Journal of Computer Applications
计算机应用,2019,39(7):1959 - 1966
ISSN 1001-9081
CODEN JYIIDU
2019-07-10
http://www.joca.cn

文章编号:1001 - 9081(2019)07 - 1959 - 08
DOI:10.11772/j.issn.1001 - 9081.2019010082

智能合约安全漏洞挖掘技术研究

付梦琳*,吴礼发,洪 征,冯文博
(中国解放军陆军工程大学 指挥控制工程学院,南京 210007)
(* 通信作者电子邮箱 fu_molly@163.com)

摘 要:近年来,以智能合约为代表的第二代区块链平台及应用出现了爆发性的增长,但频发的智能合约漏洞事件严重威胁着区块链生态安全。针对当前主要依靠基于专家经验的代码审计效率低下的问题,提出开发通用的自动化工具来挖掘智能合约漏洞的重要性。首先,调研并分析了智能合约面临的安全威胁问题,总结了代码重入、访问控制、整数溢出等10种出现频率最高的智能合约漏洞类型和攻击方式;其次,讨论了主流的智能合约漏洞的检测手段,并梳理了智能合约漏洞检测的研究现状;然后,通过实验验证了3种现有符号执行工具的检测效果。对于单一漏洞类型,漏报率最高达0.48,误报率最高达0.38。实验结果表明,现有研究涵盖的漏洞类型不完整,误报及漏报多,并且依赖人工复核;最后,针对这些不足展望了未来研究方向,并提出一种符号执行辅助的模糊测试框架,能够缓解模糊测试代码覆盖率不足和符号执行路径爆炸问题,从而提高大中型规模智能合约的漏洞挖掘效率。

关键词:区块链安全;智能合约;以太坊;漏洞挖掘;自动化工具
中图分类号:TP309;TP393.08 **文献标志码**:A

Research on vulnerability mining technique for smart contracts

FU Menglin*, WU Lifa, HONG Zheng, FENG Wenbo
(*College of Command and Control Engineering, the Army Engineering University of PLA, Nanjing Jiangsu* 210007, *China*)

Abstract: The second generation of blockchain represented by smart contract has experienced an explosive growth of its platforms and applications in recent years. However, frequent smart contract vulnerability incidents pose a serious risk to blockchain ecosystem security. Since code auditing based on expert experience is inefficient in smart contracts vulnerability mining, the significance of developing universal automated tools to mining smart contracts vulnerability was proposed. Firstly, the security threats faced by smart contracts were investigated and analyzed. Top 10 vulnerabilities, including code reentrancy, access control and integer overflow, as well as corresponding attack modes were summarized. Secondly, mainstream detection methods of smart contract vulnerabilities and related works were discussed. Thirdly, the performance of three existing tools based on symbolic execution were verified through experiments. For a single type of vulnerability, the highest false negative rate was 0.48 and the highest false positive rate was 0.38. The experimental results indicate that existing studies only support incomplete types of vulnerability with many false negatives and positives and depend on manual review. Finally, future research directions were forecasted aiming at these limitations, and a symbolic-execution-based fuzzy test framework was proposed. The framework can alleviate the problems of insufficient code coverage in fuzzy test and path explosion in symbolic execution, thus improving vulnerability mining efficiency for large and medium-sized smart contracts.

Key words: blockchain security; smart contract; Ethereum; vulnerability mining; automated tool

0 引言

区块链技术已经成为目前金融业关注度最高的技术之一,经历了以比特币应用为代表的区块链1.0时代,目前已经进入了以智能合约为标志的区块链2.0时代。智能合约适应于区块链分布式、去中心化的特点,具有独立运行、不可篡改的优良特性,可用于实现包含金融工具在内的各类分布式应用。然而,智能合约无法避免地存在安全漏洞。从2016年The DAO(Decentralized Autonomous Organization)事件中360万以太币被盗,到2017年Parity钱包因多重签名钱包合约漏洞导致上亿美元资金被冻结,到2018年美链BEC(Beauty Chain)百亿美金项目瞬时归零、BAI(Blockchain of Artificial Intelligence and Internet of things)和EDU(EduCoin)任意账户转账,再到EOS(Enterprise Operation System)漏洞允许恶意合约穿透虚拟机从而危害矿工节点,智能合约俨然成为区块链安全的重灾区。当前尚没有通用的自动化手段来验证智能合约代码是否包含漏洞,安全保障主要依靠开发者的技能水平以及基于专家经验的代码审计,这已无法满足智能合约数量及规模不断扩大、功能日益复杂、漏洞挖掘难度提升的新形势下漏洞分析与发现的需求。本文对智能合约相关知识进行了系统的阐述,根据大量调研并结合其特性分析了当前智能合约面临的安全威胁问题,从近年来发生的安全事件和代码审

收稿日期:2019-01-14;修回日期:2019-03-13;录用日期:2019-03-13。 基金项目:国家重点研发计划项目(2017YFB0802900)。
作者简介:付梦琳(1995—),女,江苏南京人,硕士研究生,主要研究方向:漏洞挖掘、区块链安全; 吴礼发(1968—),男,湖北黄石人,教授,博士,CCF会员,主要研究方向:漏洞挖掘、网络管理; 洪征(1979—),男,江苏南京人,副教授,博士,主要研究方向:协议逆向、漏洞挖掘; 冯文博(1994—),男,河南郑州人,硕士研究生,主要研究方向:协议识别、机器学习。

计工作总结出智能合约漏洞的主要类型和攻击方式。归纳了智能合约漏洞检测手段及相关工作，通过对现有工具的实验验证与对比分析，讨论目前工作存在的问题和挑战，并针对这些不足提出未来研究的改进方法与建议。

