

基于形式化方法的智能合约验证研究综述

张文博^{1,2}, 陈思敏¹, 魏立斐¹, 宋巍¹, 黄冬梅³

(1. 上海海洋大学信息学院, 上海 201306; 2. 上海市高可信计算重点实验室, 上海 200062; 3. 上海电力大学, 上海 201306)

摘要: 智能合约是区块链技术应用的一个重要场景, 智能合约技术实现了区块链的可编程化, 提高了其扩展性, 有广阔的应用前景。然而, 一系列关于智能合约的安全事件造成了大量经济损失, 削弱了人们的信心, 安全性问题已经成为制约智能合约进一步发展的关键问题。如果合约设计和代码实现的过程中存在缺陷, 可能会造成严重后果。而智能合约发布后无法修改, 因此, 在智能合约发布前对其正确性做出验证尤为重要。近年来, 国内外学者在智能合约的验证领域取得了大量成果, 但对这些研究成果的系统分析和总结相对较少。对以太坊的交易过程、gas 机制、存储结构、编写语言做了简要介绍, 在此基础上调查归纳了智能合约中常见的8种漏洞类型, 解释了漏洞产生的原因, 回顾了一些真实发生的安全事件并给出了漏洞示例代码; 根据不同的技术手段, 如符号执行、模型检测、定理证明等, 对智能合约的形式化验证工作做分类介绍, 分析了各种方法的优劣, 并选取了3个开源的自动化验证工具 Mythril、Slither 和 Oyente, 从运行效率、检测漏洞类型以及准确率等方面作出实验评估和对比; 研究了目前已有的相关综述文章, 总结了这些研究的区别与优势; 概述了智能合约的漏洞检测技术中仍存在的 key 问题, 对智能合约验证工作的现状进行了分析和展望, 提出了未来能够进一步研究的方向。

关键词: 智能合约; 形式化方法; 区块链; 以太坊; 程序验证

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2022041

State-of-the-art survey of smart contract verification based on formal methods

ZHANG Wenbo^{1,2}, CHEN Simin¹, WEI Lifei¹, SONG Wei¹, HUANG Dongmei³

收稿日期: 2022-05-16; 修回日期: 2022-07-04

通信作者: 黄冬梅, dmhuang@shou.edu.cn

基金项目: 国家自然科学基金 (61872142, 62102243, 61972241); 上海市青年科技英才扬帆计划 (21YF1417000); 上海市高可信计算重点实验室开放课题; 上海市自然科学基金面上项目 (18ZR1417300); 上海市科委部分地方高校能力建设项 (20050501900); 上海海洋大学青年教师科研启动项目

Foundation Items: The National Natural Science Foundation of China (61872142, 62102243, 61972241), Shanghai Sailing Program (21YF1417000), The Open Project of Shanghai Key Laboratory of Trustworthy Computing, Shanghai Natural Science Foundation (18ZR1417300), The Program for the Capacity Development of Shanghai Local Colleges (20050501900), Startup Foundation for Young Teachers of Shanghai Ocean University

引用格式: 张文博, 陈思敏, 魏立斐, 等. 基于形式化方法的智能合约验证研究综述[J]. 网络与信息安全学报, 2022, 8(4): 12-28.

Citation Format: ZHANG W B, CHEN S M, WEI L F, et al. State-of-the-art survey of smart contract verification based on formal methods[J]. Chinese Journal of Network and Information Security, 2022, 8(4): 12-28.

1. College of Information Technology, Shanghai Ocean University, Shanghai 201306, China
2. Shanghai Key Laboratory of Trustworthy Computing, Shanghai 200062, China
3. Shanghai University of Electric Power, Shanghai 201306, China

Abstract: Smart contract represents an essential application scenario of blockchain technology. Smart contract technology improves programmability and scalability of blockchain, and has broad development prospects. However, a series of security incidents caused a great number of economic losses and weakened users' confidence in the Ethereum platform. The security of smart contract has become a critical problem that restricts the further development of smart contract. Defects in smart contract code may cause serious consequences and cannot be modified once deployed, it is especially important to verify the correctness of smart contract in advance. In recent years, researchers have obtained many achievements in verification of smart contract, but there is a lack of systematic summary of these research results. Therefore, some basic principles of Ethereum were introduced, including the transaction, gas mechanism, storage and programming language. Eight common types of vulnerabilities in smart contract were summarized and their causes were explained. Some real security events were reviewed and some examples of vulnerability codes were presented. Then, the research work on automatic verification of smart contract based on symbolic execution, model checking and theorem proving was classified and summarized. Three open-source automated tools were selected, including Mythril, Slither and Oyente. And experiments were implemented to evaluate and compare the three tools from the aspects of efficiency, accuracy and the types of vulnerability can be detected. Furthermore, related review articles were surveyed, and the advantages of this paper compared with these works were summarized. The critical problems in the vulnerability detection technology of smart contract were also summarized and the direction of future research was proposed at last.

Keywords: smart contract, formal methods, blockchain, Ethereum, program verification

0 引言

2008年,中本聪在其发表的文章中第一次阐述了区块链技术^[1]。早期的区块链平台只能实现数字货币的简单交易,但区块链技术的去中心化、匿名性、不易篡改性等优势^[2],以及在长期实践中表现出的安全性和稳定性,使越来越多的研究者开始探索区块链技术更广阔的应用场景。智能合约便是较为成功的一个应用场景。智能合约的概念最早由密码学家 Szabo^[3]提出,是指以数字形式存储的,可以自动执行合约条款的计算机交易协议。区块链技术^[1]为智能合约的实现提供了解决方案:将智能合约的代码写入区块链中,自动按照代码内容执行合约条款,并为智能合约提供去中心化分布式的存储介质,保证智能合约代码的不易篡改性。同时,智能合约为区块链提供更丰富的业务场景。以太坊是第一个大规模应用智能合约功能的区块链平台^[2],目前已经成为仅次于比特币的第二大区块链交易平台。

智能合约在金融^[4]、医疗^[5-6]、物联网^[7]、政

府管理^[8]、能源^[9]、供应链管理^[10]等领域都显示出了广阔的应用前景,但伴随的安全问题层出不穷^[11]。攻击者能够利用合约代码中的漏洞获取较大的经济利益。2016年6月,众筹项目 The DAO 合约代码中的重入漏洞被攻击者利用,窃取了价值约4 000万美元的以太币^[12]。此次事件造成了深远影响,以太坊社区关于解决方案的分歧,导致了以太坊的硬分叉,并且引发了关于智能合约安全性的信任危机。2017年7月,多重签名钱包项目 Parity 中价值数千万美元的以太币被窃取^[13]。同年11月,Parity 钱包中价值上亿美元的以太币被冻结^[14]。以上多次漏洞事件表明,对智能合约的正确性做严格的验证是十分必要的。

形式化验证基于严格的数学基础,验证软件系统和硬件系统满足特定的安全性质,在航空航天、核电、高铁动车等安全攸关的领域中有难以取代的价值^[15]。由于智能合约代码上链后无法修改,出现漏洞则可能带来重大损失,因此在代码上链前做安全性验证十分必要。合约代码通常体量较小,能够避免形式化方法在处理大规模程序

时的效率短板。因此,形式化方法非常适合于解决智能合约的安全问题。

近年来,基于形式化方法的智能合约验证研究取得了大量研究成果。在 IEEE、ACM、Springer、Elsevier 等论文数据库中以“smart contract”“blockchain”“formal verification”等关键词的组合进行检索后,本文从中筛选出了较为重要的文章,并对这些研究成果做出系统的分析和总结。

1 以太坊概述

本节将介绍与以太坊相关的一些背景知识,这些知识是理解后文中智能合约漏洞产生原因的重要基础,对于以太坊中基本概念的更全面的介绍可参考文献[2]。

1.1 以太坊简介

2013 年,Buterin 在以太坊白皮书^[2]中描述了一个智能合约和去中心化的应用平台。以太坊已经成为规模最大的、面向用户的智能合约开发平台,也是市值仅次于比特币的区块链交易平台^[16]。以太坊的发展经历了 4 个阶段,包括 Frontier(2015 年)、Homestead(2016 年)、Metropolis(2017—2019 年)和 Serenity(2020 年至今)。

相较于比特币系统,以太坊平台不仅借助区块链技术来维护去中心化支付,并且支持用户编写智能合约以完成复杂的合约逻辑。智能合约是运行在区块链上的一段程序代码^[3],可在去中心化的环境中执行。任何用户都可编写智能合约,并将智能合约部署到区块链上^[17],这一过程是不可逆的,用户可以通过外部账户调用合约中的函数,实现与以太坊的交互。智能合约可以通过向其他合约发送消息或调用函数来完成交易,全网节点交易验证通过后,将状态改动记录在区块中。

1.2 以太坊的交易

以太坊中包含两类账户:外部账户(EOA, externally owned accounts)与合约账户(CA, contracts accounts)^[2]。EOA 即用户账户,由公-私钥对控制,公钥经过哈希运算后可得到外部账户的地址。CA 由智能合约控制,除了包括 160 位的地址信息和以太币余额信息外,还包括一段可执行的代码和一部分私有的存储空间^[3]。合约账户无法主动发起交易,但可通过合约之间的调用完

成合约功能^[17]。以太坊中两种交易的方式如图 1 所示。第一种是采用创建合约的方式,利用 Solidity 语言编写智能合约源代码,再编译为智能合约字节码,并向“0”地址发送包含智能合约字节码的交易来生成合约地址。第二种是采用消息调用的方式。

