计算机科学
COMPUTER SCIENCE

以太坊智能合约模糊测试技术研究综述

黄松，杜金虎，王兴亚，孙金磊

引用本文

黄松，杜金虎，王兴亚，孙金磊. 以太坊智能合约模糊测试技术研究综述[J]. 计算机科学, 2022, 49(8): 294-305.

HUANG Song, DU Jin-hu, WANG Xing-ya, SUN Jin-lei. Survey of Ethereum Smart Contract Fuzzing Technology Research[J]. Computer Science, 2022, 49(8): 294-305.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于 QRNN 的网络协议模糊测试用例过滤方法

Testcase Filtering Method Based on QRNN for Network Protocol Fuzzing

计算机科学, 2022, 49(5): 318-324. https://doi.org/10.11896/jsjkx.210300281

SymFuzz:一种复杂路径条件下的漏洞检测技术

SymFuzz:Vulnerability Detection Technology Under Complex Path Conditions

计算机科学, 2021, 48(5): 25-31. https://doi.org/10.11896/jsjkx.200600128

基于深度优先搜索的模糊测试用例生成方法

Fuzzing Test Case Generation Method Based on Depth-first Search

计算机科学, 2021, 48(12): 85-93. https://doi.org/10.11896/jsjkx.200800178

Android 组件间通信的模糊测试方法

Fuzz Testing of Android Inter-component Communication

计算机科学, 2020, 47(11A): 303-309. https://doi.org/10.11896/jsjkx.200100122

基于协议状态图遍历的 RTSP 协议漏洞挖掘

Mining RTSP Protocol Vulnerabilities Based on Traversal of Protocol State Graph

计算机科学, 2018, 45(9): 171-176. https://doi.org/10.11896 / j.issn.1002-137X.2018.09.028

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220500069

以太坊智能合约模糊测试技术研究综述

黄　松[1]　杜金虎[1]　王兴亚[1,2]　孙金磊[1]
1 陆军工程大学指挥控制工程学院　南京 210007
2 南京工业大学计算机科学与技术学院　南京 211816
(huangsong@aeu.edu.cn)

摘　要　运行在区块链平台之上的智能合约,完成了不同参与者之间协议的达成和自动执行,同时也管理了大量的数字资产,智能合约漏洞的频繁爆出,造成了难以估量的经济损失。模糊测试是一种有效的动态漏洞检测技术,已经被应用于智能合约安全研究。文中分析了现有综述工作对智能合约模糊测试的总结不足的问题,并提出了智能合约模糊测试的基本框架;以目前智能合约安全研究中最广泛的以太坊智能合约为例,介绍了与智能合约紧密相关的账户机制和交易结构,总结了智能合约区别于传统程序的特点;阐述了智能合约的漏洞,并对这些智能合约模糊测试技术覆盖的漏洞进行了比较;进一步地,从单交易和交易序列两个方面对已有智能合约模糊测试技术的输入生成进行了分析;从函数层面、交易层面和交易序列层面对测试输入变异进行了总结;对已有智能合约模糊测试技术的测试预言使用进行了简述;另外,还总结了智能合约模糊测试的技术评价指标。最后,提出了当前智能合约模糊测试技术研究面临的问题,并对未来的研究方向进行了展望。

关键词:以太坊智能合约;模糊测试;输入生成;输入变异;测试预言
中图法分类号　TP311

Survey of Ethereum Smart Contract Fuzzing Technology Research

HUANG Song[1],DU Jin-hu[1],WANG Xing-ya[1,2] and SUN Jin-lei[1]
1 Institute of Command and Control Engineering,Army Engineering University of PLA,Nanjing 210007,China
2 College of Computer Science and Technology,Nanjing Tech University,Nanjing 211816,China

Abstract　Smart contracts running on the blockchain platform complete the establishment and automatic execution of a greements between different participants,and also manage a large number of digital assets. The frequent exposure of smart contract loopholes has caused incalculable economic losses. Fuzzing is an effective dynamic vulnerability detection technique that has been applied to smart contract security research. This paper analyzes the problem of insufficient summarization of smart contract fuzzing in existing review work,and proposes a basic framework for smart contract fuzzing. Taking Ethereum smart contracts as an example,which are currently the most widely studied in smart contract security,the account mechanism and transaction structure closely related to smart contracts are introduced,and the characteristics of smart contracts that are different from traditional programs are summarized. The vulnerabilities of smart contracts are expounded,and the vulnerabilities covered by these smart contract fuzzing techniques are compared. Furthermore,the input generation of the existing smart contract fuzzing technology is analyzed from the aspects of single transaction and transaction sequence. The input mutation is summarized from the functional level,transaction level and transaction sequence level. The use of test oracles for existing smart contract fuzzing techniques is briefly described. In addition,the corresponding technical evaluation indicators are also summarized. Finally,the problems faced by smart contract fuzzing are proposed,and the future research directions are prospected.

Keywords　Ethereum smart contract,Fuzzing,Input generation,Input mutation,Test oracle

到稿日期:2022-05-07　返修日期:2022-06-10
基金项目:国家重点研发计划项目(2018YFB1403400);装备综合研究项目(LJ20212C011118);江苏省高等学校自然科学研究面上项目(21KJB520027);江苏省高等学校教育技术研究会高校教育信息化研究课题重点课题(2021JSETKT023);教育部产学合作协同育人项目(202002180001)
This work was supported by the National Key R&D Program of China(2018YFB1403400),Comprehensive Research on Equipment Items(LJ20212C011118),General Project of Basic Natural Science in Colleges and Universities of Jiangsu Province(21KJB520027),Key Project of University Education Information Research(2021JSETKT023) and Project of University-Industry Collaborative Education(202002180001).
通信作者:杜金虎(dujinhu@aeu.edu.cn)

1　引言

智能合约（Smart Contract）[1]指一种运行在区块链技术[2]之上的特殊程序，它借助区块链的共识机制，使得所有参与者在不借助第三方信任的情况下自动达成协议。智能合约由尼克萨博首次提出，他提出参与者在缔结财产管理合约时通过电子数据形式来描述条款，合约排除了第三方的参与并且自动执行。这个概念最初只是一个想法，没有合适的实现条件。区块链是一种在点对点的网络中，通过共识机制来实现的分布式记账[3]技术。与区块链技术的结合，使智能合约获得了执行环境，从概念转为应用。同时，智能合约也促进了区块链技术的发展，使其去中心化、可溯源、不可篡改的特点发挥了更大的作用，将区块链技术的应用从简单的价值交换扩展到了金融[4]、游戏[5]、物联网[6]、保险[7]等多个领域。

以太坊[8]是首个支持智能合约的区块链平台。在以太坊中，智能合约以合约帐户的形式存在，用于管理存储在区块链平台中的电子加密货币。截至2022年3月，由以太坊中智能合约管理的数字资产价值就超过了3 000亿美元[9]。随着智能合约的广泛应用，智能合约的安全问题也逐渐暴露出来，若智能合约存在漏洞，则攻击者可以利用这些漏洞盗走智能合约管理的电子加密货币或者使其无法取出，使合约的拥有者和参与者蒙受难以估量的经济损失，例如DAO合约事件[10]、Parity MultiSig钱包漏洞事件[11]。传统软件可以先后发布多个版本进行更新迭代，持续修复已发现的漏洞，降低损失。与此不同，由于区块链技术不可篡改的特性，智能合约一旦上链，便无法更改，即使在使用过程中发现问题也不能打补丁降低损失，即使发现漏洞被利用也无能为力。因此，在智能合约部署之前进行充分的漏洞检测，找出存在的脆弱性并更改，对于保障用户的数字资产来说至关重要，智能合约漏洞检测已经成为当前重要的研究方向。模糊测试[12]是一种有效的软件动态漏洞检测技术，该技术通过构造大量的测试用例，对被测程序进行多次重复的执行，在执行过程中监测程序的异常行为或状态，以发现程序的漏洞。模糊测试已经被成功应用于包括网络协议安全[13]、Linux内核安全[14]、物联网安全[15]和安卓系统安全[16]的多个领域。近年来，一些学者和安全团队将模糊测试引入智能合约漏洞检测中，以提升智能合约的安全性。例如：北京航空航天大学的姜博老师团队提出了智能合约模糊测试技术的首个研究工作ContractFuzzer[17]；区块链技术公司Consensys[18]提出了智能合约灰盒模糊测试的研究工作Harvey[19]；软件安全研究机构Trail of Bits[20]提出了智能合约模糊测试框架Echidna[21-22]。表1列出了本文调研的智能合约模糊测试工作，并且列出了这些工作公开发表的年份、工具的公开性以及数据的公开性。