1 智能合约概述

1.1 智能合约的概念

智能合约的诞生可以追溯到 20 世纪末，跨领域法律学者 Nick Szabo 提出智能合约的定义为 “一个智能合约是一套以数字形式定义的承诺，包括合约参与方可以在上面执行这些承诺的协议。”[1] 他描述了自动售货机在物理设定下，将产品直接给付过款的用户的场景，认为可以使用计算机代码代替机械设备来完成更为复杂的交易。然而当时缺少可信的执行环境，缺乏支持可编程合约的数字系统和技术，智能合约并未被应用于实际产业中。直到区块链技术的兴起使得该问题得以解决，区块链不仅能够支持可编程合约，并具有去中心化、不可篡改、过程透明、可追踪等优势，为智能合约提供了一个完美的解决信任问题的机制。2013 年底，Vitalik Buterin 发布的以太坊白皮书《以太坊: 下一代智能合约和去中心化应用平台》[2] 将智能合约引入了区块链，拓展了区块链在货币领域之外的应用，引领了区块链 2.0 时代的到来。如今，以太坊已成为最主流的智能合约开发和运行平台。

简而言之，智能合约是传统合约的数字化版本，它是一种计算机程序，运行在区块链数据库上。智能合约程序代码规定了合约条款，包含一些触发条件，一旦满足触发条件合约便会自行执行。

表 1 从不同维度对智能合约与传统合约进行比较，智能合约具备许多优势: 第一，智能合约不依赖中间人进行交易，例如金钱、股票、财产等具有一定价值的东西，无需第三方执行合约，大幅度缩减了成本; 第二，消除了第三方供应商，用户间得以直接进行交易，加速了合约验证和执行的整个过程; 第三，智能合约条款不可篡改的特性意味着它免于各种人为干预，降低了用户受骗的风险; 第四，智能合约存储于分布式账本，即每个联网设备均存有一份合约副本，难以出现断电、节点故障等问题。

表 1 智能合约与传统合约的区别

Tab. 1 Differences between smart contract and traditional contract

属性	智能合约	传统合约
自动化维度	自动判断触发条件	人工判断触发条件
主客观维度	适合客观性的请求	适合主观性的请求
成本维度	低成本	高成本
执行时间维度	事前预防	事后执行
违约惩罚维度	依赖于抵押资产	依赖于刑罚
适用范围维度	全球性	受限于具体辖区

1.2 智能合约的工作过程

智能合约的工作过程分为三步[3]。

1) 智能合约的构建。

首先，用户注册为区块链的用户，区块链会返回用户一个公私钥对，公钥作为用户在区块链上的账户地址，私钥作为操作该账户的唯一钥匙; 然后由区块链内的多个用户共同商定一份承诺，以电子化的方式明确双方的权利和义务，参与者分别用各自私钥进行签名以确保合约的有效性; 最后，根据承诺内容将合约传入区块链网络中。

2) 智能合约的存储。

合约通过 P2P(Peer-to-Peer) 的方式扩散到区块链全网中的每个节点。接收到合约后，区块链中的验证节点将它保存至内存并等待新一轮的共识时间，触发对该份合约的共识和处理。到达共识时间后，验证节点将近一段时间内保存的所有合约打包成集合并计算其 Hash 值，并将这个合约集合的 Hash 值组装成一个区块结构，扩散到全网。其他验证节点收到该区块结构后，取出 Hash 值，对比自己保存的合约集合，并且向其他的验证节点发送一份自己认可的合约集合。经过多轮发送和对比，在规定的时间内，所有的验证节点最终对最新的合约集合达成一致。最新达成的合约集合通过区块的形式扩散至全网，接收到合约集的节点对其逐条验证，只有通过验证的合约会被写入区块链，验证的内容主要是合约参与者的私钥签名是否与账户匹配。

3) 智能合约的执行。

智能合约定期对自动机状态进行检查，验证满足条件的事务，若达成共识则自动执行并通知用户。

1.3 以太坊与智能合约

以太坊是第一个提供完善的智能合约开发框架的区块链底层系统，提供了充足的应用程序编程接口(Application Programming Interface，API) ，支持迅速开发各种区块链应用。微软云服务上提供了智能合约工具箱，运行在于太坊区块链，其平台凭功能的多样性和较高的智能合约执行能力获得银行业和互联网金融行业的青睐，多家知名金融机构，如摩根大通、纳斯达克、VISA 等，都采用了以太坊的智能合约系统，目前开发在以太坊上的应用已有 200 多个。下文从账户模型、交易费用和以太坊虚拟机(Ethereum Virtual Machine，EVM) 三个方面介绍以太坊的运行机制。

1.3.1 账户模型

以太坊的基础单元是账户，以太坊区块链上所有的状态转换皆为账户之间价值和信息的转移。账户分为两类，分别是由私钥控制的外部账户(Externally Owned Account，EOA) 以及由合约代码控制的合约账户，它们共用同一个地址空间，每个账户都由一个长度为 160 位的地址来标识。外部账户不存储代码，但可以通过创建和使用私钥签署交易向其他外部账户或合约账户发送消息; 而合约账户无法自动发起新的交易，只有接收到外部账户的消息，合约账户内部的代码才会被激活，允许它执行转移代币、写入内存、生成新的代币、执行计算、创建新合约等操作。

1.3.2 交易费用

以太坊用户必须为合约的执行向网络支付适当的交易费用 gas，使得以太坊区块链免受无关紧要或恶意运算任务的干扰，例如无限循环和分布式拒绝服务(Distributed Denial of Service，DDoS) 攻击。发送方需要为每笔交易设置好 gas 的两个部分，即 gas limit 和 gas price。gas limit 表示用户愿意为某笔交易支付的最大 gas 量(至少为 21 000) ，gas price 表示用户承诺给每个 gas 单位的花销，一般以 gwei 为单位，因此发送

者愿意花费最多 gas limit × gas price 来执行这一交易。如果发送者账户余额中以太币的数量大于此最大值,则可以进行交易。gas used 表示执行实际花费的 gas 值总和,则实际支付的费用总额为 gas used × gas price。交易结束时,如果仍有 gas 剩余,则将之返还给发送者; 但若发送方未支付足够的 gas,导致 gas 被耗尽,则触发 out-of-gas 异常,当前交易失败,调用帧所做的所有状态修改都会被回滚,且已支付的 gas 不会被返还。

