计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210700004

以太坊 Solidity 智能合约漏洞检测方法综述

张潆藜　马佳利　刘子昂　刘　新　周　睿
兰州大学信息科学与工程学院　兰州 730000
(zhangyingli2020@lzu.edu.cn)

摘　要　以太坊 Solidity 智能合约基于区块链技术,作为一种旨在以信息化方式传播、验证或执行的计算机协议,为各类分布式应用服务提供了基础。虽然落地还不足 6 年,但因其安全漏洞事件频繁爆发,且造成了巨大的经济损失,使得其安全性检查方面的研究备受关注。首先基于以太坊相关技术对智能合约的一些特殊机制和运行原理进行介绍,并根据智能合约的自身特性对一些出现频率较高的智能合约漏洞进行分析,然后从符号执行、模糊测试、形式化验证和污点分析等方面分类阐述了传统的主流智能合约漏洞检测工具。此外,为了应对层出不穷的新型漏洞以及提高漏洞检测效率的需求,将近年来基于机器学习的漏洞检测技术依据其问题转化方式的不同做了分类总结,并从文本处理、非欧几里得图和标准图像 3 个角度进行了简要介绍。最后,针对两类检测方向的不足之处,提出了建立相关标准化、规范化信息库以及衡量指标的建议。

关键词:智能合约;区块链;安全漏洞;漏洞检测工具;机器学习
中图法分类号　TP311

Overview of Vulnerability Detection Methods for Ethereum Solidity Smart Contracts

ZHANG Ying-li,MA Jia-li,LIU Zi-ang,LIU Xin and ZHOU Rui
School of Information Science & Engineering,Lanzhou University,Lanzhou 730000,China

Abstract　Based on blockchain technology,Ethereum Solidity smart contract as a computer protocol is designed to spread,verify,or execute contracts in an informative way,and it provides a foundation for various distributed application services. Although implemented for less than six years,its security problems have frequently broken out and caused substantial financial losses,which attracts more attention in the security inspection research. This paper firstly introduces some specific mechanisms and operating principles of smart contracts based on Ethereum related techniques,and analyzes some smart contract vulnerabilities occurring frequently and deriving from the characteristics of smart contracts. Then,this paper explains the traditional mainstream smart contract vulnerability detecting tools in terms of symbolic execution,fuzzing,formal verification,and taint analysis. In addition,in order to cope with the endless new vulnerabilities and the need to improve the efficiency of detection,vulnerabilities detection based on machine learning in recent years is classified and summarized according to the various ways of problem transformation in three perspectives including text processing,non-Euclidean graph and standard image. Finally,this paper proposes to formulate more extensive and accurate standardized information database and measurement indicators towards the insufficiency of the detection methods in two directions.

Keywords　Smart contracts,Blockchain,Security vulnerability,Vulnerability detection tools,Machine learning

1　引用

智能合约(smart contracts)早在 1995 年就由尼克萨博[1]提出,目的在于以数字形式定义一个合同,当参与方执行且满足合同所需的条件时,计算机自动执行该合同内容。但是由于技术条件等限制,这一概念当时并未实现。随着近年来区块链(blockchain)技术[2]的逐渐成熟以及加密货币的快速发展,去中心化共识协议[3]和工作量证明机制[3]共同保证了

到稿日期:2021-07-01　返修日期:2021-08-19
基金项目:国家重点研发计划(2020YFC0832500);甘肃省科技重大专项创新联合体项目(项目号 1);国家自然科学基金(61402210);青海省科技计划(2020-GX-164);教育部-中国移动科研基金项目(MCM20170206);兰州大学中央高校基本科研业务费专项资金(lzujbky-2021-sp47,lzujbky-2020-sp02,lzujbky-2019-kb51,lzujbky-2018-k12)
This work was supported by the National Key R&D Program of China(2020YFC0832500),Gansu Provincial Science and Technology Major Special Innovation Consortium Project(Project No. 1),National Natural Science Foundation of China(61402210),Science and Technology Plan of Qinghai Province(2020-GX-164),Ministry of Education-China Mobile Research Foundation(MCM20170206) and Fundamental Research Funds for the Central Universities(lzujbky-2021-sp47,lzujbky-2020-sp02,lzujbky-2019-kb51,lzujbky-2018-k12).
通信作者:周睿(zr@lzu.edu.cn)

操作的去中心化、过程透明、可追踪和可信任且不可篡改的特性，由此进入了以智能合约技术为标志的区块链 2.0 时代。所谓智能合约不只是一个可以在区块链上被自动执行的程序，它本身就是一个系统参与者，可以对信息进行接收和回应，也可以存储和收发资产[4]。支持智能合约运行的区块链平台[5]很多，如 EOS，BCOS，Fabric，CITA 等，其中规模最大、历史最久同时也最具影响力的是以太坊(Ethereum)[6]。以太坊通过建立一个图灵完备的、可以允许开发人员编写任意智能合约和去中心化应用(DApps)的平台而广受欢迎。随着智能合约应用场景的丰富，如金融、保险、游戏、能源等领域，其中控制的重要金融资产不可避免地成为易受攻击的目标。此外，由于智能合约本身编程语言的不安全性以及合约整个生命过程的复杂性，再加上以太坊中每个用户都可以在没有可信赖第三方的情况下参与合约，这些都为智能合约的安全性带来了巨大的风险和隐患。例如，2017 年 6 月 18 日，智能合约 The DAO[7]中的可重入漏洞最终造成了亿万美元的损失，影响了整个以太坊网络。2020 年 1 月 16 日，攻击者通过开源项目为 RVN[8]增加了功能，并通过一系列条件判断来绕过审查，从而拥有发币功能，最终获利约千万元人民币，使得平台遭遇巨大损失。

本文第 2 节对智能合约的原理和运行机制进行了简单的介绍；第 3 节对智能合约的常见漏洞进行了总结，并给出了一些相应的防范措施；第 4 节对智能合约的自动化漏洞检测工具进行了介绍，并回顾了传统的基于符号执行、模糊测试、形式化验证以及污点分析技术的方法；第 5 节重点归纳总结了基于机器学习的漏洞检测模型；最后总结全文，讨论了现有工作存在的不足之处，并给出了进一步研究的建议。