作为一种有效的动态漏洞检测方法或者运行时的安全验证技术，模糊测试是以太坊智能合约安全研究中的一个重要方向。然而，目前并没有工作对它进行单独的总结和分析。Almakhour[31]等、Tolmach等[32]和Praitheeshan等[33]对已有的智能合约的安全研究进行了不同角度的总结。Almakhour等从对正确性的形式化验证角度，对定理证明、模型检测和运行时验证工作进行了梳理，并从质量保证的漏洞检测角度，对符号执行、抽象解释和模糊测试工作进行了介绍。Tolmach等将智能合约的安全验证技术总结为模型检测、定理证明、程序验证、符号执行以及运行时验证（测试），并对相关的工作、技术的原理以及工作关注的重点和部分技术的缺陷进行了阐述。Praitheeshan等从静态分析、动态分析和形式化验证3个角度总结了现有的智能合约安全分析方法。同时，他们还介绍了相关的工具以及产生的工具，并且对同类型的工作进行了简单的比较。本文重点关注以太坊智能合约模糊测试技术，通过深入分析智能合约区别于传统程序的特点，总结了一套对智能合约实施模糊测试的总体框架，从测试输入生成、测试输入变异和测试预言的定义3个方面梳理智能合约模糊测试的相关工作。结合智能合约的特点，本文总结了以太坊智能合约模型测试的基本框架，如图1所示。在智能合约的模糊测试中，研究重点包括：智能合约测试输入生成、测试输入的变异、使用测试预言在智能合约的执行过程中进行漏洞检测判定。

表1　智能合约模糊测试技术

Table 1　Smart contract fuzzing techniques

研究工作	年份	公开工具	公开数据集
ContractFuzzer[17]	2018	是	是
Reguard[23]	2018	否	否
ILF[24]	2019	是	否
SolidAudit[25]	2019	是	是
ContraMaster[26]	2019	是	是
Harvey[19]	2020	否	是
ETHPlOIT[27]	2020	否	否
sFuzz[28]	2020	是	否
Echidna[21-22]	2020	是	是
SMARTIAN[29]	2021	是	是
SmartGift[30]	2021	否	否

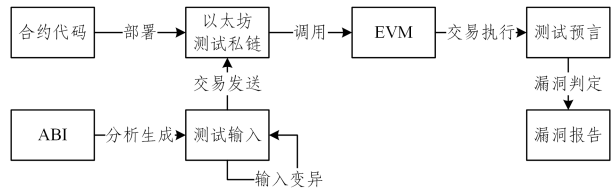

图1　智能合约模糊测试基本框架

Fig. 1　Basic framework of smart contract fuzzing

本文第2节介绍了智能合约的账户机制和交易结构以及智能合约的内容形式，阐述了智能合约区别于传统程序的特点，并对智能合约漏洞进行了总结和分析；第3节从单个交易和交易序列两个方面对现有工作的测试输入生成方法进行了梳理；第4节从函数层面、交易层面和交易序列层面对现有工作的测试输入变异方法进行了讨论；第5节对现有工作中用于漏洞判定的测试预言进行了分析；第6节从覆盖率、漏洞检测混淆矩阵指标和检测时间3个方面对智能合约模糊测试技术评价进行了梳理；最后总结全文并展望未来。

2　智能合约

2.1　以太坊智能合约

以太坊在点对点（Peer to Peer，P2P）的网络中通过共识

机制(Consensus Mechanism)就以太坊的状态达成共识,依靠交易的提交推动区块的增长[34]。在以太坊的结构中,与模糊测试过程紧密相关的主要有两个方面:账户机制和交易结构以及智能合约的内容形式。

2.1.1 账户机制和交易结构

账户是以太坊中的基本记账实体,与银行账户类似,账户中可以存有以太币。以太坊支持两种类型的账户,即外部账户和合约账户。外部账户指由密钥控制的账户,密钥由用户创建和保管,因此外部账户一般由用户使用。合约账户指部署在以太坊上的智能合约使用的账户,这种账户一般由智能合约操控,合约账户中的以太币的管理包括转入转出操作,都按照合约代码定义的逻辑完成。

交易是以太坊中账户之间互动使用的基本信息载体。以太坊中的交易可以形式化地抽象为一个六元组 T:

$$T=\langle From, To, Value, gasPrice, gasLimit, Data\rangle$$

其中,*From* 为交易的发送方地址,*To* 为交易的接收方地址,*Value* 为交易发送的以太币数量,*gasPrice* 为交易发送者愿意支付的 gas 价格,*gasLimit* 为交易发送者愿意支付的 gas 上限,*Data* 为交易附带的额外数据。

依据用途的不同,以太坊中的交易可以划分为转账、合约创建、合约调用 3 种类型,其中转账是以太坊最基础的功能。

在转账交易中,用 *Value* 指定发送的以太币数量,交易的 *Data* 可以为空。智能合约的模糊测试中,几乎不会使用到转账交易,而部署交易和调用交易需要重点关注,因为它们是合约初始化和合约调用的方式。

智能合约编译之后,需要通过交易将其部署到以太坊区块链上才能真正投入使用,具体实现的方式是,通过部署交易将编译合约得到的字节码发送到以太坊上。在这笔交易中,交易的发送方地址 *From* 为部署合约的用户地址,交易的接收方地址 *To* 为空,*Data* 中包含的就是合约的字节码。在部署合约的交易被矿工挖到并打包成区块后,以太坊会将合约部署交易的发送者地址和该地址的交易数作为输入,生成一个新地址。新生成的地址就是合约的地址,是合约在链上的唯一标识。以太坊区块链还会为该地址生成一个合约账户,并将合约代码保存在区块链数据库中。此外,在合约部署时,合约中的构造函数会被自动调用,构造函数中的语句执行对写在区块链永久存储上的合约的全局变量进行了初始化。

智能合约的调用,是围绕智能合约发生的最频繁也是最重要的活动。在合约部署完成之后,外部账户就可以发起调用交易来执行合约。在调用交易中,交易发送方地址 *From* 为发起调用的外部账户地址,交易接收方的地址 *To* 为合约地址,*Data* 中需要包含调用函数的签名和参数信息,同时还要指定交易的发送金额,设置愿意为调用合约执行花费的 *gasPrice* 和 *gasLimit*。参数信息需要遵守 ABI 中定义的规范,以保证调用信息的准确性。值得注意的是,合约代码交易调用后的实际执行是在矿工节点本地的虚拟机 EVM (Ethereum Virtual Machine)中发生的。以太坊区块链上存储着合约代码,但是区块链网络本身不执行合约。矿工的客户端接收到合约调用交易后,通过合约地址从区块链上读取存储的运行代码。根据调用交易中提供的函数签名和函数参数确定交易调用的函数,在本地 EVM 运行后,将运行结果打包提交给区块链,在得到其他节点的确认后,将结果写入区块链的永久存储。

2.1.2 智能合约的内容形式

在以太坊中,智能合约通常由高级语言编写而成。以太坊支持的智能合约编程语言有 Solidity[35], LLL[36], Serpent[37], Vyper[38] 和 Bamboo[39] 等,但绝大多数合约都是使用 Solidity 编写而成的。以 Solidity 智能合约为例,编写后的源码经过 Solc 编译器编译后可以生成字节码(Bytecode)、操作码(Opcodes)和 ABI(Application Binary Interface)。如表 2 所列,字节码是以一串十六进制数字的形式呈现,其中包含了智能合约的所有信息,是 EVM 中实际执行合约使用的代码。而操作码是合约的低级指令码,相比字节码具有更强的可读性,有助于对合约的执行情况进行理解和推断,可以用于合约的运行时分析。除此之外,在模糊测试工作 Soliaudit[25] 中,通过分析合约的操作码,可以利用机器学习模型分类的方式来判断合约中包含的漏洞。ABI 是合约调用接口,它定义了外部账户或合约账户与合约之间的交互标准,记录了合约中包含的函数以及函数对应的参数信息,以及一些日志相关的信息。在模糊测试中,通常需要根据 ABI 来确定合约中所有可以被调用的函数以及函数形参的类型,以随机地或者有策略地生成测试输入,将其打包成交后发送到链上,从而完成合约的调用执行。表 3 列出了各项模糊测试工作对智能合约代码的使用情况。

表 2 智能合约字节码、ABI 和操作码

Table 2 Bytecode, ABI and Opcodes of smart contract

bytecode: 6080604052600080546001600160a060020a03199081169091556001805490911690553480156100 2c57600080fd…
ABI: [{"constant":true,"inputs":[],"name":"owner","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"}…
opcodes: PUSH1 0x1F DUP2 ADD DUP5 SWAP1 DIV DUP5 MUL DUP6 ADD DUP5 ADD SWAP1…

表 3 已有模糊测试技术对智能合约信息的使用情况

Table 3 Use of smart contract information by existing fuzzing techniques

模糊测试技术	智能合约源码	字节码	ABI	操作码
ContractFuzzer		√	√	√
Reguard	√	√		
ILF	√	√		√
ContraMaster		√	√	√
SolidAudit	√	√	√	√
Harvey	√	√		
ETHPlOIT	√	√	√	√
sFuzz		√	√	√
Echidna	√	√	√	
SMARTIAN		√	√	√
SmartGift	√			

