计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210600117

智能合约漏洞检测工具研究综述

涂良琼　孙小兵　张佳乐　蔡　杰　李　斌　薄莉莉
扬州大学信息工程学院　江苏 扬州 225127
(tulq1336@163.com)

摘　要　智能合约是区块链平台实现交易的重要组件,为多方交易间信任问题提供了一种有效的解决方案。智能合约不仅管理高价值代币还具有不可更改等特性,导致近年来智能合约多次遭受安全威胁。目前出现了大量关于智能合约安全性的研究,其中智能合约漏洞检测成为主要关注点。文中系统分析了智能合约安全问题,从是否执行合约的角度将漏洞检测工具分为静态检测工具和动态检测工具,并对检测工具进行对比分析,重点分析现有检测工具的漏洞检测能力,介绍了16种检测技术的原理及优缺点;最后,对如何提高智能合约安全性进行展望,提出了3个可能提高智能合约安全性的研究方向。

关键词:区块链;智能合约;漏洞检测
中图法分类号　TP311

Survey of Vulnerability Detection Tools for Smart Contracts

TU Liang-qiong,SUN Xiao-bing,ZHANG Jia-le,CAI Jie,LI Bin and BO Li-li
School of Information Engineering,Yangzhou University,Yangzhou,Jiangsu 225127,China

Abstract　Smart contract is an important component of blockchain platform to realize transactions,which provides an effective solution to the trust problem between multi-party transactions. Smart contracts not only manage high value tokens but also have the characteristics of immutable,which lead to the security threats of smart contracts many times in recent years. At present,a lot of researches have devoted to the security of smart contracts,among which the vulnerability detection of smart contracts has become the main concern. This paper analyzes the security of smart contract systematically. From the perspective of whether to execute the smart contract,vulnerability detection tools are divided into static detection tools and dynamic detection tools. In particular,the vulnerability detection ability of existing detection tools is analyzed,and the principles,advantages and disadvantages of 16 detection technologies are discussed. Finally,the paper gives a prospect of how to improve the security of intelligent contract,and puts forward three research directions which may improve the security of smart contract.

Keywords　Blockchain,Smart contract,Vulnerability detection

1　引言

智能合约是区块链系统中实现交易逻辑的去中心化应用程序,可编程的智能合约让区块链可应用于不同领域。从金融服务到供应链[1-3],从物流到医疗社会保障[4-6],区块链平台逐渐渗入到各行各业中,而智能合约作为区块链中比较重要的一环且管理着高额价值代币,对其安全性的研究尤为重要。智能合约运行在区块链平台中,继承了区块链分布式、去中心化以及不可篡改的特性。由于智能合约一旦部署就无法进行更改,而软件的漏洞不可避免,因此智能合约的漏洞一旦被利用就会产生严重的后果。并且智能合约通常掌管着价值数十亿美元的虚拟币,如果智能合约遭受攻击常常会带来巨大的经济损失。2016年著名DAO攻击[7]导致了6000万美元的损失。2017年Parity钱包遭受了两次不同漏洞攻击,一次导致了6000万美元的损失[8],一次冻结了超过1.5亿美元的以太币[9]。2018年美链BEC智能合约受到攻击,攻击者可以

到稿日期:2021-06-16　返修日期:2021-08-12
基金项目:国家自然科学基金(61872312,61972335,62002309);南京大学计算机软件新技术国家重点实验室资助项目(KFKT2020B15,KFKT2020B16);扬州大学高端人才支持计划(2019);江苏省"六大人才高峰"高层次人才项目(RJFW-053);江苏省"333"工程中青年科学技术带头人项目;扬州大学畜牧学学科特区学科交叉课题支持项目(yzuxk202015);江苏省高等学校自然科学研究面上项目(20KJB520024)
This work was supported by the National Natural Science Foundation of China(61872312,61972335,62002309),Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University(KFKT2020B15,KFKT2020B16),Yangzhou University Top-level Talents Support Program (2019),Six Talent Peaks Project in Jiangsu Province (RJFW-053),Jiangsu "333" Project,Cross-Disciplinary Project of the Animal Science Special Discipline of Yangzhou University (yzuxk202015) and Natural Science Foundation of the Jiangsu Higher Education Institutions of China (20KJB520024).
通信作者:孙小兵(xbsun@yzu.edu.cn)

无限生产代币，致使代币的价格大跌至接近于零。近几年层出不穷的攻击事件让智能合约的安全性受到广泛关注。比特币[10]和以太坊[11]是目前比较流行的两大区块链平台，比特币是区块链 1.0 时代最具代表性的平台，对于比特币平台，研究者主要是从交易数据进行研究[12]；以太坊是第一个可部署和运行智能合约的区块链平台，因此基于以太坊智能合约漏洞、安全威胁的研究众多。

Feng 等[13]对常见漏洞触发逻辑进行研究，并比较使用模糊测试、符号执行和形式化验证技术的检测工具，发现这些工具不兼容以太坊平台更新。Liu 等[14]从软件安全保证和正确性验证两方面研究现有工作，通过研究发现基于形式化的验证方法能够作为验证智能合约是否准确和可信的主要方法。Ni 等[15]对智能合约运行环境的程序特性、合约安全威胁和漏洞及自动化检测漏洞技术进行分析比较，并对使用形式化验证、模糊测试、符号执行 3 种技术的优劣进行分析总结。用于检测的可选择工具数量有很多，选择哪一个工具并且如何开始使用这些工具对于用户来说比较困难，Antonio 等[16]通过实践总结出每个检测工具安装使用的难易程度，为使用者提供参考。Mehmet 等[17]对智能合约漏洞展开研究，并对这些漏洞进行分类。

然而现有工作仅从合约安全威胁或者检测手段进行分析，并没有对检测工具的检测能力进行分析，因此本文从是否运行合约程序的角度将检测工具分为静态检测工具和动态检测工具，并比较和分析了各检测工具的漏洞检测能力。目前使用不运行合约的静态检测方法较多，大部分检测工具都需要专家知识建立检测模型，导致工具检测效率较低。智能合约安全威胁包括区块链、编程语言和虚拟机 3 方面，本文也从这 3 个层面安全威胁的角度比较各检测工具的检测能力。通过对当前智能合约挖掘技术的调研，结合对检测工具漏洞检测能力的分析，总结现有合约漏洞检测的研究进展以及目前还面临的挑战。最后本文对如何提高智能合约安全性提出了展望，总结出 3 个可能改进合约安全性的研究方向：1)动态保护方法，在不影响合约的正常运行的情况下对合约进行保护；2)结合传统分析方法和机器学习方法来提高检测工具的效率；3)使用静态分析指导动态测试方法，提高漏洞检测能力。

本文第 2 节对智能合约运行环境生命周期及自身特性进行了概述；第 3 节分析了漏洞的原因并给出了代码例子；第 4 节是智能合约漏洞检测工具的方法以及优缺点的比较；最后是对智能合约漏洞检测的总结与展望。