2 智能合约简介

2.1 智能合约架构

智能合约作为一段脚本，可以使用多种高级语言进行开发，如 Solidity，Serpent，Vyper 等，其中针对以太坊的智能合约通常采用 Solidity 编写，其语法类似于 JavaScript。智能合约的编译和解释执行则通过安装在每个以太坊节点上的以太坊虚拟机(Ethereum Virtual Machine，EVM)[9]来完成，根据以太坊白皮书，以太坊系统架构如图 1 所示。每个以太坊节点架构自底向上分别是操作系统、区块链节点客户端、以太坊虚拟机和智能合约脚本。以太坊正是通过运行在不同主机上的以太坊客户端节点之间的通信来完成各种操作。编写好的合约首先通过 EVM 编译器编译生成一个 ABI 文件和一个 bin 文件，其中 ABI 文件是合约的接口描述，包括了字段名称、字段类型、参数名称、参数返回值等信息，而 bin 文件是最终运行在虚拟机上的字节码(bytecode)，即一段 EVM 指令集合。合约的部署是将编译后的字节码上传到以太坊区块链平台上，其过程与发送一笔交易类似，发起地址为发布者的地址，目标地址为零，交易数据被替换为合约对应的字节码。在进行交易打包时，将根据发布者的地址和交易序列号通过加密算法重新计算出一个地址作为这个合约的地址，调用者可通过合约地址对合约进行调用。以太坊中存在两类账户：外部账户和合约账户，外部账户由私钥控制，有账户余额，可以触发交易，但没有代码；而合约账户包含不可修改的智能合约代码，有账户余额，但不能主动发起交易，只能在被触发后执行预先编写的逻辑。因此，合约调用也分为两种类型：一种是由外部账户发起的，称为交易调用；另一种则是由一个合约发起对另一个合约的调用，称为消息调用。此外，智能合约还具有自毁操作，这也是唯一能从区块链上将合约代码移除的方式，它需要在合约编写中执行 selfdestruct 操作，之后合约账户上剩余的以太币(Ether)会被发送给指定目标，其存储的相关状态和代码也会被移除。而所谓以太币就是以太坊中的通用货币，类似于比特币。

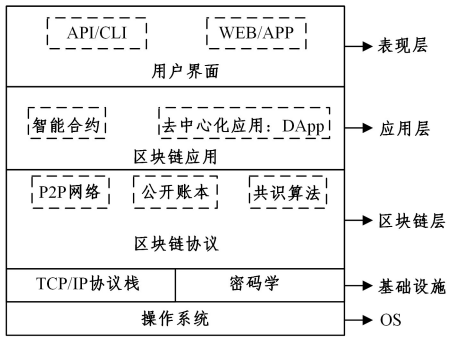

图 1 以太坊系统架构

Fig. 1 Ethereum system architecture

2.2 智能合约机制

不同于传统的应用程序，智能合约设计了一系列适用于区块链技术的机制[10]。

gas 机制：合约需要矿工的强制执行和证明，因此在打包一项交易时，即将其添加到一个区块上时，为了避免交易中包含大量循环等操作造成节点资源的浪费，合约需先行向矿工支付一笔费用(gas)。在以太坊执行一份合约需要根据内部制定的规则消耗一定量的 gas，如果交易执行后，提前支付的 gas 还有剩余，则原路返还；如果没有执行结束 gas 就被耗尽，则会触发一个 out-of-gas 异常，当前合约程序的所有执行状态都会被回滚，但是因为矿工为了执行相应计算已经付出了算力，所以已经消耗的 gas 不会被退回。

委托调用机制：合约可以通过消息调用的机制来调用其他合约，委托调用(delegatecall)就是一种特殊类型的消息调用。它和普通 call 指令的区别在于，普通 call 指令的行为是跳转到被调用合约，并在该合约中执行完相应代码再返回执行后续代码，对于调用发起者的上下文无影响；而委托调用则相当于把被调用合约中的一段代码拷贝到调用发起者合约的上下文环境中执行，修改调用发起者的信息。

异常传递机制：智能合约的函数调用方式分为内部调用和外部调用，内部函数调用指直接调用当前合约或者父合约的内部函数，这些函数调用在 EVM 中会被直接转换为简单的跳转指令；而外部函数调用指对于指定地址的外部合约函数的调用，需要靠消息调用完成[10]。其中外部调用中的一些低级调用，如 call，callcode，delegatecall 在执行中如果出错并抛出异常，该异常不会沿着函数调用栈被传递，而是只能获取一个布尔值来表示成功或者失败。此外，一些转账函数，如

call,value,send 等,当发生转账异常时,也仅返回一个布尔值而不是回滚这个操作。

3 智能合约安全问题

随着智能合约的迅速发展,其复杂性不断增加,因此也面临着越来越多的安全问题。智能合约安全漏洞事件频繁爆发,使其安全性逐渐成为热点研究问题。

相比传统的应用程序,智能合约作为运行在区块链上的去中心化应用程序,其生命周期及开发语言本身的特性带来了一些特有的安全漏洞。智能合约的源码公开透明这一特点虽然为其可信度提供了保障,但也给黑客带来了可乘之机。智能合约的可信度来源于其不可篡改性,但正因如此,使得智能合约无法像传统程序那样通过打补丁等措施来修复漏洞。

本节对当前典型的智能合约漏洞类型进行总结,并在详细介绍其原理及利用方式后,给出一些相应的防范措施。根据 Atzei 等[11]的一份调查报告,智能合约的安全漏洞可以按照高级语言 Solidity、EVM 和区块链 3 个层面进行分类。高级语言 Solidity 层面的漏洞主要为语言自身设计的缺陷以及开发者在开发过程中引入的错误;EVM 层面的安全威胁主要由以太坊智能合约字节码规范和运行机制本身的一些缺陷带来;区块链层面的问题由区块链本身的很多特性所致。具体如表 1 所列。

表 1 漏洞分类

Table 1 Vulnerability classification

分类	安全漏洞
高级语言 Solidity	整数类型错误
	未校验返回值
	权限控制问题
	拒绝服务
	资产冻结
EVM	重入漏洞
	短地址攻击
	代码注入
区块链	交易顺序依赖漏洞
	时间戳依赖漏洞
	可预测的随机处理

(1)整数类型错误。整数类型错误主要包含算数错误、截断错误和符号错误。算数错误包括整数溢出[12]、除数为零和模数为零这 3 种错误。与其他计算机程序设计语言一样,Solidity 规定使用几种固定的长度表示一定范围内的整数,一旦整数运算结果超出这个范围就会发生整数溢出。攻击者可以利用该类漏洞跳过某些条件判断或者篡改数据,例如著名的 BEC 漏洞事件就是由于合约在转账过程中发生了整数溢出,从而导致巨额的代币蒸发[13]。而使用提供安全检查的 SafeMath[14]数学计算库可以有效避免溢出漏洞。在 EVM 和旧版本 Solidity 中,除数为零和模数为零会导致运算结果为 0,并且不会触发异常。截断错误指将一个整数类型数据转换为宽度更短的整数类型数据导致的精度丢失;符号错误指将一个有符号整数类型数据转换为相同宽度的无符号整数类型数据。

(2)未校验返回值。以太坊 Solidity 语言的函数调用和其他高级语言一样,一般都会设置返回值,但是对 send,call,delegatecall 等低级别函数调用失败时不会引起事务回滚操作,只是在返回值中表示出现了异常。攻击者可以通过故意发送失败的操作来使程序执行与预期设定不同,从而造成智能合约状态混乱。开发者在开发合约时可以通过对低级别函数调用的返回值校验来确定调用是否成功,从而确保合约能以预先设定的逻辑执行。

(3)权限控制问题。Solidity 中可以使用 4 种说明符设置函数和状态变量的可见性,分别为 external,internal,public 和 private。未声明可见性的函数被默认为 public,即该函数不仅允许内部调用,还可作为合约对外接口被外部合约调用。如果一些涉及转账等敏感操作的函数没有声明可见性,就会存在被攻击者利用的风险。

(4)拒绝服务。拒绝服务(Denial-of-Service,DoS)一般指不可恢复的恶意操作或者可控制的无限资源消耗。针对以太坊合约的 DoS 攻击会导致以太币和 gas 被大量消耗,更有甚者会导致永久性地无法使用合约。2016 年的以太坊游戏 The King of the Ether Throne,攻击者利用了外部函数调用的漏洞,通过 Unexpected Revert 发动 DoS 攻击,导致该游戏运营出现重大问题[15]。