2.2 智能合约区别于传统程序的特点

相比传统程序,智能合约存在着一些独有的特点。这些

特点给智能合约带来了一些独特的漏洞，同时也给模糊测试的应用带来了挑战。需要在模糊测试流程中适当地调整策略以进行适应。

(1)合约运行的主流程

智能合约的定义类似于其他编程语言中的类，但是执行流程不像其他语言一样在主函数 *main*()中写定，而是在合约部署之后，由一系列外部账户发起的交易调用构造产生的。这些交易调用了智能合约中的各个函数，按照先后顺序，串起来看就像是合约的函数 *main*()。所有被调用的函数共享永久内存中的全局变量。但是合约中使用的外部状态信息(时间戳、区块号等)会因交易执行时区块链的状态不同而不同，并且随着新交易的调用，这个主流程还会增长。模糊测试中逐个使用的测试输入交易，就构成了合约运行的主流程。

(2)异常处理

在传统程序中，当产生异常时，传统程序会崩溃，在异常位置停止，打印出错误信息。而在智能合约中，会回退至执行之前的状态，撤回所有的状态改变，但是合约运行已经消耗的 gas 不会返还。因为没有控制台打印输出程序中变量的状态，所以合约在运行时的内部状态难以直接获取。不当的异常处理会导致智能合约出现漏洞，比如对 *assert*()的滥用可能会导致合约出现断言失败漏洞。

(3)address 数据类型

与传统编程需要不同，Solidity 自带了一种特殊的数据类型 address。开发者在编写智能合约时，需要频繁地涉及以太坊的账户。作为专门面向区块链智能合约编程的语言，除了布尔类型(bool)、无符号整数类型(uint)、字节数组(byte array)、枚举型(enum)、映射(mapping)等基本的变量类型之外，Solidity 定义了其特有的地址类型 address。address 是 20 个字节的无符号整型，用于表示以太坊中的地址，同时 address 对象具有一些特殊属性和方法。在智能合约模糊测试的测试输入生成与变异阶段，需要对 address 数据类型进行特殊的处理。

(4)外部状态信息

Solidity 智能合约在执行时除了合约内部定义的一组全局变量及其属性或方法之外，可以访问获得区块链上的一些数据，主要包括 block，msg 和 tx。其中，block 对象属性包含当前区块的信息，msg 对象包含当前交易或消息调用的相关属性，tx 对象包含当前交易中的部分信息。与全局变量不同，这些外部状态信息不能通过合约的调用执行而改变，而是根据当前的区块链状态、调用交易发起人的设置直接获取并使用。合约本身无法从内部改变这些信息，但同时，它们又是影响合约执行的重要因素。在一些智能合约中，由于使用了 block 的 number(即区块号)属性，因此导致了区块状态依赖漏洞。

(5)对货币的使用

以太坊智能合约常常涉及虚拟货币交易，例如：对调用交易发送的以太币数量进行判定；合约中在满足一定状态时向别的账户进行转账交易等。因此，虚拟货币等单位量方面也是独特定义的。以太币的单位有 wei，szabo，finney 和 ether，wei 是以太币的基本单位，也是最小单位，1 szabo 等于 10^{12} wei，1 finney 等于 10^{15} wei，1 ether 等于 10^{18} wei。在智能合约的函数中，经常出现对发送货币数量的判断。在智能合约模糊测试中，随机的输入生成方式可能导致无法产生准确的货币发送数量，从而导致货币数量判断约束下的代码无法进入，引起覆盖率低的问题，限制模糊测试的漏洞检测能力。

(6)燃油机制

燃油(gas)机制也是智能合约独有的。智能合约的运行需要消耗 gas，在调用交易发送之后，会直接按照 *gasLimit* 扣除 gas，在合约执行结束并且有剩余的情况下，多余的 gas 会退还给交易发送者。当调用交易中规定的 *gasLimit* 不足以支撑合约的执行完成时，合约会回退至执行之前的状态，但是扣除的 gas 不会返还。当 *gasPrice* 超过创世区块规定时，合约不会执行。智能合约的无气发送漏洞就与这个机制有关。

(7)无法进行内容更新和状态重置

传统程序写成的软件如果出现漏洞，可以通过打补丁的方式对已有漏洞进行修补，而智能合约一旦部署，就无法撤回或者通过某种方式对合约内容进行更改，这就意味着如果智能合约中出现漏洞，合约的开发者也无法进行修补。另外，与传统软件不同，智能合约无法简单地通过重新运行调用构造函数进行状态重置。对于已部署的合约，交易调用对合约状态的改变也是不可撤回的，这是因为区块链具有不可篡改的特性，已确认的交易无法撤回。因此，在对智能合约进行模糊测试时，如果想对智能合约的状态进行重置来使用新的测试用例，则只能重新部署合约，获取一个新的合约地址，在新部署的智能合约上进行测试操作[26]。

2.3　智能合约漏洞

若要对以太坊智能合约进行漏洞检测，则首先需要明确智能合约的漏洞类型。不同的学者提出了不同的分类方法，本节对智能合约漏洞进行了介绍和总结。

著名的去中心化应用安全项目 DASP(Decentralized Application Security Project)[40]是代码审计机构 NCC Group[41]倡议的一个开放协作项目，旨在号召安全社区的人们共同参与发现智能合约中的漏洞。DASP 列出了项目参与者公认的十大智能合约漏洞。Atzei 等[42]总结了 12 种可以被攻击者利用并盗取以太币的以太坊智能合约漏洞，并且根据漏洞引入的时机将这些漏洞分成了 Solidity 层漏洞、EVM 层漏洞和区块链层漏洞 3 种，并给出了具体的代码示例。Xia 等[43]通过对 Ethereum Stack Exchange 收集的智能合约以及相应的讨论进行了人工分析和整理，总结了 20 多种智能合约的缺陷，并根据性质对缺陷进行了分类。

如表 4 所列，本文根据现有的智能合约模糊测试的漏洞检测能力，阐述了以下 10 种以太坊的智能合约漏洞，并在此基础上，对现有工作的漏洞检测能力进行了比较。可以看到，模糊测试工作所能检测的这些漏洞主要集中于智能合约层面。对于少量区块链层面的漏洞，对以太坊客户端进行设置之后，也可以在智能合约的动态执行中触发。值得注意的是，以太坊实际存在的漏洞数量远不止 10 种，对其他漏洞的检测也是模糊测试潜在的研究方向。

表 4 以太坊智能合约漏洞

Table 4 Ethereum smart contract vulnerabilities

漏洞类型	漏洞简述	漏洞层面
重入漏洞	攻击者绕过记账状态检查，重复进入转账语句获取以太币	智能合约层面
未处理的异常	未检查低级调用的返回值，没能正确处理异常	
自杀合约	合约自毁导致上层调用无法找到依赖源或以太币，无法取出	
以太币泄露	任意攻击者可以取走以太币	
无气发送	gas 不足导致子调用失败	
资产冻结	合约只能接收以太币，无法发送以太币	
危险的委托调用	外部调用到恶意构造的代码	
整数溢出	运算结果超过数据类型表示的有效空间	
断言失败	错误地使用 assert 导致合约无法正常执行	
区块状态依赖	矿工操纵区块状态，从而改变智能合约的执行结果	区块链层面

(1)重入漏洞(Reentrancy)

重入漏洞是最著名的以太坊智能合约漏洞，它导致了以太坊社区的硬分叉。如果智能合约函数的转账发生在记账金额(Bookkeeping Balance)改动之前，且转账判定条件中包含对记账金额的判断，那么该合约就很有可能存在重入漏洞。该漏洞的攻击原理是，利用智能合约在收到转账时自动调用回调(fallback)函数的特性，可以构造一个攻击合约，使用攻击合约调用被攻击合约的转账函数，在攻击合约的 fallback 函数中再次调用这个包含漏洞的合约转账函数。当攻击合约调用被攻击合约的转账函数并收到以太币后，fallback 函数会自动触发，攻击合约就可以在记账金额改变之前重新进入该函数并再次收到以太币转账，新收到的以太币又会触发 fallback 函数，fallback 函数再取走以太币。这个过程一直循环，直到被攻击合约中的以太币被全部取走。

(2)未处理的异常(Exception Disorder/Mishandled Exception)

智能合约的低级调用方法 call()，send()，delegatecall()，callcode()中出现异常时，在回撤这些低级调用之下的执行结果后，并不能向上传递异常从而将整个交易调用的执行结果回撤，而是仅仅在这些低级调用处返回一个 false。如果这个低级调用的返回值没有被合约中的语句正确接收并处理，则会引起该漏洞。该漏洞会导致合约出现参与者无法察觉的异常，从而进一步造成合约的记账状态与实际不符的情况出现。