2 智能合约概述

2.1 智能合约运作环境

智能合约的概念于 1994 年由尼克·萨博(Nick Szabo)提出，其定义为："一个合约是一套数字定义的承诺，包括合约的参与方可以在上面执行这些承诺的协议。"[18]区块链技术的出现让智能合约有了很好的运作平台，并且区块链具有的透明、去中心化以及防篡改等特性解决了第三方信任问题，完美提高了智能合约的信任度。区块链是一个分布式系统，系统中每个节点通过 P2P(Peer-to-Peer)网络相连。节点可通过发送交易来改变区块链状态，交易通过 P2P 网络同步到全网中，由矿工进行打包确认。以太坊是第一个支持智能合约开发与运作的区块链平台，它使用 solidity[19]编程语言进行智能合约的开发，并且提供了大量的应用程序接口(API)。以太坊账户分为两类：一类是外部账户，拥有余额并可以进行合约调用；另一类是内部账户，与外部账户相比，其除了可以进行合约调用，还拥有余额和可被执行的合约代码。

智能合约运行在以太坊虚拟机(EVM)中，EVM 是以太坊实现图灵完备计算的核心结构，是一个基于栈的虚拟机，用于执行智能合约字节码。以太坊智能合约的存储空间分为两种，一种是内存存储(stroage)，一种是临时存储(memory)。智能合约被调用时，EVM 根据智能合约相应的操作指令执行操作并进行栈运算，运算过程中与存储空间进行数据交互。

区块链是智能合约的运作环境，其特点也是智能合约安全的重要影响因素。如果智能合约开发者对区块链的特性和底层技术一知半解，将会导致智能合约遭受安全威胁。

2.2 智能合约生命周期

智能合约是区块链上执行多方交易的一段程序代码，旨在验证和执行合同，与传统软件一样存在生命周期。智能合约的生命周期从开发到停止使用期间经历了设计、开发、部署、调用和销毁 5 个阶段，由于智能合约不可更改，因此它的生命周期不存在传统意义上的维护阶段。

(1)设计

需求分析是对软件问题的定义，明确软件"做什么"。设计是对定义的问题提出解决方案，知道"如何做"。软件设计分为概要设计和详细设计，概要设计的任务是确定软件的总体结构，详细设计的目的是明确软件模块的实现过程。智能合约是区块链上的应用程序，设计也是其生命周期的第一个阶段。

(2)开发

智能合约开发与传统软件开发相似，使用高级语言对其进行编码。目前支持部署智能合约的平台有 40 多个，都有相应的合约开发语言。其中大部分平台都使用 solidity 语言进行开发，以太坊区块链平台就是其一。用于开发智能合约的编程语言除了 solidity 外，还有 Vyper[20]，Idris[21]和 Rust[22]等 10 多种。开发好的智能合约代码需要经过编译。程序编译是将高级语言编写的程序代码翻译为可用于机器计算的机器代码，智能合约中用于执行的二进制代码被称为字节码，以太坊虚拟机通过将字节码翻译为相应的操作码以完成合约的执行。以太坊智能合约通过编译器编译后还会产生应用调用接口(Application Binary Interface，ABI)以供区块链解析出函数选择器从而实现合约函数的调用。

(3)部署

智能合约在区块链平台上运行，需要通过部署到区块链中才能同步到各个节点以供调用。合约的部署是通过向区块链发送一笔交易实现的，该交易的接收方为空，包含字节码等信息，经过矿工将交易打包成功后返回一个地址。该地址就是合约地址，合约的调用或者访问都需要经过这个地址，它是合约的唯一标识。

(4)调用

合约调用需要使用合约地址和应用程序二进制接口(ABI),区块链系统中的账户都可以对合约进行调用。具体的调用方式有两种:发送交易或者消息。“交易”在以太坊白皮书中的定义为外部账户签名的一串数据[17],因此普通账户即外部账户通过发送调用交易(transaction call)来调用合约。交易会被广播到以太坊网络的全部节点,并由矿工打包验证,所有交易记录都会被存储在区块链中。以太坊黄皮书中对“消息”的定义为两个账户间发送的数据(data)和值(value,即以太币)[17],合约账户(内部账户)通过消息调用(message call)完成对其他合约的调用。消息是内部参数的传递,不会同步到区块链上,因此合约间通过消息调用方式调用合约并不会被记录在链中。

(5)销毁

以太坊智能合约提供销毁功能,需要在开发时写入合约中。由于区块链数据无法被删除,因此销毁的合约仍然存在于链中,只是无法再被调用且状态被标记为销毁。

智能合约在生命周期的不同阶段完成不同任务,各阶段都可能因开发者技术上的不成熟导致一些安全问题。因此,如何提高智能合约安全性需要开发者以及合约调用者对智能合约生命周期各阶段的特性有深入了解。此外,提高合约漏洞检测工具的能力也刻不容缓。

2.3 gas 机制

以太坊智能合约的执行需要消耗资源,而且合约公开,可被任意调用,如果攻击者通过无限循环调用合约或者发送逻辑复杂、难以处理的交易对合约进行调用,会导致矿工付出巨大的代价来打包验证交易。为了避免单笔交易中智能合约的执行时间过长,导致区块链网络发生阻塞,以太坊提出了 gas(燃油)机制来限制智能合约执行所消耗的计算资源。以太坊的 gas 机制让合约调用者付出相应的成本,一定程度上保障了合约的正常运作。发送方发送交易时需要设定好 gas 的两个值,gas limit 和 gas price。gas limit 表示单笔交易支付上限,gas price 表示交易消耗成本的单价。交易执行时,被实际消耗掉的 gas(燃油量)表示为 gas used。因此发送方真正消耗掉的 gas 总费用为 gas used ×gas price。当真正消耗的总费用小于或等于用户定义的上限值时,合约被正常执行且返回剩余的 gas,否则区块链会抛出 out-of-gas 的异常。

gas 机制用于保证合约的正常执行,同时也可能存在安全隐患,如果存在恶意用户利用这一机制使程序出现恶意异常,则会影响合约的正常执行,进而引发一些安全漏洞。

3 智能合约安全漏洞

2017 年 Atzei 等[23]分析了以太坊智能合约上的安全漏洞,首次从编程语言、虚拟机、区块链 3 个层面对智能合约漏洞进行分类。同年 Dika[24]沿用 Atzei 等[23]的分类,将已知的安全问题进行归类,并进行了安全等级分配(1—3 标志低、中、高)。智能合约生命周期的安全性与编程语言、虚拟机和区块链这 3 部分的联系密不可分,不同层面的特性会导致不同漏洞,本节将对这 3 个层面产生的漏洞进行描述。详细的漏洞分类及原因如表 1 所列。