(5)资产冻结。由于合约的不可篡改性,如果开发者在进行智能合约开发时仅设置了接收以太币的功能,但没有设置任何允许以太币转出的操作,或由于某些原因使得以太币无法转出,就会造成合约内的资产被永久冻结。

(6)重入漏洞。虽然以太坊智能合约的执行是一个具有原子性和顺序性的事务操作,但是用户在调用智能合约时,如果在被调用合约中没有找到被调用的函数或者该合约只接收到以太币而没有其他任何消息时,就会调用回退函数 fallback。攻击者可以通过构造特殊的回退函数来攻击存在重入漏洞的智能合约,例如回退函数中包含重新调用被攻击合约中向攻击者转账的函数,从而通过递归调用耗尽被攻击合约的资产。导致以太坊硬分叉(ETH/ETC)的著名的事件 The DAO 就与重入漏洞有关。可以采取如下措施规避潜在的重入漏洞:先保证改变状态变量的逻辑发生后,再允许以太币从合约中转出;或将以太币发送至外部合约时,使用内置的 transfer 函数代替 call,send 函数,实现安全的转币操作。

(7)短地址攻击。短地址攻击漏洞是一种因未校验用户输入而导致的漏洞,攻击者利用虚拟机的自动补全机制,构造末尾为零的地址进行合约调用,并在传入参数时故意将地址 address 末尾的零省去,此时虚拟机会取发送代币的金额 amount 高位的 0 对地址补全,同时将 amount 低位补 0,这样就等效于 amount 左移翻倍,导致转移的代币数量超出了原来的设定。通过严格检查用户输入并拒绝接受畸形地址,可以有效避免短地址攻击。

(8)代码注入。代码注入漏洞由以太坊的委托调用机制引入。委托调用机制中使用的 delegatecall 指令允许合约在自己的上下文执行其他合约的代码片段。攻击者可以利用该漏洞向合约注入修改合约中重要状态变量等恶意操作的代码。

(9)交易顺序依赖漏洞。一笔交易被传播出去并被矿工认同写入一个区块内需要一定的时间,因此打包在区块中的交易顺序与交易生成的顺序可以不同。攻击者可通过监视网络上依赖于交易顺序的合约并发出自己的交易来改变当前的

合约状态，从而获益。例如，攻击者可以提交一个悬赏合约，允许用户通过提交难题的答案从该合约获得丰厚的奖励，攻击者可以在提交完悬赏合约后持续监听网络，若有人提交了答案且此时提交答案的交易还未被确认，则攻击者可以立刻发起一个将奖金降低到无限接近于 0 的交易并提供较高的 gas，使自己的交易先被矿工处理，这样攻击者支付很少的奖金就能获得问题的答案。

(10)时间戳依赖漏洞。在智能合约中可以使用区块时间戳作为条件判断依据，这对普通的攻击者来说没有什么利用价值，但是对于可以在一定范围内操纵时间戳的矿工来说，可以通过构造恶意的时间戳来绕过相关的判定条件。

(11)可预测的随机处理。合约开发者编写随机数生成函数时，有时会利用时间戳(block. timestamp)、区块号(block. number)等与区块有关的一些参数产生随机数，但是区块链上的上述数据都是公开的，这使得生成的随机数是可预测的，从而可能会被攻击者利用。

4 智能合约安全漏洞检测传统工具

不同于传统软件应用程序，智能合约一旦部署，即使发现缺陷和错误也无法修改。回滚交易的唯一方法就是硬分叉，即通过修改区块链中的共识协议将区块链恢复到之前的某一状态，但这与区块链的去中心化理念相悖，因此需要在合约实际部署前对其进行全面的安全测试。由于智能合约之间会进行频繁的交互和相互调用，很难在智能合约的运行期间对其进行测试，因此需要在合约实际部署前对其进行测试。

早期人工合约审计主要依赖于经验知识来检查合约代码是否存在安全漏洞与隐患，但是这种方式不仅效率低下，还存在过于主观或者经验不足等问题，因此借助一些漏洞检测工具对合约代码进行审计是十分重要的。针对智能合约的自动化漏洞检测所采取的主要方法有符号执行、模糊测试、形式化验证以及污点分析等，本节将根据上述分类对已有的一些检测工具进行介绍。

4.1 符号执行

符号执行(symbolic execution)是一种白盒分析技术，它可以通过分析程序来得到让特定代码区域执行的输入。其关键思想在于采用符号输入代替具体值输入，探索程序在不同输入下所进入的不同执行路径。使用符号执行检测智能合约漏洞的过程一般为[16]：首先将合约中不确定的值符号化，然后逐条执行程序中的指令，同时更新执行状态；分析器收集相应的路径约束后，对搜集到的路径进行约束求解，判断路径是否可达，并在特定的节点上检测变量的取值，从而寻找潜在的漏洞。

Oyente[17]是 melon. fund 于 2018 年 10 月发布的一款为以太坊智能合约开发人员构建的符号执行工具。该检测工具主要针对 4 种漏洞：条件竞争、时间戳依赖、未校验返回值以及重入漏洞。Oyente 以智能合约字节码和当前以太坊的全局状态作为输入，包含 4 个核心组件(如图 2 所示)：控制流图生成器(CFG builder)、探索器(explorer)、分析器(core analysis)和验证器(validator)。控制流图生成器对合约进行预分析，以基本块为节点，跳转关系为边，为合约构建基本的控制流图；探索器负责收集合约信息，并对合约进行符号执行，利用 Z3 求解器[18]对合约中的条件跳转进行求解，并根据求解结果决定下一步要探索的分支，探索器还会在执行过程中把上一步缺少的跳转关系补全；分析器根据探索器收集到的信息识别合约漏洞；验证器用于过滤分析器产生的误报[19]。Oyente 虽然开创性地提出使用符号执行对合约漏洞进行自动化检测，但其只进行了单次符号执行，因此只能检测由单个交易触发的漏洞。

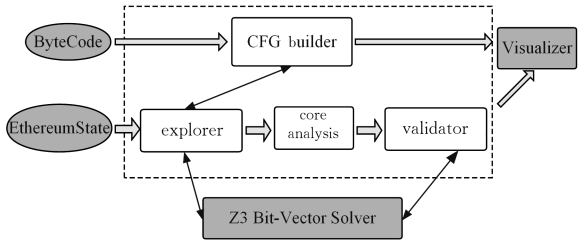

图 2 Oyente 流程图

Fig. 2 Oyente flow chart

Osiris[20]是在 Oyente 符号执行引擎基础上开发的一个用于检测整数型漏洞的漏洞检测框架，并且该工具使用污点分析排除掉不能被实际用于攻击的整数错误，以降低误报率。其主要由 3 部分组成：符号分析模块、污点分析模块和整数错误检测模块。符号分析模块首先构造控制流图，然后符号执行智能合约的不同路径，并将每一条指令的执行结果传递给其他两个组件；污点分析模块引入、传播并检查 stack，memory 和 storage 中的污点；整数错误检测模块检查执行的指令中是否存在整数错误。

