. In: Proc. of the 11th Int'l Conf. on Verified Software: Theories, Tools, and Experiments. New York: Springer, 2019. 161–179. [doi: 10.1007/978-3-030-41600-3_11]

[140] Torres CF, Iannillo AK, Gervais A, State R. The eye of horus: Spotting and analyzing attacks on Ethereum smart contracts. arXiv:2101.06204, 2021.

[141] Barboni M, Morichetta A, Polini A. SuMo: A mutation testing strategy for Solidity smart contracts. arXiv:2105.03626, 2021.

[142] 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: 116. [doi: 10.1145/3276486]

[143] Samreen NF, Alalfi MH. SmartScan: An approach to detect denial of service vulnerability in Ethereum smart contracts. arXiv:2105.02852, 2021.

[144] Nishida Y, Saito H, Chen R, Kawata A, Furuse J, Suenaga K, Igarashi A. Helmholtz: A verifier for Tezos smart contracts based on refinement types. In: Proc. of the 27th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems. Luxembourg: Springer, 2021. 262–280. [doi: 10.1007/978-3-030-72013-1_14]

[145] Perez D, Livshits B. Smart contract vulnerabilities: Vulnerable does not imply exploited. In: Proc. of the 30th USENIX Security Symp. USENIX Association, 2021. 1325–1341.

附中文参考文献:

[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] 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]

[74] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33–61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

董伟良(1998－), 男, 硕士生, 主要研究领域为智能合约安全.

黎立(1989－), 男, 博士, 高级讲师, 博士生导师, 主要研究领域为智能化软件工程, 软件安全.

刘哲(1986－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为密码学, 密码工程, 后量子密码, 侧信道攻击与防御.

葛春鹏(1987－), 男, 博士, 副教授, 主要研究领域为网络空间安全, 软件安全.

刘逵(1988－), 男, 博士, 副教授, 主要研究领域为智能化软件工程, 软件安全.

黄志球(1965－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 系统软件, 形式化方法.