引用格式:闫凯伦,刁文瑞,郭山清. 智能合约安全漏洞及检测技术综述[J]. 信息对抗技术,2023, 2(3):1-17.[YAN Kailun, DIAO Wenrui, GUO Shanqing. A survey of smart contract vulnerabilities and detection techniques[J]. Information Countermeasure Technology, 2023, 2(3):1-17. (in Chinese)]

智能合约安全漏洞及检测技术综述

闫凯伦[1,2],刁文瑞[1,2*],郭山清[1,2]

(1. 山东大学网络空间安全学院,山东青岛 266237;
2. 山东大学密码技术与信息安全教育部重点实验室,山东青岛 266237)

摘 要 智能合约是去中心化生态中的重要组件,它降低了多方合作的信任成本,因而广泛应用于数字货币和金融等领域。智能合约在区块链上自动执行,具有不可修改和不可中止的特性,合约常常持有大量数字资产,一旦存在漏洞就有可能会造成巨大损失。随着智能合约技术的发展,合约漏洞开始从简单的语法漏洞向复杂的逻辑漏洞转变,触发漏洞的条件也可能从单一的交易演变为特定的交易序列。目前,各种针对合约的攻击层出不穷,因此开发出有效的合约漏洞检测工具显得尤为重要。为此,首先介绍了 11 个著名的智能合约漏洞;然后从静态分析和动态分析 2 个方面介绍了 21 个合约漏洞检测技术和工具,并从检测方法、研究对象、检测能力等方面对比这些工具,讨论了它们的优点和不足;最后,结合当前合约的安全现状展望了未来的研究工作。

关键词 智能合约;区块链;漏洞检测;自动化工具

中图分类号 TP 311 **文章编号** 2097-163X(2023)03-0001-17

文献标志码 A **DOI** 10.12399/j.issn.2097-163x.2023.03.001

A survey of smart contract vulnerabilities and detection techniques

YAN Kailun[1,2], DIAO Wenrui[1,2*], GUO Shanqing[1,2]

(1. School of Cyber Science and Technology, Shandong University, Qingdao 266237, China; 2. Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao 266237, China)

Abstract As crucial components of decentralized ecosystems, smart contracts can reduce the trust cost of multi-party cooperation, so they have been widely applied in fields of digital currency, finance, etc. Smart contracts are non-censorship, immutable, and automatically executed on the blockchain. Contracts often hold a large number of digital assets, which may cause huge losses once they are breached. With the development of smart contracts, vulnerabilities have changed from simple syntax errors to complex logic problems. The trigger conditions have also evolved from a single transaction to a specific transaction sequence. At present, there are endless attacks against contracts, so it is particularly important to develop effective contract vulnerability detection tools. Therefore, in this paper, eleven well-known smart contract vulnerabilities were introduced and twenty-one vulnerability detection tools were investigated. These investigated detection tools were compared from the aspects of static analysis, dynamic analysis, detection methods, research objects, capabilities, etc. and their

收稿日期:2023-01-10 **修回日期**:2023-02-04
通信作者:刁文瑞,E-mail:diaowenrui@link.cuhk.edu.hk
基金项目:山东省泰山学者青年专家项目(tsqn202211001)

strengths and weaknesses were also discussed. Finally, the future trend of the smart contract was prospected based on current research works.

Keywords smart contracts; blockchain; vulnerability detection; automated tools

0 引言

区块链技术的应用正在迅速改变金融、物流、医疗等领域的运作方式[1],它可以为这些行业提供更高效、更透明、更安全的解决方案,从而打破传统中心化机制下的瓶颈和难题。作为一个去中心化的分布式数据库,区块链利用了对等网络、密码学、工作量证明等机制,使得每个参与者都持有一个相同的副本,从而保证了数据的一致性和安全性。以太坊[2]作为目前第二大加密货币,市值超过了 5 000 亿美元。以太坊是一个开源的区块链平台,它对比特币进行了拓展,实现了图灵完备的智能合约编程。智能合约部署和运行在区块链上,它可以根据预先设定的规则自动执行任务,并将结果永久地保存在链上。得益于区块链,智能合约具有去中心、易于审计、不可篡改等优点,不仅可以降低仲裁成本、避免欺诈损失,而且使得参与者无需可信的第三方介入就可以实现多方合作。

基于智能合约技术的发展,以太坊上涌现出许多新概念和新应用。去中心化自治组织(decentralized autonomous organization, DAO)通过智能合约技术实现了整个组织在没有中心化管理者的情况下运作。去中心化应用(decentralized application, DApp)可以在保护个人隐私和数据安全的前提下提供各种服务。在数字货币领域[3-5],智能合约在保证安全性的前提下大大降低了发行加密货币的门槛,任何人都可以通过 ERC-20 合约发布自己的代币(token)。在数字资产领域[6-8],非同质化代币(non-fungible token, NFT)合约可以用来管理各种数字资产,例如虚拟物品、游戏道具、音乐作品、艺术品等。2022 年,NFT 市场突破 24 亿美元[9]。NFT 提供了一种安全可靠的方法来交易数字资产,也为艺术家和其他创作者提供了新的平台来出售他们的作品。去中心化金融(decentralized finance, DeFi)[10-11]打破了传统金融机构的限制,DeFi 合约根据既定的规则自动处理交易,任何人都可以通过 DeFi 进行借贷、储蓄、投资等金融活动,而不需要依赖券商、交易所或银行。截至 2022 年底,整个 DeFi 市场市值超过了 397 亿美元[12]。

随着智能合约的广泛应用,合约开始频繁遭受攻击,其中最著名的是 DAO 攻击[13]。该攻击导致 360 万个以太币被盗,占到当时以太币发行总量的 14%,最终以太坊不得不进行分叉来降低该攻击的影响。据不完全统计,仅 2021 年就发生 231 起区块链安全事件,其中 170 起和智能合约直接相关[14],DeFi 合约因为安全漏洞损失超过 13 亿美元[15]。2022 年 3 月,Ronin 合约遭受攻击[16],价值 6.25 亿美元的加密货币被盗,此次损失超过了 2021 年 8 月的 Poly Network 的 6.1 亿美元[17],成为目前因为安全漏洞损失最大的 DeFi 合约。智能合约容易遭受攻击主要源于以下 4 个原因:首先,合约一旦部署到区块链后就无法修改,即使发现漏洞,也无法按照传统软件更新的方式进行修补和重新部署;其次,区块链是开放的,任何人都可以匿名调用链上的智能合约,这意味着没有有效的手段来追踪攻击者;再次,为了取得参与者的信任,合约通常会开源代码,这进一步方便了攻击者寻找漏洞;最后,大部分合约都缺少第三方的安全审计,有报告指出,在被攻击的合约中,仅有 43%的合约经过了审计[18]。

区块链是智能合约执行的载体,智能合约拓展了区块链的功能,构建了丰富的去中心化服务。然而,与传统软件不同,智能合约部署到区块链后无法通过暂停服务、重新部署或在线补丁等方式修复漏洞。除非采取极端措施,例如区块链分叉,否则合约无法中止或修改。因此,在部署前对合约的安全性进行广泛的测试和审计至关重要。智能合约的安全问题一直是研究热点,相关学者已经提出了许多检测技术和工具[19-23]。其中,Naga 是一个自动化工具[24],它可以检测合约中由权限控制引发的中心化安全风险。QIAN 等[25]介绍了 5 类主要的合约漏洞检测技术,并对比了这些方法的可检测类型、准确率以及效率等。TU 等[26]则详细介绍了目前较为流行的 16 种检测工具的技术原理,对比了这些检测工具的优缺点。LI 等[27]系统性地回顾了区块链系统的

安全性，分析了以太坊中智能合约的安全问题。ATZEI 等[28]调研了智能合约已发生的安全攻击和未来可能面临的挑战。然而，由于智能合约技术更新迭代非常快，许多工作往往忽视了智能合约的实际发展情况。因此，本文针对现有的智能合约研究工作，结合当前合约技术的发展现状，总结了智能合约面临的安全风险和应对手段，系统地介绍了合约漏洞检测工具。

1　智能合约概述

智能合约概念最早可以追溯到 1994 年，由 NICK 提出[29]。以太坊是首个支持智能合约运行的平台，也是目前最大的去中心化平台。图 1 是以太坊的架构，由图可见，以太坊中每个账户包括交易数量（nonce）、余额（balance）、存储的哈希值（storage hash）和代码哈希值（code hash）4 个部分。每个区块头中存储着 3 个梅克尔树（Merkle tree）的树根，即状态树、交易树、收据树，具体的数据保存在数据库中。梅克尔树利用哈希的单向性，在密码学上保证了数据的不变性。智能合约通常由 Solidity[30]等高级语言编写，然后编译为字节码运行在以太坊虚拟机上。下面将从合约的运行环境和合约编程 2 个方面介绍背景知识。