Tikhomirov 等[21]提出了一种智能合约的静态分析工具 SmartCheck，该工具通过将 Solidity 解析生成 XML 解析树，并使用 XPath 模式来检测漏洞。Krupp 等[22]开发了 TEETHER 工具，通过 EVM 字节码重构控制流图，并利用路径遍历生成约束模块来查找漏洞。

HoneyBadger[23]同样是基于符号执行的智能合约自动化检测工具，但其主要针对蜜罐合约——一种包含隐藏陷阱的恶意合约的检测。MAIAN[24]用于针对性地分析贪婪合约、浪子合约、自杀合约这 3 种合约的调用序列，并取得了较好的检测效果。此外，Manticore[25]和 Vandal[26]等工具也提供了基于符号执行技术的漏洞检测方案。

基于符号执行的合约漏洞自动化检测，借助约束求解从而较为精确地探索程序的执行路径，并通过分析交易之间的依赖来求解出合适的交易顺序。但是符号执行在合约规模较大、跳转指令较多时，存在着状态空间爆炸等问题。实践中，通常不需要对所有的可执行路径进行检查，而是针对涉及智能合约中易产生高危漏洞的指令路径进行重点检查，从而有效地缩减漏洞空间，提高效率。

4.2 模糊测试

模糊测试技术是一种动态的软件安全检测技术，通过构造随机的测试用例，同时监视程序是否会出现内存损坏等异常来判断程序是否存在潜在安全隐患[27]。但智能合约有许多区别于普通程序的特性，这也为针对智能合约的模糊测试带来了困难。测试用例的生成是模糊测试的关键[28]。传统

应用程序在执行测试用例时，不同测试用例的执行结果互相独立，而智能合约中不仅内部函数可以相互调用，也可以调用外部合约中的函数，合约的状态变量不仅会受自身影响，还会受不可预测的外部调用影响，这导致其难以生成智能合约的测试样例。又因为智能合约中的大部分漏洞不会出现程序崩溃这类的严重结果，从而更难以判定安全隐患的存在，例如当出现整数溢出等问题时，合约却能够继续执行。

ContractFuzzer[29]是首个基于以太坊平台的智能合约安全漏洞模糊测试框架，该工具包括一个线下的 EVM 插桩工具和一个线上的模糊测试工具。该工具支持对 7 种漏洞的检测，但是由于随机生成的测试样例达不到理想的覆盖率，因此容易出现漏报。

sFuzz[30]借鉴传统的模糊测试工具 AFL 并采用反馈自适应模糊策略来提高测试用例的覆盖率。该模糊测试工具主要由 3 个模块组成：runner，libfuzzer 和 liboracles。runner 模块建立了一个私有的测试网络，用来部署测试合约并执行交易；libfuzzer 组件负责生成测试用例；liboracles 组件负责检测测试用例的执行并检查测试样例是否存在漏洞。sFuzz 不仅支持对大多数常见漏洞的检测，还能挖掘出以太币冻结等业务逻辑方面的漏洞。

CONFUZZIUS[31]是首个用于智能合约的混合模糊测试工具，通过测试智能合约的浅层逻辑，约束求解可用于生成满足复杂条件的输入，从而到达更深的测试路径。虽然它利用动态数据相关性分析来有效地生成事务序列，但是这些事务序列更有可能导致其隐藏错误的合同状态。

Harvey[32]基于灰盒并有针对性地根据需求驱动对交易序列的模糊测试，从而提高模糊测试的代码覆盖率以及复杂漏洞的检测效率。

已有的模糊测试工具虽然可以较好地检测合约漏洞，但是其生成的测试样本较为随机，导致代码的覆盖率不够，效率降低。尽管一些工具会针对特定的漏洞生成测试样例，例如 ContractFuzzer 针对重入漏洞设计了一种攻击合约来尝试触发该类漏洞，但是这些工具往往难以进行大规模的扩展，这也是使用模糊测试的漏洞检测工具所面临的问题。

4.3 形式化验证

形式化验证方法是基于形式语言、语义和推理证明的形式逻辑，对计算机软硬件系统进行描述、开发和验证的技术。形式化验证使用逻辑语言对智能合约进行形式化建模，通过严密的数学推理逻辑来证明智能合约功能的正确性和运行时的安全性。基于形式化验证的方法[25]对智能合约进行安全审计的主要流程为：首先通过合适的建模语言和建模工具描述合约，然后设置合约验证性质，最后对合约的状态空间进行搜索，以发现合约存在的安全问题。

Securify[33]可以完全自动化地针对给定的性质来验证智能合约的行为是否安全。该工具可以将智能合约源代码编译为字节码，或者直接将智能合约字节码作为输入，通过符号分析得到依赖关系图，进而得到精确的语义事实，并将语义事实用 Datalog 语法进行描述，然后将这些事实匹配遵从和违反模式，进而将合约行为分为违规、警告和遵从 3 类；最后根据给定的特征分析合约语义信息是否遵循或违背这些特征，来判断合约是否存在漏洞。

VaaS 工具[34]是链安公司采用自有知识产权独立研发的自动形式化验证工具，目前有以太坊智能合约安全检测的在线免费版和可通过 VS Code 插件市场免费获取的插件。该工具能够为智能合约和区块链提供军事级形式化验证服务，可精确到有风险的代码位置并指出存在风险的原因，可有效检测出智能合约的常规安全漏洞、安全属性和功能正确性。

Idelberger 等[35]提出了一种将智能合约可执行规范转换为逻辑语句的形式化验证工具，降低了合约错误的风险。Permenev 等[36]设计了自动验证器 Verx，其主要使用 3 种技术：将时序性简化为可达性检查、符号执行技术以及将谓词抽象和符号执行相结合的延迟谓词抽象技术，这 3 种技术均可以很好地验证合约的功能。智能合约的语义状态[37]同样可以抽象为有限状态机来验证智能合约的正确性。Xu 等[38]建立了系统化安全模型，该模型将智能合约的参与者行为抽象为有限状态机，并对合约中的函数建模，之后使用时序逻辑，查找包括 gas 耗尽、重入漏洞等在内的安全性问题。

形式化验证虽然有可能完全覆盖代码的运行期行为，可以确保一定范围内的绝对安全[39]，在一定程度上弥补了合约审计的局限性，但是此方法需要研究人员拥有较强的数学功底，并且现有的大部分基于自动化的验证方法都是有条件限制的，需要专业人员参与调试，不利于该方法的推广应用。此外，尽管新的形式化验证方法不断出现，目前仍然缺少衡量其优劣程度的标准。

4.4 污点分析

在漏洞挖掘中，通常将程序外部的输入数据标记为污点，然后通过跟踪与污点数据相关的信息流向，就可以知道它们是否会影响某些关键的程序操作，进而挖掘出程序漏洞。污点分析技术的主要作用是实现更加精确的数据流分析，通常需要和其他技术相结合。

EasyFlow[40]是一种基于动态污点分析来判断以太坊智能合约中是否存在整数溢出漏洞的方法，并对 SafeMath 及其可能存在的变种进行了识别，有效降低了误报率。Mythril[41]集成了混合符号执行、污点分析和控制流检查，可以检测出常见的安全漏洞，但是无法检测出智能合约的业务逻辑问题。