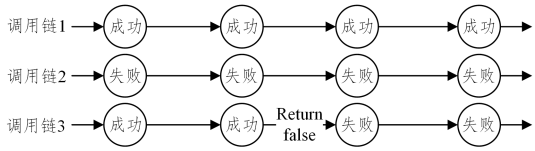

图 2 交易调用的函数调用链

Fig. 2 Call chains of function invoke transactions

图 2 中，如调用链 1 所示，当调用交易正常执行时，整条调用链都会执行成功。如调用链 2 所示，当调用交易在函数调用链的深层出现异常时，整条交易的执行结果都会沿着函数调用链从深层向外层回撤，直到整条链的执行结果被全部回撤。如调用链 3 所示，当调用交易在函数的调用链深层出现异常时，假设调用链上的第三个节点就是引起未处理的异常漏洞低级调用，那么交易调用的执行结果不能随着异常的向上传递沿着函数调用链全部回撤，在第三个节点向上反馈时只会抛出一个 false。如果没有正确处理，那么上层的执行并不会回撤，最后表现为调用交易执行成功。

(3)自杀合约(Suicidal Contract)

合约的开发者在编写合约时可以使用合约的自毁函数(suicide 或者 self-destruct)，设置合约在适当的情况下自毁或者被合约的拥有者调用自毁。当合约开发者设置了错误合约的自毁函数调用条件或者对合约自毁函数的调用权限设置有误时，攻击者可以通过恶意的交易调用销毁合约。合约被销毁之后，不仅合约本身无法再对用户提供服务，其他合约对该合约的调用也会出现异常。

(4)以太币泄露(Leaking/Ether Leak)

当任何帐户通过任意的地址都可以从一个智能合约中盗取以太币时，就称这个合约出现了以太币泄露的漏洞。以太币泄露漏洞会导致合约管理的以太币流向合约参与者之外的用户。对于通过该智能合约管理以太币的其他用户来说，这个漏洞的危害巨大，可能会导致他们的数字资产蒙受损失。

(5)无气发送(Gasless Send)

无气发送漏洞是一种特殊类型的未处理异常漏洞。该漏洞出现的原因是合约使用底层调用 *send*() 发送以太币时，接收合约的 fallback 函数会被调用。但是由于 EVM 为 *send*() 指定了固定的 gas 值(通常是 2 300)，如果接收合约的 fallback 函数内容过于复杂，需要的 gas 超过这个固定的 gas 值，那么就会出现 gas 耗尽异常。这种情况下，*send*() 调用下的合约执行会被回撤。然而，异常在向上层传递时只会在 *send*() 的调用处返回一个 false。如果这个 false 没有被正确地接收和处理，*send*() 的以太币发送会失败，但是由于异常没有被上层捕捉到，记账金额会正常扣除，这样合约就会错误地保留了以太币。

(6)资产冻结(Freezing Ether/Locking)

包含资产冻结漏洞的智能合约一般具有这样的特点：合约本身可以接收以太币，但是合约中没有定义将合约保管的以太币转出的函数。这种合约有时通过 *delegatecall*() 将其他合约的相关代码动态加载到当前的执行环境中执行以完成转账，或者直接没有任何转出以太币的操作。如果是前者，当 *delegatecall*() 依赖的合约调用了自毁函数，那么包含该漏洞的合约便再也无法将自身管理的以太币转出。如果是后者，那么该合约就只能接收以太币，无法转出以太币，任何转入该合约的以太币都会被锁定。当漏洞出现时，对于通过该合约管理以太币的用户来说，存放在合约中的资产就被冻结了。

(7) 危险的委托调用(Dangerous Delegatecall)

Solidity 中提供了操作码 Call 和 DelegateCall 来进行外部代码的调用，以提升代码的重用性。Call 发起的调用会在被调用合约的上下文环境中执行完成，而 DelegateCall 会将被调用合约的字节码嵌入到调用合约的字节码中。DelegateCall 的使用意味着合约可以在运行时从不同的地址动态加载代码并在当前的上下文中运行，可以通过动态加载的代码操作当前合约的永久存储(Storage)。因此，恶意攻击者可以设计精心的攻击代码，调用 DelegateCall 目标合约中的函数直接修改调用合约的全局状态。通过修改全局状态，攻击者

可以成为合约的拥有者或者参与方,从而参与以太币的分配。

(8)整数溢出(Integer Over-flow/Integer Under-flow)

整数溢出是一种常见的漏洞类型。计算机程序设计语言一般都有这种漏洞,主要原因是编程语言对特定类型的数据的存储空间长度有限制,一旦数据经过运算得到的结果超出了特定类型保存数据大小的范围,就会发生整数溢出漏洞。但是在智能合约中,这种漏洞的类型尤其危险。如果用户没有对计算的结果进行检查并设置异常处理,则很容易发生这种漏洞,并且在 Solidity 语言中没有可以表示负数的数据类型。通常使用的数据类型为无符号整型,无符号整型数值下溢会变成一个接近该数据类型表示上限的无符号整型值。如果该漏洞发生在钱包合约中,用户就可以在调用取款时设置一个比记账余额大的取款值,使记账余额值下溢变成一个更大的值,然后继续取款,盗取合约管理的以太币。

(9)断言失败(Assert Failure)

在 Solidity 中,*assert*()语句用于断言不变量,正常运行的代码永远不应到达失败的断言语句。当断言的条件没有得到满足时,会触发断言失败漏洞。出现该漏洞的原因有多种,如合约中存在允许其进入无效状态的错误、对测试输入进行了断言检查等。该漏洞会导致合约在输入无误时无法正常执行。

(10)区块状态依赖(Block State Dependency)

区块状态依赖漏洞又可以细分为时间戳依赖漏洞(Timestamp Dependency)和区块号依赖(Block Number Dependency)漏洞。当智能合约中利用时间戳和区块号这两种状态信息作为一些关键操作(如以太币转账)的判定条件的一部分时,就可能出现区块状态依赖漏洞。漏洞产生的原因是,这两种状态信息都可以被以太坊矿工控制,矿工可以在一个较短的时间间隔内随意设置时间戳,同时区块号在一定范围内也是可以控制的。如果有一定的利益激励,矿工可以通过操纵这两种信息来使合约的调用交易产生对自己有利的执行结果,从而谋取不正当的利益。

已有智能合约模糊测试工作的漏洞检测能力比较结果如表 5 所列。可以看出,针对前文总结的漏洞,sFuzz[28] 和 Smartian[29] 这两项技术的漏洞检测能力相对比较出色,Smartian 更是能够覆盖本文阐述的所有漏洞,ETHPLOIT[27],ReGuard[23] 和 Echidna[21] 能够检测的漏洞种类相对较少。其原因是,ETHPLOIT 定义的测试预言较少,ReGuard 是专门针对重入漏洞的,因此从设计上来说,ReGuard 对于检测其他漏洞并不是很有效。而 Echidna 是一个框架,其漏洞检测能力很大程度上依赖于用户的个性化定义。SmartGift[30] 并不是一个完整的模糊测试工作,它解决的是智能合约模糊测试的测试输入生成问题,但是没有完成智能合约模糊测试的其他步骤,因此该工作不具备独立的漏洞检测能力。

表 5　已有智能合约模糊测试漏洞检测技术覆盖的漏洞比较

Table 5　Comparison of vulnerabilities covered by existing smart contract fuzzing techniques

研究工作	漏洞类型									
	重入漏洞	未处理的异常	自杀合约	以太币泄露	无气发送	资产冻结	危险的委托调用	整数溢出	断言失败	区块状态依赖
ContractFuzzer	√	√			√	√	√			√
Reguard	√									
ILF		√	√	√		√	√			√
ContraMaster	√	√			√			√		
SolidAudit	√	√	√					√	√	
Harvey	√	√						√	√	
ETHPLOIT				√						√
sFuzz	√	√			√	√	√	√		√
Echidna									√	
Smartian	√	√	√	√	√	√	√	√	√	√
SmartGift	—	—	—	—	—	—	—	—	—	—

3　测试输入生成

测试输入的质量决定了模糊测试对智能合约的代码覆盖率的高低。已有智能合约模糊测试工作对智能合约测试输入生成的处理方式有两类,一类是只考虑单个合约函数调用的单交易(Single Transaction)生成,另一类是考虑连续合约函数调用之间相互影响的交易序列(Transaction Sequence)生成。其中单交易的形式化定义前文已经给出,为一个六元组。交易序列可以形式化地定义为一个序列 TS:

$TS=(T_1,T_2,T_3,\cdots)$

序列中的每一个项 T_n 都是一个完整的交易。在使用交易序列 TS 对待测合约进行测试时,需要严格按照序列中各个元素的顺序逐个发送序列中的交易。序列的长度不固定,一般都是由模糊测试研究工作自行定义。

单交易测试输入的生成方式是直接针对合约中的函数,利用 ABI 中的函数规格信息,直接产生符合函数参数信息的测试输入。然后,通过一定的方式生成发送者地址 *From*、gas 上限 *gasLimit*、gas 价格 *gasPrice* 和交易发送的以太币数量 *Value* 等信息。将这些信息组合,形成一个完整的交易。