表 1 漏洞分类和漏洞原因

Table 1 Vulnerability classification and vulnerability causes

漏洞分类	漏洞名称	漏洞原因
编程语言	严格 balance 比较	balance 状态变量可被任意更改
	未检查返回值	发送以太币未检查返回值
	拒绝服务	程序异常导致无法响应正常请求
	整数溢出	整数范围错误
	tx. origin 依赖漏洞	错误使用 tx. origin 全局变量
	贪婪合约	永久冻结以太币
虚拟机	重入漏洞	使用 fallback 函数递归调用同一个函数
	危险的代理调用	使用 delegatecall 函数注入恶意代码
	短地址攻击	合约地址长度不一致
区块链	区块状态依赖	使用区块相关数据作为随机数种子
	交易顺序依赖	交易执行顺序不一致导致产生条件竞争问题

3.1 编程语言层面

目前编写智能合约使用最多的高级语言是 solidity,它设计了大量 API 为开发者提供便利,但由于其自身的程序特性以及开发者使用不当使合约存在安全漏洞,容易受到攻击者攻击。下面对编程语言层面产生的漏洞进行描述,并给出代码例子进行说明。

3.1.1 严格 balance 比较

严格 balance 比较是指合约执行逻辑依赖于合约余额与一个确切值严格相等,但是合约的余额值是透明且可以被人为更改的。当一个合约被销毁时(如合约调用 selfdestruct 函数)可以向任何合约发送以太币,并且接受合约无法拒绝此次转账操作。如果合约存在严格 balance 比较这样的漏洞,则攻击者可以更改合约余额从而导致合约执行逻辑出现错误。

如图 1 所示,对于 this. balance 与 23ether 的比较,使用第 3 行大于等于的比较形式比使用第 1 行严格等于的比较形式更为安全。

```
1.  if(this.balance==23ether);//bad
2.      dosomething();
3.      if(this.balance>=23ether);//good
4.  dosomething();
```

图 1 严格 balance 比较的例子

Fig. 1 Strict balance comparison vulnerability example

3.1.2 未检查返回值

未检查返回值是在合约转账时发生的一类漏洞,是智能合约代码中没有对转账操作的执行情况进行检查导致的。solidity 通过 transfer(),send(),call()这 3 种函数实现转账操作。当转账失败时,transfer()会自动抛出异常,而 send()和 call()函数不会抛出异常且继续执行剩余代码。如果未对转账函数的执行情况进行检查,会给攻击者留下可乘之机,攻击者可以通过某种方式故意让转账失败,以此影响合约的正常执行。

如图 2 所示,pay 函数的作用是合约向地址为 address 的账户支付一定数量的代币并且将 paid 的状态设置为 true。第 3 行中 address. send(amount)将一定数量的代币转给该地址,然后在第 5 行将 paid 状态改为 true。假设此时地址为 address 的账户是一个攻击者,该恶意账户使用某种方式导致第 3 行执行失败返回 false,并继续执行下一行代码。实际上以太币并没有被转出而 paid 的状态仍被设置为 true,pay 函数

与其预期的执行状态不相符。因此第 4 行使用 require(address.send(amount))的编写方式更安全，只有第 4 行成功执行才会继续执行下一行代码，避免出现未检查返回值的漏洞。

```
1.  function pay(unit245 amount){
2.      require(! paid);
3.      address.send(amount);//bad
4.        //require(address.send(amount);//good
5.        paid = true;
6.  }
```

图 2 未检查返回值的例子

Fig. 2 Unchecked send example

3.1.3 拒绝服务

拒绝服务(DOS)是指用户请求得不到正确响应，程序无法提供正常服务，是一种较为常见的漏洞。智能合约中的拒绝服务漏洞会让合约自身的逻辑无法正确实现，扰乱合约正常交易，使合约陷入无法向其他用户提供服务的异常状态中。攻击者对智能合约进行拒绝服务攻击有多种实现方式，如无限循环 DOS 攻击和 gas-limit send DOS 攻击等。由于开发者编写合约时考虑不周使得合约在一段时间后出现拒绝服务的漏洞，这种漏洞被称为被动拒绝服务，是在智能合约编写过程中引入的。另一种拒绝服务漏洞被称为主动拒绝服务，是恶意用户的恶意行为导致区块链系统繁忙而无法正常响应其他用户的请求。本文主要关注被动拒绝服务，由于智能合约无法篡改，因此智能合约一旦出现拒绝服务的现象，合约将永久瘫痪，无法修复。

图 3 是一个竞标王位的合约，如果有人想要竞争王位，则需要付出比现有的竞标价即 hightestBid 更高的金额，现有的竞标价会归还给当前的王位拥有者，然后将王位转交给竞标者，并且将竞标价进行更新。

```
1. contract auction{
2.      unit256 highestBid;
3.      function bid(address king) pubilc payable{
4.      require(msg.vaule>highestBid);
5.      require(king.send(highestBid));
6.      king = msg.sender;
7.      highestBid = msg.vaule;
8. }
9. contract attack{
10.     …
11.     function bid(address king){
12.       king.bid(msg.sender)
13.     }
14.     function ()payable{…}
15.     …
16. }
```

图 3 拒绝服务的例子

Fig. 3 DOS example

目前，合约状态的改变依赖于外部函数执行的结果，即第 5 行执行成功才会执行第 6－7 行代码以更改合约状态值，并且没有考虑到外部函数执行失败而导致交易回滚的情况(即只能执行到第 5 行)。若此时存在一个恶意用户，首先成为当前的王位拥有者，当其他用户出更高金额来争夺王位时，会将现有竞标价归还给恶意用户，恶意用户故意使 king.send(highestBid)执行失败，无论其他用户出多高的金额，合约都无法将王位转让出去。

3.1.4 整数溢出

编程语言对不同类型整数设置了不同长度的存储空间，因此不同整数类型有不同大小的表示范围。整数溢出是指在运算时结果超过了其本身的表示范围，出现越界值。智能合约中整数溢出漏洞容易发生于发币合约中，现实中合约由于整数溢出被攻击的案例层出不穷，属于高风险漏洞。攻击者利用这一漏洞可使用少量代币向地址转入大量的代币，造成严重的经济损失。

如图 4 所示，第 2 行 amount 是两个 unit256 长度的数值相乘，如果给一个非常大的数值 value 导致计算结果出现溢出，amount 的值变为 0，则第 3 行中 require(_value>0&&balances[msg.sender]>=amount)检查逻辑失效，因为正常情况下 amount 应该是一个非零数值。

```
1. function paidout(address[] _buyer,unit256 _value){
2.      unit256 amount = unit256(_buyer.length) * _value;
3.      require(_value>0&&balances[msg.sender]>=amount);
4.      …
5.      return true;
6. }
```

图 4 整数溢出的例子

Fig. 4 Integer overflow

3.1.5 tx.origin 依赖漏洞

solidity 中的 tx.origin 是一个全局变量，表示发起本次交易的地址，如合约 A 发起一笔交易调用合约 B，合约 B 又调用了合约 C，整个调用链可抽象为 A→B→C，则 tx.origin 为合约 A 地址，msg.sender 为合约 B 地址。tx.origin 依赖漏洞是指使用 tx.origin 去判断控制权限，而非使用 msg.sender。攻击者可以利用这个漏洞实现身份伪装，对调用合约的用户进行攻击。