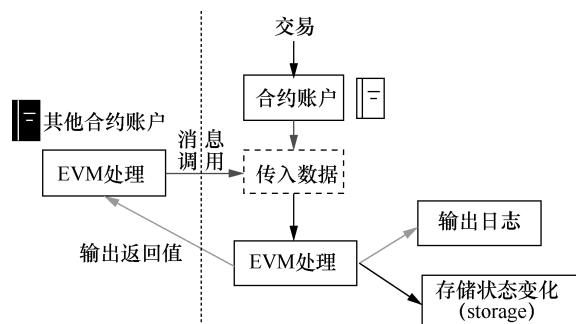


图 1 以太坊中的交易方式
Figure 1 Transactions way in Ethereum

相较于比特币系统直接进行的简单交易,在以太坊中,外部账户将经过数字签名的数据包发送到另外的账户,账户状态发生变化并被记录到区块链中,在此过程中合约账户通过消息调用与其他合约账户交互。交易结束后系统状态发生改变。以太坊中的去中心化应用程序(Dapp, decentralized application)可由交易或消息触发,消息可由交易或以太坊虚拟机(EVM, Ethereum virtual machine)字节码触发。消息触发方式有 4 种,如图 2 所示。当某些被调用的功能是恶意的,消息触发不当可能会导致漏洞。交易具有原子性,无法撤回和打断执行过程。以太坊账户发起的交易包括 3 类:创建新合约;直接向其他合约或用户转账;调用其他合约中的某个可执行函数。当出现第 3 类交易时,合约代码将载入以太坊虚拟机中运行。当交易被矿工节点打包提交并得到其他节点的共识,区块链上所有的节点都会执行该合约代码^[18]。

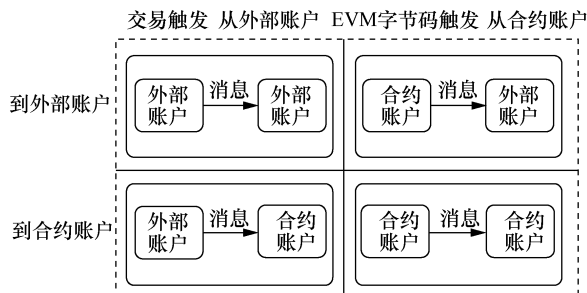


图 2 4 种消息触发方式
Figure 2 Four ways of triggering a message

1.3 以太坊的 gas 机制

EVM 字节码中除了 return 与 stop 以外的任意指令都需要支付 gas 费。读操作指令不改变区块链的状态，执行速度较快，gas 费较低^[19]；而写操作指令会导致账户的存储状态改变，执行代价较高，因此 gas 消耗量较高。交易的创建者可以指定 gasPrice 和 gasLimit 两个变量。其中 gasPrice 表示在交易执行时消耗的每单元 gas 所支付的以太币数额，gasLimit 表示交易执行所消耗的以太币上限。而当消耗的以太币达到上限时，程序将无法继续执行并回退到初始状态。gas 机制保证了合约程序总会在有限步之内终止。

以太坊的 gas 机制如图 3 所示，以太坊中的不同操作会消耗 gas，如操作指令、创建合约或进行消息调用、使用存储结构等，且不同情况下 gas 的消耗量也是不同的。当操作回退时，多余的 gas 会返还给账户，但已消耗的 gas 无法退还。

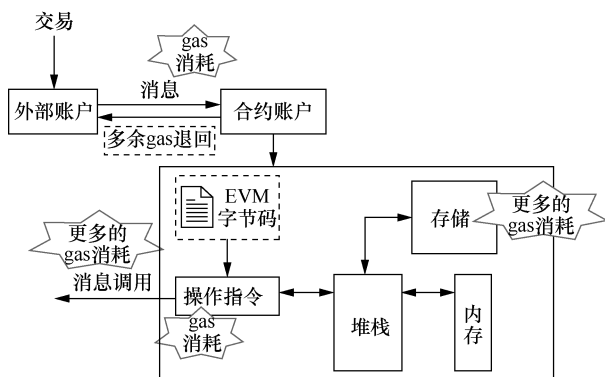


图3 以太坊的 gas 机制
Figure 3 The Ethereum's gas mechanism

类似于传统的区块链系统，以太坊中的记账工作是由独立的矿工节点完成的^[18]。矿工节点收集用户提交的交易信息，将交易信息打包成区块并提交到链上。这些交易信息可能会触发对合约代码的调用。合约代码执行时消耗的 gas 费会成为矿工的收益，因此矿工节点通常优先打包 gasPrice 值较高的交易。

1.4 以太坊虚拟机的存储结构

EVM 作为智能合约和以太坊操作系统的中介，将由高级程序语言编写的智能合约源代码编译为可在 EVM 上自动执行的、低级的机器操作码^[3]，并将机器码与执行环境隔离。EVM 中包含 3 种数据存储结构：易失性堆栈 (stack)、易失性内

存 (memory) 以及非易失性存储结构 (storage)^[20]。了解 EVM 的存储结构有助于理解整数溢出^[11]、变量类型声明不当等漏洞产生的方式和原因。

EVM 的堆栈用于存储值类型的局部变量，是易失性的存储结构。从图 4 可以看出，堆栈深为 1 024，当递归调用深度超过 1 024 时，会产生相应的溢出错误。对栈的访问仅限于其顶部的 16 层，且不会造成 gas 消耗。当合约中声明保存的局部变量超过 16 时，会造成栈溢出的错误，EVM 可以使用除 STOP、JUMPDEST 和 INVALID 外的操作码对堆栈进行数据的读取或者写入，使用 POP、PUSH、SWAP 等指令进行操作。

EVM 的内存与普通 CPU 的内存类似，用于临时保存代码运行时的指令和数据，如缓存的数据或子合约调用时函数产生的临时变量^[20]。如图 4 所示，内存位宽为 8 bit，按字节数组形式存储，并按字节级寻址。内存作为易失性的线性存储结构，当 EVM 重新开始执行时，内存中保存的数据将被清空，程序计数器归 0。在以太坊的内存中进行数据的读取和写入时，可选择 256 bit 或 8 bit 作为单位，对 gas 的消耗相较于存储更低，通过 MSTORE、MSTORE8、MLOAD 指令访问。

EVM 的存储作为非易失性存储器，所记录的数据被同步到区块链中，如图 4 右侧所表示的存储区的结构，存储区的数据按照键值对应存储，并以 256 bit 进行读取或写入，当没有更小的存储空间时，单值存储 uint8 与 uint256 类型的数据消耗的 gas 相同，因此智能合约验证过程中，会判断是否存在 gas 消耗可优化的情况。存储区中保存了合约状态和合约代码，如合约中声明的公共状态变量和越过局部作用域后无法访问的局部变量。EVM 通过 SLOAD、SSTORE 等指令对存储进行操作，storage 类型的参数通常采用指针传递，对存储进行读写会消耗大量 gas。

1.5 智能合约高级语言

Solidity 是以太坊中最常用的编写智能合约的高级语言^[17]，其语法类似于 Javascript。Solidity 支持如继承、库函数、用户自定义类型等较为复杂的特性。

当智能合约的编写中对变量的引用类型声明错误时，会造成安全漏洞。如果在编写合约时，

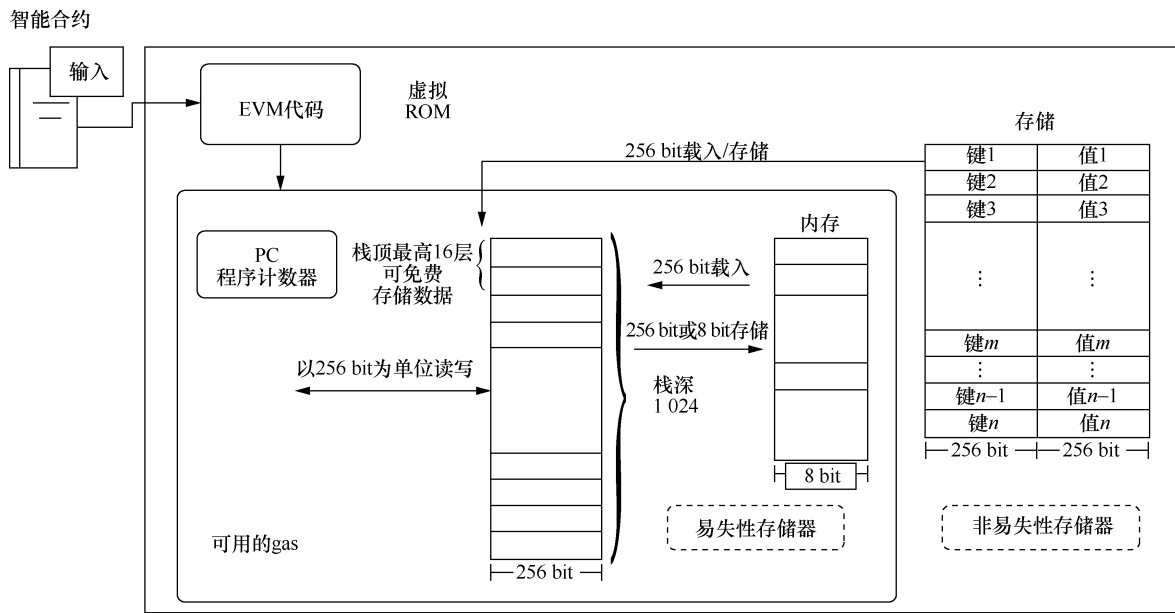


图 4 以太坊的执行资源及存储结构
Figure 4 Execution resources and storage structure of Ethernet

混乱地使用引用类型，将会造成函数的类型声明错误，进而影响合约运行。Solidity 中数据的类型通常分为值类型（如布尔类型、地址、函数等）和引用类型（如数组、结构体等）。对于引用类型，需要说明数据的存储位置。编写合约时，不同的引用类型可按照默认位置进行存储，也可以通过声明类型名关键字 `storage` 或 `memory` 的方式修改存储位置。其中，编译合约时会将会合约声明的公共状态变量强制为 `storage` 的存储结构，交易通过