1.3.3　以太坊虚拟机

Solidity 是开发以太坊智能合约的主流语言,它的语法类似于 JavaScript。类似于 Java 程序通过 Java 虚拟机(Java Virtual Machine, JVM)将代码解释成字节码执行,以太坊智能合约经由以太坊虚拟机(EVM)解释成为字节码方可执行。换言之,EVM 是被沙箱封装起来、完全与外界隔离的以太坊智能合约运行环境,因此运行于 EVM 的所有智能合约都无法访问托管虚拟机的计算机上运行的网络架构、文件系统或其他进程。EVM 是一个图灵完备的虚拟机,能够完成的计算量实际上受限于用户提供的 gas 数量。

EVM 基于堆栈架构,一切运算都在栈上进行,栈上的数值可能是指令需要取用的参数,也可能是被压入栈的运算结果。堆栈的最大深度为 1 024,每个栈条目的大小为 256 位,意味着 EVM 是一个 256 位的机器。高级语言编写的智能合约编译成 EVM 字节码后方可执行,EVM 字节码是一串十六进制数字编码的字节数组。字节码的解析以一个字节为单位,每个字节都表示一个 EVM 指令或一个操作数据。保留最小规模的 EVM 指令集合,从而尽量减少引起共识问题的错误实现。所有 EVM 指令操作都在 256 bit 的基本数据类型上进行,具备常用的算术、位、逻辑和比较操作,也可以做到条件和无条件跳转。

2　智能合约安全性问题

随着智能合约数量的增多,去中心化应用(Decentralized Application, DApp)的推广,智能合约涉及的数字资产呈指数级别增长。相比传统软件,智能合约的安全问题更加棘手,现实情况也更加严峻。

1) 智能合约的可信度源自其不可篡改性,一旦被部署上线便无法修改。任何人都可对合约存在的安全漏洞发起攻击,如果合约没有相应的防御措施,便将无法遏止安全问题的恶化,从而严重损害合约本身的经济价值以及公众对项目的信任。

2) 很多项目会公开智能合约源码。源码的公开透明虽能提升用户对合约的信任度,却也大幅度降低了黑客攻击的成本,每一个暴露在开放网络上的智能合约都有可能成为专业黑客团队的金矿和攻击目标。

3) 智能合约的开发过程存在纰漏。由于起步晚,发展时间短,智能合约本身就有很多不足; 同时市面上专业的技术人员严重匮乏,不严谨的代码参考、拷贝和修改等人为因素都会引起漏洞。

NCC Group 总结出智能合约中出现频率最高的 10 类安全问题为分别: 代码重入、访问控制、整数溢出、未严格判断不安全函数调用返回值、拒绝服务(Denial of Service, DoS)、可预测的随机处理、竞争条件/非法预先交易、时间戳依赖、短地址攻击以及其他未知漏洞类型[4]。安比(SECurity of BITcoin, SECBIT)实验室安全团队深度扫描检测了当前正在运行的 23 357个智能合约源代码,通过智能合约安全审计模型扫描,实验室安全专家发现其中大量合约代码存在着不同程度的安全隐患。扫描结果显示,这 23 357 个智能合约源代码中共有 405 882处不符合安全开发规范,平均每个合约有超过 17 个规范违反项。其中低级别(Low)的安全问题有 26 821 个,主要集中在未指明版本号以及高 gas 消耗等问题; 中级(Medium)安全问题有 7 202 个,主要集中在整数溢出、除法、依赖时间戳、区块哈希的运算; 中高危(High)安全问题有 572 个,主要集中在代码重入、短地址攻击、强制转账、使用合约余额来做判断、高地址脏数据、tx-origin 的误用上[5]。

下文将依据已发生的安全事件和已有的代码审计工作,对部分智能合约漏洞类型及利用方式进行分析,并提出一些防范方法。

2.1　代码重入

以太坊智能合约能够调用和使用其他外部合约的代码。合约也通常能够处理以太币,因此常常会把以太币传送到外部账户的地址。调用外部合约或将以太币发送至某指定地址的操作要求合约提交一个外部调用,然而这些外部调用若是被攻击者劫持,就会迫使合约进一步执行其他恶意代码,包括回调自己,这就等同于代码执行了重新进入合约的操作。The DAO 事件即是重入攻击的典型案例,可以采取多种方法规避潜在的重入漏洞: 第一种是将以太币发送至外部合约时,使用内置的 transfer( ) 函数,该函数仅发送的 2 300 个 gas 不足够目的地址上的合约调用另一个合约; 第二种方法是先保证所有改变状态变量的逻辑发生后,再允许以太币被从合约或任何外部调用发送出去; 第三种方法是引入一个互斥系统,即添加一个代码执行期间锁定合约的状态变量,从而防止重新入口的调用[6]。

2.2　权限控制问题

智能合约的权限控制问题大多是由于未能明确函数的可见性,或者未能作充足权限检查,导致攻击者能够访问或修改到不该访问的函数或变量。Solidity 为函数和状态变量设置了 external、public、internal 和 private 四种可见性说明符,函数默认可见性为 public,即对应函数既允许内部调用,也可以作为合约对外接口的一部分被外部合约调用访问,因此,若转账等重要和敏感函数未能明确可见性,很可能导致合约中的资金流失。另外,智能合约中存在管理员 Owner 的概念,Owner 一般拥有超级权限,而合约的初始化和 Owner 地址绑定则是由智能合约的构造函数实现的。一方面,如果构造函数声明方式有误,就会成为普通函数,任何人都可以调用,并将自己设为合约的管理员; 另一方面,部分敏感函数需要 onlyOwner 权限,若未给该函数添加 onlyOwner 函数修饰器,则任何人能够操纵该函数,破坏合约的执行逻辑。2017 年 11 月,Parity 钱包合约越权访问漏洞被触发,攻击者通过调用钱包初始化函数,使自己成为 walletlibrary 的管理员,随后以管理员身份调用 kill 函数,杀死 walletlibrary,导致价值超过 1.5 亿美元的