如图 5 所示，存在两个合约，Wallet 合约和 Attack 合约。Wallet 是一个钱包合约，调用者通过调用该合约实现取钱。攻击合约通过使用 getMoney 函数调用钱包合约，由于第 7 行中使用 tx.origin == owner 作为判断条件，而 tx.orgin 代表最初发送交易的地址，此时攻击合约可以成功欺骗钱包合约，获得转账资格盗取以太币。

```
1. contract Wallet{
2.      address owner;
3.      constructor(){
4.          owner = msg.sender;
5.      }
6.      function sendMoney(address reciver){
7.          require(tx.origin == owner);
8.          reciver.transfer(2ether);
9.      }
10. }
11. contract Attack(){
12.     function getMoney(address victim,address attack)
13.     payable{
14.     Wallet(victim).sendMoney(attack) ;
15.     }
16. }
```

图 5 tx.origin 依赖漏洞的例子

Fig. 5 tx.origin dependency vulnerability example

3.1.6 贪婪合约

贪婪合约漏洞会导致资金冻结问题,这类漏洞出现的原因是有些合约依赖于其他合约(如库合约)实现向其他地址转账的功能,但自身无法实现转账操作。如果库合约被销毁,则合约中的以太币无法被转出,将致使资金永久冻结。智能合约有一个显著的特点,拥有高额价值,一旦出现贪婪合约漏洞,后果无法挽回。

3.2 虚拟机层面

虚拟机是智能合约的执行环境,由于其自身的运行机制或开发者不恰当的使用方式,程序也会出现漏洞。

3.2.1 重入漏洞

重入漏洞是一类危险系数较高的安全漏洞,著名的 DAO 攻击就是利用了该漏洞。该漏洞通常发生在向外部用户地址转账的过程中,由外部用户地址递归调用合约同一个函数而引发。solidity 默认合约中存在一个没有函数名和参数的函数,称为回调函数(fallback),此函数在收到转账时会自动被触发。当合约转账操作修改合约 Storage 状态变量操作之前就可能受到重入攻击,因为攻击者可以注入一个恶意 fallback 函数再次调用合约的同一个函数。对于重入漏洞存在多种规避方式,比如将修改余额操作放在转账操作之前,再使用 send 或 transfer 对 gas 有限制的转账方式。

如图 6 所示,代码片段表示可用于撤回资金的受害者合约。合约中存在一个 withdraw 函数用于 receiver 撤回资金,第 5 行首先会检查接收者合约的余额是否有足够金额,若余额充足则将金额转给接收者(第 6 行),第 7 行接收者的余额会减去相应的转账金额。

```
1. contract Victim{
2.    mapping(address=>unit 256) balances;
3.    …
4.    function withdraw(unit256 amount) public {
5.    require(balances[msg. sender]>=amount);
6.    msg. sender. call. value(amount)();
7.    balances[msg. sender]-=amount;
8.    }
9.    …
10.}
```

图 6 重入漏洞例子——受害者合约

Fig. 6 Example of reentry vulnerability—victim contract

图 7 中存在一个攻击者合约,第 4 行首先调用受害者合约 withdraw 函数撤回一定数量资金,并且在第 7 行 fallback 函数中再次调用 withdraw 函数。

```
1. contract Attack{
2.    …
3.    function transfer(){
4.       victim. withdraw (this. amount);
5.    }
6.    function() payable {
7.       victim. withdraw (this. amount);
8.    }
9.    …
10.}
```

图 7 重入漏洞-攻击者合约

Fig. 7 Example of reentry vulnerability-attack contract

由于图 6 中转账操作和修改账户余额之前(第 5—6 行),图 7 中的第 7 行就可以重复调用 withdraw 函数,直到将受害者合约资金转完或者 gas 消耗完才会停止转账操作,因此重入漏洞会造成不可挽回的巨大经济损失。

3.2.2 危险的代理调用

solidity 提供了一个 delegatecall 函数为库函数的实现提供了基础,delegatecall 是在自己上下文的环境中执行他人的代码。如果 delegatecall 函数使用不当,则会导致合约执行恶意代码。如当 delegatecall 函数调用目标地址和目标函数作为输入的函数时,攻击者可以使受害者合约执行任意一段代码,甚至可以轻易更改受害者合约中的状态值。

3.2.3 短地址攻击

短地址攻击是由于虚拟机加载输入参数,对长度不符合要求的字段在末尾加零自动补位,从而改变了数据的原始大小。攻击者利用虚拟机自动补位的特性,输入一个地址位数不满足条件的地址参数值,若此时第二个参数恰好以零开头,则 EVM 在加载第二个参数时通过自动补全机制扩大了第二个参数的数值范围,并且地址保持不变,如果该攻击者想通过此类漏洞进行资金交易,则攻击者可以得到更多的货币。

3.3 区块链层面

区块链是分布式系统,具有不可篡改和数据透明等特性。智能合约存储在区块链上同样不可更改且透明。这些特性解决了合约的信任危机,但同时也增大了合约被攻击的风险。

3.3.1 区块状态依赖

区块状态依赖是指合约中使用了时间戳,将区块数等与区块相关的变量值作为随机种子。但与区块相关的变量受矿工的影响是可操控的,由于区块链数据公开透明,随机数是可以进行预测的。若是使用这些变量去生成随机数,在某种程度上,矿工可以控制与区块相关的变量值,则生产随机数去控制程序执行就失去其原始的公平性,可能也会面临安全威胁。

3.3.2 交易顺序依赖

区块链中交易的顺序决定了此时整个链的状态,区块链中一个区块包含的是一段时间内产生的交易集合,而交易的顺序由矿工进行重新排序,用户对交易的执行顺序无法提前预知。假设在一个悬赏合约中悬赏能够解答难题的人,若 A 先找到答案并发送了一笔交易,并广播给以太坊网络中所有节点,此时攻击者 B 剽窃了答案,以更高代价发起自己的交易,则攻击者的交易可能更容易被先执行,而 A 付出了努力却无法得到回报。

4 漏洞检测工具

以太坊智能合约部署且运行在区块链上,区块链不可篡改的特性也相应继承到合约上。这一特性给合约带来可信度的同时也带来很大的安全隐患,并且合约较传统软件存储有更大的价值,因此智能合约的漏洞检测尤为重要。从是否运行程序的角度进行分类,漏洞检测技术可以分为静态检测和动态检测两大类。对于合约漏洞的检测,从是否执行合约分类,也可以将合约漏洞检测工具分为静态检测工具和动态检测工具。