调用函数修改状态变量的值，从而改变交易状态。局部变量默认为存储^[2]。在存储中，状态变量通过引用传递赋值给局部变量，对局部变量的修改可影响状态变量，因此不能将内存中的变量赋值给局部变量，否则会影响状态变量的值。图 5 反映了账户与交易、区块的联系，并且展示了区块中包含的信息。

`storage` 中包含合约代码中全局变量的值，当一个交易执行成功后，全局变量更新后的值将被

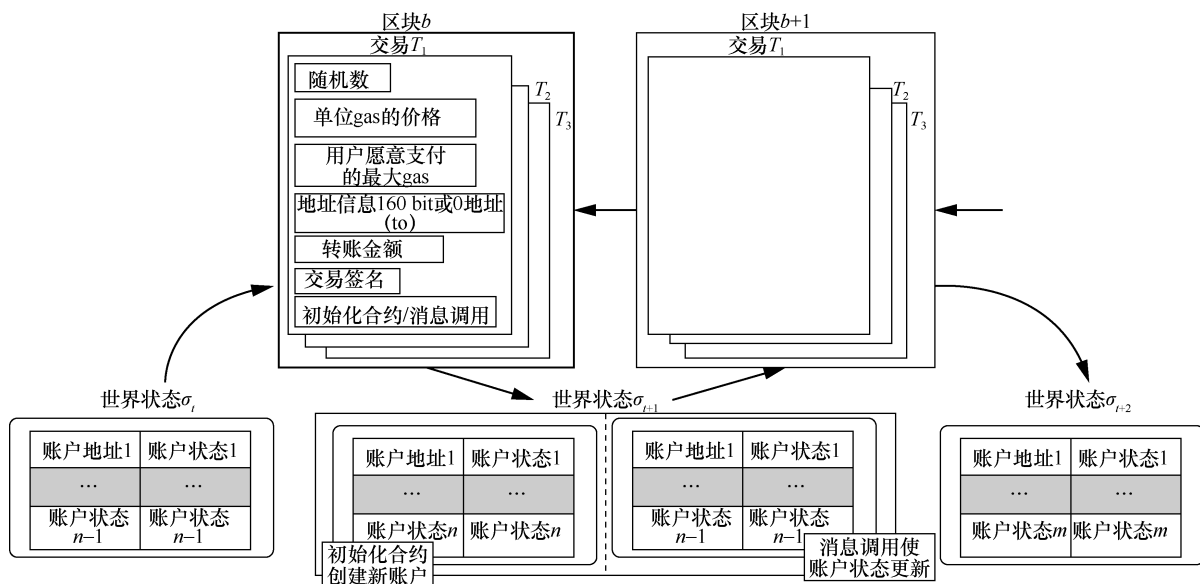


图 5 以太坊中的交易与世界状态
Figure 5 Transactions and state of the world in Ethereum

记录在区块链上。`storage` 和账户余额 `balance` 共同构成了合约账户当前的状态，整个 EVM 的状态被称为世界状态，是从账户地址到账户状态的映射。

2 智能合约的漏洞分类

智能合约中漏洞产生的原因不尽相同。本节参考了 SmartContractSecurity 和 CWE 漏洞库，将智能合约的常见漏洞进行分类，包括未声明函数的可见性、重入漏洞、整数溢出漏洞、代理调用漏洞、`tx.origin` 认证漏洞、异常处理不当、时间戳依赖、交易顺序依赖等。以下将分别对这些漏洞做简要介绍。

2.1 未声明函数

Solidity 的合约代码中规定了函数的 4 种可见性声明，包括 `public`、`private`、`external`、`internal`，在未指定智能合约函数访问权限时默认为 `public`，声明为 `public` 函数的允许被内部函数和外部函数调用；声明为 `external` 的函数可被其他外部函数调用，但不能被合约内的其他函数调用；声明为 `internal` 的函数可被本合约及其子合约调用；声明为 `private` 的函数仅可在本合约中被调用，外部合约不可访问此类函数或更改数据。表 1 中对不同函数的可见性做出总结。

表 1 函数的可见性总结
Table 1 Summary of function visibility

函数可见性	本合约	子合约	外部合约
public(默认)	√	√	√
private	√	×	×
internal	√	√	×
external	×	×	√

注：“√”表示可被调用，“×”表示不可被调用、访问或更改。

在编写合约代码时，不正确地声明函数的可见性，尤其是不恰当地使用可见性默认为 `public` 的函数，可能会导致黑客能够随意调用函数，破坏智能合约的执行逻辑，产生安全漏洞。

除了函数的可见性声明可以约束函数的调用，修饰器 `modifier` 也能起到约束函数行为的作用。在合约逻辑层面，一般使用关键字 `modifier` 添加函数约束，用于在函数执行之前对所有权的检查，如修饰符 `onlyOwner` 会检查当前是否为部

署者执行函数调用，未添加修饰符可能会造成非合约部署者操作关键函数，从而导致漏洞。例如，为了实现代码复用，以太坊中提供了 `delegatecall` 指令用于调用外部的库函数。库函数中关键函数被默认为 `public`，任意攻击者都可以调用函数窃取账户余额，导致 `parity` 多重签名钱包的项目中，被窃取了价值 3 100 万左右美元的以太币^[13]。

2.2 重入漏洞

以太坊中的智能合约可以相互调用完成交易。当合约 C1 在调用合约 C2 后，必须要等 C2 执行完毕，C1 才会继续执行。重入漏洞是指 C2 在被调用后利用合约 C1 所处状态可能存在的不一致问题，对合约 C1 发起攻击。在 The DAO^[12] 事件中，攻击者利用的就是合约代码中的重入漏洞。图 6 表示一个重入漏洞的代码示例。恶意用户可以构建合约 `Attacker`，先调用 `Wallet` 合约的 `deposit` 函数存入一定的金额，再调用 `withdrawAll` 函数将金额取出，`withdrawAll` 函数的第 7 行执行向 `Attacker` 转账的命令。在以太坊中，当合约收到以太币的转账时会自动执行一个无函数名、无参数以及无返回值的回退（`fallback`）函数。`Attacker` 合约的第 7 行到第 9 行重写了回退函数，再次调用 `Wallet` 的 `withdrawAll` 函数，但此时 `Wallet` 合约还没有执行第 8 行对 `Attacker` 账户余额的置 0 操作，这个暂时的不一致状态被 `Attacker` 所利用，导致 `Wallet` 合约的所有余额被转入 `Attacker` 的账户中。重入漏洞造成的损失较大，是以太坊中频繁发生的安全事故，因此，重入漏洞是各类工具检测的重点。

1) contract Wallet{	1) contract Attacker{
2) mapping (address => uint) balances;	2) Wallet dao;
3) function deposit() public payable{	3) function attack() payable{
4) balances[msg.sender] += msg.value;	4) dao.deposit.value(1)();
5) }	5) dao.withdrawAll();
6) function withdrawAll() public{	6) }
7) require(msg.sender.call.value(msg.sender));	7) function() public payable{
8) balances[msg.sender]=0;	8) dao.withdrawAll();
9) }	9) }
10) }	

图 6 重入漏洞示例

Figure 6 An example of the reentrancy bug

2.3 整数溢出漏洞

以太坊虚拟机与通用计算机的存储空间特性类似, 编程语言中的整数类型长度在存储空间中具有存储限制。Solidity 语言中的整型变量步长以 8 递增, 支持从 uint8 到 uint256 类型的数。图 4 表示了 EVM 的存储结构。EVM 中的存储空间最多存储 uint8 的无符号整数类型, 当需要存储的数据超过 EVM 数据空间的范围时, EVM 会执行截断, 可能会造成溢出。溢出类型通常有乘法溢出、加法溢出、减法溢出。当智能合约缺少完善的数值检测语句, 在算术运算时容易出现上溢或下溢的错误。例如, 2018 年, 美链合约未使用标准数学运算库 SafeMath 库中定义的运算, 在交易过程中直接使用了乘法运算符而没有进行相应的溢出检测, 造成整数溢出漏洞^[21], 最终导致 BEC 合约的价值蒸发。

2.4 代理调用漏洞

以太坊中存在多个调用函数类型, 其中委托调用函数 `delegatecall` 可直接使用外部合约的库代码, 以实现代码复用。攻击者可以在外部合约中恶意注入有害代码, 当被调用的函数违反合约逻辑时, 会造成极大的安全隐患。

2018 年 4 月, parity 多重签名钱包经历了第二次攻击^[14], 如图 7 所示, 代理调用漏洞可能伴随函数未声明可见性的漏洞。由于 `walletLibrary` 可见性声明为 `public`, 攻击者通过调用 `WalletLibrary` 合约中的 `initWallet()` 函数, 成为该库的 `owner`, 随机调用 `kill()` 函数删除该库函数。钱包内的以太坊转出完全依赖于 `walletLibrary`, `walletLibrary` 的销毁使得钱包余额被冻结。

```

1) contract Wallet{
2)   address constant walletLibrary;
3)   function withdraw(){
4)     walletLibrary.delegatecall(msg.data)
5)   }
6) }
```