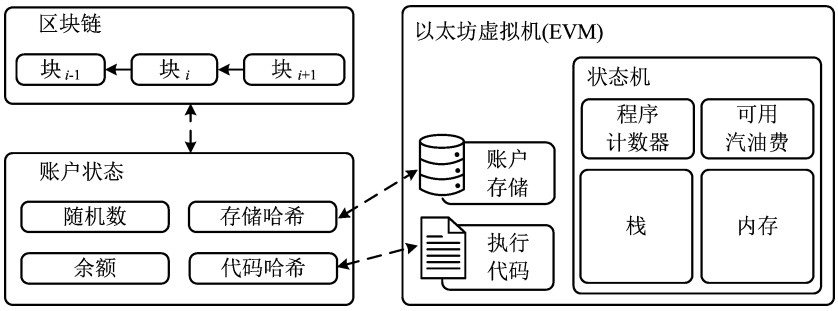

图 1　以太坊架构

Fig. 1　Ethereum architecture

1.1　以太坊虚拟机

以太坊虚拟机（Ethereum virtual machine，EVM）是智能合约的运行环境，它是一个完全隔离的沙盒，在 EVM 中运行的代码无法访问网络、文件系统或其他进程。狭义上，EVM 是指以太坊虚拟机中的状态机（machine state），由图 1 可见，它包括程序计数器（program counter，PC）、可用汽油费（gas available）、栈（stack）以及内存（memory）4 个部分。广义上还包括账户存储（account storage）和执行代码（EVM code）。EVM 设计时考虑到了方便密码学计算，因此具有 256 位字宽（32 字节）。

合约开发时需要考虑数据的存储位置，EVM 中数据存储分为 3 种：栈上存储、临时存储和永久存储。作为一个基于栈的虚拟机，EVM 的所有操作都是在栈上进行的，栈上存储是免费的。临时存储占用 EVM 的内存，在合约执行完成后释放。永久存储则是将数据存储到区块链上，需要调用者支付一定的费用。合约调用本质上和普通的转账交易是一样的，转账金额可以为 0。EVM 会检测目标账户中是否存在可执行的字节码，如果存在，则通过 EVM 中的解释器加载和执行字节码。由于每条指令的运行都需要消耗计算资源，所以每笔交易都需要收取一定量的手续费。以太坊使用汽油费（gas）的数量来衡量每条指令消耗的资源。调用者需要支付以太币购买汽油费，再把汽油费作为奖励分给矿工。用户可以指定他们愿意为执行交易支付的汽油费单位价格和最大限额（gas limit），最终支付的手续费为实际使用的汽油费乘以汽油费单价。多余的手续费会退还给用户，但如果最大汽油费限额不够，则抛出汽油耗尽异常（out-of-gas）。合约执行出现异常时，EVM 会回滚已执行的状态，同时为了防止 DoS 攻击，已经消耗的汽油费不会返还给合约调用者。

以太坊中有 2 种账户：外部账户（external account）和合约账户（contract account），他们共用一个地址空间，每个账户是长度为 20 字节的地址。外部账户的地址由私钥产生，合约账户地址是在合约创建时根据创建者的地址以及该地址的交易数量（nonce）确定的。从 EVM 视角来看，外部账户和合约账户不存在任何区别，都存在以

太币余额和一个键值存储，只是合约账户中存在可执行字节码。合约的执行是通过交易发起的，EVM 会自动检查账户中是否存在代码并自动执行。执行完成后，EVM 会将执行结果（即合约状态）更新到对应的账户存储中。

1.2 智能合约编程

智能合约通常由高级语言编写，例如 Solidity[30]、Rust[31]、Vyper[32]、Move[33]。高级语言编写的源代码需要先编译成字节码，编译后会同时产生合约的应用二进制接口（application binary interface，ABI）。然后，开发者通过交易的方式将字节码部署到以太坊网络。ABI 是从区块链外部与合约进行交互以及合约与合约间进行交互的一种标准方式。

以太坊征求意见稿（Ethereum request for comments，ERC）是以太坊开发者提交的协议提案。ERC 中定义了技术要求，描述了合约开发必须遵循的规范。ERC 后面的数字是提案的编号，目前常见的 ERC 标准是 ERC-20、ERC-721 以及 ERC-1155。ERC-20 定义了可替换代币标准，允许开发者在以太坊网络上创建自己的代币；ERC-721 是不可替代代币标准，在 ERC-721 合约中，每个代币都是独一无二的，因而被广泛应用于数字藏品领域；ERC-1155 将 ERC-20 和 ERC-721 这 2 种标准相结合，同时支持可替换和不可替换代币，与此同时，ERC-1155 还支持批量转移，显著降低了交易费用。

Solidity 是目前最流行编程语言，它是一种基于 ECMAScript 语法的面向对象编程语言。在 Solidity 中，合约类似于面向对象编程语言中的“类”。每个合约包含状态变量、函数、函数修饰器、事件等。Solidity 是静态类型的，支持继承、库和复杂的用户定义类型以及其他功能。Solidity 中的变量类型和普通编程语言类似，不同的是开发者在定义变量时需要考虑存储的位置。此外，Solidity 提供了单位变量和全局变量。单位包括以太币单位（如 wei、ether）和时间单位（如 second、year）。全局变量主要提供合约和区块链的信息，例如合约可以调用 msg. sender 获取当前合约地址，调用 block. timestamp 获取当前区块时间戳等。

合约可以互相调用，分为 CALL 和 DELEGATECALL 2 种调用方式。前者在调用中会把上下文切换到目标合约中，而后者调用的代码会运行在当前合约的环境中，这意味着合约可以从其他地址动态加载代码，实现库的功能。合约调用为开发提供了便利，但也带来了额外的安全风险，例如外部合约可能消耗资源过多导致合约运行失败，或者引发重入漏洞。合约的每次调用都会增加栈的深度，为了避免调用产生死循环，EVM 将栈的执行深度限制为 1 024，从而对合约的调用次数进行了限制。

由于合约编程和传统编程存在差异性，导致合约往往会出现开发者预期之外的行为，产生安全隐患。例如，合约在转账时，如果目标地址是一个合约，且存在执行代码，则可能耗尽汽油费并且抛出异常，导致转账失败。针对这些问题，Solidity 提供了例如提款模式等一些通用编程模式来限制访问状态机。这些模式有助于规范编程，减少逻辑漏洞。

2 智能合约漏洞类型

以太坊是一个开放的、匿名的去中心化平台，任何人都可以调用区块链上的智能合约。合约经部署就无法更改，因此一旦遭受攻击，往往会造成巨大损失，并且难以追责。根据合约漏洞引发的原因，可以将其分为 3 个层面：编程层面、虚拟机层面、区块链层面。这 3 个层面涉及的 11 个典型漏洞如表 1 所列。

表 1 智能合约漏洞

Tab. 1 Smart contract vulnerabilities

漏洞分类	漏洞名称	漏洞原因
编程层面	整数溢出	数据类型范围错误
	算术精度	精度丢失
	重入攻击	递归调用
	未处理异常	未检查调用结果
	以太币锁定	访问权限和不安全设置
	合约初始化错误	代码错误或访问权限
	拒绝服务	外部输入或调用不安全
虚拟机层面	tx. origin 漏洞	tx. origin 不能用于鉴权
	短地址攻击	EVM 未检查输入且自动补位
区块链层面	区块链依赖	使用区块数据产生随机数
	交易顺序依赖	交易竞争

2.1 编程层面漏洞

编程层面的漏洞指的是由于开发者失误，导致合约代码中存在错误，包括语法错误、逻辑错

误、数据边界处理错误等。这些错误可能导致合约执行失败或执行出错，从而被攻击者利用。本节结合 Solidity 的开发现状，详细介绍智能合约在编程层面上存在的安全问题。

2.1.1　整数溢出

整数溢出是编程中最常见的漏洞，分为上溢和下溢，加法、减法、乘法都会造成整数溢出。在 Solidity 中，合约通常使用无符号定长的整数类型，从 uint 8 到 uint 256 按 8 位递增。EVM 在整数越界时，会根据数据类型从高位截断，因此可能会产生整数溢出。攻击者可以利用整数溢出绕过合约中的条件限制语句（如 require、if 等）实现超额转账。2018 年 4 月 22 日，知名代币 Beterchip(BEC)遭受攻击[34]，攻击者利用乘法溢出凭空转出了远超发行量的 BEC 代币，引起持有者的恐慌和抛售。此次攻击最终导致 BEC 近 65 亿人民币的市值几乎归零。

应对措施：为了避免合约中的整数溢出，开发者应该在编写合约代码时格外注意数据类型的选择和使用。在使用 Solidity 0.8 及以前版本时，需要主动检查是否存在溢出，或可利用 OpenZeppelin 库提供的安全整数类型 SafeMath。SafeMath 可以确保在数学运算中没有整数溢出，并在检测到可能的溢出时抛出异常。Solidity 在其 0.8 版本之后的版本中加入了溢出检查，如果出现溢出，会直接回滚交易。