4.1 静态检测工具

静态检测是在不运行程序的情况下分析软件程序进行漏洞检测，检测方法主要分为静态分析和程序验证。静态分析从程序的语义或语法层面进行分析，一般会将程序代码或者二进制文本翻译为中间表示，对中间表示进行漏洞分析。静态分析方法有很多，最基本的两种分析方法是控制流分析和数据流分析[25]。针对智能合约进行静态分析的方法主要是通过控制流分析构建控制流图。程序验证通过将程序进行形式化或建立抽象模型以验证程序不存在某种缺陷或程序的正确性。通常使用模型检验、定理证明或符号执行3种方法进行形式化验证。在智能合约安全分析领域，这3种方法目前都存在很多相关工具的研究，下面将对这些研究工作分别进行描述。

4.1.1 静态分析

Smartcheck[26]是对以太坊智能合约的漏洞进行检测的工具。它对合约源代码进行语法和词法的分析，使用语法分析器 ANTLR 工具结合 solidity 语法生成 XML 解析树作为中间表示，并使用 Xpath 在中间表示上进行查询来匹配安全属性。由于分析的目标源码能够完全翻译为中间表达，并且 Xpath 能够匹配到所有元素，因此该工具能达到代码全覆盖。Smartcheck 对简单的漏洞修复工作能提供更好的帮助，但该工具亦存在一些局限性，如无法检测某些复杂的漏洞，需要更为复杂的技术如污点分析等。

Slither[27]是既支持以太坊智能合约漏洞检测又提供理解合约代码功能的分析框架。它创建了一个名为 SlithIR 的中间表示，用于表达合约源码。该工具的中间表达通过合约源代码的抽象语法树(AST)进行构建，在构建中间表示之前会尽可能恢复合约信息如合约继承图、控制流图等。漏洞检测以及代码理解都是在中间表示的基础上进行分析，提高了工具的通用性，可以实现其他区块链平台的分析。Slither 漏洞检测目前能够检测出20多个 bug，包含重入、自杀合约、资金锁住等常见的 bug。

Vandal[28]是用于以太坊智能合约的安全分析框架。与其他研究工作不同，Vandal 使用逻辑驱动的静态程序分析方法。它从智能合约字节码中提取出语义逻辑关系进行安全漏洞分析，主要由两部分构成。第一部分是将字节码翻译为逻辑关系，用于安全分析的输入；第二部分是一组安全问题的逻辑规范，作为安全分析的规则基础。该工具能较为高效地检测出重入、未检查 send 等多种合约漏洞。

4.1.2 程序验证

Zeus[29]使用抽象解释和符号模型对智能合约进行漏洞检测，并基于 LLVM 生成规则的形式化验证工具。Zeus 主要由3部分构成，分别是规则生成器、源代码翻译器、验证器。规则生成器基于 solidity 语义使用 XACML 生成验证规则，源代码翻译器将智能合约源代码翻译为 LLVM 中间语言，最后验证器利用 Seahorn[30]验证框架构建符号模型进行形式化验证。Zeus 可以对5种安全漏洞进行检测。由于 Zeus 使用了 LLVM 位码进行操作的验证器，而编写智能合约的大多数编程语言都有较为成熟的 LLVM 位码转换器，使得 Zeus 能够支持不同的区块链平台。但是该工具也存在缺点，即在将智能合约程序代码转换为 LLVM 的过程中，无法保留智能合约程序的所有语义，也就无法对智能合约一些特有的性质进行表达，降低了该工具的性能。

Orisis[31]是基于符号执行和污点分析的静态分析框架，主要关注以太坊智能合约整数溢出漏洞检测。该工具的输入为智能合约的字节码或源码，后者可被编译为字节码。Orisis 有3大分析组件：符号分析组件、污点分析组件、整数 bug 检测组件。符号分析组件构建控制流图，然后符号化执行合约的不同路径，将结果传送给污点分析组件和整数 bug 检测组件；污点分析组件通过引入污点对结果进行跟踪，一定程度上降低了工具的误报率，相较于其他研究工作其误报率得到了很大的改进。

Scompile[32]基于符号执行对以太坊智能合约漏洞进行检测，优化了路径的选择，提高了检测效率。该工具将以太坊字节码作为输入，构建控制流图，并枚举出所有路径，识别出与货币相关的路径，并根据它是否违反预先定义的安全属性进行评分排序。然后使用符号执行过滤掉不可行的关键路径，最后将检查结果以及相关路径呈现给用户。

Securify[33]是基于事实和规则的智能合约形式化验证工具。该工具的整个验证过程分为3步：首先将 EVM 字节码转换为静态单一赋值(SSA)形式；然后从中推断出合约程序的语义事实，并将语义事实用一种基于逻辑的编程语言 Datalog 进行描述；最后根据预定义的安全属性规则进行检查。Securify 定义了7种安全漏洞属性规则，每一种属性规则都定义了两种模式，分别为遵从模式和违背模式。根据语义事实与两种模式的匹配情况进行合约安全性验证。

KEVM[34]是使用 K 框架生成可执行语义对 EVM 字节码进行形式化验证的工具，主要用于证明那些高额合约是否遵循了 ERC20 发币标准。文献[35-36]都基于 F * 对以太坊字节码进行形式化验证，文献[36]只将小部分的字节码进行形式化，而文献[35]可形式化更多的字节码。文献[37]基于 Isabelle/HOL 提出将以太坊字节码构建成相应的程序逻辑作为形式化验证的工具。

4.2 动态检测工具

动态分析方法通过运行被测程序，获取执行信息以进行安全漏洞分析。目前较为常用的3种动态漏洞检测方法主要是动态符号执行、模糊测试和动态污点分析[38]。

4.2.1 动态符号执行

符号执行是将程序的输入变量进行符号化，然后逐步分析程序的执行流程。早期的符号执行是静态的，不需要执行程序，而动态符号执行是在执行程序的过程中进行符号执行。动态符号执行的基本过程是将初始输入进行符号化，通过收集程序执行的路径约束进行约束求解来判断路径是否可达。基于符号执行检测智能合约漏洞的大部分工具将交易数据变量值进行符号化作为输入，约束求解过程中多数使用 Z3 求解方法。

Oyente[39]是第一个对智能合约进行安全分析的工具，它使用动态符号执行技术进行安全漏洞检测。该工具采用模块化设计，主要由4大组件组成，分别为控制流图构建组件(CFG Builder)、探索组件(Explorer)、核心分析组件(Core Analysis)和验证组件(Validator)。控制流图构建组件基于合约字节码构建控制流图，控制流图的节点表示为程序的基本

块,边表示为基本块间的跳转关系。由于有些边无法在此阶段通过静态分析确定出来,因此在后续的符号执行过程中会进一步完善边的关系。探索组件会基于控制流图进行符号执行,并使用 Z3 求解器确定执行路径。符号执行阶段是一个主要过程,为漏洞分析过程提供合约路径信息基础。该工具能检测出 4 种类型的漏洞,核心分析组件根据搜集到的路径信息分别对 4 种类型的漏洞构建相应的分析子组件。在将结果输出前,验证组件会将误报的结果过滤掉,但其并没有完全模拟以太坊的执行环境,因此过滤的效果并不明显。