图 7 代理调用漏洞示例
Figure 7 An example of mishandled exception

2.5 tx.origin 认证漏洞

Solidity 语言中, `tx.origin` 作为全局变量, 可

以返回最初发送交易的地址。当合约使用 `tx.origin` 授权声明所有者的权限时, 只能返回最初的交易发起者的账户地址。例如, 用户 A 调用合约 C5, 合约 C5 再调用合约 C6。对于合约 C6 来说, `msg.sender` 地址唯一, 为合约 C5 的地址, 但实际上调用 C6 的是 `tx.origin` 所声明的用户 A 的地址。具体的 `tx.origin` 漏洞示例如图 8 所示。合约 `Wallet` 的 `withdraw` 函数中用 `tx.origin` 来确认交易的发起者就是账户的所有者。此时, 如果合约 `Wallet` 向 `Attacker` 发起任意一笔转账时, 会自动调用 `Attacker` 合约第 3 行的回调函数, 进一步调用合约 `Wallet` 的 `withdrawAll` 函数。由于交易最初的发起人确实是 `Wallet` 的所有者, 因此第 7 行的条件验证可以通过, 合约 `Wallet` 的所有余额都将被转给 `Attacker` 合约。攻击者利用 `tx.origin` 声明所有者的权限, 通过恶意调用绕过授权检查, 可能导致重入漏洞。

```

1) contract Wallet{                               1) contract Attacker{
2)   address user;                                   2)   address user;
3)   constructor()public{                             3)   function ()payable{
4)     user = msg.sender;                             4)     Wallet(msg.sender).
5)   }                                                withdraw All(user);
6)   function withdrawal (address 5)  }
a) public{                                           6) }
7)   require(tx.origin==owner);
8)   a.transfer(this.balance)
9)   }
10) }
```

图 8 tx.origin 认证漏洞示例
Figure 8 An example of mishandled exception

2.6 异常处理不当

智能合约在以太坊的运行中具有顺序性和原子性, 当存在如下 3 种情况时会抛出异常: 执行过程中耗尽 `gas` 值; 调用栈深超过 1 024; 采用 `assert` 与 `require` 函数进行条件检查后抛出 `revert` 函数或 `throw` 关键字。

执行合约出现异常时会进行回滚。合约调用其他合约时可能无法识别其他合约的错误标识。例如, 智能合约的底层调用函数 `send`、`call` 和 `delegatecall` 等发生异常时只返回 `false`, 而不会抛出异常。在以太坊中, 当合约 C1 在调用合约 C2 时, 如果 C2 在执行过程中出现了异常, C2 的状

态将回退并返回 `false`，此时合约 C1 应当检查智能合约 C2 的返回情况再继续执行，否则可能造成合约状态的不一致。KoET 智能合约是存在异常处理不当问题的典型例子^[22]。KoET 是基于以太坊的智能合约游戏，参与游戏的玩家需支付一定数量的以太币给现任“国王”称号的所有者。图 9 是一段 KoET 合约代码的示例。这段代码的问题在于没有检查第 7 行中的调用是否执行成功，攻击者可以故意构造出异常来获利。例如，通过以太坊虚拟机对调用栈深度不超过 1 024 的限制，攻击者可以通过自调用使调用深度达到 1 023 再调用 `claimThrone` 函数竞争“国王”称号，此时转账将会因为超出栈的调用深度而失败，原“国王”称号的所有者将失去称号但无法收到相应的报酬。上述情况说明，每当出现外部调用时都应当对调用结果做检查，当调用成功后再继续执行后续操作。

```

1) contract KingOfTheEtherThrone{
2)     address public currentKing;
3)     uint public claimPrice;
4)     function claimThrone (){
5)         if (currentKing != msg.sender && claimPrice <=
           msg.value)
6)             uint compensation = generateCompensation();
7)             currentKing.call.value(compensation)();
8)             currentKing= msg.sender;
9)     }
10) }
```

图 9 异常处理不当示例
Figure 9 An example of mishandled exception

2.7 时间戳依赖攻击

时间戳依赖指智能合约在选择程序执行路径时将时间戳作为关键的随机变量。矿工在打包区块时要设置区块的时间戳信息，通常设置为矿工节点的本地时间，但恶意矿工可能会有意将时间戳设置为特定的值。如果合约以时间戳作为随机数来决定是否转账，那么很可能被矿工节点利用。

2.8 交易顺序依赖

假设在一个区块中，有两个不同的交易事务 T_1 和 T_2 调用了同一个合约 C。合约 C 的起始状态记作 s ，并且 $s \xrightarrow{T_1} s_1 \xrightarrow{T_2} t_1$ ， $s \xrightarrow{T_2} s_2 \xrightarrow{T_1} t_2$ ，

如果 t_1 与 t_2 两个状态不同，那么 C 是一个交易顺序依赖的合约。当合约出现交易顺序依赖时，攻击者在发起交易时可以通过调整交易的 `gasPrice` 值来影响该交易与其他交易的顺序，也可以通过直接参与记账来影响不同交易的执行顺序。例如，攻击者 Alice 可以发布一份悬赏交易：当完成某项任务后以一定的以太币作为奖励。此后 Alice 不断监听网络中的交易信息，当发现用户 Bob 完成任务的交易并提交后，Alice 立即发布一项降低奖励额度的交易，并设置较高的 `gasPrice` 值来鼓励矿工节点优先打包 Alice 的交易，这时 Bob 就有很大的概率无法得到应得的奖励。

2.9 小结

本节总结了 8 种常见的智能合约漏洞。从中可以看出，依靠传统测试的方法保证智能合约的安全性是很困难的。智能合约间存在相互调用的情况，开发者无法预知其他合约的执行逻辑，智能合约的实际执行情况可能会超出合约开发者的想象。大量软件存在缺陷的根本原因是开发者对语义的理解与智能合约系统的实际语义之间存在语义鸿沟^[23]。在保证智能合约安全性的工作中引入形式化方法显得十分必要。

3 智能合约的形式化验证

形式化验证方法包括符号执行^[24]、模型检测^[25]、定理证明^[26]等。本节将分别介绍不同验证方法的代表研究成果。

3.1 符号执行

符号执行是较常见的智能合约分析技术。符号执行的主要思想是符号化地执行程序，程序的输出为变量的表达式。每一条路径对应一个路径可达的条件公式，通常需要约束求解器来判定程序路径的可达性。符号执行能够遍历程序的路径空间，从而检查程序是否满足给定的安全性质。Luu 等^[23]开发的漏洞检测工具 Oyente 是利用符号执行技术验证智能合约的开创性工作。Oyente 以智能合约的 EVM 字节码和当前以太坊的全局状态作为输入，分析合约是否存在安全问题，并返回存在问题的程序执行路径。Oyente 能够检测交易顺序依赖漏洞、时间戳依赖漏洞、异常处理不当、重入漏洞等智能合约漏洞。Oyente 定义了一个简化的 EVM 操作语义，通过分析程序生成控

制流图,并符号化执行合约代码,以 Z3^[27]作为求解器判定程序路径可达性,针对每一类漏洞类型定义了相应的代码模式,根据符号执行的结果判断是否满足其中一类模式。Mueller^[28]开发的漏洞检测工具 Mythril 与 Oyente 的技术路线基本一致,基于以太坊官方的操作语义^[19],并引入了污点分析技术。Nikolić 等^[29]在 Oyente 的基础上开发了验证工具 Maian,相对于 Oyente 只考虑一个合约被单次调用后的行为,Maian 对一个合约多次被调用的情况建模,侧重检测程序的轨迹缺陷。Maian 重点研究了两类安全性质(资金泄露与账户销毁)和一类活性性质(资金冻结)。Maian 以 EVM 字节码为输入,支持 133 个字节码指令中的 121 个,遇到不支持的指令将跳过该路径并回溯,因此不能覆盖所有路径。Mossberg 等^[30]用动态符号执行技术开发了验证工具 Manticore,其优势是使用较为简单。Krupp 等^[31]定义了易受攻击的合约并基于此开发了验证工具 TEETHER。VerX 是一个基于符号执行的工具^[32],将时序逻辑的安全性验证归约为程序可达性分析,因而能够验证表达为时序逻辑的安全性质。Torres 等开发的符号执行验证工具 Osiris^[33]和 HoneyBadger^[34]分别能够识别整数相关的漏洞、蜜罐合约。Chen 等^[35]开发的 TokenScope 分析了智能合约的不一致问题。

Tsankov 等^[36]开发了一个基于数据流、控制流分析的可扩展验证工具 Securify。该研究基于如下观察:如果合约代码满足或违反了一个性质,它大概率满足或违反若干更加简单的模式,如超过九成的重入漏洞代码有在外部函数调用后修改 storage 的行为。该研究定义了一个描述代码模式的领域专用语言 Securify (与验证工具同名),在此基础上定义了以太坊流动性、调用操作后没有写操作等 7 类代码模式。用户可以利用 Securify 语言自定义新的代码安全模式。Securify 验证工具以 EVM 字节码和自定义安全性质为输入,验证合约代码满足或违反给定的性质,其分析过程分为 3 步:首先,对 EVM 字节码反汇编,使代码不依赖于栈结构;之后,借助 Datalog^[37]和 Souffle^[38]推导合约代码的语义,得到数据流和控制流的依赖关系;最后,根据合约的语义信息验证是否满足给定的代码模式。Securify 的缺点在

于无法处理与整数相关的性质。Kalra 等^[39]开发了基于抽象解释的检测工具 Zeus。一个交易的执行过程一定是若干 public 函数的连续调用,Zeus 应用此特点显著减少了待验证的状态空间,提高了验证效率。

Feist 等^[40]开发了静态分析框架 Slither,该框架将 Solidity 代码转换为中间表示 SlithIR,可利用数据流分析、污点分析等技术检测程序漏洞。Tikhomirov 等^[41]开发了静态分析工具 SmartCheck。SmartCheck 分析 Solidity 代码生成 XML 解析树,根据路径语言 Xpath 检测漏洞。Schneidewind 等^[42]开发了基于静态分析技术的验证工具 EThor。EThor 利用 Horn 逻辑分析程序的可达性,并由此分析程序的安全性质。Grech 等^[43]基于静态分析技术开发了检测工具 MadMax,其能够识别由 gas 耗尽带来的安全问题。