本节介绍的现有的工具检测的漏洞类型大都不完整，已有的工具检测的漏洞大都聚焦于重入漏洞、交易顺序依赖漏洞、整数溢出漏洞和时间戳依赖漏洞等智能合约漏洞，如表 2 所列，并且漏洞检测效率较低，误报率和漏报率均较高，例如著名的 Oyente 和 Securify 等工具的准确率均不足 60%。

表 2 检测漏洞工具对比

Table 2 Comparison of vulnerability detection tools

漏洞检测工具	检测漏洞类型
Oyente	交易顺序依赖、时间戳依赖、未校验返回值、重入漏洞
Osiris	整数错误相关漏洞
Securify	重入漏洞、交易顺序依赖、未校验输入、以太币冻结
ContractFuzzer	gas 耗尽终止、异常处理混乱、重入、时间戳依赖、未校验返回值、以太币冻结等
Hydra 框架[41]	整数溢出漏洞
Mythril	重入漏洞、整数溢出、未校验返回值、交易顺序依赖、时间戳依赖等

由于智能合约的漏洞种类较多且成因各异，并且一次针对智能合约的恶意攻击通常由多个漏洞构造而成，这些因素都增加了漏洞检测工具检测复杂合约漏洞和实现更大程度自动化的难度。目前一些工具已经在 GitHub 上发布了源码，如 Oyente[1)]，Slither[2)] 等。

5 基于机器学习的漏洞检测模型

随着被部署在以太坊上的智能合约数量呈爆炸性增长，对合约中的潜在漏洞进行更加准确而更高效的检查变得愈发重要。而上文提到的传统自动漏洞检测工具都需要大量的计算开销，有时甚至需要探索所有路径或是到达一定深度才能够检测到漏洞，非常耗时且检测效率不高。而基于机器学习的检测方法通过对不同形式的合约代码进行自动化的特征提取和恰当的模型训练，显著提高了检测效率，使其具有更好的泛化能力，适用于更多的应用场景，并且相比现有工具在一定程度上提高了对常见漏洞的检测准确率。而处理漏洞检测问题时，将合约视为不同的对象有其对应的不同解决方案。本节将分别根据以下几类问题的转化方式来介绍基于机器学习的漏洞检测方法。第一类做法较为广泛，通常将合约编译后的操作码或者字节码看作是一段文本，从而将其转化为一个文本分类问题；第二类做法的思路是将代码转化为一个非欧几里得图，该图一般包含合约中的数据流和控制流依赖关系，因此可以很好地表征原合约语义，进一步将漏洞检测问题转化为图拓扑结构的识别检测问题；第三类做法是将合约的字节码转化为图片的编码，使得漏洞检测问题被转化为图像的分类识别问题。

5.1 基于自然语言处理方式的漏洞检测

如果将合约代码视为文本，分析其语义和语法关系以及控制流和数据流依赖，再通过恰当的学习模型和任务目标，就可以有效实现漏洞检测、代码相似性检测等。同时，由于传统的漏洞检测方法通过分析执行代码来发现漏洞，因此可以认为分析识别代码的能力与漏洞识别能力正相关。而机器学习模型可以很好地从大量数据集中学习某种模式，因此可以将漏洞检测问题转化为一个有监督的多分类（multi-class）问题或多标签（multi-label）分类问题。自然语言处理的常见流程是先获取语料，然后对语料进行清理，如分词、词性标注、去停用词等，再进行特征化处理并映射为向量，最后选取模型进行训练。类似于文本分类[42]问题，漏洞检测问题也转化为 3 个阶段：首先对源代码或是编译后的操作码进行文本处理，然后进行特征提取得到其向量表示[43]，最后选取合适的模型进行分类。

在文本处理阶段，常见的有以下几类做法：

（1）直接对源代码或是编译后的操作码进行简单规范化，如 Contractward[44]，Soliaudit[45]，ESCORT[46] 等。这种方案尽可能保留了原有的结构和语义特征，通常用于多种类型漏洞的检测。

（2）针对某种漏洞或是语义特性形成代码片段，如文献[47]和文献[48]。前者针对可重入漏洞，去除了注释以及一些和可重入漏洞无关的函数，使其能更好地捕捉控制流和数据依赖；后者则使用数据流分析得到函数调用，从而形成代码分片。这种文本处理方式使得后续的处理更具针对性。

（3）对文本做进一步的处理，如形成抽象语法树、JSON 文件等。如文献[49]、SmartEmbed[50] 和文献[51]都是由解析器生成抽象语法树（Abstract Syntax Tree，AST），作为漏洞查找的对象，其中每个节点表示合约代码的语法元素，提供关于源代码特性的所有细节。Eth2Vec[52] 考虑到编程语言与自然语言的不同，分析 EVM 字节码，从而创建不同级别的 JSON 文件作为输入，以增添语法相关信息。

在特征提取阶段，参考自然语言处理方法，获取代码中间表示通常有以下几种方案：

（1）采用一些经典模型，如 n-gram、tf-idf、Word2Vec、Embedding 层和 PV-DM（Distributed Memory of Model Paragraph Vectors）。Contractward[44]，Soliaudit[45]，Eth2Vec[52] 等分别采用上述模型进行特征化。这种方案具有鲜明的统计学意义，计算简单高效，同时能较好地获取语义信息。

（2）随着循环神经网络（Recurrent Neural Networks，RNN）在自然语言处理领域取得了杰出的成果，对于序列相关的数据，普遍使用在此基础上衍生出来的可以保留更早期信息的长短期记忆网络（Long Short-Term Memory，LSTM）和门控循环单元（Gate Recurrent Unit，GRU）进行特征提取。如 ESCORT[46] 采用嵌入层和门控循环单元层的叠加，既可以很好地捕捉上下文语义和语法关系，又考虑到时序相关信息。文献[53]将循环神经网络中的全局最大池化层的结果作为中间表示。这类特征提取方法采用深度学习方式，使得捕捉的特征信息更全面，后续的应用也更灵活，既可作为传统机器学习的向量输入，也可作为整个训练网络的一部分，从而实现端到端的检测。

（3）针对合约的特性自定义特征，如文献[49]在 AST 基础上抽取了 17 个特征，这些特征被分为两类：一类代表程序执行路径，如函数调用；另一类是代码复杂性的启发式猜测。

分类器选择一般分为传统机器学习模型和基于神经网络两种方法，前者诸如支持向量机（Support Vector Machine，SVM）、逻辑回归（Logistic Regression，LR）、K 最近邻（K-Nearest Neighborhood，KNN）、决策树（Decision Tree，DT）、随机森林（Random Forest，RF）、XGBoost（eXtreme Gradient Boosting）和 Adaboost（Adaptive Boosting），后者较为常见的方案为多层感知器（Multi-Layer Perceptron，MLP）结合 softmax 分类器。基于传统机器学习分类器的方案实现简单、效果好，但比较依赖于文本处理以及特征提取。如 Contractward[44] 针对 6 种常见的安全漏洞预测结果的 Micro-$F1$ 和 Macro-$F1$ 均超过 96%，Soliaudit[45] 对前文提到的几种漏洞进行检测，最终的平均准确率达到 90%。而基于神经网络的判别器可以结合特征提取网络进行整体训练，适配性高、泛化能力好。如 ESCORT[46] 针对自毁检查、断言违规、可重入等漏