Manticore[40]是基于动态符号执行的程序分析工具,不仅可以用于以太坊智能合约分析,还可以分析 Linux ELF 二进制文件。智能合约执行的本质是状态的转移,Manticore 将智能合约程序执行过程定义为 3 种状态:就绪(Ready)、忙碌(Busy)、终止(Terminated),3 种状态可以相互转换。该工具将合约交易进行符号化作为输入,选择一个就绪状态并执行,直到程序执行结束或者出现异常,停止执行。该工具可以对多个相互作用的合约进行分析,而其他工具都只是针对某个合约进行分析。

4.2.2 模糊测试

模糊测试又称随机测试,其基本原理是通过构造大量的随机数据作为输入,基于程序运行过程中产生的信息进行软件故障分析。它通过两种方法来进行测试输入数据的生成,一种是基于现有的数据进行变异产生,另一种是通过特定的算法或者协议进行生成。由于目前智能合约缺乏现有的缺陷数据样本,因此基于模糊测试进行智能合约漏洞检测的方法大部分都是基于第一种方法来构造测试用例。

ContractFuzzer[41]是第一个使用模糊测试方法对以太坊智能合约进行漏洞检测的工具。它首先对智能合约的应用程序二进制接口(ABI)进行分析,解析出函数参数类型等信息,并依据解析出的相关信息随机生成测试用例去调用合约,通过监听 EVM 来获取合约执行日志,并分析这些日志来报告安全漏洞。该工具对 7 种类型漏洞定义了相应的漏洞测试预言,能够较为高效地检测出合约漏洞,并且与 oyente 工具相比,其漏洞误报率得到很大的改善。

ReGuard[42]是基于模糊测试对以太坊智能合约进行漏洞检测的工具,主要检测重入漏洞。ReGuard 既支持源代码检测也支持字节码检测,其会将源代码或字节码翻译为 C++ 语言。该工具主要有 3 大模块:合约翻译器(Contract Transformer)、模糊引擎(Fuzzing Engine)以及检测器(Core Detector)。合约翻译器基于源代码的抽象语法树(AST)或者字节码的控制流图(CFG)实现合约的翻译。模糊引擎使用运行时覆盖率信息作为反馈,不断迭代生成随机字节运行程序。检测器基于程序运行过程的踪迹信息来报告漏洞。

4.2.3 动态污点分析

动态污点分析是在静态污点分析的基础上演变而来的,该分析方法是在运行程序的情况下,监控数据流和控制流,以此对程序进行安全分析。它的基本原理是将来自网络、文件等非信任渠道的数据标记为污染源,通过基于数据流分析或者控制流分析的方法追踪到被污染的属性。区块链通过发送交易调用智能合约,动态污点分析常将交易作为输入,标记输入污点,在合约执行的过程中追踪污点传播途径,以此进行漏洞检测。

EasyFlow[43]是对智能合约整数溢出这类漏洞进行检测的工具,不仅能够将合约分为安全合约和存在溢出漏洞的合约,还能够自动生成触发整数溢出漏洞的交易。该工具修改了以太坊客户端(go-ethereum),在交易执行过程中追踪污点传播。它将合约参数作为污点,观察与数值运算相关的操作数是否为标记的污点,如果其被标记为污点则生成以太坊交易去触发溢出漏洞,能够成功触发则报告为显式溢出漏洞,没有成功触发则报告为潜在溢出漏洞。如果标记的不是污点并且没有发生溢出操作则报告为安全合约。

从是否执行智能合约的角度可将检测工具分为静态检测工具和动态检测工具,各检测工具的分析技术、具体方法及研究对象如表 2 所列。

表 2 漏洞检测工具

Table 2 Vulnerability detect tools

工具分类	分析技术	研究工作	具体方法	研究对象	漏洞种类
静态检测工具	静态分析	smartcheck[26]	语法分析	源代码	7
		Slither[27]	语法分析	源代码	4
		Vandal[28]	语义分析	字节码	3
	程序验证	Zeus[29]	模型检验	源代码	6
		Orisis[31]	符号执行和污点分析	字节码	1
		KEVM[34]	形式化验证	字节码	1
		Grishchenko[36]	形式化验证	字节码	4
		Amani[37]	定理证明	字节码	2
		Bhargavan[35]	形式化验证	源代码/字节码	1
		Securify[33]	形式化验证	字节码	2
		Scompile[32]	符号执行	字节码	1
动态检测工具	动态符号执行	Oyente[39]	符号执行	字节码	4
		Manticore[40]	符号执行	源代码	2
	模糊测试	ContractFuzzer[41]	模糊测试	源代码/字节码	6
		ReGuard[42]	符号执行和模糊测试	源代码/字节码	1
	动态污点分析	Easyflow[43]	污点分析	字节码	1

4.3 检测工具的比较

4.3.1 漏洞检测能力比较

智能合约安全性一直都是研究热点,近年来也出现了相当多的研究工作。本文通过对检测工具能检测出的漏洞进行统计来比较检测工具的检测能力,并通过实验比较检测工具的检测效率。

(1)漏洞检测不完全,检测工具能检测出的漏洞类型不全面。

从表 3 可以看出,目前的检测工具对漏洞种类的覆盖面是比较窄的,纵向比较,检测工具最多能检测出 7 种漏洞。重入漏洞是大多数工具都能检测出的一类漏洞,但有一些漏洞是无法被检测工具检测出来的。整体来看,大部分检测工具都重点关注编程语言层面的漏洞检测,由于编程语言层面的漏洞与开发者编程规范密切联系,因此开发人员需要对编程语言特性、漏洞特征了如指掌,加强安全编程能力。现有工具大多只能检测出一些低级别、攻击逻辑较为简单的漏洞,难以发现需要不同合约间调用产生的安全问题。

表 3 漏洞检测能力的比较

Table 3 Comparison of vulnerability detection capability

漏洞分类	漏洞名称	研究工具											
		[26]	[27]	[29]	[31]	[28]	[33]	[32]	[39]	[40]	[41]	[42]	[43]
编程语言	严格 balance 比较	√	×	×	×	×	×	×	×	×	×	×	×
	未检查返回值	√	×	√	×	√	×	×	×	×	√	×	×
	拒绝服务	√	√	×	×	×	×	×	√	×	√	×	×
	整数溢出	√	×	√	√	×	×	×	×	×	×	×	√
	tx. origin 依赖漏洞	√	×	√	×	√	×	×	×	×	×	×	×
	贪婪合约	×	√	×	×	×	√	×	×	√	×	×	×
虚拟机	重入漏洞	√	√	√	×	√	√	√	√	×	√	√	×
	危险的代理调用	×	×	×	×	×	×	×	×	×	×	×	×
	短地址攻击	×	×	×	×	×	×	×	×	×	×	×	×
区块链	区块状态依赖	√	√	√	×	×	×	×	√	×	√	×	×
	交易顺序依赖	×	×	√	×	×	×	×	√	×	√	×	×