以符号执行为代表的程序分析技术对于合约开发人员避免常见漏洞是很好的辅助工具,但无法作出安全性的保证。这些工具通常需要依赖于专家对危险代码模式的定义,但简单的模式匹配并不能完全刻画复杂的安全性质,会带来不同程度的误报和漏报。

3.2 模型检测

模型检测是形式化验证的一类重要验证技术。首先,将待验证系统抽象为状态空间有限的形式模型;然后,通过搜索有限的状态空间分析形式模型的行为是否满足给定的时序性质,当性质不满足时给出违反该性质的执行路径。

模型检测技术已经发展得比较成熟,有许多优秀的模型检测工具,如 SPIN^[44]、UPPAAL^[45]、NuSMV^[46]等。目前已有一些将这些方法应用在智能合约验证工作中的尝试,如 Nehai 等^[47]用 NuSMV 语言对智能合约建模,建模分为 3 个层次,分别是内核层、应用层和环境层。内核层建模以太坊的行为,应用层建模智能合约的行为,环境层建模合约的执行环境框架,待验证的性质被刻画为 CTL 逻辑公式,借助 NuSMV 工具验证模型是否满足给定性质。Abdellatif 等^[48]用模型检测语言 BIP 建模智能合约,不仅对合约做建模,还包括用户和简化的区块链环境的行为。Bai 等^[49]将智能合约建模为状态迁移系统,将待验证性质表

示为 LTL 公式,并用模型检测工具 SPIN 进行验证。智能合约程序中可能涉及一些与时间相关的执行逻辑。Andrychowicz 等^[50]用时间自动机建模,并提出了在模型检测工具 UPPAAL 中验证的方法。

智能合约不存在并发行为^[51]。在合约间发生调用时,只有当被调用合约执行完毕后,原合约才会继续执行。但 Sergey 等^[51]指出,智能合约的一些特性类似于并发系统,如合约之间的相互调用类似于并发系统中不同进程间的通信行为;交易顺序的不确定性对交易结果的影响也类似于并发系统中的数据竞争问题。Sergey 等^[51]认为可以引入传统并发模型的研究方法来解决智能合约的验证问题。受此观点影响,有许多用传统并发模型如进程演算、Petri 网对智能合约建模的研究工作。例如, Bartoletti 等^[52]定义了领域专用语言 BitML 帮助用进程演算来建模智能合约; Laneve 定义了进程演算 scl 对智能合约建模^[53]; Qu 等^[54]用通信顺序进程建模合约代码,并用模型检测工具 FDR (failure divergence refinement)^[55]验证合约正确性; Li 等^[56]将智能合约建模为 Pi 演算的一个变种,并用 Tamarin 工具做验证。Wang 等^[57]和 Liu 等^[58]的建模方法都基于着色 Petri 网,以 ASK-CTL 表示待验证性质,在模型检测工具 CPN Tools 中验证以太坊系统的性质。

应用模型检测技术能够严格证明合约是否满足给定的安全性或活性,是较为严格的验证技术。但是模型检测方法存在一些局限性,如要求待验证的模型是有限状态的,并且由于模型检测存在状态爆炸问题,验证效率会受到影响。

3.3 定理证明

定理证明通常将待验证系统和性质统一描述在合适的逻辑系统中。根据公理和推理规则,借助定理证明器去证明待验证性质,目前的定理证明器大多是交互式的,如 Coq^[59]、Isabelle/HOL^[60]。

首次基于定理证明的智能合约验证工作是由 Hirai^[61]提出的。Hirai 首先用 Lem 语言^[62]描述 EVM 字节码, Lem 语言支持当前大多数定理证明器,再利用 Isabelle/HOL 验证合约的相关简单安全性质。Annenkov 等^[63]提出了基于元编程的方式,将智能合约代码转换为 Coq 程序,构建了验证框架 ConCert。Bhargavan 等^[64]分别将 Solidity

代码和编译后的 EVM 字节码翻译到 F* 语言,验证翻译后结果的一致性。Amani 等^[65]利用以太坊的 gas 机制对合约终止性的保证,定义了一个类似 Hoare 逻辑的程序逻辑,并提出了基于 Isabelle/HOL 的面向 EVM 字节码的验证方法。Yang 等^[66]基于 Coq 定理证明器,在 Solidity 语言的一个子集上^[67]开发了 FEther, FEther 包含部分自动化策略,提高了验证的自动化程度。除了 Solidity 和 EVM 字节码外,还有一些面向智能合约的中间语言的研究。例如, Li 等^[68]给出了基于 Isabelle/HOL 验证智能合约中间语言 Yul 的方法。Sergey 等^[69]提出了基于 Coq 验证智能合约时序性质的方法。Sun 等^[70]提出了基于 Coq 验证 BNB 智能合约的方法。

相比于模型检测,定理证明技术的优势是可以验证无限状态系统。但是目前的工作都依赖于交互式的定理证明器,在验证过程中需要有相关知识背景的专家参与,因此缺少高度自动化的验证工具。

3.4 其他工作

智能合约严格的形式语义是符号执行、定理证明等工作的基础。以太坊发布的黄皮书是描述 EVM 语义的官方文档^[19],但未过多考虑实际应用时的复杂场景,其定义并不完备,无法支撑形式化验证的工作。基于 F* 语言^[71], Grishchenko 等^[72]定义了 EVM 字节码的完备的小步语义。基于 K 框架^[73], Hildenbrandt 等^[74]定义了 EVM 的可执行的形式语义 KEVM。

不同于此前大部分语义定义中的 EVM 字节码, Jiao 等^[75]根据 Solidity 语言的语义,将 Solidity 源码编译成中间语言。合约开发者可以用 Solidity 语言编写合约,再编译成便于验证器操作的中间语言,验证合约代码的正确性,如 Sergey 等^[76]定义的中间语言 Scilla。Scilla 的设计遵循几个原则:程序中合约内部的计算被设计为独立的状态迁移,这一方式区分了合约的内部计算和合约间的交互;划分了改变状态的计算和不改变状态的计算;便于识别普通调用和续体传递^[77]。这表明 Scilla 中间语言适用于验证,且由于其代码可自动生成,能够避免合约开发者在编写中引入漏洞。代码自动生成方面的代表工作是 Mavridou 等^[78-79]提出的 FSolidM 框架,用户可以将合约行为定义

为有限状态机, 利用 FSolidM 能够生成对应的合约代码。在 FSolidM 的基础上 Mavridou 等^[80]又提出了 VeriSolid 框架, 进一步扩展了语法和语义, 并集成了基于模型检测的工具, 如 nuXmv 和 BIP 等。存在专门针对某类安全漏洞的工作, 如重入漏洞^[81-82]、资金冻结问题^[83]、交易顺序依赖漏洞^[84]等。此外, 将形式化方法与模糊测试^[85-86]、运行时验证^[87]、博弈论^[88]等方式结合起来验证智能合约安全性的研究逐渐成为主流。

4 验证工具的总结与实验分析

第 3 节介绍了智能合约形式化验证的代表工

作, 表 2 对这些工作做了一个简要的总结。本节选择其中具有代表性的工具进行实验, 并对分析结果进行对比。基于模型检测及定理证明方法的智能合约验证工具多数需要建模或将智能合约源代码、EVM 字节码转换成中间语言, 自动化程度较低, 实验时需要较多的人为干预。且部分工具如 Zeus 无可用开源代码。基于以上原因, 本节选取 3 个开源的基于符号执行的智能合约检测工具 (Mythril、Slither 以及 Oyente), 对 Smartbugs 数据集中的 143 个合约文件进行检测。该数据集中的合约包含 208 个标记的漏洞, 可用于评估合约工具的准确性。

表 2 智能合约工具
Table 2 Smart contract tools

分析方法	检测工具	检测对象	自动化程度
符号执行/静态分析	Oyente	EVM 字节码+以太坊的全局状态	全自动
	Zeus	Solidity 源码→LLVM 的中间语言	全自动
	Securify	EVM 字节码→静态单一赋值形式	全自动
	Mythril	Solidity 源码+EVM 字节码	全自动
模型检测	SPIN	Solidity 源码→PROMELA 语言, LTL 公式	半自动
	UPPAAL	带时间约束的 Solidity 源码	半自动
	NuSMV	Solidity 源码建模	半自动
	F*	Solidity 源码+EVM 字节码	半自动
定理证明	Isabelle/HOL	EVM 字节码→Lem 语言	半自动
	Coq	Solidity 源码建模	半自动
	F*	Solidity 代码和 EVM 字节码	半自动
其他	Slither	Solidity 源代码→SlithIR 中间语言	全自动

在 Smartbugs 数据集中已经开源了其中 69 份已标记漏洞的合约。本节使用 Mythril 和 Slither 工具对这 69 份合约进行检测, 根据实验结果, 对 69 份合约 (除处理超时或版本错误的合约文件) 中之前未被检测出的漏洞进行补充标记, 结果如表 3 所示。

Mythril 作为以太坊官方推荐的智能合约静态分析检测工具, 使用了污点分析、控制流验证等技术。表 3 列举了在实验过程中, 数据集中的合约新检测出的漏洞。Mythril 适合检测任意存储位置的写入漏洞、重入漏洞、无保护的以太币提款漏洞。Mythril 在 Oyente 工具的基础上进行了扩展, 利用多次符号执行可以还原实际合约执行过程中复杂的调用情况。