2.1.2　算术精度

Solidity 不支持浮点型计算，在整数除法中，小数位会自动取 0 从而导致精度损失。因此，不同的运算顺序可能产生不同的结果。例如，在图 2 中，开发者的本意是根据预先设定的费率 feePercentage，收取 10%的手续费，然而在第 3 行中可以看出，10/100＝0，而不是 0.1，因此手续费的计算结果将始终为 0。

```
1 uint feePercentage = 10;
2 function contribute() payable public {
3     uint fee = msg.value * (feePercentage / 100);
4     /** ... **/
5 }
```

图 2　算术精度

Fig. 2　Arithmetic precision

应对措施：合约中涉及运算时，开发者在编程前，应当明确计算过程和顺序，以确保计算的准确性和一致性，尽可能避免由于顺序不当、精度损失而引起的错误。

2.1.3　重入攻击

2016 年 6 月 17 日，攻击者利用重入漏洞盗取 DAO 合约中价值 5 000 万美元的以太币。这是区块链历史上最严重的一次攻击，直接导致了以太坊的分叉。智能合约可以由外部账户调用，也可以由合约账户调用。每个合约都隐式包含了一个回调函数 fallback()。如果调用的函数不存在或接收到以太币，合约会自动执行 fallback()函数。攻击者通过重写合约中的 fallback()函数，比如让它再次调用转账函数，从而实现重入攻击。目前重入攻击主要存在 3 种形式[35]：跨函数(cross-function)、委托调用(delegated)和基于构造函数(create-based)。图 3 给出了一个存在重入漏洞的示例，withdraw()函数是一个提款函数，用户可以通过调用它取回自己的以太币。合约使用 balances 记录每个账户的余额（第 3 行），withdraw()函数使用 call()方法转移以太币（第 4 行），并且在执行转移操作之后对账户余额进行清零（第 5 行）。攻击者首先调用 withdraw()函数向自己的另一个合约账号转账，并且在执行第 5 行命令之前，利用目标合约中的 fallback()函数多次调用 withdraw()函数。通过重入不断超额取款，直至取走合约账户内全部的以太币。

```
1 mapping(address=>uint) public balances;
2 function withdraw() public{
3     uint256 _amount = balances[msg.sender];
4     payable(msg.sender).call{value:_amount}("");
5     balances[msg.sender] = 0;
6 }
```

图 3　存在重入和未处理异常漏洞的取款函数

Fig. 3　A withdraw() function with re-entrancy and unhandled exception vulnerabilities

应对措施：智能合约中有 transfer()、send()和 call() 3 种转账方法，前 2 种限定消耗 2 300 gas，因此没有多余的汽油费用于执行其他指令，而 call()默认不限制汽油费，这可能导致重入攻击。因此，如果使用 call()函数，应当限定汽油费。此外，合约可以实现互斥锁保护函数代码，从而防止递归调用，例如，在调用其他合约之前设置一个标志位，只有当标志位被清除后才能继续执行合约。互斥锁也可以通过继承 OpenZepplin 中的 ReentrancyGuard 函数并使用 nonReentrant 修饰

符来实现。

2.1.4 未处理异常

Transfer()指令执行失败会抛出异常并回滚交易,而 send()和 call()则会返回一个布尔值,并继续执行后续代码。图 3 中不仅存在重入漏洞,还存在未处理异常,withdraw()函数没有处理第 4 行 call()调用的返回值,如果调用失败(例如汽油费数量不够),第 5 行仍然会将账号的余额清零,从而导致合约状态和实际情况不一致。

应对措施:合约需要主动对 send()和 call()的调用结果进行判断,不能默认调用成功。对于返回结果为 False 的情况,需要进行处理,避免实际情况和合约状态不一致。Solidity 0.8.0 版本以上的编译器会对上述未处理的返回值发出警告,开发者不应该忽视这些警告。

2.1.5 以太币锁定

智能合约拥有自己的余额,可以接收以太币。然而如果合约存在漏洞,则可能会将合约中所有的以太币锁定。2022 年 4 月,一个名为 AkuDreams 的 NFT 项目由于合约存在逻辑错误,锁定了参与拍卖人员的 11 539.5 枚以太币[36]。2017 年,Parity 钱包合约由于漏洞导致价值超过 2.8 亿美元的以太币被锁定在合约中[37]。Parity 合约使用了代理模式将合约的数据存储和业务逻辑分离,用一个合约存储数据,然后通过 DELEGATECALL 指令调用另一个合约来实现业务处理。这种模式可以实现代码复用,从而降低合约的部署费用。然而 Parity 的业务合约存在权限漏洞,攻击者可以获取业务合约的管理员权限,并在获得权限后执行自毁(SELFDESTRUCT)指令,导致 Parity 的存储合约无法执行任何操作,从而将所有的以太币锁定在合约中。

应对措施:合约一旦部署就无法修改,因此合约开发者应当提前设置好安全有效的提币方式。开发者不仅要考虑当前合约的权限设置,还要注意委托调用的合约权限是否安全。

2.1.6 合约初始化错误

在 Solidity 中,合约有 2 种初始化方式:构造函数和自定义初始化函数。在小于 0.4.22 版本的 Solidity 中,构造函数的名称应该和合约名称一致。在开发过程中,如果构造函数名称写错,那么原本的构造函数就会变成人人都可以调用的普通函数。MorphToken 合约就曾因为构造函数大小写不一致问题,致使合约存在所有权丢失的风险[38]。在 0.4.22 版本之后的版本中,合约使用了 Constructor 作为构造函数的名称。然而一些合约为了避免处理复杂的构造函数参数,使用了自定义的 Initialize 函数对合约进行初始化。然而,Initialize 函数跟普通函数并无本质区别,并不能像 Constructor 一样只能执行一次,开发者如果没有正确设置函数的访问权限,则 Initialize 函数可能可以被任何人再次调用。例如 2021 年 3 月 9 日,DODO 的合约遭遇攻击[39],造成约 200 万美元损失,原因在于初始化函数没有校验调用者,并且可以多次调用。

应对措施:Solidity 0.4.22 及更高版本,支持使用 Constructor 关键字来声明构造函数,而无需再和合约名一致,因此可以避免构造函数名称写错的问题。对于自定义的初始化函数,可以使用 Modifier 修饰器来限定函数的访问权限和调用次数。

2.1.7 拒绝服务

合约拒绝服务是指攻击者通过破坏合约中原有的逻辑,消耗以太坊中的汽油费或计算资源,从而使合约在一段时间或永久无法正常执行。针对智能合约的拒绝服务攻击主要有以下 3 种:

1) 基于代码逻辑的拒绝服务攻击。由于合约代码存在漏洞导致合约暂时或永久无法使用。最典型的是当合约中存在对传入的映射或数组循环遍历操作时,由于外部的输入长度是不可控的,攻击者可能传入过大的映射或数组。然而每个以太坊区块都设定了汽油费上限,如果外部传入的参数导致后续的交易运行所需的汽油费超过区块上限,会导致交易一直处于失败状态。

2) 基于外部调用的拒绝服务攻击。这种情况是由于合约没有正确处理外部调用。例如,如果合约依赖于外部函数执行的结果,但合约没有对外部函数执行失败进行处理,此时如果外部调用失败或者由于外部原因而被拒绝时,会导致每次执行交易时都会因为这个失败问题导致交易回滚,合约无法继续执行。

3) 合约 Owner 导致的拒绝服务攻击。一些合约中设置了 Owner 作为管理员角色,Owner 具有很高的权限,例如可以开启或关闭转账功能。如果合约 Owner 的密钥泄露、丢失,则可能让合

约遭受非主观意愿上的拒绝服务攻击。

应对措施：合约的拒绝服务攻击存在多种形式，首先，在编写智能合约时，应当避免出现循环遍历操作，并且需要限制外部传入参数的长度，避免超出可接受范围。其次，在合约中对外部函数调用进行处理时，应当对函数返回值进行检查，确保返回值符合预期。如果外部函数执行失败，需要对失败情况进行适当的处理，例如进行回滚操作，避免导致合约无法继续执行。最后，在权限设置上应当遵循最小权限原则，在 Owner 设置上可以采用多签名策略，避免因单一私钥泄露而引发的风险。

2.2　虚拟机层面漏洞

虚拟机层面的合约漏洞指的是由以太坊虚拟机 EVM 产生的错误。EVM 是以太坊平台上智能合约的执行引擎，如果 EVM 中存在漏洞，则执行的合约也可能受到影响。智能合约通常由高级语言编写，然后编译成字节码在以太坊虚拟机上执行。由于高级编程语言和低级字节码之间存在语义差异，因此智能合约可能会引发安全问题。此外，虚拟机本身的设计缺陷也可能会引发合约漏洞。虚拟机层面的漏洞主要有 2 种：tx. origin 漏洞和短地址攻击，前者是由于开发者错误使用 tx. origin 进行鉴权造成的，后者则利用了虚拟机本身的设计缺陷。