1) https://github.com/oyente/oyente

2) https://github.com/crytic/slither

洞的 F1-score 为 95%。文献[47]针对可重入漏洞的平均检测准确率可以达到 73.83%，平均 F1-score 为 67.89%。文献[54]对文中定义的 3 种漏洞的检测平均准确率均达到 91.0%，平均 F1-score 为 90.0%。

5.2 基于非欧几里得图的漏洞检测

智能合约由于编程语言的特殊性，直接进行自然语言处理可能存在词汇来源复杂、依赖关系不只来自于上下文等问题，因此需要进行一些特殊处理。而将源码抽象为一个图，再由图神经网络生成图嵌入，进而完成一些特定的漏洞检测任务，不仅符合程序语言自身的特点，还能更好地捕捉数据流和控制依赖关系。基于图的漏洞检测需重点关注 3 个方面：如何恰如其分地用图来表示代码、如何对图进行特征化表示以及选择恰当的模型来进行漏洞检测。整个流程如图 3 所示。

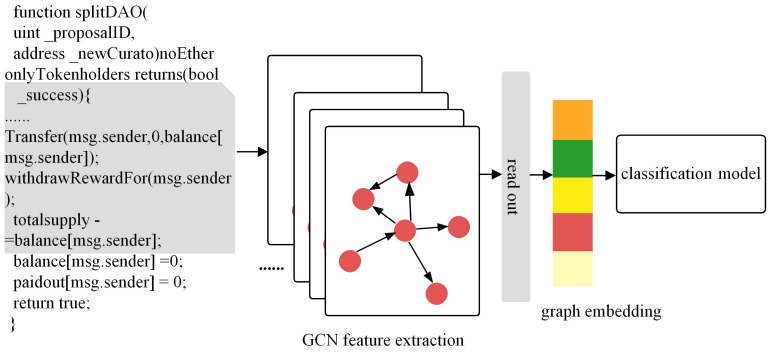

图 3 基于非欧几里得图的漏洞检测流程图

Fig. 3 Flow chart of vulnerability detection based on non-Euclidean graph

为了既能保留完整的语法和结构信息，又能针对目标任务有所侧重地对源代码构建图表征，CCGraph[55]基于程序依赖图(Program Dependency Graph，PDG)进行代码克隆检测。该方法对于合约中代码克隆检测问题同样具有极大的启发性。该模型使用的程序依赖图是一种由源代码构造而来的带标签的有向图，以显示函数之间的依赖关系。其中节点分为语句节点和系统节点，每个语句节点包含语句以及对应类型。边则分为数据依赖边、控制依赖边和执行依赖边。文献[56]提出将增强函数调用图(Enhanced Function Call Graph，EFCG)用于分析函数调用的行为特征，从而进行恶意软件检测。类似地，合约同样因具有频繁调用的特征而导致漏洞频发。具体而言，函数调用图(Function Call Graph，FCG)中每个节点代表一个函数，每一条边代表一个函数调用，但是因为节点不包含函数属性或者功能的含义，仅仅捕捉调用行为很难理解整个程序行为，所以需要为每个函数训练对应的向量表示。文献[57]提出了一种基于图神经网络的智能合约漏洞检测方法，根据 3 种重点检测的漏洞，在由合约转化为图的过程中，设计了对应的节点和边。节点分为主要节点、次要节点和回调函数节点。主要节点一般为自定义函数或内置函数的调用，次要节点用于标记一些关键变量，而回调函数节点则表示触发 fallback 函数行为。边的构造分为控制流、数据流、前向和后向 4 个类型，边上的时序标号表示在函数中执行的顺序。文献[58]采用有向无环图(Directed Acyclic Graph，DAG)来检测合约逻辑漏洞，具体做法为，将待检测的智能合约作为最小可信单元，多个最小可信单元拼接组合形成用户的业务智能合约。这些最小可信单元即为 DAG 中的每个节点，根据节点事件的相互触发形成关联的边，触发的方向决定了 DAG 中边的方向。最终，最小可信单元的区块链智能合约组合模式可以映射为一个 DAG 模型，针对合约逻辑的校验可以在这个 DAG 模型中展开。

不同于图片等可在欧氏空间中表示的数据，越来越多的应用程序和场景中的数据来自非欧几里得域，并被表示为具有复杂依赖关系的图。随之而来的问题是，如何将这种不规则数据映射到向量空间以得到其特征化表示。图卷积网络(Graph Convolutional Network，GCN)[59]于 2016 年被提出，分为基于频谱的方法(spectral-based approaches)和基于空间的方法(spatial-based methods)，并得到广泛应用。在获得图之后，可利用图神经网络来获取对应的图嵌入或者实现图分类。文献[57]在基于图神经网络的智能合约漏洞检测中分别采取了两类方法：无度图神经网络(Degree-free Graph Convolutional Neural Network，DR-GCN)和消息传播网络(Message Passing Neural Network，MPNN)学习图的特征表示，并结合 softmax 分类器实现最终的图分类。BGNN4VD[60]利用双向图神经网络来学习图特征，文献[56]也采用了 GCN 进行图嵌入的学习，从而更好地捕捉图的拓扑信息和节点属性信息。除了直接利用 GCN 进行分类外，还可以在生成图的基础上直接采取近似图匹配等算法，如 CCGraph[55]采用改进的 WL 图核(Weisfeiler-Lehman Graph Kernels)分别对图进行图核计算，再计算图之间的相似度，从而进行克隆检测。随着近年来智能合约数量激增和大批相似合约的产生，原有的单个漏洞随着复制粘贴可能会扩散到成百上千个合约中，因此对合约的克隆检测也逐渐成为保障合约安全的重要任务。

5.3 基于图像的漏洞检测

不同于常规方法致力于定义和提取特征，文献[61]提出了一种新的检测思路，将待检测二进制文件处理为大小固定的彩色图像，利用卷积神经网络(Convolutional Neural Networks，CNN)自动完成特征提取和学习。借鉴此思路，将其应用到智能合约的漏洞检测上，文献[62]将合约的字节码译为 RGB 色彩编码，从而将合约转化为固定大小的编码图像，并将其作为输入，同样通过 CNN 进行训练和检测。但是由于这类方法将源码直接转化为图像，因此卷积和池化操作可能会造成原本不相关的字节码变得相关或是破坏上下文

关系。为了解决这一问题，需要通过实验来选取恰当的 CNN 模型结构（如 AlexNet，GoogLeNet 等）进行优化。

5.4 小结

本节总结了针对不同智能合约漏洞检测问题的转化方式，并指出了针对该种数据进行处理所具有的优势以及一些经典的处理思路。而随着检测要求的进一步提高，集各家之长将逐渐成为未来基于神经网络的漏洞检测模型设计的一个重要方向。如腾讯科恩实验室在二进制相似性检测的研究中，利用 Bert 更全面地捕捉了非欧几里得图节点的语义信息[63]。这种做法区别于传统的人工定义节点特征的方式[64]，可以更好地针对检测任务学习特征，从而进一步优化整个图嵌入的生成结果。类似地，文献[65-66]针对智能合约设计了一套基于专家规则提取语义信息的前馈神经网络和一套提取合约图特征的图神经网络，将这两个特征依次输入具有注意力机制的自动编码器中，从而实现为不同特征自动分配权重，以达到优化检测结果的目的。此外，BGNN4VD[60]在传统 GCN 提取特征的基础上进一步使用 CNN 提取特征，并通过分类器分类来提升检测效果。