以太币被冻结。

规避权限控制漏洞的一般方法是，正确添加函数可见性说明符或修饰符来控制函数调用的范围和权限，并对敏感操作作出严格的权限检查。

2.3 整数溢出

以太坊虚拟机用几种固定长度的数据类型表示整数。这意味着一个整数变量只可以表示一定范围的数字。以变量类型 uint8 为例，其变量长度为 8 bit，支持存储的数字范围是 [0，255]，若试图将大小超过这个范围的数据存储到 uint8 中，虚拟机则自动截断高位，导致运算结果异常。整数溢出包括加法溢出、减法溢出和乘法溢出三种类型。整数溢出漏洞属于高危漏洞，若被攻击者利用，很容易造成超额铸币、下溢增持、高卖低收等事故。已曝出的安全事件如 SMT（SmartMesh Token）、BEC、EDU、BAI 漏洞，都是由于转账逻辑中产生了整数溢出，导致代币的无限增发或任意转账。防范整数溢出漏洞的一般方法是使用或构建数学库来替代标准的数学运算符，OpenZeppelin[7] 提供了有安全检查的 SafeMath 数学计算库，使用 SafeMath 库函数进行开发能够有效避免溢出漏洞。

2.4 异常处理问题

Solidity 在执行到 gas 耗尽、调用栈溢出、执行到 throw 语句这几种场景下会抛出异常，并通过回退状态的方式来处理异常，即撤销当前调用及其所有子调用所改变的状态，同时给调用者返回一个错误标识。当子调用中发生异常时，异常通常会自动向上传播；但是 send 以及底层调用函数 call、delegatecall、callcode 发生异常时，只是返回 false，而不抛出异常，因此不能仅仅根据有无异常抛出判断合约是否成功执行，在使用底层方法时，必须通过检查返回值，对可能的异常进行处理。

2.5 短地址攻击

短地址攻击针对基于 ERC20 类型的代币转发问题。转发代币需要调用函数 transfer（address addr，uint amount），addr 表示发送代币的目标地址，amount 为发送代币的数额。对于 EVM 层面，交易的消息输入实际上是传入了一串 16 进制字节码，分为三部分，分别是 4 字节的方法名哈希值、32 字节的目标以太坊地址以及 32 字节的代币数额。假如账户地址 addr 以 0 结尾，而攻击者故意少输入末尾的 0，EVM 解析时就会从下一个参数，即代币数额 amount 的高位取缺少的编码位数对地址进行补全，而 amount 高位又恰好是 0，这样地址就会保持不变；然后，EVM 对于 amount 参数从末位补 0 直至正常的编码位数，这意味着将 amount 左移了数位，从而实现转移超出实际应该转发的 token 数。实践中预防短地址攻击的措施是对用户输入进行严格的检测，避免接受畸形地址。

2.6 操纵区块时间戳

有些智能合约的执行依赖于当前区块的时间戳，时间戳直接或通过产生随机数间接影响着执行的结果，但是这存在很多安全问题：一方面，时间戳由矿工打包交易时设置，矿工很可能利用自主权对时间戳作些微的改动；另一方面，以太坊未来有可能会对出块时间上作出调整，因此通过块高度来预估时间是存在不合理性[8]，因此，区块时间戳不应该用于熵或产生随机数，例如，它们不应成为赢得一场比赛或一个重要状态转移的决定性因素。

2.7 拒绝服务攻击

针对智能合约的拒绝服务（DoS）攻击本质是让用户在一小段时间内或在某些情况下永久性地无法使用合约，这可能会使合约中的以太币（Ether）永远无法提取出来。攻击方法有三种[8]：第一种是通过（Unexpected）Revert 发动 DoS 攻击，在外部函数执行的结果决定着智能合约状态的转移而这个执行一直失败的情况下，若不采取防护措施，则该智能合约可能会受到 DoS 攻击。2016 年以太坊游戏 The King of the Ether Throne 遭到的攻击即为典型案例，DoS 攻击下退位君王的补偿以及未接受款项均无法退回玩家的钱包。第二种是通过区块 gas limit 发动 DoS 攻击。以太坊给每个区块能消耗的 gas limit 设定了上限，超过此上限交易便会失败，即便没有蓄意的攻击也可能产生问题。然而，更糟糕的情况是如果攻击者操控了 gas 的花销，导致达到区块 gas limit 的上限，整个转账的操作也会以失败告终。第三种是所有者操作发动 DoS 攻击。很多代币合约的 Owner 账户拥有开启或暂停交易的权限，若是没有妥善保管 Owner 账户，代币合约交易可能会被永久冻结，形成非主观的拒绝服务攻击。

2.8 伪随机问题

Solidity 无法创建随机数，实际上，每个创建随机数的算法都是伪随机的，合约开发者编写随机数生成函数时，往往利用区块号（block. number）、区块时间戳（block. timestamp）、区块难度（block. difficulty）和区块 gas 限制（block. gaslimit）等区块头相关的参数或者其他区块信息，如合约中存储的数据来产生随机数；但是，区块头参数可以被矿工获知的特性和链上数据公开的问题，都导致生成的随机数是可预测的，攻击者可以利用这一点进行攻击。

防范伪随机问题的措施是，使得随机数的来源不依赖于区块参数，而是尽量从区块链之外，例如 Oraclize 等第三方服务来获取随机数。

2.9 delegatecall 委托调用

Solidity 中提供了 delegatecall 函数，用于实现合约之间的相互调用以及交互，它的特点是调用后内置变量 msg 的值不会修改为调用者，但执行环境为调用者的运行环境，不当使用 delegatecall 会导致非预期代码的执行。例如，若 delegatecall 的调用地址和调用的字符序列均由用户传入，则完全可以调用任意地址的函数。另外，由于 delegatecall 的执行环境为调用者环境，调用者与被调用者拥有相同变量的情况下，若被调用的函数修改该变量值，则修改的是调用者中的变量。Parity Multisig 钱包的第二次攻击就是 delegatecall 滥用的代表案例。防护措施是谨慎使用此类底层函数，对用户输入的调用发起地址及调用参数作出严格限定。

2.10 竞争条件/非法预先交易

用户可以通过竞争代码的执行，得到非预期的状态。以太坊中，各节点把发生的交易汇集起来并形成区块，一旦矿工解决了共识机制，就把这些交易认定为有效的。解决该区块的矿工根据 gas price，选择把该矿池中的哪些交易记录到该区块中。这里就形成了一个潜在的攻击面：攻击者通过监视有可能包含问题解决方案的交易池，更改合约中对攻击者不