Slither 将 Solidity 源码转换为混合中间语言 SlithIR, 利用 Solidity 抽象语法树生成合约的继承关系。Slither 中包含 20 多种对不同类型的漏洞进行检测的探测器, 并且用户可以使用第三方工具基于中间语言 SlithIR 来建立更高级的分析。如表 3 所示, Slither 对使用过时的编译器版本漏洞、重入漏洞、未声明状态变量可见性漏洞、时间戳依赖攻击漏洞、使用密码学上较弱的伪随机数发生器漏洞的检测效果较好。Slither 工具的检测结果, 可以帮助用户和审查员理解代码。用户还可以通过第三方接口与 Slither 检测工具进行交互, 辅助代码的审查。

通过实验发现, Slither 工具对代码的覆盖率较高。表 4 的实验结果说明, Slither 的平均处理速度快于 Mythril 工具, 且检出的漏洞种类较多。

表 3 对 smartbugs 数据集中合约的漏洞检测进行补充标记
Table 3 Additional flagging of vulnerability detection for contracts in the smartbugs dataset

Mythril 新检测出的漏洞	合约名	Slither 新检测出的漏洞	合约名
SWC-124 CWE-123 任意存储位置的写入	arbitrary_location_write_simple.sol mapping_write.sol	SWC-124CWE-123 任意存储位置的写入	arbitrary_location_write_simple.sol
SWC-116 CWE-829 时间戳依赖攻击	crypto_roulette.sol parity_wallet_bug_2.sol roulette.sol spank_chain_payment.sol	SWC-102CWE-937 使用过时的编译器版本	arbitrary_location_write_simple.sol blackjack.sol crypto_roulette.sol dos_address.sol dos_number.sol dos_simple.sol
SWC-107 CWE-829 重入漏洞	unchecked_return_value.sol etherstore.sol governmental_survey.sol dos_address.sol reentrancy_dao.sol spank_chain_payment.sol	SWC-119CWE-710 合约内容矛盾	arbitrary_location_write_simple.sol blackjack.sol ether_lotto.sol
SWC-120 CWE-330 基于链属性的弱随机性	etherpot_lotto.sol lottery.sol open_address_lottery.sol	SWC-107 CWE-829 重入漏洞	auction.sol blackjack.sol crypto_roulette.sol eth_tx_order_dependence_minimal.sol etherpot_lotto.sol list_dos.sol lotto.sol lucky_doubler.sol modifier_reentrancy.sol odds_and_evens.sol wallet_02_refund_nosub.sol
SWC-104 CWE-252 未检查的调用返回值	king_of_the_ether_throne.sol	SWC-135 CWE-1164 无效代码	BECToken.sol random_number_generator.sol
SWC-105 CWE-284 无保护的以太坊提款	multiowned_vulnerable.sol odds_and_evens.sol old_blockhash.sol open_address_lottery.sol modifier_reentrancy.sol reentrance.sol roulette.sol simple_dao.sol wallet_04_confused_sign.sol	CWE-338 使用密码学上较弱的伪随机数发生器	blackjack.sol crypto_roulette.sol lucky_doubler.sol etherpot_lotto.sol open_address_lottery.sol random_number_generator.sol roulette.sol
SWC-106 CWE-284 无保护的自毁指令	parity_wallet_bug_2.sol simple_suicide.sol	SWC-108 CWE-710 未声明状态变量可见性	blackjack.sol crypto_roulette.sol FindThisHash.sol etherstore.sol guess_the_random_number.sol etheraffle.sol king_of_the_ether_throne.sol list_dos.sol send_loop.sol
SWC-112 CWE-829 委托调用非可信合约	proxy.sol	SWC-116 CWE-829 时间戳依赖攻击	crypto_roulette.sol etherstore.sol guess_the_random_number.sol king_of_the_ether_throne.sol roulette.sol Timelock.sol
SWC-110 CWE-670 触发 assert 断言	spank_chain_payment.sol	SWC-104 CWE-252 未检查的调用返回值	etherpot_lotto.sol list_dos.sol lotto.sol lucky_doubler.sol
SWC-113 CWE-703 失败调用引发的 DoS 攻击	send_loop.sol	SWC-133 CWE-294 变长参数导致的哈希冲突	blackjack.sol crypto_roulette.sol dos_address.sol king_of_the_ether_throne.sol

表 4 Mythril, Slither 和 Oyente 工具的处理结果
Table 4 Processing results for Mythril,
Slither and Oyente tools

工具	平均处理时间/s	实验样本(合约数)	准确率
Mythril	94	143	79.02%
Slither	5	143	81.12%
Oyente	36	69	15%

对于重入漏洞的检测, Slither 工具检测的准确率高于 Mythril; 在算数漏洞中, Mythril 工具的处理结果准确率更高。而 Oyente 工具由于长时间未更新, 实验效果并不理想。

此次实验表明, 用户使用第三方接口基于中间语言自定义漏洞类型进行检测, 可以结合不同的测试方法, 扩展检测漏洞的类型^[89]。在后续工作中将会对漏洞类型进一步细分, 明确分类标准, 并尝试在现有的一些工具中, 自定义第三方接口, 提高代码检测的覆盖率和准确率。

5 相关工作总结

本节对相关的综述文章做一个总结。

Atzei 等^[11]对智能合约可能遭受的攻击方式的总结是早期的一个经典工作。Delmolino 等^[90]在马里兰大学做了教学实验, 总结了学生在智能合约编码中常犯的一些错误。Chen 等^[91]对智能合约攻击和防御方式的总结是一个比较详尽的补充。

Durieux 等^[92]对包括 Oyente、Mythril、Securify、Smartcheck 在内的各种测试工具做了详细的实验比较。从准确度, 对不同类型漏洞的检测能力, 以及运行效率等方面分析了各类工具。Angelo 等^[93]总结了基于符号执行的漏洞检测工具。Imeri 等^[94]的综述则主要侧重模型检测方面的结果。Parizi 等^[95]总结了不同类型的智能合约领域专用编程语言。

Grishchenko 等^[96]的综述涵盖了形式语义, 性质定义, 检测工具等不同方面的内容。Tolmach 等^[97]给了一个相对全面的综述, 对已有的形式化建模工作分为两类。一类是合约层面, 将合约实现细节忽略, 着重考虑合约与其他合约或外部环境之间的交互关系; 另一类是程序层面, 更关注合约在执行时的内部细节。

近年来已有一些中文的综述文章^[98-100]。相较

于这些工作, 本文进行以下 3 点补充。

1) 对以太坊的机制以及智能合约的常见漏洞作了较为详细的介绍。在此基础上, 对已有验证工作的理论总结更加深入。

2) 涵盖了 2021—2022 年新工作进展, 对形式化验证工作的总结更加全面。

3) 对现有的基于不同理论的智能合约检测工具进行了对比实验, 结合实验结果对检测工具作出更加客观的评价。

6 总结与展望

在系统总结了基于形式化方法的智能合约验证的研究进展后, 发现智能合约仍存在大量安全问题, 阻碍了区块链技术的大范围落地。利用形式化方法检测智能合约漏洞已经有了大量研究成果, 但目前仍有部分关键问题需要解决。

(1) 检测工具的覆盖率较低

部分工具需要提取智能合约 Solidity 源代码或 EVM 字节码的语义编译成中间语言, 转换时语义信息不全面、不准确。

检测工具对代码的覆盖率低, 难以对代码中所有复杂的调用情况和判断条件做到检测和推断, 对于跨合约的调用、实际复杂环境中的执行过程、执行过程中恶意代码的攻击等, 无法进行全面准确的执行前预测。

(2) 检测工具的自动化程度以及可扩展性较低

目前检测工具的自动化程度较低, 部分工具的检测需要进行人工干预、核验与专家分析, 对检测结果进行手动检查, 筛选出漏洞的误报结果。此外, 编写智能合约的主流语言为 Solidity 语言, 但仍有其他语言如: Serpent^[101]和 LLL^[102]可支持合约的编写, 对智能合约检测工具的适配性和可扩展性造成极大的挑战。

(3) 智能合约的语义鸿沟问题

智能合约验证中大部分缺陷出现的根本原因是智能合约系统的实际语义与开发者对合约的理解之间存在偏差。为解决语义鸿沟的问题, 利用领域特定语言(DSL)来规范合约的语义, 设计漏洞检测的模式, 仍是未来研究的难点。此外, 有不少研究常用中间语言表达合约逻辑和语义的准确性^[40-43], 提高中间表示的通用性, 规

范中间语言的表示形式，仍是未来研究中需要关注的关键问题。

(4) 关于智能合约验证的评估体系不规范

由于目前基于形式化验证技术的部分工具的自动化程度不高，缺乏测试用例集等原因，对智能合约验证工作的评价准则仍不规范。因此，需要建立较为标准的综合评价体系来判定验证方式的有效性。

在上述研究的基础上，有以下几个方向是值得进一步研究的。

1) 自动生成合约代码。基于用户对合约功能的定义自动生成代码，能够避免不规范的合约编写引入错误。此前的大量工作聚焦于如何检测合约代码中的漏洞，关于代码生成的研究比较少。目前的研究主要基于简单的有限状态系统^[78-79]，该方向还有很大的提升空间。

2) Solidity 编译器的正确性验证。即便 Solidity 代码是安全的，如果编译器本身存在漏洞，那么可能在编译后的代码中引入安全问题。关于 Solidity 编译器在 gas 消耗方面，目前还没有对编译器的正确性验证的研究^[103]。

3) 中间语言的一致性验证。目前有一些中间语言被提出^[76,104]。在 Solidity 代码转换为中间语言，以及中间语言转换为 EVM 字节码时需要保证语义的一致性。目前还缺乏对语义一致性的研究。

4) 开发新的智能合约高级语言。另一个方向是重新设计一门更加安全的智能合约语言，减少合约开发者出错的可能性。

5) 规范智能合约漏洞数据集及工具的性能评估标准。对各类智能合约检测工具的评估所用的数据集较少且不统一，不同工具的性能对比存在误差，且工具的评估标准没有统一的规定，无法准确分析有效性。