结束语 优化智能合约漏洞检测工具及算法在未来依然是重要的研究方向，这些工作将有助于形成一个更加安全可靠的以太坊智能合约环境。传统的面向智能合约的自动化漏洞检测技术虽然已经取得了一定的成果，但仍有不足之处。

现有的传统漏洞检测工具很多，大多采用符号执行、模糊测试和形式化验证等较为成熟的技术。但是它们支持检测的漏洞种类有限，检测手段较为单一，容易出现漏报率和误报率高等问题，因此离不开人工审计的辅助，对于检测出来的疑似漏洞仍需要人工分类为真阳性和假阳性，导致漏洞检测效率较低。未来可以研究将多种技术按照需求综合使用，从而解决之前提到的大规模扩展问题，如使用符号执行辅助弥补模糊测试技术生成样例较为随机的缺陷。并且目前尚未有标准的漏洞测试集，现有的漏洞检测工具都使用自己制作的测试集，未来可以由具有丰富智能合约审计经验的专家根据已存在的漏洞归纳出一个更加完善的智能合约漏洞库，并根据误报率、漏报率、支持检测漏洞类型的多少以及检测效率等指标总结出一套行业评价标准。

近年来，随着人工智能产业的兴起，大量基于机器学习的漏洞检测研究应运而生，覆盖了静态分析、动态分析以及动静态分析结合等典型场景。虽然机器学习方法效率高、泛化能力好，对于特定漏洞的检测准确率更高，但是机器学习方法非常依赖数据，目前各项研究都依赖于各自构建的数据集，如文献[67]致力于构建针对 IEEE 软件异常分类标准[68]中的几种常见漏洞的检测标准和相关数据集，有助于开发者更好地设计漏洞检测工具。文献[69]从安全指示性、可用性、性能、可维护性和可重用性等角度定义了合约缺陷并公开了相应数据集。但是由于部分类型的漏洞并未被频繁利用，可用于训练的数据较少，因此仍未建立一套广受认可的可作为基准的开源数据集。

除此之外，未来的漏洞检测将不仅限于编程语言以及平台自身特性所导致的一些漏洞，会更关注一些更复杂的交易逻辑所带来的安全隐患。

参 考 文 献

[1] WEI A，HUANG Z Y，ZHOU M A. Research on Smart Contract Security and Implementation Specifications[J]. Information Security and Technology，2020，11(3)：44-49.

[2] YUAN Y，WANG F Y. Current status and prospects of blockchain technology development[J]. Acta Automatica Sinica，2016，42(4)：481-494.

[3] YIN M，MALKHI D，REITER M K，et al. HotStuff：BFT Consensus in the Lens of Blockchain[J]. arXiv：1803.05069，2019.

[4] KIAYIAS A，MILLER A，ZINDROS D. Non-interactive proofs of proof-of-work[C]// International Conference on Financial Cryptography and Data Security. Cham：Springer，2020：505-522.

[5] BUTERIN V. A next-generation smart contract and decentralized application platform[EB/OL]. https://the-blockchain.com/docs/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[6] FU M L，WU L F，HONG Z，et al. Research on mining technology of smart contract security vulnerabilities[J]. Journal of Computer Applications，2019，39(7)：1959-1966.

[7] SHIER C，MEHAR M I，GIAMBATTISTA A，et al. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain：The DAO Attack[J]. Social Science Electronic Publishing，2017.

[8] SeeBug[EB/OL]. https://paper.seebug.org/.

[9] BUTERIN V. Ethereum：a next-generation smart contract and decentralized application platform[EB/OL]. https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform-1390528211/.

[10] NI Y D，ZHANG C，YIN T T. A Review of Research on Smart Contract Security Vulnerabilities[J]. Journal of Information Security，2020，5(3)：78-99.

[11] ATZEI N，BARTOLETTI M，CIMOLI T. A Survey of Attacks on Ethereum Smart Contracts (SoK)[C]// International Conference on Principles of Security and Trust. 2017：164-186.

[12] SUN J，HUANG S，ZHENG C，et al. Mutation testing for integer overflow in ethereum smart contracts[J]. Tsinghua Science and Technology，2022，27(1)：27-40.

[13] HESSENAUER S. Batch Overflow bug on Ethereum ERC20 token contracts and SafeMath[EB/OL]. https://blog.matryx.ai/batch-overflow-bug-on-ethereum-erc20-token-contracts-and-safemath-f9ebcc137434.

[14] ARIAS L，SPAGNUOLO F，GIORDANO F，et al. OpenZeppelin[EB/OL]. https://github.com/OpenZeppelin/openzeppelin-Solidity.

[15] KotET-Post-MortemInvestigation[EB/OL]. https://www.kingoftheether.com/postmortem.html.

[16] YE Z B，YAN B. Summary of symbolic execution research[J]. Computer Science，2018，45(s1)：28-35.

[17] LOI L，DUC-HIEP C，HRISHI O，et al. Making Smart Contracts Smarter[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '

16). New York, NY, USA: Association for Computing Machinery, 2016: 254-269.

[18] MOURA L D, LOPES N, WINTERSTEIGER C M. Z3Prover/z3: The Z3 Theorem Prover [EB/OL]. https://github.com/Z3Prover/z3.

[19] ZOU Q C, WU R P, MA J X, et al. Research progress of constraint solving problems in symbolic execution [J]. Journal of Beijing University of Technology, 2019, 39(9): 957-966.

[20] TORRES C F, SCHÜTTE J, STATE R. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts [C]// Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC'18). New York, NY, USA: Association for Computing Machinery, 2018: 664-676.

[21] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. SmartCheck: Static Analysis of Ethereum Smart Contracts [C]// 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). 2018: 9-16.

[22] KRUPP J, ROSSOW C. TEETHER: gnawing at ethereum to automatically exploit smart contracts [C]// Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18). USA: USENIX Association, 2018: 1317-1333.

[23] TORRES C F, STEICHEN M, STATE R. The art of the scam: demystifying honeypots in ethereum smart contracts [C]// Proceedings of the 28th USENIX Conference on Security Symposium (SEC '19). USA: USENIX Association, 2019: 1591-1607.

[24] NIKOLIC I, KOLLURI A, SERGEY I, et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale [C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018: 653-663.

[25] MOSSBERG M, MANZANO F, HENNENFENT E, et al. Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts [C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2019: 1186-1189.

[26] BRENT L, JURISEVIC A, KONG M, et al. Vandal: A Scalable Security Analysis Framework for Smart Contracts [EB/OL]. https://arxiv.org/abs/1809.03981.

[27] ZHANG X, LI Z J. Review of fuzzy testing technology [J]. Computer Science, 2016(5): 1-8.

[28] GODEFROID P, LEVIN M, MOLNAR D, et al. Automated whitebox fuzz testing [C]// Proceedings of the Network and Distributed System Security Symposium, San Diego. https://patricegodefroid.github.io/public_psfiles/ndss2008.pdf.