(2)检测效率低,检测工具存在较高的误报率和漏报率,正确率比较低。

本文对能够检测出较为全面的漏洞类型的开源工具进行实验比较,选择了 5 个检测工具,分别为 Smartcheck[26],Slither[27],Securify[33],Oyente[39] 和 ContractFuzzer[41]。实验数据集采用 Chen 等[44]人工验证且公开的数据集,从中选取了 150 个合约源码样本,具体漏洞数量如表 4 所列。实验对未检查返回值、拒绝服务、tx. origin 依赖漏洞、重入漏洞、区块状态依赖这 5 种漏洞,从真阳性(TP)、假阳性(FP)和假阴性(FN) 3 个方面统计检测结果,具体如表 5 所列。

表 4 脆弱性合约数量

Table 4 Number of vulnerable contracts

漏洞名称	漏洞合约数/合约数量
未检查返回值	22/150
拒绝服务	5/150
tx. origin 依赖漏洞	3/150
重入漏洞	11/150
区块状态依赖	42/150

表 5 漏洞检测的实验结果比较

Table 5 Comparisonal experimental results of vulnerability detection

漏洞名称	评估指标	检测工具				
		[26]	[27]	[33]	[39]	[41]
未检查返回值	TP	2	1	1	16	0
	FP	10	12	8	5	0
	FN	19	11	21	6	12
拒绝服务	TP	0	—	0	—	0
	FP	2	—	1	—	0
	FN	5	—	5	—	5
tx. origin 依赖漏洞	TP	0	—	—	—	—
	FP	0	—	—	—	—
	FN	3	—	—	—	—
重入漏洞	TP	4	10	0	2	1
	FP	1	0	0	3	2
	FN	7	12	11	9	10
区块状态依赖	TP	3	—	0	11	2
	FP	0	—	0	6	0
	FN	39	—	42	31	40

Smartcheck[26] 和 Slither[27] 使用静态程序分析技术,两者能够检测出的漏洞种类较多,但前者比后者的检测准确度更高。Securify[33] 基于程序验证方法对重入漏洞检测的效果最佳。对于应用动态分析方法的工具 ContractFuzzer[41] 和 Oyente[39],前者检测能力低于后者。Oyente[39] 能更好地检测出未检查返回值和区块链状态依赖这两类漏洞。但对于 tx. origin 依赖漏洞和拒绝服务这两种漏洞,现有的检测工具都不能很好地检测出来。总体来看,检测工具正确报告漏洞的能力仍然比较薄弱。

4.3.2 检测技术比较

从是否运行合约的角度,智能合约漏洞检测工具可分为静态检测工具和动态检测工具。相对来说,目前的大多数检测工具都是在不运行程序的情况下,使用静态分析方法或者使用程序证明来验证程序不存在某种漏洞,而动态检测方法是近几年才开始受到关注。静态检测工具能够达到比较高的覆盖率,从而检测出更多的漏洞,但目前基于形式化验证和静态分析的检测工具存在误报率高、检测准确率不高的问题,需要耗费大量的资源进行人工验证。使用动态检测方法,能够发现静态检测方法无法检测的漏洞,如需要多个合约间调用才能触发的漏洞,这类逻辑较为复杂的漏洞无法通过静态检测工具检测出来。但目前动态检测工具的覆盖率并不高,有些漏洞需要特定的交易序列才能被触发,因此如何多角度生成测试用例进而提高覆盖率是动态检测工具需要解决的难题。绝大部分研究工作的分析对象是字节码,有部分工具虽然可以将源码作为输入,但其实还是将合约源码编译为字节码作为分析对象。智能合约以字节码形式存储在区块链系统中,相较于合约源码字节码更易获取,并且在以太坊系统中只有 1%左右的合约源码是公开的[45]。所以基于字节码的分析得到了大量的研究,但由于字节码失去了很多语义信息,且对字节码进行反编译时存在很多困难和问题,所以以字节码为研究对象的检测工具利用率较低。研究展望区块链的安全性存在 4 个方面的威胁[46],其中应用程序错误是比较常见并且易于检查和解决的。智能合约作为区块链上可编程的应用程

序,有很大的不可预知性,并且智能合约较传统软件拥有更大的价值以及合约不可更改等特征,存在更大的风险。智能合约中存在大量重复代码[47-48],因此提高合约的安全性有利于智能合约的后续开发。对提高智能合约的安全性,可以从对智能合约本身实行动态保护,提高检测工具的效率等方面进行改善。

(1)动态保护

易被攻击的合约往往是极小部分[49],对于智能合约的保护可以重点关注被攻击的漏洞,针对漏洞原因,使用技术手段对智能合约进行保护。如文献[50]在不影响合约正常运作的情况下,通过字节码重写部署过滤器,将可能引发漏洞的不良输入过滤掉。对已部署的合约进行动态保护,这是一种比较新颖的保护方式,合约开发者了解到更多漏洞的引发机制,才能更好地对症下药。

(2)提高检测效率

目前的检测工具比较多,但难以被普遍利用,并且检测效率不高,能够检测出来的漏洞也较少。大多数检测工具基于合约的控制流图进行半动态检测,没有考虑到多次合约间的调用,导致漏洞检测效率低。Wesley[51]提出使用机器学习长短时记忆模型对智能合约漏洞进行顺序学习,提高了发现漏洞的效率。现有检测工具大多基于比较传统的检测分析方法,需要专家知识预定义检测模型,导致工具局限性较高、扩展性差等问题,将传统检测方法与机器学习相结合能在一定程度上提高检测工具的通用性,进一步提高漏洞检测工具的效率。

(3)动静态结合

通过对检测工具进行比较,可以发现当前检测工具的技术手段比较单一,检测能力比较薄弱。静态检测技术有较高的误报率,动态检测技术有较高的漏报率。使用动静结合的检测技术,取长补短,可以实现更好的检测效果。有些漏洞需要特定的交易序列才能被触发,因此使用静态分析方法指导生成测试用例进行动态分析,可以提高代码覆盖率,从而提高漏洞检测能力。

结束语 在区块链 2.0 时代,以太坊是比较流行的区块链,并且也是首个支持部署智能合约的平台。本文基于以太坊平台对智能合约运作环境、生命周期以及安全漏洞进行概述,并且从是否运行合约的角度对静态检测和动态检测工具进行了描述和比较,总结了现有合约漏洞检测的研究进展,并提出了合约漏洞检测面临的挑战以及可以提高智能合约安全性的研究方向。

参 考 文 献

[1] SCHÄR F. Decentralized finance:On blockchain-and smart contract-based financial markets[J]. FRB of St. Louis Review, 2021,103(2):153-174.

[2] MOOSAVI J,NAENI L M,FATHOLLAHI-FARD A M,et al. Blockchain in supply chain management:a review,bibliometric, and network analysis[C]// Environmental Science and Pollution Research. 2021:1-15.