利的状态或者修改攻击者的权限。攻击者于是能够获得此交易的数据,以更高的 gas price 创建自己的交易,并且将该交易包含在原始数据之前的区块中,从而获得优势竞争条件得以预先交易。

3　智能合约漏洞检测手段及相关工作

出于“代码即规则”,智能合约一旦被部署便不可更改,即便有恶意交易被记录下来,也不可以将其从区块链中删除。回滚交易的唯一方法是执行硬分叉,即通过修改区块链中的共识协议把区块链中的数据恢复到过去某一状态,而这无法回避开发者客观上存在滥用“专业垄断”的质疑,一定程度上冲击了区块链系统去中心化的理念,所以必须在智能合约上线之前,对其进行全面深入的代码安全审计与测试,充分分析潜在的安全威胁,尽可能规避漏洞。

针对智能合约安全问题,应该从开发人员使用安全库进行开发、安全团队开展合约测试、合约审计这三个角度采取措施。其中,合约测试指用大量的测试用例引导合约的执行,验证其是否符合预期行为,但测试用例无法确保覆盖所有可能的路径,所以即便测试结果没有发现问题,也不能排除漏洞存在的可能性。合约审计是安全团队通过专业的手段,检查出合约的代码实现和业务逻辑中存在的漏洞和隐患,并向项目方提出业务逻辑上的指导和建议。这虽能发现大部分常见的漏洞和风险,但由于现有的审计工作在一定程度上依赖于审计人员的主观判断和经验,无法根除安全风险和漏洞。借鉴传统软件漏洞检测方法,可以采用以下几种手段检测智能合约漏洞。

3.1　形式化验证

形式化验证用逻辑语言对智能合约文档和代码进行形式化建模,通过严密的数学推理逻辑和证明,检查智能合约的功能正确性和安全属性[9],克服了用传统测试手段无法穷举所有可能输入的缺陷,能完全覆盖代码的运行期行为,可以确保在一定范围内的绝对正确,弥补了合约测试和合约审计工作的局限性,因此形式化验证已初步应用于高铁、航天、核电等安全攸关的领域,并且取得了非常好的效果。另外,一些较为复杂的业务逻辑以及更高阶的性质,例如经济学、博弈论领域的问题,很难通过安全测试或审计手段得到有效验证,所以对于规模较小但功能设计复杂的智能合约来说,形式化验证必定是维护其安全性的有效方法之一。形式化验证方法包括模型检验和定理证明两种。模型检验是列举出系统所有可能的状态并逐一进行检验,这种方法全自动化却只适用于小型系统;而定理证明首先把系统代码提取成抽象的数学模型,然后对定理进行证明,这种方法能够适应大规模系统,但需要首先将系统的运作方法转换成验证系统能够理解的语言。成都链安科技研发的 VaaS(Verification as a Service)[10] 是全球首个同时支持 EOS、以太坊区块链智能合约的自动形式化验证平台,支持多种合约开发语言,可以对整型溢出、可重入攻击、异常可达状态、多签名钱包、Tx. origin 漏洞等 10 多种常规安全漏洞进行“一键式”自动化检查。

由耶鲁大学、哥伦比亚大学安全团队研发的 Certik[11] 是一个形式化验证框架,经过 Certik 验证的智能合约、DApp 以及区块链会被附上证书形式的标志,来表示其正确性和安全性。Certik 平台的主要功能包括应用深度学习技术来构建智能标签框架;通过层级分解将复杂的证明任务分解为更小的任务;可插拔的验证引擎;认证的 DApp 库等。

Bhargavan 等[12] 提出了一个智能合约分析和验证框架,该框架通过 Solidity* 和 EVM* 工具将智能合约源码和字节码转化成函数编程语言 F* ,以便分析和验证合约运行时安全性和功能正确性。目前,Coq[13]、Isabelle/HOL[14]、Why3[15] 等工具也实现了 EVM 的语义表示,并做了一些形式化验证智能合约的工作。

3.2　模糊测试

模糊测试是一种通过构造非预期的输入数据并监视目标软件在运行过程中的异常结果来发现软件故障的方法[16]。对智能合约进行模糊测试时,利用随机引擎生成大量的随机数据,构成可执行交易,参考测试结果的反馈,随机引擎动态调整生成的数据,从而探索尽可能多的智能合约状态空间。基于有限状态机分析每一笔交易的状态,检测是否存在攻击威胁。自动化工具 Echidna[17] 采用了模糊测试技术来对 EVM 字节码进行检测,但是不能保证 API 功能的稳定性。

文献 [18] 中提出的 ContractFuzzer 是第一个基于 Ethereum 平台的智能合约安全漏洞模糊测试框架,支持 gas 耗尽终止、异常处理混乱、重入、时间戳依赖、区块号依赖、危险的 delegatecall 调用和以太币冻结七种漏洞的检测。ContractFuzzer 包含离线 EVM 测试工具和在线模糊测试工具。构建了一个 Web 爬虫,从 Etherscan 网站上获得部署在以太坊上的智能合约,爬虫可以提取合约创建代码(智能合约的二进制文件)、应用程序二进制接口(Application Binary Interface, ABI) 以及这些合约的构造函数参数。和其他智能合约漏洞检测工具相比,ContractFuzzer 支持更多的漏洞类型,有效降低了误报率;但由于测试用例生成的随机性,所能涵盖的系统行为有限,无法达到理想的路径覆盖率,很难找出所有的潜在错误。

3.3　符号执行

符号执行的核心思想是使用符号值代替具体值执行程序。对于程序分析过程中任意不确定值的变量,包括环境变量和输入等,都可以用符号值代替。符号执行中的“执行”是指解析程序可执行路径上的指令,根据其语义更新程序执行状态,等同于解释执行[19]。借助符号执行检测智能合约漏洞的一般过程为,首先将按需将智能合约中不确定值的变量符号化,然后逐条解释执行程序中的指令,在解释执行过程中更新执行状态、搜集路径约束,并在分支节点处做 fork 执行,以完成程序中所有可执行路径的探索,发现安全问题。约束求解技术能够对符号执行中搜集的路径约束进行求解,判断路径是否可达,并在特定的程序点上检测变量的取值是否符合程序安全的规定或者可能满足漏洞存在的条件。