6) 除以太坊外，还有一些支持智能合约的区块链平台如超级账本、EOS、区块链底层技术开源平台（BCOS, be credible, open & secure）等。以超级账本为例，编写超级账本合约用的不是领域专用语言，而是通用语言如 Go、Java 等，可能出现一些与具体编程语言相关的漏洞^[105]，和本文介绍的漏洞类型有较大区别。由于缺乏一个影响力较大的公共平台，出现的漏洞事件相对较少，相关的智能合约

验证的研究也不多^[105-106]。但是以超级账本为代表的平台受到来自企业的关注越来越多，在这些平台上验证智能合约的正确性将是一个重要的方向。

参考文献：

- [1] NAKAMOTO S. Bitcoin: a peer-to-peer electronic cash system[J]. Consulted, 2008:1-49.
- [2] BUTERIN V. A next-generation smart contract and decentralized application platform[R]. White Paper, 2014.
- [3] SZABO N. Formalizing and securing relationships on public networks[J]. First Monday, 1997, 2(9).
- [4] PETERS G W, PANAYI E. Understanding modern banking ledgers through blockchain technologies: future of transaction processing and smart contracts on the internet of moneybanking beyond banks and money[M]. Springer, 2016.
- [5] AZARIA A, EKBLAW A, VIEIRA T, et al. MedRec: using blockchain for medical data access and permission management[C]//Proceedings of 2016 2nd International Conference on Open and Big Data (OBD). 2016: 25-30.
- [6] GRIGGS K N, OSSIPOVA O, KOHLIOS C P, et al. Healthcare blockchain system using smart contracts for secure automated remote patient monitoring[J]. Journal of Medical Systems, 2018, 42(7): 130.
- [7] CHRISTIDIS K, DEVETSIKIOTIS M. Blockchains and smart contracts for the Internet of Things[J]. IEEE Access, 2016, 4: 2292-2303.
- [8] ØLNES S, UBACHT J, JANSSEN M. Blockchain in government: benefits and implications of distributed ledger technology for information sharing[J]. Government Information Quarterly, 2017, 34(3): 355-364.
- [9] MENGELKAMP E, NOTHEISEN B, BEER C, et al. A blockchain-based smart grid: towards sustainable local energy markets[J]. Computer Science-Research and Development, 2018, 33(1): 207-214.
- [10] ABEYRATNE S A. Blockchain ready manufacturing supply chain using distributed ledger[J]. International Journal of Research in Engineering and Technology, 2016, 5(9): 1-10.
- [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] The DAO smart contract[EB]. 2016.
- [13] An in-depth look at the parity multisig Bug[EB]. 2017.
- [14] The parity wallet vulnerability[EB]. 2017.
- [15] 王戟, 詹乃军, 冯新宇, 等. 形式化方法概貌[J]. 软件学报, 2019, 30(1): 33-61.
WANG J, ZHAN N J, FENG X Y, et al. Overview of formal methods[J]. Journal of Software, 2019, 30(1): 33-61.
- [16] BONNEAU J, CLARK J, GOLDFEDER S. On bitcoin as a public randomness source[J]. IACR Cryptology ePrint Archive, 2015: 1015.
- [17] DANNEN C. Introducing Ethereum and solidity[M]. Berkeley: Apress, 2017.
- [18] ANDRYCHOWICZ M, DZIEMBOWSKI S. Distributed crypto-

- graphy based on the proofs of work[J]. IACR Cryptology ePrint Archive, 2014: 796.
- [19] WOOD G. Ethereum: a secure decentralised generalised transaction ledger[J]. Ethereum Project Yellow Paper, 2014, 151: 1-32.
 - [20] BUTERIN V. Dagger: a memory-hard to compute, memory-easy to verify script alternative[R]. 2013.
 - [21] BeautyChain Integer Overflow[EB]. 2018.
 - [22] KingOfTheEtherThrone smart contract[EB].
 - [23] LUU L, CHU D H, OLICKEL H, et al. Making smart contracts smarter[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 254-269.
 - [24] KING J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.
 - [25] CLARKE E M, GRUMBERG O, PELED D. Model checking[M]. MIT Press, 1999.
 - [26] BIBEL W. Automated theorem proving[M]. Wiesbaden: Vieweg+Teubner Verlag, 1987.
 - [27] Microsoft Corporation. The Z3 theorem prover[EB].
 - [28] MUELLER B. Smashing Ethereum smart contracts for fun and real profit[C]//In 9th Annual HITB Security Conference (HITBSec-Conf). 2018.
 - [29] NIKOLIĆ 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.
 - [30] MOSSBERG M, MANZANO F, HENNENFENT E, et al. Manticores: a user-friendly symbolic execution framework for binaries and smart contracts[C]//Proceedings of 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2019: 1186-1189.
 - [31] KRUPP J, ROSSOW C. TEETHER: gnawing at Ethereum to automatically exploit smart contracts[C]//Proceedings of 27th USENIX Security Symposium. 2018: 1317-1333.
 - [32] PERMENEV A, DIMITROV D, TSANKOV P, et al. VerX: safety verification of smart contracts[C]//Proceedings of 2020 IEEE Symposium on Security and Privacy. 2020: 1661-1677.
 - [33] 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. 2018: 664-676.
 - [34] TORRES C F, STEICHEN M. The art of the scam: demystifying honeypots in Ethereum smart contracts[C]//Proceedings of the 28th USENIX Security Symposium. 2019: 1591-1607.
 - [35] CHEN T, ZHANG Y F, LI Z H, et al. TokenScope: automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 1503-1520.
 - [36] 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. 2018: 67-82.
 - [37] ULLMAN J D. Principles of database and knowledge-base systems[J]. Computer Science, 1988.
 - [38] JORDAN H, SCHOLZ B, SUBOTIĆ P. Soufflé: on synthesis of program analyzers[C]//Computer Aided Verification. 2016: 422-430.
 - [39] KALRA S, GOEL S, DHAWAN M, et al. ZEUS: analyzing safety of smart contracts[C]//Proceedings 2018 Network and Distributed System Security Symposium. 2018: 1-12.
 - [40] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]//Proceedings of 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). 2019: 8-15.
 - [41] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. SmartCheck: static analysis of ethereum smart contracts[C]//Proceedings of 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). 2018: 9-16.
 - [42] SCHNEIDEWIND C, GRISHCHENKO I, SCHERER M, et al. Ethor: practical and provably sound static analysis of Ethereum smart contracts[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 621-640.
 - [43] GRECH N, KONG M, JURISEVIC A, et al. MadMax: surviving out-of-gas conditions in Ethereum smart contracts[J]. Proceedings of the ACM on Programming Languages, 2018, 2: 1-27.
 - [44] HOLZMANN G J, CHECKER S M. The: primer and reference manual[J]. Septiembre del, 2003.
 - [45] BENGTTSSON J, LARSEN K, LARSSON F, et al. UPPAAL—a tool suite for automatic verification of real-time systems[C]//International Hybrid Systems Workshop. 1995: 232-243.
 - [46] CAVADA R, CIMATTI A, JOCHIM C A, et al. Nusmv 2.4 user manual[J]. CMU and ITC-IRST, 2005.
 - [47] NEHAÏ Z, PIRIOU P Y, DAUMAS F. Model-checking of smart contracts[C]//Proceedings of 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data. 2018: 980-987.
 - [48] ABDELLATIF T, BROUSMICHE K L. Formal verification of smart contracts based on users and blockchain behaviors models[C]//Proceedings of 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). 2018: 1-5.
 - [49] BAI X M, CHENG Z J, DUAN Z B, et al. Formal modeling and verification of smart contracts[C]//Proceedings of the 2018 7th International Conference on Software and Computer Applications. 2018: 322-326.
 - [50] ANDRYCHOWICZ M, DZIEMBOWSKI S, MALINOWSKI D, et al. Modeling bitcoin contracts by timed automata[C]//International Conference on Formal Modeling and Analysis of Timed Systems. 2014: 7-22.
 - [51] SERGEY I, HOBOR A. A concurrent perspective on smart contracts[C]//International Conference on Financial Cryptography and Data Security. 2017: 478-493.
 - [52] BARTOLETTI M, ZUNINO R. BitML: a calculus for bitcoin smart contracts[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 83-100.
 - [53] LANEVE C, COEN C S, VESCHETTI A. On the prediction of smart contracts' behaviors[M]//From Software Engineering to Formal Methods and Tools, and Back. 2019: 397-415.
 - [54] QU M X, HUANG X, CHEN X, et al. Formal verification of smart contracts from the perspective of concurrency[C]//International Conference on Smart Blockchain. Springer, Cham, 2018: 32-43.
 - [55] ROSCOE A W. Understanding concurrent systems[M]. London: Springer London, 2010.
 - [56] LI X Y, SU C, XIONG Y, et al. Formal verification of BNB smart contract[C]//Proceedings of 2019 5th International Conference on Big Data Computing and Communications (BIGCOM). 2019: 74-78.
 - [57] WANG D, HUANG X, MA X F. Formal analysis of smart contract based on colored petri nets[J]. IEEE Intelligent Systems, 2020,