[29] MASI M. ContractFuzzer: fuzzing smart contracts for vulnerability detection [J]. Computing Reviews, 2019, 60(12): 467-468.

[30] NGUYEN T D, PHAM L H, SUN J. SFuzz: an efficient adaptive fuzzer for solidity smart contracts [C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20). New York, NY, USA: Association for Computing Machinery, 2020: 778-788.

[31] TORRES C F, LANNILLO A K, GERVAIS A, et al. ConFuzzius: A Data Dependency-Aware Hybrid Fuzzer for Smart Contracts [C]// 6th IEEE European Symposium on Security and Privacy. https://akiannillo.github.io/misc/publications/EUROSP2021_Torres.pdf.

[32] WÜSTHOLZ V, CHRISTAKIS M. Harvey: A Greybox Fuzzer for Smart Contracts [C]// Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). New York, NY, USA: Association for Computing Machinery, 2020: 1398-1409.

[33] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: Practical Security Analysis of Smart Contracts [C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). New York, NY, USA: Association for Computing Machinery, 2018: 67-82.

[34] VaaS [EB/OL]. https://www.lianantech.com/.

[35] IDELBERGER F, GOVERNATORI G, RIVERET R, et al. Evaluation of Logic-Based Smart Contracts for BlockchainSystems [C]// International Symposium on Rules and Rule Markup Languages for the Semantic Web. 2016: 167-183.

[36] PERMENEV A, DIMITROV D, TSANKOV P, et al. VerX: Safety Verification of Smart Contracts [C]// 2020 IEEE Symposium on Security and Privacy (SP). 2020: 1661-1677.

[37] ZHU J, HU K, ZHANG B J. A review of formal verification methods for smart contracts [J]. Acta Electronica Sinica, 2021, 49(4): 792-804.

[38] XU W, GLENN A F. Building Executable Secure Design Modelsfor Smart Contracts with Formal Methods [EB/OL]. (2019-12-09) [2021-06-20]. https://arxiv.org/abs/1912.04051.

[39] WANG J, ZHAN N J, FENG X Y, et al. Overview of formal methods [J]. Journal of Software, 2019, 30(1): 33-61.

[40] GAO J, LIU H, LIU C, et al. EASYFLOW: Keep Ethereum Away from Overflow [C]// 2019 IEEE/ACM 41st International Conference on Software Engineering. 2019: 23-26.

[41] Mythril: Security analysis tool for Ethereum smart contracts [EB/OL]. [2021-06-20]. https://pypi.org/project/mythril/.

[42] BREIDENBACH L, DAIAN P, TRAM F, et al. Enter the hydra: towards principled bug bounties and exploit-resistant smart contracts [C]// Proceedings of the 27th USENIX Security Symposium. 2018: 1335-1352.

[43] ZHAO J S, SONG M X, GAO X. Overview of the development and application of natural language processing [J]. Information Technology and Informatization, 2019(7): 142-145.

[44] WANG W, SONG J, XU G, et al. ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts [J]. IEEE Transactions on Network Science and Engineering, 2021, 8(2): 1133-1144.

[45] LIAO J, TSAI T, HE C, et al. SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testing [C]// 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). 2019: 458-465.

[46] LUTZ O, CHEN H, FEREIDOONI H, et al. ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning [EB/OL]. [2021-06-

20]. https://arxiv.org/abs/2103.12607.
[47] QIAN P, LIU Z, HE Q, et al. Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models[J]. IEEE Access, 2020, 8: 19685-19695.
[48] WANG R, YE K J, XU C Z. A smart contract vulnerability detection method based on deep learning: China Patent, 2019112576541[P]. 2020-05-15.
[49] MOMENI P, WANG Y, SAMAVI R. Machine Learning Model for Smart Contracts Security Analysis[C]// 2019 17th International Conference on Privacy, Security and Trust (PST). 2019: 1-6.
[50] GAO Z, JAYASUNDARA V, JIANG L, et al. SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding[C]// 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2019: 394-397.
[51] WENG J, CHEN X K, LI M, et al. An intelligent contract security vulnerability detection method based on machine learning: China Patent, 2019109045392[P]. 2020-01-31.
[52] ASHIZAWA N, YANAI N, CRUZ J P, et al. Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts[C]// Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '21). New York, NY, USA: Association for Computing Machinery, 2021: 47-59.
[53] ZHOU Y J. A method, device and storage medium for fuzz testing of smart contracts: CN Patent, 112131115[P]. 2020-09-23.
[54] GOGINENI A K, SWAYAMJYOTI S, SAHOO D, et al. Multi-Class classification of vulnerabilities in Smart Contracts using AWD-LSTM, with pre-trained encoder inspired from natural language processing[EB/OL]. (2020-03-21)[2021-06-20]. https://arxiv.org/abs/2004.00362v1.
[55] ZOU Y, BAN B, XUE Y, et al. CCGraph: a PDG-based code clone detector with approximate graph matching[C]// 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020: 931-942.
[56] MC A, YUAN J, CG A, et al. Learning features from enhanced function call graphs for Android malware detection[J]. Neurocomputing, 2021, 423: 301-307.
[57] ZHUANG Y, LIU Z, QIAN P, et al. Smart Contract Vulnerability Detection using Graph Neural Network[C]// Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. 2020: 3283-3290.
[58] HAN Z G, YUAN L, GENG J H, et al. Smart contract vulnerability detection methods, devices and electronic equipment: China Patent, 2018101589863[P]. 2020-07-17.
[59] WU Z, PAN S, CHEN F, et al. A comprehensive survey on graph neural networks[J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 32(1): 4-24.
[60] CAO S C, SUN X B, BO L B, et al. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection [J]. Information and Software Technology, 2021, 136: 106576.
[61] HUANG H D, KAO H Y. R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections [C]// 2018 IEEE International Conference on Big Data (Big Data). 2018: 2633-2642.
[62] HUANG H D. Hunting the Ethereum Smart Contract: Color-inspired Inspection of Potential Attacks[J]. arXiv: 1807.01868, 2018.
[63] YU Z, CAO R, TANG Q, et al. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection[C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2020: 1145-1152.
[64] XU X, CHANG L, QIAN F, et al. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection [C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17). 2017: 363-376.
[65] HUANG B T, DING J, QIAN P, et al. An interpretable method for smart contract vulnerability detection based on codec: China Patent, 2020108267923[P]. 2020-12-04.
[66] LIU Z, QIAN P, WANG X, et al. Smart Contract Vulnerability Detection: From Pure Neural Network to Interpretable Graph Feature and Expert Pattern Fusion[C]// Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence. 2021: 2751-2759.
[67] ZHANG P, XIAO F, LUO X. A Framework and DataSet for Bugs in Ethereum Smart Contracts[C]// 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2020: 139-150.
[68] IEEE Computer Society. IEEE Standard Classification for Software Anomalies[S]. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), 2010.
[69] CHEN J, XIA X, LO D, et al. Defining Smart Contract Defects on Ethereum[J]. arXiv: 1905.01467, 2019.

ZHANG Ying-li, born in 1997, postgraduate. Her main research interests include web security and blockchain security.

ZHOU Rui, born in 1981, associate professor. His main research interests include distributed systems, embedded systems and machine learning.

（责任编辑：柯颖）