2.2.1　tx. origin 漏洞

一些合约错误地使用 tx. origin 进行鉴权。tx. origin 会递归栈的调用，找到交易调用的最初发起者。因此，使用 tx. origin 进行鉴权时存在钓鱼攻击的风险。图 4 演示了利用 tx. origin 漏洞的钓鱼攻击。如图所示，Alice 拥有一个钱包合约（wallet contract）用于管理自己的以太币，钱包合约中存在一个转账函数 transfer()，使用了 tx. origin 进行鉴权。Bob 发现钱包合约存在漏洞，因此他构造了一个钓鱼合约（phishing contract），并引诱 Alice 调用钓鱼合约的 buy()函数。一旦 Alice 调用了 buy()函数，buy()函数就会自动调用 Wallet. transfer()函数。Bob 可以利用合约最初签名者是 Alice，从而绕过 require(tx. origin == owner)，盗取钱包合约中的以太币。

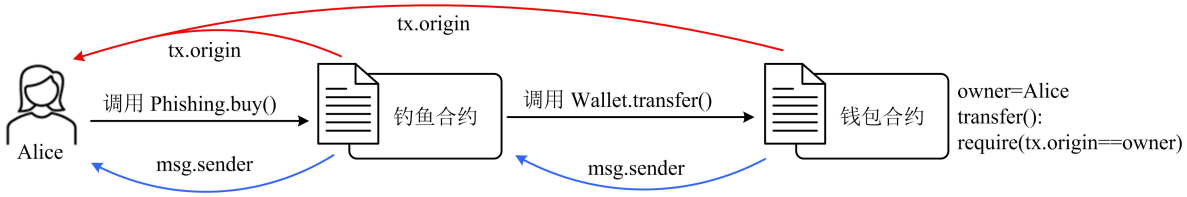

图 4　利用 tx. origin 漏洞的钓鱼攻击

Fig. 4　A phishing attack using the tx. origin vulnerability

应对措施：Solidity 中常用的 2 种验证发送方地址的方式是 tx. origin 和 msg. sender。应优先使用 msg. sender 进行鉴权，而不是 tx. origin。msg. sender 指代的是当前函数调用的直接调用者，而不是交易的最初发起者，因此可以有效避免钓鱼攻击的风险。另外，需要注意在合约调用链中进行鉴权时，应该使用存储在合约中的用户地址信息，而不是使用 msg. sender 进行鉴权，防止被中间合约恶意篡改。

2.2.2　短地址攻击

短地址攻击利用了 EVM 的自动补全机制。例如在调用 ERC-20 合约的 transfer()函数时，正常情况下，需要输入 68 字节的字节码，其中地址和金额 2 个参数相邻，并且都是高位补 0 的 32 字节。然而，EVM 并不会严格校验地址的位置，并且还会对缺失的位数自动补全。调用者可以首先通过碰撞生成一个末几位为 0 的收款地址（假定是 2 位）。然后在调用合约中的转账功能时，故意少写收款地址的后两位，此时，EVM 会自动用后续金额的高位对地址进行补齐。但由于后续金额的位数不足，EVM 会在参数的低位自动补齐，假定原来金额的值为 0x1，自动补齐后就变成了 0x100，扩大了 256 倍。

应对措施：钱包、交易所需要在前端对用户的输入进行校验，检查地址长度，以避免短地址攻击。

2.3　区块链层面漏洞

区块链上的数据是完全透明公开的，任何人都可以访问。此外矿工也拥有一定的权利，例如选择打包的交易顺序和调整区块的时间戳。如果开发者在编写合约时忽略了区块链的这些特性，就可能导致合约存在区块链层面的漏洞。

2.3.1 区块链依赖

合约可以通过 Solidity 全局变量访问到当前区块的相关数据，例如区块号、区块哈希值、区块时间戳等，合约执行时应当避免依赖这些数据。最典型的错误是使用这些数据生成随机数。合约通常是公开的，因此攻击者可以根据随机数的生成算法推测出随机数从而发动攻击。例如 2018 年 11 月 11 日，EOS. WIN 遭受随机数攻击[40]，损失了 20 000 枚 EOS 代币。图 5 演示了一个存在不安全随机数的合约。Victim 合约使用当前区块的难度值 block. difficulty 和时间戳 block. timestamp 作为随机种子生成随机数，同时，bet()函数根据随机数是否为偶数判断参与者是否获胜。由于任何合约都可以访问当前区块的信息，攻击者可以构造一个合约 Attack，使用同样方法生成随机数，并判断当前是否可以获胜。若获胜，则合约会自动调用受害者的 bet()函数下注，否则跳过。事实上，攻击者可以在 attack()函数中设置一个循环，一旦获胜就不断地调用 bet()函数，直到取光受害者合约中所有的以太币。

```
Contract Victim {
    function _getRandom() private view returns(bool){
        uint256 random = uint256(keccak256(abi.
            encodePacked(block.difficulty,block.
            timestamp)));
        uint256 rand = random % 2;
        if(rand == 0){ return false; }
        else { return true; }
    }
    function bet() external payable{
        bool randLucky = _getRandom();
        /** Transfer Ethers to Winners **/
    }
    /****/
}

contract Attack {
    function _getRandom() private view returns(bool){
        //Same as Victim._getRandom()
    }
    function attack(address victim) external payable{
        if (_getRandom()){
            victim.call(abi.encodeWithSignature("bet()"));
        }
    }
}
```

图 5 存在不安全随机数的合约

Fig. 5 A contract with unsafe random numbers

应对措施：由于链上数据是公开的，因此一旦随机数生成算法被公开，其他人就可以提前预知随机数。如果需要随机数，可以从链外的数据源获取随机数，例如使用 Oraclize 外部预言机，这可以确保数据源的可信度。合约执行应当避免依赖链上数据，因为矿工在一定程度上可以调整区块的时间戳，也可以延迟出块，这都会引起安全风险。

2.3.2 交易顺序依赖

矿工在打包交易时，通常会优先选择手续费高的交易。在同一个区块中，如果 2 个交易交换执行顺序会影响最终的结果，那么此合约就存在交易顺序依赖。这让合约的执行存在不确定性，并且可能导致合约执行失败或出现意料之外的结果。例如在一个谜题奖励合约中，合约拥有者可以监听交易池，当发现有参与者提交结果后，立刻发布一个相同的答案或调用合约减少奖励，通过更多的手续费让矿工优先打包到区块中，利用交易顺序依赖漏洞来实现不公平行为。

应对措施：为了避免交易顺序依赖漏洞，合约编写者应该在设计和编写合约时考虑到交易执行的顺序对合约执行结果的影响，并采取适当的措施来确保合约的稳定性和可靠性。

3 智能合约漏洞检测工具

目前，自动化检测工具主要从源代码(source code)和字节码(byte code)2 个方面来分析合约漏洞。源代码可以保留合约的语义信息，并检测出由于开发失误造成的逻辑漏洞，如不安全的随机数、拒绝服务等。字节码分析则更贴近虚拟机层面，更容易检测出合约运行时可能产生的不安全行为，例如以太币锁定。主要的分析方法包括静态分析(static analysis)和动态分析(dynamic analysis)2 种方法。静态分析是指针对合约源代码或字节码，通过模式匹配或语义分析等方法来检测合约中是否存在易受攻击的行为模式；动态分析则是在合约运行时进行漏洞检测，通过输入测试用例来尝试触发漏洞。主要的验证方法包括形式化验证(formal verification)、符号执行(symbolic execution)和模糊测试(fuzzing)。目前，大部分研究工作都是结合了多种分析方法通过优势互补以获得更好的效果，下面从静态分析和动态分析 2 个方面来介绍这些工具。

3.1 静态检测工具

静态分析方法是一种针对合约源代码或字节码进行的漏洞检测方法，通过分析代码的控制流和数据流来检测易受攻击的行为模式，从而发现预定义的漏洞。静态分析的优点在于其易于实现自动化，并且对于给定的合约代码，可以有

效地发现合约漏洞。但是静态分析也有一定的局限性。首先,它不能发现合约执行时动态产生的漏洞,例如,当合约使用外部数据进行计算时,可能会因为恶意数据而产生漏洞,但静态分析往往对此无能为力。其次,静态分析对合约中的动态调用或复杂逻辑分析效果也不佳。

3.1.1　基于规则匹配的 SmartCheck

SmartCheck[41]将合约的 Solidity 源代码解析为 XML,然后使用预定义的 Xpath 来匹配合约是否存在漏洞。由于源代码可以完全翻译为 XML 中间表达,并且 Xpath 可以匹配到所有元素,因此 SmartCheck 达到了代码的全覆盖。虽然 SmartCheck 的检测效率很高,但是受限于匹配的局限性,往往难以检测出复杂逻辑的漏洞。因此 SmartCheck 更多的则是作为对其他检查方法的一个补充。