Jiang 等[17] 提出了学术界智能合约模糊测试的首个工作 ContractFuzzer。该工作针对待测智能合约使用单交易生成的方式产生测试输入。根据合约的 ABI 对合约中函数的描述,为每个函数生成待选输入集。针对一个函数中的多个参数,为每个参数生成 k 个候选值,将所有参数的 k 个候选值进行组合,产生函数的测试输入集合。由于参数的数据类型分为固定长度(如 uint256 类型)和不固定长度(如 string 类型)两种情况,因此 Jiang 等对此采取了不同的策略。对于固定长度的参数类型,通过在有效值域中随机生成的方式产生一个候选集合,另一方面利用合约中常用的数据生成另一个候选集合,将两个集合合并,就产生了固定长度参数的候选

集合。对于不定长度的参数类型，首先为参数随机生成一个长度值，然后通过在有效值域中随机选取值的方式产生候选集合。这种单交易测试输入方式以随机为主，使用了部分合约中定义的数值，这种测试输入生成方法比较平实，存在一定的改进空间。

Zhou 等[30]在 Jiang 等的工作的基础上，使用自然语言处理的方法对智能合约的实用测试输入生成方法进行了研究，提出了 SmartGift。其中，实用测试输入指很有可能在漏洞检测过程中达到更高覆盖率的测试输入。SmartGift 基于的假设是待测智能合约中的函数可以被已有数据集中相似函数的测试输入所验证。SmartGift 将待测合约中的某个具体函数的函数名称和参数类型名称合并生成符号序列，然后符号化生成表示向量。通过与已有输入数据集中的函数及其参数进行余弦相似度计算，得到一个按照相似度排序的函数列表，在此基础上进一步匹配待测合约函数和已有测试输入的函数，最终产生待测合约函数的可用测试输入。该研究通过替换 ContractFuzzer 中的测试输入生成方法发现，其漏洞检测能力得到了提升，验证了自身的有效性。这种测试输入生成方式类似于测试输入推荐，其生成的测试用例也仅仅针对单个合约函数。打包之后生成的交易属于单次函数调用交易，对于内部逻辑相似度较大的合约函数而言，这种方式可以取得一定的效果。

交易序列测试输入的生成是以函数调用交易序列为测试输入的生成目标。由于智能合约中的函数共享永久存储下的全局变量，某些函数的执行路径会受到全局变量的控制约束，而全局变量的改变又依赖于其他函数的执行。不考虑函数之间执行顺序的单交易测试输入往往难以进入这些特殊的路径。考虑函数对全局变量的读写关系，使用一定策略来生成合约的调用交易序列，可以提升模糊测试对合约路径的覆盖率，增加触发漏洞的可能性。

He 等[24]认为，智能合约通常含有在多个状态约束下才能进入的深层状态，必须要产生特定的交易序列才能进入合约的深层状态。他们在工作 ILF 中提出了使用模仿学习的方式，学习符号执行专家针对已有合约产生的优秀交易序列，将待测合约的交易序列生成的过程抽象为马尔可夫决策过程。通过全连接神经网络和循环神经网络学习出一个概率策略来将交易序列生成决策的累计奖励最大化。通过对构成交易的函数名称、函数参数、交易发送者地址和交易发送金额进行采样来产生新的交易，持续延长交易序列。这种测试输入生成方法基于统计推理，与训练集相似度较大的合约有可能进入控制约束下的代码块。其优点在于，不用专门针对合约花费大量的时间进行约束求解，测试输入的产生速度快。

Liao 等[25]认为，函数调用序列对于触发智能合约漏洞十分重要。他们在工作 SoliAudit 中，使用两种方式来产生函数调用序列，一种是随机的函数调用序列，另一种是用户定义的函数调用序列。随机的函数调用序列有利于触发未知漏洞，而用户定义的函数调用序列在针对特定类型漏洞时一般会有更好的表现。对于每个函数调用交易，根据 ABI 中定义的参数类型生成随机值和极值，将交易中的 *gasPrice* 和 *gasLimit* 也作为变量数据进行生成。此外，在测试输入生成时设置了 3 种账户角色地址，分别是账户创建者地址、合约地址和其他账户地址。在生成交易时，随机使用这 3 种地址作为交易的发送地址。该方法产生的测试输入仍以随机为主，用户经验的加入可以提升测试输入的有效性，将发送地址分类并随机使用可以增加测试输入的多样性。

Wüstholz 等[19]认为，交易序列的可能组合随着序列长度的增加呈指数级增长，因此采用随机的方式生成交易序列并不可取。他们在工作 Harvey 中，使用了需求驱动的交易序列生成方法。该方法的核心思想是，交易序列的其他交易的作用就是改变全局变量来为测试最后一个交易调用的函数做准备。序列生成的目的就是为了探索最后一个交易调用的函数的新路径，从而发现更多的漏洞。通过在交易序列生成中增加一种激进(Aggressive)模式来直接模糊全局变量，判断其是否能够增加覆盖率。Wüstholz 等为激进模式的触发设置了一定的概率，以保证在大部分情况下使用的仍然是常规的交易序列生成方法。如果激进模式增加了最后一个交易调用的函数的覆盖率，那么对应的序列会用于生成更长的交易序列，有利于进一步提升覆盖率。

Zhang 等[27]将智能合约模糊测试用例定义为一个交易序列，序列中的每个交易按照顺序执行。他们在工作 ETHPLOIT 中，将交易序列的生成划分为 3 个步骤，分别是函数选择、参数生成和区块链属性生成。在函数选择中，通过分析函数的调用权限、对全局变量的使用情况以及污点关系来选择函数。在函数参数生成中，结合使用伪随机方法和基于反馈的备选种子集。在区块属性生成中，从预定义的账户中选择交易的发送地址，其他属性的生成仍使用随机的方法。该工作的测试输入生成方式思路比较清晰，从函数到交易进行逐层分析，特点是使用了污点关系来筛选函数。参数生成过程中结合使用合约执行的反馈来提升新输入的有效性。与 SoliAudit 工作对交易发送地址进行分类不同，该工作直接定义出可能使用到的账户预定义，生成的交易发送地址可能更为精准。

Nguyen 等[28]提出了使用遗传算法来根据分支覆盖率引导优化测试套件的工具 sFuzz。他们将交易定义为带有固定实参的函数调用，通过对待测合约中的每个函数单独生成一个交易序列，来保证合约的函数覆盖率为 100%。每个序列中的合约部署交易后的首个交易调用的函数都是待测合约中不同的函数。其次，对于序列中的每个交易调用的函数参数，使用随机方式生成。该方法生成的测试输入可以保证每个函数都被覆盖到，后续会在此基础上对测试输入进行进一步的优化。

Choi 等[29]认为，智能合约与传统程序的区别在于，智能合约中的函数共享永久存储的全局变量，需要考虑不同函数对全局变量的读写关系，以生成交易序列。他们在工作 SMARTIAN 中，以交易序列为函数测试输入的生成目标。为了生成有效的种子交易序列，首先需要导出有效的函数调用顺序。该工作通过迭代的方式将不同的函数加入已有序列，然后通过检查是否增加了新的数据流的方式来判断序列的有效性。通过这种方式，尽可能地找出所有有效的交易序列。在函数序列的基础上，检查每个函数中是否有对调用者

的判断，据此指定交易的发送者。这种交易序列的生成方式具有很强的目的性，可以将改变全局变量的函数放在使用全局变量的函数之前优先调用，在这些种子交易序列基础之上的变异输入就可以大大增加进入全局变量控制约束的代码块的可能性。同时，针对调用者判断的检查和处理可以使测试输入轻易地进入用户检查控制约束下的代码块内。该方法产生的交易序列可以达到比随机产生的交易序列更高的覆盖率。

4　测试输入变异

测试输入变异是模糊测试中根据已生成的测试输入或者原始种子输入产生新的测试用例的步骤，是产生大量测试输入的关键。智能合约模糊测试输入的变异可以划分为 3 个层面：函数层面的变异、交易层面的变异和交易序列层面的变异。函数层面的变异指，对交易调用的合约函数中的输入参数进行变异，更改测试输入中的函数参数值。交易层面的变异指，对函数调用交易中的交易信息进行变异，更改函数调用交易中的交易发送者地址、交易金额、gas 价格和 gas 上限信息。交易序列层面的变异指，根据一定的策略对调用交易序列进行延长、缩减或顺序调换。这 3 个层面的变异可以单独使用，产生新的测试输入，也可以根据一定的线索结合使用。表 6 列出了智能合约模糊测试技术的变异层面比较情况。

表 6　部分智能合约模糊测试技术变异层面比较

Table 6　Comparison of mutation level of some smart contract fuzzing techniques

研究工作	变异层面		
	函数层面	交易层面	交易序列层面
ContractFuzzer[17]	√		
ContractMaster[26]	√	√	√
Harvey[19]	√		√
sFuzz[28]	√		√
SMARTIAN[29]	√	√	√