Luu 等[20] 开发了一种基于符号执行的静态分析工具 Oyente,可以直接运行在 EVM 字节码上,而无需访问 Solidity 或 Serpent 等高级语言。Oyente 支持检测交易顺序依赖、时间戳依赖、代码重入、错误处理异常等漏洞。Oyente 覆盖了大部分的 EVM 操作码,但由于缺失类型、不同函数调用对相同字

节码的重用等上下文信息，仅从 EVM 字节码很难重建开发意图，因此 Oyente 无法验证一些公平性和正确性问题（包括整数溢出等）。此外，Oyente 简化地处理循环，通过限制循环次数防止路径爆炸，这导致了部分缺陷的漏报。

Manticore[21] 是另一款基于符号执行且支持 EVM 的动态二进制分析工具，可以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，并验证关键功能的安全性，标记出整数溢出、未初始化内存等安全问题的类型。

上述工具都致力于检测低级别的安全违规行为和漏洞，例如整数溢出、可重入和未处理的异常等，大多只对单个调用行为进行建模，均没有对合同执行流进行推理。为解决此问题，Nikolic 等开发了基于符号执行的 MAIAN[22] 工具，针对贪婪合约、浪子合约、自杀合约，精确定义这三类漏洞合约可核查的 trace 特性，着重分析对合约的调用序列，测试了近 100 万个合约，以较快的速率发现大量真阳性漏洞。

Tsankov 等开发了一种基于符号抽象的合约静态分析工具 Securify[23]，改进了市面上其他工具无法确切判定合约是否包含漏洞的缺陷，针对各安全属性定义了遵从模式和违反模式两种模式。Securify 从合约的字节码（或者源代码，可以编译为字节码）开始，符号化地分析合约的依赖关系图，提取出精确的语义事实，并使用这些事实匹配遵从和违反模式。根据检查结果，Securify 将所有合约行为分为违规、警告和遵从三类，所有与违规模式匹配的行为报告为违规，所有与遵从模式匹配的报告为遵从，其他行为报告为警告，因此，用户需要手动分类为真阳性或假阳性的行为只有警告行为（标记为 warning），减少了大约 65.9% 的工作量。

Johannes Krupp 等开发的 TEETHER[24] 工具，在仅提供 EVM 字节码的情况下支持智能合约的自动漏洞识别和漏洞利用。TEETHER 的工作流程为：首先根据 EVM 字节码，使用反向切片迭代地重建控制流图（Control Flow Graph，CFG）；其次寻找关键指令，共有四条，要从合约中提取以太币必须至少执行其中一条，CALL 和 SELFDESTRUCT 能够导致以太币的直接传输，CALLCODE 和 DELEGATECALL 指令允许在合约的上下文中执行任意 EVM 字节码；接着对于提取出的每条关键指令，根据其关键参数计算后向程序切片，随后使用 A^* 算法探索路径，其中路径的成本定义为该路径在 CFG 中遍历的分支数；一旦生成关键路径，路径约束生成模块就以符号方式执行路径，收集一组路径约束；根据约束求解结果，推断出导致智能合约利用的事务列表。相比其他工具，TEETHER 不仅实现了漏洞检测，还实现了漏洞的自动利用，但其局限性在于只支持检测单个合约内的漏洞，而不能发现调用其他合约产生的漏洞。

3.4 污点分析

本质上来说，污点分析是针对污点变量的数据流分析技术。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。Bernhard Muller 研发了一种全新的智能合约安全分析工具 Mythril[25]，Mythril 集成了混合符号执行、污点分析和控制流检查，可以检测出整数溢出、不可信合约的外部调用等一系列的常见安全问题；但它无法检测出智能合约的业务逻辑问题，且极易产生误报。

4 总结与展望

4.1 现有研究的不足

如前所述，近年来虽出现了一些智能合约漏洞自动化检测工具，但都存在一定的缺陷，主要体现在以下几个方面。

1）各工具涵盖的漏洞类型不完整，总结部分现有工具支持的漏洞类型如表 2 所示。

可以看出，各工具检测手段单一，涵盖的漏洞类型不完整，它们大多只能检测低级别的安全违规行为和漏洞，缺乏对合约执行流的推理，难以发现不同合约间调用产生的安全问题，且无法有效检测公平性等逻辑缺陷，因此，使用单个工具对合约进行更全面的安全验证是一个待解决的技术难题。

表 2 智能合约漏洞检测工具对比

Tab. 2 Comparison of smart contract vulnerability detection tools

工具/平台	漏洞类型
Oyente	交易顺序依赖、时间戳依赖、代码重入、未处理的异常
Mythril	重入、危险的 delegatecall 调用、整数溢出、未检查的 CALL 返回值、交易顺序依赖、时间戳依赖等
MAIAN	贪婪合约、浪子合约、自杀合约
Securify	重入、以太币冻结、未验证的输入、交易顺序依赖、未处理的异常、无限制的以太币流
ContractFuzzer	gas 耗尽终止、异常处理混乱、重入、时间戳依赖、区块号依赖、危险的 delegatecall 调用、以太币冻结
VaaS	整型溢出、可重入攻击、多签名钱包、Tx. origin 漏洞等 10 多种常规漏洞

2）代码覆盖率低，漏报率和误报率高。本文选取了 150 个智能合约 Solidity 源码样本，就交易顺序依赖（Transaction-Ordering Dependence，TOD）、代码重入（Reentrancy）、未处理的异常（Mishandled Exception）三种漏洞类型，对 Securify、Oyente 和 Mythril 三种工具的检测效果进行对比，结果如表 3 所示。其中，TW（True Warning）、FW（False Warning）和 UV（Unreported Vulnerability）分别表示正确警告、错误警告和未报告的漏洞占样本总数的比例。