3.1.2　形式化验证框架 Zeus

Zeus[42]是一个基于抽象解释(abstract interpretation)和符号模型检测(symbolic model checking)的自动形式化验证框架。Zeus 基于 XACML 生成验证规则,并以断言(assert)的形式在源代码中插入违反规则(policy),然后将合约源代码转化为 LLVM 中间语言,最后利用 SeaHorn[43]作为符号执行的后端进行形式化验证。Zeus 验证合约的正确性和公平性。正确性是指基础的安全性编程,公平性是指高级的逻辑正确,即合约可能对拥有者单方面有利,影响参与者的公平性。Zeus 支持用户定义自己合约漏洞检测规则。

3.1.3　基于形式化验证的 Securify

Securify[44]是一个轻量级和可扩展的以太坊智能合约验证器,可以证明合约行为对于给定的属性是否安全。Securify 基于事实规则进行形式化验证,它注意到智能合约中的漏洞常常违反一些简单的属性,而这些简单属性更容易以纯自动化的方式检测。Securify 首先对合约依赖图(dependency graph)进行符号化分析,提取代码中精确的语义信息,然后检查它们的合规性(compliance)和违例模式(violation patterns)。

3.1.4　基于形式化验证的 VeriSmart

VeriSmart[45]是一个高精度的形式化验证工具,可以自动推断智能合约中的不变量,并在验证合约算术安全性时使用它们。VeriSmart 包括 1 个生成器和 1 个验证器,并基于 GEGIS 风格的验证算法来迭代的搜索验证相关的不变量。由于 VeriSmart 注意到了代码中的不变量,因此假阳性很低。图 6 为一个带有不变量合约的示例。由图可见,n 的初始值为 1,函数 add 每次调用都会对 n 进行累加。但第 6 行对 n 的最大值进行了限制,因此第 4 行的断言永远不会因为上溢而失败,而其他工具中往往忽略了这些条件。

```
1 contract ConVar {
2     uint public n = 1;
3     function add() public{
4         assert(n + 1 >= n);
5         n = n + 1;
6         if(n >= 100){ n = 1;}
7     }
8 }
```

图 6　带有不变量的合约

Fig. 6　A contract with invariant

3.1.5　基于污点分析的 Easyflow

Easyflow[46]通过检测字节码来发现合约是否存在溢出。它将合约参数作为污点,追踪数值运算相关操作,通过分析相关污点的传播来识别出安全合约、显性溢出、良好保护的溢出和潜在溢出。Easyflow 具有较少的时间开销,并可以自动生成触发溢出漏洞的交易。

3.1.6　静态分析框架 Osiris

Osiris[47]是一个结合了符号执行和污点分析的静态分析框架,主要用于分析智能合约中的整数错误问题,包括:算术错误(arithmetic bugs)、截断错误(truncation bugs)以及符号错误(signedness bugs)。Osiris 在字节码层面进行分析,它使用符号分析组件构建控制流图,象征性地执行合约的不同路径。然后,符号分析组件将每个指令的执行结果传递给污点分析组件和整数错误检测组件。接着,污点分析组件会在栈、内存和存储之间引入、传播和检查污点,整数错误检测组件则检查执行的指令中是否可能存在整数错误。最后,根据这些结果汇报合约中是否存在整数错误。对比其他工具,Osiris 支持检测更多的整数错误问题,并具有较低的假阳性。

3.1.7　基于关键路径的 sCompile

sCompile[48]通过分析控制流来发现合约的安全问题。首先,根据合约中的函数以及合约间函数的调用生成控制流图。然后,根据控制流生成相应的路径(文中称为有界的函数调用序列)。

为了解决路径爆炸问题，sCompile 将以太币相关的路径设置为关键路径，静态地识别这些关键路径并计算路径长度。接着，根据预先定义的属性来识别所有可能的违反路径。随后，根据路径长度和违反路径 2 个方面来计算每个路径的得分，并根据路径得分进行排序。最后，使用符号执行来依次检查这些路径的可达性，确定是否存在和以太币相关的漏洞。

3.1.8 漏洞生成器 teEther

teEther[49]提出了一种针对智能合约特定漏洞生成输入用例的方法。teEther 在字节码层面分析合约，复现合约的控制流图，然后在控制流图中搜索关键指令，并构建攻击路径。在构建攻击路径的过程中，teEther 利用符号执行将这些路径转为约束，并通过约束求解得到攻击者触发漏洞需要执行的交易序列。

3.1.9 结合静态分析和符号执行的 MPro

MPro[50]关注了符号执行过程中，函数序列组合爆炸问题。一些合约漏洞往往需要一组特定的交易才能触发，而 Depth-n 是指需要调用特定序列的 n 个函数才会触发漏洞。由于函数序列的组合非常多，当 n 较大时，现有的自动化分析器会非常耗时。本文提出应首先对源代码进行静态分析，找到数据的依赖关系，然后再生成有效的函数执行序列，最后使用符号执行进行漏洞检测。MPro 的符号执行工具来自开源工具 Mythril-Classic[51]，因此和 Mythril-Classic 具有相同的精度，但在执行 Depth-n 的漏洞检测时，提高了 n 倍的执行速度。

3.1.10 有界模型检测工具 EthBMC

EthBMC[52]是一个基于符号执行的有界模型检测工具(bounded model checker)，可对以太坊进行精确建模。通过对近年来智能合约静态分析工具的分析，可以发现它们至少在一个方面存在不精确推理。因此本文提出了 EthBMC，通过内存复制式操作、提高合约间通信和引入新的编码方式 3 种方法，实现合约中内存建模、合约调用、加密哈希函数 3 个方面的精确推理，从而取得更高精确度的分析结果。此外，EthBMC 还利用了符号执行来搜索合约的状态空间。

3.1.11 静态分析框架 Slither

Slither[53]是一个 Solidity 静态分析框架，它将合约编译得到的抽象语法树(abstract syntax tree，AST)作为输入，分析出合约的继承图及控制流图，并将合约代码转换为 SlitherIR 中间表示，从而实现高精度的分析。SlitherIR 支持静态单赋值(static single assignment，SSA)形式，因此可以实现精确的污点分析和变量追踪。Slither 根据预定义的检测器，结合变量读写、函数依赖、权限控制等条件来分析常见的合约漏洞。

3.1.12 基于图神经网络的 TMP 和 DR-GCN

ZHUANG 等[54]提出了一种时序消息传播(temporal message propagation，TMP)网络和无度图卷积网络(degree-free graph convalutional network，DF-GCN)来自动检测智能合约漏洞。这种方法将合约源代码中的函数描述为接触图，并对接触图进行显示规范化，以突出关键节点。考虑到代码片段之间丰富的依赖关系，本文探索使用图神经网络进行合约漏洞检测的可能性。

3.2 动态检测工具

动态分析是在合约实际运行时检测漏洞的方法，通常会通过输入测试用例来尝试触发漏洞，目的是尽可能地覆盖合约的代码，以便最大程度地检测到合约中的漏洞。在动态分析过程中，合约会真正地运行，因此相比静态分析，动态分析的准确率更高，但是代价也更大，因为合约运行会花费更多的时间和资源，因此动态分析的效率往往远低于静态分析。

3.2.1 针对重入漏洞的 Sereum

Sereum[35]在 EVM 运行时监控和验证合约，基于 Sereum 修改的 EVM 可以保护已部署的合约避免重入攻击。Sereum 详细讨论了各种重入攻击场景，总结出了跨函数、委托调用和基于构造函数的 3 种重入攻击。Sereum 不需要依赖任何的语义信息，它通过构建动态调用树并为特定变量添加写锁来防止重入攻击。Sereum 基于现有的 EVM，扩展实现了通过一个污点引擎和重入攻击检测器来检测上述 3 种重入漏洞。

3.2.2 基于动态符号执行的 Oyente

Oyente[55]使用动态符号执行来寻找合约中的安全漏洞。它基于合约的字节码构建控制流图，图中的节点为程序的基本块，边为程序块间的跳转关系，并在符号执行过程中进一步完善静态分析中缺少的边关系。Oyente 的探索组件使用 Z3 求解器进行符号执行来通过控制流图中的路径，分析组件根据执行的路径信息分析相应的

漏洞,并使用验证组件过滤掉假阳性。Oyente 没有对以太坊环境进行建模,但支持用户输入外部以太坊的状态信息。由于字节码往往难以反映出原始的语义信息,因此 Oyente 的检测效果有限。

3.2.3　模糊测试工具 ContractFuzzer

ContractFuzzer[56]是基于模糊测试的合约漏洞检测工具,它根据智能合约的 ABI 接口解析出函数参数类型等信息,然后随机生成测试用例。ContractFuzzer 监听 EVM 获取合约的执行日志,并通过分析这些日志的内容来检测合约中存在的漏洞。