对于测试输入的函数层面变异，文献[17]的做法是，围绕有效边界对函数的参数做一些变化。与之不同，文献[26]使用了与 AFL[44] 中位翻转（Bit Flip）操作类似的位否定（Bit Negation）操作来对输入参数进行改变，以产生新函数输入，对于特殊数据类型 Address，使用了枚举的方式。针对合约函数，为了进入控制语句约束的新路径，文献[19]使用了插桩的方式来计算当前执行和最优执行之间的距离，通过将距离最小化的方式使函数测试输入到新的执行路径。具体来讲，每次仅仅是变异函数输入中的一个参数，通过变异前后两个输入在同一方向上的距离差异预测，就能够将该距离最小化的测试输入。文献[28]使用 6 种与 AFL 中类似的变异操作符对函数参数进行变异，同时根据参数类型是否是固定长度进行了不同的区分。为了使变异的测试输入进入条件分支，文献[29]使用了灰盒螺旋测试技术[45]（Grey-box Concolic Testing），在解决约束进入的同时，避免了复杂求解和插桩带来的额外开支。

对于测试输入的交易层面变异，文献[26]专门针对交易中的 *gasLimit* 进行了变异，针对交易进行最大 gas 消耗估计和固有 gas 消耗估计，将两者构成的区间进行 n 等分，然后从每个区间中随机选取 *gasLimit* 值来构成交易。文献[29]使用经典变异操作符（如位翻转）对交易中的参数（交易金额、交易发送者地址）进行变异。

对于交易序列层面的变异，文献[26]使用了数据流、控制流和动态合约状态来引导交易序列的变异，并且提出了 4 种变异操作，分别是操作两个相同状态变量的交易的顺序、随机替换某个交易、随机选择一个交易删除和随机在一个交易之前插入新交易。文献[19]提出了 3 种变异操作，分别是模糊某个交易所调函数的参数、在某个交易之前插入一个新交易和用一个交易序列替换某个交易之前的所有交易。文献[28]提出了 3 种操作符来变异交易序列，3 种操作符针对交易序列中交易的函数调用，分别做函数调用修剪、函数调用增加和函数调用交换 3 种操作。文献[29]定义了 3 种操作符，分别对交易序列做插入随机交易、移除随机交易和交换两个随机交易 3 种操作。

5　测试预言

在智能合约模糊测试工作中，测试预言（Test Oracle）[46] 专门用于对智能合约进行漏洞判定。测试预言的质量决定了模糊测试中的漏洞检测能力，包括检测漏洞的种类和检测的准确性。一般来讲，在模糊测试技术中，对于每种漏洞都要单独设置一个测试预言。智能合约的每一次调用执行都要检查测试预言是否被触发，若触发则说明出现了对应的漏洞，相应的测试用例可以被保留，便于以后进行复现和分析。表 7 列出了部分智能合约模糊测试技术的测试语言定义情况。

表 7　部分智能合约模糊测试技术的测试预言

Table 7　Test oracles defined by some smart contract fuzzing techniques

研究工作	测试预言
ContractFuzzer	无气发送预言、未处理异常预言、重入预言、时间戳依赖预言、区块号依赖预言、危险的委托调用预言、资产冻结预言
Harvey	重入预言、整数溢出预言
ContractMaster	余额不变量预言、交易不变量预言
ETHPLOIT	金额增加预言、自毁预言、代码注入预言
sFuzz	无气发送预言、未处理的异常预言、时间戳依赖预言、区块号依赖预言、危险的委托调用预言、重入预言、整数溢出预言、以太币冻结预言
SMARTIAN	断言预言、任意写入预言、区块状态依赖预言、控制流劫持预言、以太币泄露预言、资产冻结预言、整数溢出预言、未处理的异常预言、多次发送预言、重入预言、要求违反预言、自杀合约预言、交易源使用预言

ContractFuzzer[17] 的主要贡献之一是提出了多个用于运行时漏洞检测的测试预言。针对无气发送漏洞、未处理异常漏洞、重入漏洞、时间戳依赖漏洞、区块号依赖漏洞、危险的委托调用和资产冻结漏洞，逐一定义了测试预言。每个测试预言定义了函数在执行时出现的行为对应触发的漏洞。在测试预言的实际使用中，ContractFuzzer 需要收集 3 种信息，分别是合约中 call 和 delegatecall 的属性信息、合约执行中调用的操作码信息和合约执行中的状态信息。由于智能合约的执行过程的相关信息不能如传统程序一样直接在控制台打印输出，ContractFuzzer 通过对 EVM 进行插桩，将测试预言判定所需的结果通过端口发送到服务端，由服务端进行最终的漏洞判定。从实验结果来看，测试预言的效果很好，但是测试

预言对于时间戳依赖漏洞和区块号依赖漏洞的检测不能做到百分百的准确，有可能出现误判。

Harvey[19]中定义了重入漏洞和整数溢出漏洞等多种漏洞检测的测试预言，也支持自定义的测试预言，但是对于测试预言的具体描述和实现细节并没有给出过多描述。

与其他工作不同，ContraMaster[26]中未针对每种需检测的漏洞分别使用独特的测试用例，而是设计了一种通用的测试预言，在语义层面检测智能合约中出现的异常。这个测试预言实现了两种合约交易执行中必须遵守的不变量，即余额不变量和交易不变量。ContraMaster 的研究者 Wang 等认为，智能合约主要用于进行资产的转移工作和合约参与者之间资产转移的记账(bookkeeping)工作。为了完成记账工作，合约往往会使用记账变量来记录参与者的余额。余额不变量表明，在一个交易前后，合约的真实余额和所有交易者总记账余额之间的差值保持不变。交易不变量表明，在合约的记账余额中扣除的金额始终会转入收款人的余额中。如果合约的执行结果违背了这两个不变量，则说明产生了异常，执行过程中很有可能触发了漏洞。测试预言中包含了对这两个不变量成立与否的判别，用于交易执行前后对合约的状态检查。Wang 等还对重入漏洞、无气发送漏洞和整数溢出漏洞进行了详细的分析，说明了当这些漏洞发生时，都会违反不变量，从而被他们提出的测试预言检测到。对于资产管理的合约，ContraMaster 提出的测试预言不仅可以一次性检测多种漏洞，还发现了 3 种新的漏洞攻击方式。可以看出，该测试预言针对带有内部记账的资产管理合约的漏洞检测效果显著。

ETHPLOIT[27]使用了 3 种测试预言来检测智能合约执行过程中的漏洞利用情况，分别是金额增加预言、自毁预言和代码注入预言。ETHPLOIT 的研究者 Zhang 等在对 EVM 插桩时设置了一组攻击者账户，金额增加预言检测合约在一次执行过后攻击者的账户余额是否有增加，如果增加，则说明出现了攻击者利用漏洞盗取数字资产的情况。自毁预言检查智能合约在一个测试用例的执行中是否使用了 SELFDESTRUCT 操作码，该操作码的使用意味着合约出现了自毁的情况。代码注入预言检查在合约一次执行中，是否使用 callcode 和 delegatecall 这两种危险的低级调用对攻击者合约进行了调取和使用，以判断合约是否存在代码注入的风险。通过使用这 3 种测试预言，ETHPLOIT 有能力检测未检查的转账金额、脆弱的权限控制和暴露的秘密 3 种漏洞。其中，未检查的转账金额漏洞和脆弱的权限控制这两种漏洞产生的结果类似于前文总结的重入漏洞、资产冻结漏洞、以太币泄露和合约自毁等，是在不同角度对智能合约漏洞的总结。而暴露的秘密是一种新发现的漏洞，对它的检测是 ETHPLOIT 拥有的区别于其他工作的一项能力。

sFuzz[28]参考了包括 ContractFuzzer 在内的其他工作中定义的 8 种测试预言，用于检测无气发送、未处理的异常、时间戳依赖、区块号依赖、危险的委托调用、重入、整数溢出(上溢和下溢)和以太币冻结漏洞。通过使用 EVM 支持的 hook 机制来监测测试用例的执行过程，使用这些测试预言来判断漏洞的发生。在漏洞的检测实践中，有可能会出现误报。

SMARTIAN[29]支持对断言失败、重入、资产冻结等 13 种漏洞的检测。针对其中的 11 种漏洞，SMARTIAN 有针对性地定义了测试预言；针对资产冻结漏洞和任意写入漏洞，分别使用了 ContractFuzzer 和 Harvey 中定义的测试预言。SMARTIAN 同样是对 EVM 进行了插桩处理，从而在测试用例执行过程中检测漏洞，在测试预言的使用中还结合污点分析方法来提升漏洞检测的效果。进一步地，针对 sFuzz 中测试预言对整数溢出的检测只关注数值的加法和减法操作进行了改进，SMARTIAN 在自身的测试预言中支持对数值乘法可能造成的溢出进行检查，提升了整数溢出漏洞检测的准确性。总的来看，SMARTIAN 中对测试预言的定义和使用相比其他工作更加全面和完善，这也使得 SMARTIAN 的漏洞检测能力更加强大。

6 评价指标

在智能合约模糊测试中，不同的工作使用了相似的方法来衡量模糊测试技术能力。本文从覆盖率、漏洞检测混淆矩阵指标和检测时间 3 个方面对智能合约模糊测试能力的评价方法进行了总结。表 8 列出了已有智能合约模糊测试工作采用的评价指标。