表 3 检测结果比较

Tab. 3 Comparison of experimental results

漏洞类型	工具	TW	FW	UV
TOD	Oyente	0.12	0.05	0.48
	Mythril	0.35	0.14	0.25
	Securify	0.47	0.03	0.13
Reentrancy	Oyente	0.45	0.38	0.13
	Mythril	0.54	0.12	0.04
	Securify	0.56	0.20	0.02
Mishandled Exception	Oyente	0.15	0.00	0.18
	Mythril	0.19	0.00	0.14
	Securify	0.24	0.00	0.09

与 Oyente 和 Mythril 相比，Securify 在判定真阳性漏洞上具有显著优势，但三种工具均有较多的错误警告以及漏报的漏洞。

3) 现有研究工作大多不能做到完全自动化,无法明确报告合约行为是否违规,对于检测出来的所有疑似漏洞,需要用户手动将其分类为真阳性或假阳性,这大幅度增加了智能合约安全检测的工作量,因此,下一步研究需要对于挖掘出的疑似漏洞,使用工具直接验证其是否能实际造成安全威胁,减少人工验证环节工作量,并尽可能地提供漏洞利用方案,提出漏洞修补措施和智能合约安全编程建议。

4) 审计时间较长,漏洞挖掘效率低下。对于每个合约的检测时间,Mythril 平均花费 60 s,Oyente 大约为 30 s,而 Securify 大约为 20 s。面对数目和规模与日俱增的智能合约,提高漏洞发现效率也是亟需解决的重难点。

4.2　未来研究方向与改进思路

1) 扩展形式化验证的应用范围。

对于目前学术界颇为关注的形式化验证方法,用数学推演来验证复杂系统,安全有效但难度很高。未来的研究应针对不同的业务目标定制对应的验证规范描述,突破成本昂贵、不适应大规模合约等技术限制,并扩展形式化验证的应用范围,从验证一般功能属性和安全属性、检测常见漏洞到逐步实现经济学、博弈论范畴中复杂业务逻辑及公平性等高阶性质的证明。

2) 提取重点路径,缩减路径空间。

基于攻击者目标是非法窃取加密货币的假设,结合现有智能合约审计经验和已曝漏洞分析,寻找智能合约中易产生漏洞的高危指令,如 SUICIDE、CALL、ORIGIN、ASSERT_FAIL 等,定义涉及这些操作码的路径为重点路径。为了提高漏洞挖掘效率,实践中不必对所有可能的执行路径进行检查,仅符号执行关注的重点路径并进行漏洞验证,可以有效地缩减路径空间。

3) 符号执行辅助的模糊测试。

现有的工具通常是对一种典型方法的具体实现,但是在执行具体漏洞挖掘任务时,因需求和重点不同,使用不同的辅助工具或者不同的检测方法组合往往能达到更好的效果,可以考虑符号执行辅助的模糊测试方法: 一方面,对于已有的智能合约模糊测试工具 Echidna 和 ContractFuzzer,虽简单有效,执行效率高,适合发现隐藏的未知漏洞,但局限性在于随机性强,生成的测试用例的质量影响着代码覆盖率,难以对合约进行全面的测试,并且缺乏对智能合约语义的洞察力,很难发现多阶段安全漏洞、多点触发漏洞、访问控制漏洞、糟糕的逻辑设计等。另一方面,对于已有的基于符号执行的工具,如 Oyente和 Manticore,在处理智能合约中较多的不定常数组时,或是合约规模庞大、跳转指令较多时,容易产生路径爆炸,导致测试效率不高。

因此未来可以研究动态符号执行辅助的模糊测试技术,使用动态符号执行弥补模糊测试理解语义的缺失,推断出到达特定程序状态的约束条件,通过约束求解产生能够触发测试者所关注逻辑的合理输入。据此恰当地改变模糊测试的输入,提供额外的测试用例,触发先前未覆盖的代码区域,因此本文设计一种符号执行辅助的智能合约模糊测试框架,如图 1 所示。

框架采用模糊测试作为前端处理,利用种子测试用例来驱动被检测智能合约的执行; 采用 EVM 字节码覆盖率监测处理作为中间层处理,其中在被测试程序的执行过程中,记录被测试合约覆盖的基本块,由此计算模糊测试的覆盖率; 在符号执行处理中生成重点关注代码区域的新测试用例,然后将新的测试用例反馈至模糊测试,使得模糊测试利用新的测试用例来驱动被测试合约的执行。这样既能融合模糊测试快速高效、低消耗的优势与符号执行探索能力强的优势,又克服了模糊测试代码覆盖率不足和符号执行路径爆炸的缺陷,可以有效提高大中型规模智能合约的漏洞分析与验证效率。

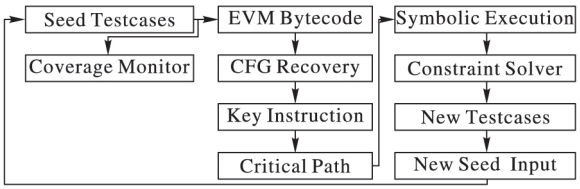

图 1　符号执行辅助的模糊测试框架

Fig. 1　Symbolic-execution-based fuzzy test framework

4) 完善智能合约漏洞库,建立漏洞挖掘工具效率评价方法。

当前关于智能合约的测试尚未有标准的案例集,因此,为了验证智能合约漏洞挖掘工具的有效性,同时给智能合约的安全开发提供参考,下一步工作需要根据已爆发的安全事件以及合约审计经验,总结归纳出涵盖类型完善的智能合约漏洞库。综合考虑漏报率、误报率、检测时间、支持漏洞类型和漏洞的危害等级等因素,建立漏洞挖掘工具效率的综合评价指标,将市面上的相关工具进行对比分析,验证其有效性,并为新工具的研发和改进提供指导。

5　结语

智能合约极大地扩展了区块链的应用场景与现实意义,但频发的安全事故严重阻碍了它的发展。本文梳理了智能合约面临的安全问题,讨论了形式化验证、模糊测试和符号执行等主流的智能合约漏洞检测手段,通过对几个现有漏洞挖掘工具的验证分析发现,当前研究主要存在涵盖的漏洞类型不完整、误报及漏报率高、检测效率低下、依赖人工复核等缺陷,并针对相关工作中存在的不足,提出几点未来研究方向。下一步工作的重难点在于采用不同的检测方法组合,着力提高漏洞挖掘的准确性、效率和自动化程度,满足智能合约规模和复杂度与日俱增的漏洞挖掘需求。

参考文献 (References)

[1] 马昂,潘晓,吴雷,等. 区块链技术基础及应用研究综述 [J]. 信息安全研究,2017,3(11):968-983. (MA A, PAN X, WU L, et al. A survey of the basic technology and application of block chain [J]. Journal of Information Security Research, 2017, 3(11): 968-983.)