3.2.4　针对重入漏洞的 ReGuard

ReGuard[57]是一个用于检测合约重入漏洞的检测工具。它基于源代码的抽象语法树或字节码的控制流图对合约进行翻译,然后迭代的随机生成不同的交易对合约进行模糊测试,跟踪程序运行过程来识别重入漏洞。

3.2.5　基于模仿学习的 ILF

ILF[58]在模仿学习的框架下,通过符号执行生成大量高质量的输入作为数据集。基于生成的数据集使用神经网络训练出模糊策略,用来对其他合约进行模糊测试。实验表明,ILF 具有很高的效率以及良好的代码覆盖率。

3.2.6　基于漏洞生成的 EthPloit

EthPloit[59]是一个基于模糊测试的智能合约漏洞攻击生成器,主要解决执行路径中的硬约束和区块链环境模拟 2 个问题。EthPloit 关注合约层面漏洞,主要包括秘密公开(exposed secret)、未检查的交易值(unchecked transfer value)和脆弱的访问控制(vulnerable access control)3 种漏洞。Ethploit 注意到产生这些漏洞需要:1) 一个关键交易;2) 在关键交易之前执行一组交易来修改合同状态。因此,EthPloit 采用静态污点分析来生成交易序列,采用动态种子策略来传递硬约束(即模糊测试的输入),并通过一个改进的以太坊虚拟机模拟区块链环境。

3.2.7　混合模糊测试工具 ConFuzzius

ConFuzzius[60]是一个结合符号执行和模糊测试的混合模糊测试工具,具有很高的代码覆盖率。ConFuzzius 从 3 个方面对传统模糊测试进行改进:首先,基于遗传算法生成模糊测试的输入值,当传统的模糊测试遇到无法通过的路径时,启用符号执行的约束求解来通过路径,并把求解值放到函数的突变池中。ConFuzzius 注意到合约中写后读(read-after-write)数据依赖特性,并基于此生成特定的交易序列。此外,ConFuzzius 将区块链环境变量和合约返回值都视为可模糊的输入。

3.2.8　灰盒模糊测试工具 Harvey

Harvey[61]是一个工业级的灰盒模糊测试工具,主要关注传统模糊测试中随机输入和智能合约的状态空间探索这 2 个问题,并从输入预测和以需求驱动的交易序列模糊测试 2 个方面来解决这些问题。Harvey 主要检测了断言失败和内存访问错误的问题。在一个小规模的数据集上进行了实验并取得良好的效果。

3.2.9　模糊测试工具 sFuzz

sFuzz[62]是一种自适应自动化模糊测试工具,它将 AFL 模糊测试和一种高效的轻量级自适应种子选择策略相结合。对比其他工具,sFuzz 在效率上快 2 个数量级,并具有很高的代码覆盖率。

3.3　检测工具对比

表 2 列出了本文介绍的 21 种合约漏洞检测工具。从表中可以看到,大部分合约漏洞检测工具都结合了多种方法,总体来说,主要包括以下 6 种方法:

1) 形式化验证。如表 2 所列,有 4 种工具使用了形式化验证。形式化验证将源代码转换为形式化的模型,列举出模型所有可能存在的状态并根据预定义的规则检测漏洞。形式化验证基于定理证明的思想,采用了逻辑公式描述系统和性质,因此它是从数据的角度来证明合约的正确性。形式化验证的优点在于可以实现代码的全覆盖,缺点是它只能按预先定义的规则进行验证,无法检测到特殊情况。

2) 符号执行。在合约漏洞检测中,无论是静态检测还是动态检测,符号执行都是最广泛的使用的方法,如表 2 所列,有 9 个工具使用了符号执行。符号执行使用符号值来替代真实值,针对目标代码,分析出其路径约束,然后通过约束求解得到可以触发目标代码的具体值。然而符号执行面临路径爆炸、内存建模等问题。合约漏洞可能需要特殊的交易序列才能触发,而交易的搜索空间无疑是巨大的。为了解决这个问题,MPro

和 ConFuzzius 都引入了变量依赖，通过变量间的依赖关系生成交易序列，大大减少了搜索空间。此外，符号执行受制于约束求解器，大部分工具都使用了 Z3 求解器。然而约束求解器只能解决线性问题，对于非线性问题，例如哈希函数则无能为力[52]。

表 2 合约漏洞检测工具对比

Tab. 2 Comparison of contract vulnerability detection tools

类型	检测工具	具体分析方法	研究对象	检测的漏洞
静态工具	SmartCheck	形式化验证，中间表示	源代码	3 个安全漏洞和其他问题
	Zeus	形式化验证，符号执行，中间表示	源代码	6 类正确性和公平性漏洞
	Securify	形式化验证，符号执行	源代码	7 类安全漏洞
	VeriSmart	形式化验证	源代码	整数溢出
	Easyflow	污点分析	字节码	整数溢出
	Osiris	符号执行，污点分析	字节码	算术相关的 3 类问题
	sCompile	符号执行	源代码	货币交易相关漏洞
	teEther	符号执行	字节码	以太币恶意转移
	MPro	符号执行	源代码	9 类安全漏洞
	EthBMC	符号执行	字节码	3 类底层漏洞
	Slither	中间表示，语法分析	源代码	多种漏洞，可自定义
	TMP，DF-GCN	图神经网络	源代码	3 类安全漏洞
动态工具	Sereum	污点分析	字节码	3 种重入漏洞
	Oyente	符号执行	字节码	4 类安全漏洞
	ContractFuzzer	模糊测试	源代码，字节码	7 类安全漏洞
	ReGuard	符号执行，模糊测试	源代码，字节码	重入漏洞
	ILF	深度学习，模糊测试	源代码	6 类安全漏洞
	EthPloit	静态污点分析，模糊测试	源代码	3 类安全漏洞
	ConFuzzius	符号执行，模糊测试	字节码	10 类安全漏洞
	Harvey	灰盒模糊测试	源代码	3 种安全漏洞
	sFuzz	模糊测试	源代码	9 类安全漏洞

3）模糊测试。模糊测试是动态工具中的代表性分析方法，它通过向合约输入非预期的值并监视合约状态来发现漏洞。智能合约提供了应用程序二进制接口（ABI），模糊测试工具可以直接根据 ABI 来生成测试用例。然而模糊测试也存在输入空间过大的问题，例如对于一个 if（num==42）语句，符号执行可以轻易得出 num 的值为 42，但对于模糊测试而言，理论上需要尝试 2^{256} 次。因此，许多模糊测试工具[48,57]结合了符号执行来通过复杂路径。还有一些模糊测试工具结合了深度学习[58]、静态污点分析[59]、遗传算法[60]等方法以得到更好的输入值。

4）污点分析。污点分析主要通过标记源代码或字节码中的关键值，分析它们的数据流来判断是否存在漏洞。在合约检测中，既存在静态污点分析[46-47,59]，也存在动态污点分析[35]。静态污点分析主要将合约的输入值作为污染源，动态污点分析则将合约中的特殊变量标记污染源，两者都通过分析污点传播来判断漏洞。污点分析可以精确地追踪变量间的关系，但是它一般只用于检测特定的漏洞，例如在文献[46-47]中，主要用于检测整数溢出，而在文献[35]中只用于检测重入漏洞。

5）中间表示。中间表示是指将合约源代码

转译为中间表示[42,53]，例如 LLVM，然后在中间表示的基础上进行漏洞检测。中间表示的优点可以基本保留原本的代码语义，并且可以优化常量传播，删除重复代码等，而且更容易查找出变量间的依赖关系。

6）深度学习。在漏洞检测中，深度学习主要应用于 2 个方面：一是直接检测，通过将合约中的字节码、控制流图或数据流图进行转化，输入到深度学习模型中进行训练和检测[54]；二是辅助检测，例如 ILF[58] 利用深度学习生成更有效的输入，从而提高模糊测试的效率。深度学习进行漏洞检测效果较好，并且检测效率也比较高，但是缺少可靠的理论依据。

在研究对象上，大部分问题都可以从源代码上进行分析，但是一些特定的问题在字节码上会有更好的效果。例如在货币交易上，源代码分析需要考虑各种实际场景，因此检测效果常常有限。对比而言，字节码可以直接追踪转账相关的指令，从而更精确地分析货币漏洞。在漏洞检测上，一些工具侧重于单个或单类安全问题，主要包括整数溢出、重入漏洞、货币交易 3 个方面。不过，这些单一漏洞检测工具往往方法和侧重点都不同，例如 VeriSmart[45] 和 Osiris[47] 都是针对算术安全问题的，前者基于形式化验证分析合约的 Solidity 源代码，后者基于符号执行和污点分析检测合约的字节码，两者都需要用到约束求解。还有一些工具则面向更广泛的合约安全漏洞，例如 Zeus[42] 和 Slither[53] 等工具还支持用户自定义检测器从而检测更多的漏洞。