表 8 已有智能合约模糊测试工作采用的评价指标

Table 8 Evaluation indicators adopted by ixisting smart contract fuzzing techniques

研究工作	评价方法
ContractFuzzer	真阳性率(TP Rate)、误报数(FP)、漏报数(FN)
Reguard	误报数(FP)、漏报数(FN)
ILF	指令覆盖率(Instruction Coverage)、真实漏洞率(Percentage of True Vulnerability)
ContraMaster	时间消耗(Time Taken)
SolidAudit	准确性(Accuracy)、精确性(Precision)、召回率(Recall)、F1 得分(F1 Score)
Harvey	指令覆盖(Instruction Coverage)、缺陷触发时间(Time to Bug)
ETHPlOIT	生成的攻击次数(Count of Generated Exploits)
sFuzz	覆盖率(Coverage)、真阳性率(TP Rate)
Echidna	覆盖率(Coverage)
SMARTIAN	漏洞检测数(Bugs found)、指令覆盖(Instruction Coverage)、真阳性数(TP)，误报数(FP)
SmartGift	成功执行率(Successful Execution Rate)、漏洞检测数(Vulnerability Detected)

覆盖率[47]是一种传统的软件测试度量方法，用于对测试的有效性进行度量，更高的覆盖率意味着更高的测试充分性。在智能合约模糊测试中，覆盖率常常用来衡量模糊测试生成输入的有效性。ILF[24]从对待测合约(包括小型合约和大型合约)的指令覆盖和基本块覆盖两个方面，分别与其他模糊测试工具和符号执行工具进行了对比，发现在合理的检测时间内，都可以达到更高的覆盖率，验证了其生成的交易序列的有效性。在 Harvey[19]中，通过使用指令覆盖数量，验证了基于距离度量的输入预测在测试输入生成中的有效性。同时还发现，随着时间的增加，数据驱动的交易序列生成方法可以提高模糊测试对待测合约的指令覆盖能力。在 sFuzz[28]中，使用了分支覆盖来衡量其为待测智能合约生成的测试套件的效果。在智能合约模糊测试框架工作 Echidna[21]中，为了对比不同配置下该框架生成测试输入的覆盖率，分别设置了不同的运行时间和不同的交易序列长度。实验结果表明，延长运行

时间和增加交易序列长度都可以提升该框架对待测合约代码的覆盖率。

表 9 列出了智能合约漏洞检测的混淆矩阵，其中根据合约漏洞的真实存在情况和合约的漏洞检测结果，产生了真阳性(TP)、漏报(FN)、误报(FP)和真阴性(TN)4 个指标元素。已有的智能合约模糊工作使用了该矩阵中的部分元素来作为模糊测试技术漏洞检测准确性的评价指标。ContractFuzzer[17]使用真阳性率来评价其漏洞检测准确性，实验结果表明，除了时间戳依赖漏洞和区块号依赖漏洞，其他漏洞检测的真阳性率都为 100%。并且将其对时间戳依赖漏洞和重入漏洞检测的误报数和漏报数与 Oyente[48]进行了对比，并分析了误报和漏报的原因。Reguard[23]同样使用了误报数和漏报数，将其与 Oyente 对重入漏洞的检测准确性进行了对比。SoliAudit[25]使用了准确率、精确率、召回率和 F1 得分来对不同模型对合约数据集的分析结果进行了比较。sFuzz[28]工作中，针对 sFuzz 检测能力范围内的 9 种漏洞，使用真阳性率指标与 ContractFuzzer 和 Oyente 进行了漏洞检测准确性的比较。实验结果表明，在绝大多数漏洞上，sFuzz 都获得了更高的漏洞检测效果。SMARTIAN[29]中，针对智能合约数据集的漏洞检测，使用了真阳性数与误报数指标，与 ILF，sFuzz，Manticore[49]和 Mythril[50]工作进行了比较。

表 9　漏洞检测混淆矩阵

Table 9　Confusion matrix of vulnerability detection

		合约检测到漏洞	
		是	否
合约存在漏洞	是	True Positive (TP)	False Negative (FN)
	否	False Positive (FP)	True Negative (TN)

时间指标也是模糊测试工具漏洞检测能力的一项重要指标。在同样或者类似的漏洞检测准确性下，时间指标更强意味着工具的漏洞检测效率更高。ContraMaster[26]使用漏洞检测时间与 ContraAFL(只使用控制流的 ContraMaster 变体)进行了对比。实验结果表明，ContraMaster 的时间性能优于 ContraAFL，并且这种优势随着时间限制的放宽而逐渐明显。Harvey[19]使用了缺陷到达时间这一指标来验证技术中输入预测步骤的必要性，实践表明该步骤可以降低其测试输入达到漏洞的时间，提升了漏洞检测效率。

7　问题总结和研究展望

7.1　总结

目前，智能合约模糊测试的研究十分火热，但仍存在以下两个问题。

(1) 智能合约覆盖率不够高

该问题产生的原因有两个方面。一方面，已有模糊测试技术具有智能合约中对简单约束的求解能力，但是缺乏对复杂约束的突破，其中关注到合约路径中的条件可能会影响合约测试的覆盖率，进一步影响漏洞检测效率的工作有 Harvey[19]，ILF[24]和 ETHPLOIT[27]。Harvey 通过在合约中插桩，然后通过距离计算来对输入进行预测，对于线性的条件约束具有较好的效果，但是在复杂约束条件下，难以产生有效的输入。ILF 通过强化学习的方法来学习符号执行对已有智能合约产生良好交易序列的决策知识，用于待测智能合约的交易序列生成，虽然能一定程度上进入简单约束，但是当待测合约中出现不可预期的复杂约束时，基于统计推理的强化学习方法同样束手无策。ETHPLOIT 通过使用污点分析辅助的动态种子策略来生成能通过强约束的种子，为复杂约束的进入提供了一些解决思路。

另一方面，大多数工作仍是采用以随机为主的方法来生成交易序列，这种方法难以产生启发式的交易序列种子。少部分工作对种子交易序列的生成进行了改进，Harvey[19]中每条种子交易序列只针对一个函数，并且把针对的函数固定放在交易序列的最后一个交易中，尝试用其他的交易对全局状态进行改变来进入最后一个函数，从而得到更高的覆盖率。SMARTIAN[29]通过观察数据流增加的引导方式来保留可以相互影响的交易，有利于产生高覆盖的交易序列。目前，大多工作对种子交易序列生成的关注度不够高。事实上，种子交易序列的生成十分重要，相比变异策略的改进，良好的交易序列对覆盖率的提升更为明显。变异只是对测试输入的一种扩展和补充方法，可以在一定范围内提升覆盖率，但是并不能起到决定性作用。

(2) 测试预言的效果有待提升

从前文的总结比较可以看出，当前智能合约模糊测试工作的漏洞检测结果仍存在误报和漏报，这种情况是由测试预言的定义不够准确而引起的。当测试预言的定义过于粗糙时，可能使合约执行时的漏洞判定范围过大，正常的合约执行状态也被囊括在测试预言的判定集合中，从而产生误报。在研究者定义测试预言之前，对漏洞出现时合约的执行状态梳理不够全面，导致一些漏洞出现的情形没有被测试预言覆盖，可能造成合约执行的异常状态不能被测试预言顺利地捕捉，从而造成漏洞的漏报。

7.2　研究展望

结合对已有智能合约漏洞检测工作的分析和总结，对智能合约的漏洞检测工作还可以从以下角度和方向开展。

(1)根据合约内容预先生成优秀交易序列

现有的研究缺乏好的策略来产生质量高的种子交易序列。在质量差的种子交易序列的基础上进行变异，可能会花费大量的时间但无法取得很好的效果。可以尝试在对智能合约进行静态分析的基础上产生满足全局变量之间约束的优秀交易序列，使智能合约的模糊测试有一个较高的起点，使得在更短的时间内有更高的覆盖率，从而达到更好的漏洞检测效果。

(2) 测试预言改进和扩展

目前的工作中定义的测试预言还存在不够精准的问题，存在漏洞的误报。通过对智能合约漏洞触发时的表现或者内部状态进行更加细致的分析和梳理，可以定义出更加准确的测试预言，从而提升模糊测试对智能合约漏洞检测的准确性。另一方面，在智能合约的发展中，新的漏洞又不断涌现出来，因此可以有针对性地对新漏洞进行分析，定义出针对单个漏洞的测试预言，或者总结一类漏洞触发时的共同表现，定义出覆盖多个漏洞类型的通用测试预言(类似于 ContraMaster[26]

中的测试预言工作),从而增强智能合约的安全性。

(3)从合约类型的角度来检测漏洞

Tolmach 等[51]将智能合约分类为金融(Financial)合约、公证(Notary)合约、游戏(Game)合约、钱包(Wallet)合约和库(Library)合约 5 类。不同类型智能合约之间差异巨大,但是同类型合约之间基本功能十分相似,比如钱包合约一般都包含全局变量钱包拥有者 Owner、存款函数 deposit()和取款函数 withdraw()。金融合约一般都会涉及代币(Token)的创建(Token Initialization)、代币的焚毁(Token Burn)、代币的转账(Token Transfer)。在同类型的合约中,触发漏洞的环节和触发漏洞的类型可以十分相似。针对这一点,设计对相同类型智能合约的模糊测试,有望取得良好的效果,解决一类合约的漏洞检测问题。

结束语 以太坊对智能合约的实现,促进了区块链技术的进一步发展。智能合约的模糊测试漏洞检测技术,也成为了一个重要的研究方向。本文以模糊测试的实施开展为出发点,对以太坊智能合约进行了介绍;总结了智能合约区别于传统程序的特点;介绍了重入、未处理的异常等 10 个漏洞,并且将现有工作对漏洞的检测能力进行了比较;从测试输入生成、测试输入变异、测试预言和评价指标 4 个方面对现有工作进行了分析;总结了现有工作,并对其进行了比较,提出了当前工作面临的对待测合约的覆盖率不够高和测试预言效果有待提升两个问题;对智能合约模糊测试未来的方向进行了展望,提出了根据合约内容预先生成优秀交易序列、测试预言改进与扩展和从合约类型的角度来检测漏洞 3 个潜在的研究方向。

参 考 文 献

[1] NICK S. The Idea of Smart Contract[EB/OL]. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html.