- 35(3): 19-30.
- [58] LIU Z T, LIU J. Formal verification of blockchain smart contract based on colored petri net models[C]//Proceedings of 2019 IEEE 43rd Annual Computer Software and Applications Conference. 2019: 555-560.
- [59] The coq proof assistant reference manual[EB].
- [60] NIPKOW T, WENZEL M, PAULSON L C. Isabelle/HOL[M]. Berlin, Heidelberg: Springer, 2002.
- [61] HIRAI Y. Defining the Ethereum virtual machine for interactive theorem provers[C]//International Conference on Financial Cryptography and Data Security. 2017: 520-535.
- [62] MULLIGAN D P, OWENS S, GRAY K E, et al. Lem: reusable engineering of real-world semantics[J]. ACM SIGPLAN Notices, 2014, 49(9): 175-188.
- [63] ANNENKOV D, NIELSEN J B, SPITTERS B. ConCert: a smart contract certification framework in Coq[C]// Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2020: 215-228.
- [64] BHARGAVAN K, DELIGNAT-LAUDAUD A, FOURNET C, et al. Formal verification of smart contracts: short paper[C]//Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. 2016: 91-96.
- [65] AMANI S, BÉGEL M, BORTIN M, et al. Towards verifying Ethereum smart contract bytecode in Isabelle/HOL[C]//Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2018: 66-77.
- [66] YANG Z, LEI H. FEther: an extensible definitional interpreter for smart-contract verifications in coq[J]. IEEE Access, 2019, 7: 37770-37791.
- [67] YANG Z, LEI H. Lolisa: formal syntax and semantics for a subset of the solidity programming language[J]. arXiv: 1803.09885, 2008.
- [68] LI X M, SHI Z P, ZHANG Q Y, et al. Towards verifying ethereum smart contracts at intermediate language level[M]//Formal Methods and Software Engineering. 2019: 121-137.
- [69] SERGEY I, KUMAR A, HOBOR A. Temporal properties of smart contracts[C]//International Symposium on Leveraging Applications of Formal Methods. 2018: 323-338.
- [70] SUN T Y, YU W S. A formal verification framework for security issues of blockchain smart contracts[J]. Electronics, 2020, 9(2): 255.
- [71] SWAMY N, HRIȚCU C, KELLER C, et al. Dependent types and multi-monadic effects in F[C]//Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2016: 256-270.
- [72] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. A semantic framework for the security analysis of Ethereum smart contracts[C]//Principles of Security and Trust. 2018: 243-269.
- [73] STEFĂNESCU A, PARK D, YUWEN S J, et al. Semantics-based program verifiers for all languages[J]. ACM SIGPLAN Notices, 2016, 51(10): 74-91.
- [74] HILDENBRANDT E, SAXENA M, RODRIGUES N, et al. KEVM: a complete formal semantics of the Ethereum virtual machine[C]//Proceedings of 2018 IEEE 31st Computer Security Foundations Symposium. 2018: 204-217.
- [75] JIAO J, KAN S L, LIN S W, et al. Semantic understanding of smart contracts: executable operational semantics of solidity[C]//Proceedings of 2020 IEEE Symposium on Security and Privacy. 2020: 1695-1712.
- [76] SERGEY I, KUMAR A, HOBOR A. Scilla: a smart contract intermediate-level language[J]. arXiv preprint arXiv:1801.00687, 2018.
- [77] REYNOLDS J C. Definitional interpreters for higher-order programming languages[C]//Proceedings of the ACM Annual Conference. 1972: 717-740.
- [78] MAVRIDOU A, LASZKA A. Designing secure ethereum smart contracts: a finite state machine based approach[C]//Financial Cryptography and Data Security, 2018: 523-540.
- [79] MAVRIDOU A, LASZKA A. Tool demonstration: FSolidM for designing secure Ethereum smart contracts[C]//International Conference on Principles of Security and Trust. 2018: 270-277.
- [80] MAVRIDOU A, LASZKA A, STACHTIARI E, et al. VeriSolid: Correct-by-design smart contracts for Ethereum[C]//International Conference on Financial Cryptography and Data Security. 2019: 446-465.
- [81] GROSSMAN S, ABRAHAM I, GOLAN-GUETA G, et al. Online detection of effectively callback free objects with applications to smart contracts[J]. Proceedings of the ACM on Programming Languages, 2018, 2: 1-28.
- [82] XUE Y X, MA M L, LIN Y, et al. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts[C]//Proceedings of 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020: 1029-1040.
- [83] BARTOLETTI M, ZUNINO R. Verifying liquidity of Bitcoin contracts[C]//8th International Conference on Principles of Security and Trust, POST 2019 Held as Part of the European Joint Conferences on Theory and Practice of Software. 2019: 222-247.
- [84] KOLLURI A, NIKOLIC I, SERGEY I, et al. Exploiting the laws of order in smart contracts[C]//Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2019: 363-373.
- [85] JIANG B, LIU Y, CHAN W K. ContractFuzzer: fuzzing smart contracts for vulnerability detection[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018: 259-269.
- [86] NGUYEN T D, PHAM L H, SUN J, et al. sFuzz: an efficient adaptive fuzzer for solidity smart contracts[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020: 778-788.
- [87] LI A, CHOI J A, LONG F. Securing smart contract with runtime validation[C]//Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020: 438-453.
- [88] BIGI G, BRACCIALI A, MEACCI G, et al. Validation of decentralised smart contracts through game theory and formal methods[M]//Programming Languages with Applications to Biology and Security. Springer, Cham, 2015: 142-161.
- [89] 田国华, 胡云瀚, 陈晓峰. 区块链系统攻击与防御技术研究进展[J]. 软件学报, 2021, 32(5): 1495-1525.
- TIAN G H, HU Y H, CHEN X F. Research progress on attack and defense techniques in block-chain system[J]. Journal of Software, 2021, 32(5): 1495-1525.
- [90] DELMOLINO K, ARNETT M, KOSBA A, et al. Step by step towards creating a safe smart contract: Lessons and insights from a

- cryptocurrency lab[C]//International conference on Financial Cryptography and Data Security. 2016: 79-94.
- [91] CHEN H S, PENDLETON M, NJILLA L, et al. A survey on Ethereum systems security[J]. ACM Computing Surveys, 2021, 53(3): 1-43.
- [92] DURIEUX T, FERREIRA J F, ABREU R, et al. Empirical review of automated analysis tools on 47, 587 Ethereum smart contracts[C]//Proceedings of 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). 2020: 530-541.
- [93] DI ANGELO M, SALZER G. A survey of tools for analyzing Ethereum smart contracts[C]//Proceedings of 2019 IEEE International Conference on Decentralized Applications and Infrastructures. 2019: 69-78.
- [94] IMERI A, AGOULMINE N, KHADRAOUI D. Smart Contract modeling and verification techniques: a survey[C]//Proceedings of the 8th International Workshop on ADVANCES in ICT Infrastructures and Services (ADVANCE 2020). 2020: 1-8.
- [95] PARIZI R M, DEGHANTANHA A. Smart contract programming languages on blockchains: An empirical evaluation of usability and security[C]//International Conference on Blockchain. 2018: 75-91.
- [96] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. Foundations and tools for the static analysis of Ethereum smart contracts[C]//International Conference on Computer Aided Verification. 2018: 51-78.
- [97] TOLMACH P, LI Y, LIN S W, et al. A survey of smart contract formal specification and verification[J]. ACM Computing Surveys, 2021, 54(7): 1-38.
- [98] 钱鹏, 刘振广, 何钦铭, 等. 智能合约安全漏洞检测技术研究综述[J]. 软件学报, 2022, 33(8): 3059-3085.
QIAN P, LIU Z G, HE Q M. Smart Contract vulnerability detection technique: a survey[J]. Journal of Software, 2022, 33(8): 3059-3085.
- [99] 章峰, 史博轩, 蒋文保. 区块链关键技术及应用研究综述[J]. 网络与信息安全学报, 2018, 4(4): 22-29.
ZHANG F, SHI B X, JIANG W B. Review of key technology and its application of blockchain[J]. Chinese Journal of Network and Information Security, 2018, 4(4): 22-29.
- [100] 朱健, 胡凯, 张伯钧. 智能合约的形式化验证方法研究综述[J]. 电子学报, 2021, 49(4): 792-804.
ZHU J, HU K, ZHANG B J. Review on formal verification of smart contract[J]. Acta Electronica Sinica, 2021, 49(4): 792-804.
- [101] Ethereum Foundation. The serpent contract-oriented programming language[EB].
- [102] Lisp-Like language[EB].
- [103] CHEN T, LI X Q, LUO X P, et al. Under-optimized smart contracts devour your money[C]//Proceedings of 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. 2017: 442-446.
- [104] O'CONNOR R. Simplicity: a new language for blockchains[C]//Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security. 2017: 107-120.
- [105] YAMASHITA K, NOMURA Y, ZHOU E C, et al. Potential risks of hyperledger fabric smart contracts[C]//Proceedings of 2019 IEEE

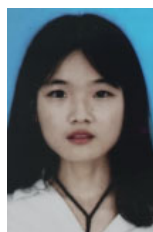
International Workshop on Blockchain Oriented Software Engineering. Piscataway: 2019: 1-10.

- [106] BECKERT B, HERDA M, KIRSTEN M, et al. Formal specification and verification of Hyperledger Fabric chaincode[C]//3rd Symposium on Distributed Ledger Technology (SDLT-2018) co-located with ICFEM. 2018: 44-48.

[作者简介]



张文博 (1992-), 男, 河南洛阳人, 博士, 上海海洋大学讲师, 主要研究方向为形式化验证、理论计算机科学。



陈思敏 (1998-), 女, 安徽合肥人, 上海海洋大学硕士生, 主要研究方向为形式化验证、区块链。



魏立斐 (1982-), 男, 浙江绍兴人, 博士, 上海海洋大学副教授, 主要研究方向为密码学与云计算安全。



宋巍 (1977-), 女, 山西大同人, 博士, 上海海洋大学教授、博士生导师, 主要研究方向为计算机视觉、海洋大数据分析、软件工程等。



黄冬梅 (1964-), 女, 河南郑州人, 上海电力大学教授、博士生导师, 主要研究方向为大数据分析、机器学习及智能服务。