4　总结与展望

智能合约拓展了区块链的应用场景，丰富了去中心化生态。智能合约作为去中心化应用的后端，控制了去中心化应用的业务逻辑，管理着参与者的数字资产。开发人员的疏忽往往可能造成巨大的经济损失，因此合约安全性越来越受到研究者的关注。本文系统地介绍了以太坊智能合约的安全漏洞，包括主要的攻击手段和常见的应对措施，列举了相关有影响力的安全事件。基于分析方法、研究对象、检测范围等方面，对比了 21 种最先进的漏洞检测工具，归纳出 6 种最具代表性的合约漏洞检测方法。然而，当前工作也存在一些不足，主要有以下 2 个方面：

1）当前合约技术发展迅速，从最早以 ERC-20 为代表的 Token 合约到现在的 ERC-721、ERC-1155 合约，以及当前最流行的 DeFi 合约，新的应用不断出现。随着智能合约在不同领域的应用，一些新型合约中特有的安全问题开始浮现。除了利用传统的漏洞进行攻击外，还出现了针对 OpenSea 上 NFT 合约的钓鱼攻击[63]，针对 X2Y2 合约可升级漏洞的攻击[64]，利用空投漏洞[65]，以及通过闪电贷攻击 DeFi 合约[66-67] 等。然而，目前大部分的检测工具只关注于以 ERC-20 为代表的代币合约安全问题，缺少对这些新型攻击的研究和关注。

2）研究者和开发者存在脱节。目前最常用的合约编程语言是 Solidity，它的版本更新非常快，不断推出新特性和修复已有问题，几乎每 6～12 个月就会有一次大的更新，截至 2022 年 12 月底，Solidity 的最新版本是 0.8.17。0.8 以上的版本默认进行溢出检查，从根本上解决了整数溢出问题。0.5.7 版本将构造函数的生成方法由 function ContractName() 强制改为 Constractor()，因此由于命名错误产生的构造函数可访问问题也不会再产生，但许多使用静态分析方法的文章中仍在讨论上述问题，许多文章分析合约代码还停留在 0.4 或 0.5 版本。

智能合约作为自动执行的去中心化程序，在许多领域得到了应用，特别是在常规性、重复性的流程中，智能合约可以有效提高效率，降低交易成本，并保证安全性。随着智能合约技术的发展，越来越多的新特性会不断添加到智能合约中，这必将招致更多的安全风险。结合当前的合约发展现状，未来智能合约的研究方向有以下 3 个方面：

1）目前大部分基于源代码的研究工作都选择了 Solidity 编程语言作为研究对象，事实上支持编写智能合约的语言有几十种，例如最近备受关注的 Move 语言。其他一些传统编程语言例如 GO、C、Java、Python 也支持编写智能合约。不同的编程语言的特性和语法规则往往不同，从而可能避免或产生特定的安全漏洞。因此，有必要面向更广泛的编程语言进行合约安全研究。中间表示方法可以将不同的编程语言转译到同一个层面上，这对于对比不同的编程语言间的合约安全漏洞差异具有重要意义。

2）如上文所述，合约的安全问题已开始从低级的编程错误转向更为高级、复杂的逻辑错误，然而当前对于复杂漏洞的关注还远远不够。随着 DeFi 合约的兴起，闪电贷攻击、钱包劫持等问题也不断涌现。虽然 Zeus[42]、Slither[53]等工具可以自定义漏洞检测，但对于复杂的逻辑漏洞，检测效果并不理想，因此研究者有必要扩展现有的工作，并提出新的策略来应对这些新型攻击。另一方面，Osiris[47]、Sereum[35]等单一漏洞检测工具均表现出优秀的检测效果，因此可以针对这些新型攻击开发特定的漏洞检测工具。此外，合约的公平性[42]、中心化安全风险[24]也值得研究者进一步关注。

3）由于合约通常会与其他合约交互，还可能会频繁地调用链下服务，从而导致静态检测产生大量误报，智能合约很难在运行时测试。针对这些问题，EthBMC[52]对区块链进行了建模，Oyente[55]则支持输入区块链的状态。研究者可以考虑采用动静结合的手段以实现优势互补，尝试结合多个层面的不同手段来有效地发现合约漏洞。例如可以通过对合约进行语义提升来检测复杂的逻辑漏洞，对区块链进行建模提高检测的准确率，对字节码分析从而发现低层次的安全问题。

参 考 文 献

[1] NAKAMOTO S. Bitcoin: a peer-to-peer electronic cash system[EB/OL]. (2008-11-01)[2023-02-04]. https://bitcoin.org/bitcoin.pdf.

[2] BUTERIN V. A next-generation smart contract and decentralized application platform[EB/OL]. (2023-01-12)[2023-02-04]. https://ethereum.org/en/whitepaper/.

[3] VICTOR F, LÜDERS B K. Measuring ethereum-based erc20 token networks[C]//Proceedings of International Conference on Financial Cryptography and Data Security. [S. l.]: Springer, Cham, 2019: 113-129.

[4] CHEN W, ZHANG T, CHEN Z, et al. Traveling the token world: a graph analysis of Ethereum erc20 token ecosystem[C]//Proceedings of the Web Conference 2020. [S. l.: s. n.], 2020: 1411-1421.

[5] SHIROLE M, DARISI M, BHIRUD S. Cryptocurrency token: an overview[C]//Proceedings of 2019 International Conference on Blockchain Technology. [S. l.: s. n.], 2019: 133-140.

[6] ANTE L. The non-fungible token (NFT) market and its relationship with Bitcoin and Ethereum[J]. Social Science Electronic, 2022, 1(3): 216-224.

[7] REHMAN W, ZAINAB H E, IMRAN J, et al. NFTs: applications and challenges[C]//Proceedings of the 22nd International Arab Conference on Information Technology (ACIT). [S. l.]: IEEE, 2021: 1-7.

[8] GONSERKEWITZ P, KARGER E, JAGALS M. Non-fungible tokens: use cases of NFTs and future research agenda[J]. Risk Governance and Control: Financial Markets and Institutions, 2022, 12: 8-18.

[9] COINMARKETCAP. Highest price NFT stats[EB/OL]. [2023-02-04]. https://coinmarketcap.com/nft/.

[10] SRIMAN B, KUMAR S G. Decentralized finance (DeFi): the future of finance and DeFi application for Ethereum blockchain based finance market[C]//Proceedings of 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI). [S. l.]: IEEE, 2022: 1-9.

[11] ZHOU L, QIN K, CULLY A, et al. On the just-in-time discovery of profit-generating transactions in DeFi protocols[C]//Proceedings of 2021 IEEE Symposium on Security and Privacy. [S. l.]: IEEE, 2021: 919-936.

[12] COINMARKETCAP. Top DeFi tokens by market capitalization[EB/OL]. [2023-02-04]. https://coinmarketcap.com/view/defi/.

[13] MEHAR M I, SHIER C L, GIAMBATTISTA A, et al. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack[J]. Journal of Cases on Information Technology, 2019, 21(1): 19-32.

[14] MUHAIMIN O. Crypto industry loses $9.8 bn to hacks, ransomware attacks in 2021[EB/OL]. (2021-12-29)[2023-02-04]. https://www.cryptopolitan.com/crypto-industry-loses-9-8bn-to-hacks/.

[15] ELIZA G. Funds lost to DeFi hacks more than doubled to $1.3 B in 2021: Certik[EB/OL]. (2022-01-13)[2023-02-04]. https://www.coindesk.com/business/2022/01/13/funds-lost-to-defi-hacks-more-than-doubled-to-13b-in-2021-certik/.

[16] YANG Z, MAN G, YUE S. Understanding security audits on blockchain[C]//Proceedings of the 5th International Conference on Blockchain Technology and Applications. [S. l.: s. n.], 2022: 10-15.

[17] TOMMASO G. The Poly network Hack explained[EB/OL]. (2021-08-12)[2023-02-04]. https://research.kudelskisecurity.com/2021/08/12/the-poly-network-hack-explained/.

[18] SHARKTEAM. The Web3 security quarterly report

of 2022[EB/OL]. (2022-07-26)[2023-02-04]. https://learnblockchain.cn/article/4439.

[19] HEWA T, YLIANTTILA M, LIYANAGE M. Survey on blockchain based smart contracts: applications, opportunities and challenges[J]. Journal of Network and Computer Applications, 2021, 177: 102857.

[20] KHAN S N, LOUKIL F, GHEDIRA-GUEGAN C, et al. Blockchain smart contracts: applications, challenges, and future trends[J]. Peer-to-Peer Networking and Applications, 2021, 14(5): 2901-2925.

[21] WANG Z, JIN H, DAI W, et al. Ethereum smart contract security research: survey and future research opportunities[J]. Frontiers of Computer Science, 2021, 15(2): 1-18.