[2] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system [EB/OL]. https://bitcoin.org/bitcoin.pdf.

[3] SUNYAEV A. Distributed ledger technology [M]// Internet Computing. Cham: Springer, 2020: 265-299.

[4] TAPSCOTT A, TAPSCOTT D. How blockchain is changing finance[J]. Harvard Business Review, 2017, 1(9): 2-5.

[5] MIN T, WANG H, GUO Y, et al. Blockchain games: A survey [C]// 2019 IEEE Conference on Games(CoG). IEEE, 2019: 1-8.

[6] REYNA A, MARTÍN C, CHEN J, et al. On blockchain and its integration with IoT. Challenges and opportunities[J]. Future Generation Computer Systems, 2018, 88: 173-190.

[7] RAIKWAR M, MAZUMDAR S, RUJ S, et al. A blockchain framework for insurance processes[C]// 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2018: 1-4.

[8] WOOD G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum Project Yellow Paper, 2014, 151 (2014): 1-32.

[9] Etherscan. Total Ether Supply[EB/OL]. https://cn.etherscan.com/stat/supply.

[10] CSDN. The reason for the Ethereum fork: the famous The DAO event [EB/OL]. https://blog.csdn.net/mrRqAEr7ci9s2v0/article/details/84949088.

[11] Zhihu. Analysis of Parity MultiSig Wallet Freezing[EB/OL]. https://zhuanlan.zhihu.com/p/31000130?from_voters_page=true.

[12] LI J, ZHAO B, ZHANG C. Fuzzing: a survey[J]. Cybersecurity, 2018, 1(1): 1-13.

[13] KAKSONEN R, LAAKSO M, TAKANEN A. Software security assessment through specification mutations and fault injection [M]// Communications and Multimedia Security Issues of the New Century. Boston: Springer, 2001: 173-183.

[14] SCHUMILO S, ASCHERMANN C, GAWLIK R, et al. kafl: Hardware-assisted feedback fuzzing for {OS} kernels[C]// 26th {USENIX} Security Symposium ({USENIX} Security 17). 2017: 167-182.

[15] ZHENG Y, DAVANIAN A, YIN H, et al. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation[C]// 28th {USENIX} Security Symposium ({USENIX} Security 19). 2019: 1099-1114.

[16] LIU B, ZHANG C, GONG G, et al. {FANS}: Fuzzing Android Native System Services via Automated Interface Analysis[C]// 29th {USENIX} Security Symposium({USENIX} Security 20). 2020: 307-323.

[17] JIANG B, LIU Y, CHAN W K. Contractfuzzer: Fuzzing smart contracts for vulnerability detection[C]// 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2018: 259-269.

[18] Consensys. Homepage of Consensys [EB/OL]. https://www.consensys.net/.

[19] 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. 2020: 1398-1409.

[20] Trail of Bits. Homepage of Trailofbits[EB/OL]. https://www.trailofbits.com/.

[21] GRIECO G, SONG W, CYGAN A, et al. Echidna: effective, usable, and fast fuzzing for smart contracts[C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020: 557-560.

[22] GROCE A, GRIECO G. echidna-parade: a tool for diverse multicore smart contract fuzzing[C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021: 658-661.

[23] LIU C, LIU H, CAO Z, et al. Reguard: finding reentrancy bugs in smart contracts[C]// 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). IEEE, 2018: 65-68.

[24] HE J, BALUNOVIĆ M, AMBROLADZE N, et al. Learning to fuzz from symbolic execution with application to smart contracts [C]// Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 531-548.

[25] LIAO J W，TSAI T T，HE C K，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）. IEEE，2019：458-465.

[26] WANG H，LIU Y，LI Y，et al. Oracle-supported dynamic exploit generation for smart contracts[J]. IEEE Transactions on Dependable and Secure Computing，2022，19(3)：1795-1809.

[27] ZHANG Q，WANG Y，LI J，et al. Ethploit：From fuzzing to efficient exploit generation against smart contracts[C]// 2020 IEEE 27th International Conference on Software Analysis，Evolution and Reengineering（SANER）. IEEE，2020：116-126.

[28] 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.

[29] CHOI J，KIM D，KIM S，et al. SMARTIAN：Enhancing Smart Contract Fuzzing with Static and Dynamic Data-Flow Analyses[C]// 2021 36th IEEE/ACM International Conference on Automated Software Engineering（ASE）. IEEE，2021：227-239.

[30] ZHOU T，LIU K，LI L，et al. SmartGift：Learning to Generate Practical Inputs for Testing Smart Contracts[C]// 2021 IEEE International Conference on Software Maintenance and Evolution（ICSME）. IEEE，2021：23-34.

[31] ALMAKHOUR M，SLIMAN L，SAMHAT A E，et al. Verification of smart contracts：A survey[J/OL]. Pervasive and Mobile Computing，2020，67：101227. https：//doi. org/10. 1016/j. pmcj. 2020. 101227.

[32] TOLMACH P，LI Y，LIN S W，et al. A survey of smart contract formal specification and verification[J]. ACM Computing Surveys（CSUR），2021，54(7)：1-38.

[33] PRAITHEESHAN P，PAN L，YU J，et al. Security analysis methods on ethereum smart contract vulnerabilities：a survey[J]. arXiv：1908. 08605，2019.

[34] VUJIČIĆ D，JAGODIĆ D，RANDIĆ S. Blockchain technology，bitcoin，and Ethereum：A brief overview[C]// 2018 17th International Symposium Infoteh-jahorina（Infoteh）. IEEE，2018：1-6.

[35] Ethereum. Solidity[EB/OL]. https：//docs. soliditylang. org/en/v0. 8. 13/.

[36] Ben Edgington. LLL Complier Documentation[EB/OL]. https：//lll-docs. readthedocs. io/en/latest/lll_introduction. html.

[37] Ethereum. Serpent[EB/OL]. https：//github. com/ethereum/serpent.

[38] Vyperlang. Pythonic Smart Contract Language for the EVM[EB/OL]. https：//github. com/vyperlang/vyper.

[39] CornellBlockchain. Bamboo：a morphing smart contract language[EB/OL]. https：//github. com/cornellblockchain/bamboo.

[40] DASP. Decentralized Application Security Project （or DASP） Top 10 of 2018[EB/OL]. https：//www. dasp. co/＃item-7.

[41] NccGroup. Homepage of NccGroup[EB/OL]. https：//www. nccgroup. com/.

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

[43] CHEN J，XIA X，LO D，et al. Defining smart contract defects on ethereum[J]. IEEE Transactions on Software Engineering，2022，48(1)：327-345.

[44] ZALEWSKI M. American fuzzy lop[EB/OL]. https：//github. com/google/AFL.

[45] CHOI J，JANG J，HAN C，et al. Grey-box concolic testing on binary code[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering（ICSE）. IEEE，2019：736-747.

[46] BARR E T，HARMAN M，MCMINN P，et al. The oracle problem in software testing：A survey[J]. IEEE transactions on software engineering，2014，41(5)：507-525.

[47] AMMANN P，OFFUTT J. Introduction to software testing[M]. Cambridge：Cambridge University Press，2016.

[48] 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.

[49] 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）. IEEE，2019：1186-1189.

[50] MUELLER B. Mythril-Security analysis tool for EVM bytecode[EB/OL]. https：//github. com/ConsenSys/mythril.

[51] TOLMACH P，LI Y，LIN S W，et al. A survey of smart contract formal specification and verification[J]. ACM Computing Surveys（CSUR），2021，54(7)：1-38.

HUANG Song，born in 1970，Ph.D，professor，Ph. D supervisor，is a senior member of China Computer Federation. His main research interests include software testing and software reliability.

DU Jin-hu，born in 1998，postgraduate. His main research interests include smart contract security and fuzzing.

（责任编辑：何杨）