[2] BUTERIN V. Ethereum: a next-generation smart contract and decentralized application platform [EB/OL]. (2014-01-23) [2018-09-08]. https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform-1390528211/.

[3] 长铗,韩锋,杨涛. 区块链: 从数字货币到信用社会 [M]. 北京: 中信出版社,2016: 62-73. (CHANG J, HAN F, YANG T.

Blockchain: From Digital Currency to Credit Society [M]. Beijing: China CITIC Press, 2016: 62 - 73.)

[4] NCC Group. Decentralized application security project top 10 of 2018 [EB/OL]. (2018-07-08) [2018-09-08]. https://www.dasp.co/index.html.

[5] SECBIT. Frequent smart contracts events, security development requires standardization [EB/OL]. (2018-05-07) [2018-10-20]. https://www.jianshu.com/p/9d78f5110af1.

[6] MANNING A. Solidity security: comprehensive list of known attack vectors and common anti-patterns [EB/OL]. (2018-05-30) [2019-01-03]. https://blog.sigmapri-me.io/solidity-security.html.

[7] ARIAS L, SPAGNUOLO F, GIORDANO F, et al. OpenZeppelin [EB/OL]. (2016-07-31) [2018-12-06]. https://github.com/OpenZeppelin/openzeppelin-Solidity.

[8] ATZEI N, BARTOLETTI M, CIMOLI T. A survey of attacks on Ethereum smart contracts (SoK) [C]// Proceedings of the 2017 International Conference on Principles of Security and Trust. Berlin: Springer, 2017: 164 - 186.

[9] FEY G. Assessing system vulnerability using formal verification techniques [C]// Proceedings of the 2011 International Conference on Mathematical and Engineering Methods in Computer Science. Berlin: Springer, 2011: 47 - 56.

[10] CSDN Research and Development Technology. Formal verification is a sharp weapon for smart contracts safety [EB/OL]. (2018-06-12) [2018-09-08]. https://blog.csdn.net/CDLianan/article/details/80665163.

[11] Certik. 用形式化验证的方式构建安全的智能合约和区块链生态系统 [EB/OL]. (2018-07-11) [2019-01-06]. https://baijiahao.baidu.com/s?id=1605131670683321304&wfr=spider&for=pc. (Certik: Constructing secure smart contract and block chain ecosystem by formal verification [EB/OL]. (2018-07-11) [2019-01-06]. https://baijiahao.baidu.com/s?id=1605131670683321304&wfr=spider&for=pc.)

[12] BHARGAVAN K, SWAMY N, ZANELLA B S, et al. Formal verification of smart contracts: short paper [C]// Proceedings of the 2016 Association for Computing Machinery Workshop. New York: ACM, 2016: 91 - 96.

[13] YANG X, YANG Z, SUN H Y, et al. Formal verification for Ethereum smart contract using Coq [J]. International Journal of Information and Communication Engineering, 2018, 12(6): 125 - 130.

[14] HIRAI Y. pirapira/eth-isabelle [EB/OL]. (2016-04-24) [2018-12-18]. https://github.com/pirapira/eth-isabelle.

[15] MARCHE C, MELQUIOND G, FILLIATRE J C, et al. AdaCore/why3 [EB/OL]. (2009-11-29) [2018-12-18]. https://github.com/AdaCore/why3.

[16] BEKRAR S, BEKRAR C, GROZ R, et al. Finding software vulnerabilities by smart fuzzing [C]// Proceedings of the 4th International Conference on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2011: 427 - 430.

[17] SMITH J P, PEREZ B, CHRISTIE C, et al. Trailofbits/echidna [EB/OL]. (2018-06-12) [2018-09-08]. https://github.com/trailofbits/echidna.

[18] JIANG B, LI Y, CHAN W K. ContractFuzzer: fuzzing smart contracts for vulnerability detection [C]// Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering. New York: ASE, 2018: 259 - 268.

[19] 吴世忠,郭涛,董国伟. 软件漏洞分析技术 [M]. 北京: 科学出版社, 2014: 215 - 268. (WU S Z, GUO T, DONG G W. Software Vulnerability Analysis Technology [M]. Beijing: Science Press, 2014: 215 - 268.)

[20] 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. New York: ACM, 2016: 254 - 269.

[21] MOSSBERG M, IVNITSKIY Y, SMITH J P, et al. trailofbits/manticore [EB/OL]. (2017-02-12) [2018-09-08]. https://github.com/trailofbits/manticore.

[22] NIKOLIC I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale [EB/OL]. (2018-02-06) [2018-11-14]. https://arxiv.org/pdf/1802.06038.pdf.

[23] TSANKOV P, DAN A, COHEN D D. Securify: practical security analysis of smart contracts [C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2018: 67 - 82.

[24] KRUPP J, ROSSOW C. teEther: gnawing at Ethereum to automatically exploit smart contracts [C]// Proceedings of the 27th USENIX Security Symposium. Berkeley, CA: USENIX Association, 2018: 1317 - 1333.

[25] MUELLER B, HONIG J, PARASARAM N, et al. ConsenSys/mythril [EB/OL]. (2017-09-17) [2018-12-04]. https://github.com/ConsenSys/mythril.

This work is partially supported by the National Key Research and Development Program of China (2017YFB0802900).

FU Menglin, born in 1995, M. S. candidate. Her research interests include vulnerability mining, blockchain security.

WU Lifa, born in 1968, Ph. D., professor. His research interests include vulnerability mining, network management.

HONG Zheng, born in 1979, Ph. D., associate professor. His research interests include protocol reverse, vulnerability mining.

FENG Wenbo, born in 1994, M. S. candidate. His research interests include protocol identification, machine learning.