[22] CHEN H, PENDLETON M, NJILLA L, et al. A survey on Ethereum systems security: vulnerabilities, attacks, and defenses[J]. ACM Computing Surveys, 2020, 53(3): 1-43.

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

[24] YAN K L, ZHANG J L, LIU X Y, et al. Bad apples: understanding the centralized security risks in decentralized ecosystems[C]//Proceedings of the 32nd ACM Web Conference. [S. l.: s. n.], 2023: 2274-2283.

[25] QIAN P, LIU Z G, HE Q M, et al. Smart contract vulnerability detection technique: a survey[J]. Journal of Software, 2022, 33(8): 3059-3085.

[26] TU L Q, SUN X B, ZHANG J L, et al. Survey of vulnerability detection tools for smart contracts[J]. Computer Science, 2021, 48(11): 79-88.

[27] LI X, JIANG P, CHEN T, et al. A survey on the security of blockchain systems[J]. Future Generation Computer Systems, 2020, 107: 841-853.

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

[29] NICK S. Smartcontracts[EB/OL]. (1994-01-01)[2023-02-04]. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[30] DANNEN C. Introducing Ethereum and solidity[M]. Berkeley: Apress, 2017.

[31] NIKHIL B. How to write and deploy a smart contract in Rust[EB/OL]. (2020-11-01)[2023-02-04]. https://learn.figment.io/tutorials/write-and-deploy-a-smart-contract-on-near.

[32] CHATTERJEE K, GOHARSHADY A K, GOHARSHADY E K. The treewidth of smart contracts[C]//Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. [S. l.: s. n.], 2019: 400-408.

[33] FYNN E, BESSANI A, PEDONE F. Smart contracts on the move[C]//Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]: IEEE, 2020: 233-244.

[34] TIAN Z, TIAN J, WANG Z, et al. Landscape estimation of solidity version usage on Ethereum via version identification[J]. International Journal of Intelligent Systems, 2022, 37(1): 450-477.

[35] RODLER M, LI W, KARAME G O, et al. Sereum: protecting existing smart contracts against reentrancy attacks[C]//Proceedings of the 26th Annual Network and Distributed System Security Symposium. San Diego, USA: [s. n.], 2019: 24-27.

[36] WILLIAMS C. AkuDreams faces nightmare as team locked from $34 M takings[EB/OL]. (2022-04-23)[2023-02-04]. https://cryptobriefing.com/akudreams-faces-nightmare-team-locked-from-34m-takings/.

[37] AGGARWAL S, KUMAR N. Attacks on blockchain[M]. Amsterdam: Elsevier, 2021.

[38] ZHANG Z, YIN J, HU B, et al. CLTracer: a cross-ledger tracing framework based on address relationships[J]. Computers & Security, 2022, 113: 102558.

[39] ROB B. Explained: the DODO Dex Hack[EB/OL]. (2021-03-12)[2023-02-04]. https://halborn.com/explained-the-dodo-dex-hack-march-2021/.

[40] SAKO K, MATSUO S, MORI T. Distributed random number generation method on smart contracts[C]//Proceedings of the 4th Blockchain and Internet of Things Conference. [S. l.: s. n.], 2022: 1-10.

[41] 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. [S. l.: s. n.], 2018: 9-16.

[42] KALRA S, GOEL S, DHAWAN M, et al. Zeus: analyzing safety of smart contracts[C]//Proceedings of the 25th Annual Network and Distributed System Security Symposium. California, USA: [s. n.], 2018: 18-21.

[43] GURFINKEL A, KAHSAI T, KOMURAVELLI A, et al. The SeaHorn verification framework[C]//Proceedings of International Conference on Computer Ai-

ded Verification. [S. l.]: Springer, Cham, 2015: 343-361.

[44] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: practical security analysis of smart contracts[C]//Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2018: 67-82.

[45] SO S, LEE M, PARK J, et al. VeriSmart: a highly precise safety verifier for Ethereum smart contracts [C]//Proceedings of 2020 IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2020: 1678-1694.

[46] GAO J, LIU H, LIU C, et al. Easyflow: keep Ethereum away from overflow[C]//Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings. [S. l.]: IEEE, 2019: 23-26.

[47] 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. [S. l. :s. n.], 2018: 664-676.

[48] CHANG J, GAO B, XIAO H, et al. sCompile: critical path identification and analysis for smart contracts [C]//Proceedings of International Conference on Formal Engineering Methods. [S. l.]: Springer, Cham, 2019: 286-304.

[49] KRUPP J, ROSSOW C. TeEther: gnawing at Ethereum to automatically exploit smart contracts[C]//Proceedings of the 27th USENIX Security Symposium. [S. l. :s. n.], 2018: 1317-1333.

[50] ZHANG W, BANESCU S, PASOS L, et al. MPro: combining static and symbolic analysis for scalable testing of smart contract[C]//Proceedings of the 30th International Symposium on Software Reliability Engineering. [S. l.]: IEEE, 2019: 456-462.

[51] CONSENSYS. Mythril-Classic [EB/OL]. [2023-02-04]. https://github.com/ConsenSys/mythril.

[52] FRANK J, ASCHERMANN C, HOLZ T. EthBMC: a bounded model checker for smart contracts [C]// Proceedings of the 29th USENIX Security Symposium. [S. l. :s. n.], 2020: 2757-2774.

[53] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]//Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain. [S. l.]: IEEE, 2019: 8-15.

[54] ZHUANG Y, LIU Z, QIAN P, et al. Smart contract vulnerability detection using graph neural network [C]//Proceedings of the 29th International Joint Conference on Artificial Intelligence and 17th Pacific Rim International Conference on Artificial Intelligence. [S. l. :s. n.], 2020: 3283-3290.

[55] LUU L, CHU D H, OLICKEL H, et al. Making smart contracts smarter [C]//Proceedings of 2016 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2016: 254-269.

[56] JIANG B, LIU Y, CHAN W K. ContractFuzzer: fuzzing smart contracts for vulnerability detection [C]//Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering. [S. l.]: IEEE, 2018: 259-269.

[57] LIU C, LIU H, CAO Z, et al. ReGuard: finding reentrancy bugs in smart contracts[C]//Proceedings of the 40th International Conference on Software Engineering: Companion. [S. l.]: IEEE, 2018: 65-68.

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

[59] ZHANG Q, WANG Y, LI J, et al. EthPloit: from fuzzing to efficient exploit generation against smart contracts[C]//Proceedings of the 27th International Conference on Software Analysis, Evolution and Reengineering. [S. l.]: IEEE, 2020: 116-126.

[60] FERREIRA T C, IANNILLO A K, GERVAIS A. ConFuzzius: a data dependency-aware hybrid fuzzer for smart contracts[C]//Proceedings of 2021 IEEE European Symposium on Security and Privacy. [S. l. :s. n.], 2021:103-119.

[61] 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. [S. l. : s. n.], 2020: 1398-1409.

[62] NGUYEN T D, PHAM L H, SUN J, et al. sFuzz: an efficient adaptive fuzzer for solidity smart contracts [C]//Proceedings of the 42nd International Conference on Software Engineering. [S. l. :s. n.], 2020: 778-788.

[63] DAS D, BOSE P, RUARO N, et al. Understanding security issues in the NFT ecosystem[C]//Proceedings of 2022 Conference on Computer and Communications Security. [S. l. :s. n.], 2022: 667-681.

[64] ROSCOE A W. Specification is law: safe creation and upgrade of Ethereum smart contracts[C]//Proceedings of the 20th International Conference on Software

Engineering and Formal Method. Berlin, Germany: Springer Nature, 2022: 220-227.

[65] HE D, DENG Z, ZHANG Y, et al. Smart contract vulnerability analysis and security audit[J]. IEEE Network, 2020, 34(5): 276-282.

[66] WANG D, WU S, LIN Z, et al. Towards a first step to understand flash loan and its applications in defi ecosystem[C]//Proceedings of the 9th International Workshop on Security in Blockchain and Cloud Computing. [S. l.: s. n.], 2021: 23-28.

[67] QIN K, ZHOU L, LIVSHITS B, et al. Attacking the DeFi ecosystem with Flash loans for fun and profit [C]//Proceedings of International Conference on Financial Cryptography and Data Security. [S. l.]: Springer, 2021: 23-32.

作 者 简 介

闫凯伦

男，1994 年生，博士研究生，研究方向为智能合约安全

E-mail：kailun@mail. sdu. edu. cn

刁文瑞

男，1988 年生，博士，教授，博士研究生导师，山东省泰山学者青年专家，研究方向为软件与系统安全

E-mail：diaowenrui@link. cuhk. edu. hk

郭山清

男，1976 年生，博士，教授，博士研究生导师，研究方向为软件与系统安全

E-mail：guoshanqing@sdu. edu. cn

责任编辑　董　莉