[3] JIANG Y,ZHONG Y,GE X. Smart contract-based data commodity transactions for industrial Internet of Things[J]. IEEE Access,2019,7:180856-180866.

[4] LI Q,WANG L. Research on the information sharing in the linkage between manufacturing and logistics industry based on blockchain[J]. Journal of Physics,2021,1774(1):012055.

[5] AL-JOBOURY I M,AL-HEMIARY E H. Automated Decentralized IoT Based Blockchain Using Ethereum Smart Contract for Healthcare[C]// Enhanced Telemedicine and e-Health: Advanced IoT Enabled Soft Computing Framework. 2021: 179-198.

[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):1-7.

[7] BUTERIN V. Critical update re:Dao vulnerability[OL]. (2016-06-17). https://blog. ethereum. org/2016/06/17/critical-update-re-dao vulnerability/.

[8] The Multi-sig Hack:A Postmortem. Blockchain Infrastructure for the Decentralised Web[OL]. https://www. parity. io/the-multi-sig-hack-apostmortem/,Jul. 2017.

[9] KASHISYN D. A Postmortem on the Parity Multi-Sig Library Self-Destruct[OL]. (2017-09-15). https://www. parity. io/a-postmortem-on-the-parity-multi-sig-library-self-destruct.

[10] LAUMEISTER M. BitListen,2019[OL]. https://www. bitlisten. com/.

[11] WOOD G. Ethereum:A secure decentralised generalized transaction ledger [OL]. https://gavwood. com/paper. pdf.

[12] CHEN W L,ZHENG Z B. Blockchain Data Analysis:A Review of Status,Trends and Challenges[J]. Journal of Computer Research and Development,2018,55(9):1853-1870.

[13] FENG X,WANG Q,ZHU X,et al. Bug searching in smart contract[J]. arXiv:1905. 00799,2019.

[14] LIU J,LIU Z. A survey on security verification of blockchain smart contracts[J]. IEEE Access,2019,7:77894-77904.

[15] NI Y D,ZHANG C,YIN T T. A Survey of Smart Contract Vulnerability Research[J]. Journal of Cyber Security,2020,5(3): 78-99.

[16] LÓPEZ V A,CASTEDO A T,SANDOVAL O A L,et al. Smart Contracts:A Review of Security Threats Alongside an Analysis of Existing Solutions[J]. Entropy,2020,22(2):203.

[17] DEMIR M,ALALFI M,TURETKEN O,et al. Security smells in smart contracts[C]// 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE,2019:442-449.

[18] SZABO N. Formalizing and Securng Relationships on Public Networks[J]. First Monday,1997,2(9):1-21.

[19] DANNENC. Solidity Programming[M]//Introducing Ethereum and Solidity. Berkeley,CA:Apress,2017:69-88.

[20] Vyper—Vyper documentation [OL]. https://vyper. readthedocs. io/en/ latest/.

[21] Idris | A Language with Dependent Types [OL]. https://www. idris-lang. org/.

[22] Rust | A Language with Dependent Types [OL]. https://www. rust-lang. org/.

[23] 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.

[24] DIKA A. Ethereum smart contracts:Security vulnerabilities and

security tools[D]. Trondheim: Norwegian University of Science and Technology, 2017.
[25] CAI J, ZHOU P, HE J, et al. A software vulnerability detection method based on static analysis and dynamic symbolic execution [J]. Computer Engineering & Science, 2016, 38(12): 2536-2541.
[26] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. Smartcheck: Static analysis of ethereum smart contracts [C]// Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. 2018: 9-16.
[27] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]// 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019: 8-15.
[28] BRENT L, JURISEVIC A, KONG M, et al. Vandal: A scalable security analysis framework for smart contracts [J]. arXiv: 1809.03981, 2018.
[29] KALRA S, GOEL S, DHAWAN M, et al. ZEUS: Analyzing Safety of Smart Contracts[C]// Ndss. 2018: 1-12.
[30] GURFINKEL A, KAHSAI T, KOMURAVELLI A, et al. The SeaHorn verification framework[C]// International Conference on Computer Aided Verification. Cham: Springer, 2015: 343-361.
[31] 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.
[32] CHANG J, GAO B, XIAO H, et al. sCompile: Critical path identification and analysis for smart contracts [C]// International Conference on Formal Engineering Methods. Cham: Springer, 2019: 286-304.
[33] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: Practical security analysis of smart contracts [C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 67-82.
[34] PARK D, ZHANG Y, SAXENA M, et al. A formal verification tool for Ethereum VM bytecode[C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 912-915.
[35] BHARGAVAN K, DELIGNAT-LAVAUD 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.
[36] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. A semantic framework for the security analysis of ethereum smart contracts[C]// International Conference on Principles of Security and Trust. Cham: Springer, 2018: 243-269.
[37] SIDNEY A, MYRIAM B, MAKSYM B, et al. Towards Verifying Ethereum Smart Contract Bytecode in Isabelle/HOL[C]// Proceedings of the 7th ACM International Conference on Certified Programs and Proofs (CPP 2018). 2018.
[38] LI Z J, ZHANG J X, LIAO X K, et al. Survey of Software Vulnerability Detection Techniques[J]. Chinese Journal of Computers, 2015, 38(4): 717-732.
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] GAO J, LIU H, LIU C, et al. Easyflow: Keep ethereum away from overflow[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2019: 23-26.
[44] CHEN J, XIA X, LO D, et al. Defining smart contract defects on ethereum[J]. arXiv: 1905.01467, 2020.
[45] FROWIS M, BOHME R. In code we trust? Measuring the control flow immutability of all smart contracts deployed on Ethereum[J]. LNCS, 2017, 10436: 357-372.
[46] SAYEED S, MARCO-GISBERT H, CAIRA T. Smart Contract: Attacks and Protections [J]. IEEE Access, 2020, 8: 24416-24427.
[47] CHEN X, LIAO P, ZHANG Y, et al. Understanding Code Reuse in Smart Contracts[C]// 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2021: 470-479.
[48] PIERRO G A, TONELLI R. Analysis of Source Code Duplication in Ethreum Smart Contracts[C]// 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2021: 701-707.
[49] PEREZ D, LIVSHITS B. Smart contract vulnerabilities: Does anyone care? [J]. arXiv: 1902.06710, 2019.
[50] WANG Z, DAI W, CHOO K K R, et al. FSFC: An input filter-based secure framework for smart contract[J]. Journal of Network and Computer Applications, 2020, 154: 102530.
[51] TANN W J W, HAN X J, GUPTA S S, et al. Towards safer smart contracts: A sequence learning approach to detecting security threats[J]. arXiv: 1811.06632, 2018.

TU Liang-qiong, born in 1996, postgraduate. Her main research interests include smart contract security and so on.

SUN Xiao-bing, born in 1985, Ph.D, professor, is a senior member of China Computer Federation. His main research interests include software analysis, maintenance and evolution.