以太坊智能合约的漏洞自动化修复技术研究

傅紫薇,沈子牛,陈云芳,张 伟
(南京邮电大学 计算机学院,江苏 南京 210023)

摘 要:以太坊等公链上的智能合约可以实现各种去中心化应用,但频发的安全事件导致用户的财产遭受威胁。智能合约安全问题极大地影响用户对去中心化应用的信任度,且链上信息具有不可篡改的特性,使得智能合约在部署前的安全审计和漏洞修复过程必不可少,但当前的安全研究大多聚焦于智能合约漏洞检测技术。文章首先介绍了智能合约相关背景并比较了其与传统应用程序的差异,提出了包含漏洞识别和补丁生成两大关键步骤的智能合约部署前漏洞自动化修复流程,然后分析并阐述了常见的漏洞类型和漏洞检测技术,深入讨论了基于字节码和源码生成智能合约常见漏洞补丁的研究进展,最后对智能合约漏洞补丁生成技术面临的有效性、成本、可扩展性等性能问题以及漏洞自动修复技术的未来方向进行了展望。

关键词:区块链安全;以太坊;智能合约;漏洞检测;自动化修复
中图分类号:TP393 **文献标识码**:A **文章编号**:1673-629X(2023)02-0110-09
doi:10.3969/j.issn.1673-629X.2023.02.017

Research on Automatic Vulnerability Repair Technology of Smart Contracts on Ethereum

FU Zi-wei,SHEN Zi-niu,CHEN Yun-fang,ZHANG Wei
(School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210023,China)

Abstract:Smart contracts on public blockchain such as Ethereum can realize various decentralized applications,but frequent security incidents threaten users´ property. The security problems of smart contracts greatly affect users´ trust in decentralized applications,and the immutable nature of on-chain information makes the security audit and vulnerability repair process of smart contracts essential before deployment. However,most of the current security research focuses on the vulnerability detection technology of smart contracts. Firstly,the background of smart contracts is introduced and the differences between smart contracts and traditional application are compared. The automatic vulnerability repair process before deployment of smart contracts is proposed,which includes two key steps of vulnerability identification and patch generation. Besides,common vulnerability types and vulnerability detection technologies are analyzed,the research progress of patch generation based on bytecode and source code is discussed in detail. Finally,the effectiveness,cost,scalability and other performance problems,as well as the future direction of vulnerability automatic repair technology are prospected.

Key words:blockchain security;Ethereum;smart contracts;vulnerability detection;automation repair

0 引 言

区块链技术依据去中心化、不可篡改和公开透明等特性[1],在近年受到了广泛关注。区块链 2.0 开始,智能合约与区块链技术相结合,智能合约的引入使得各种去中性化应用程序 DApps 迅速发展,极大地丰富了区块链的应用场景。但智能合约也带来了一系列安全问题。以太坊作为目前最流行的智能合约平台之一,已经发生了多起重大安全事件,在这些安全事件中,绝大多数是由于智能合约被攻击所致[2]。例如,2018 年 4 月美链 BEC 的智能合约因其整数溢出漏洞而遭受攻击[3],导致 60 亿 BEC 代币转移到恶意账户,使得 BEC 市值几乎归零,给 BEC 市场交易带来了严重的影响。2020 年 4 月,黑客对 DeFi 交易平台 Lendf.Me[4] 的智能合约实施了攻击,损失金额达 2500 万美元。

智能合约正处于发展阶段,在设计、开发和应用过程中常存在某些缺陷,且很多基于智能合约的去中心化应用涉及资金流转,存在漏洞的合约更易成为攻击

收稿日期:2022-03-31 **修回日期**:2022-08-02
基金项目:国家重点研发计划资助(2019YFB2101700)
作者简介:傅紫薇(1997-),女,硕士研究生,研究方向为区块链智能合约安全;通讯作者:陈云芳(1976-),男,硕导,副教授,研究方向为区块链、网络信息安全;张 伟(1973-),男,博导,教授,研究方向为区块链、恶意代码分析与网络安防。

者重点关注的目标,因而智能合约安全上链前的合约代码安全审计和修复工作十分关键。到目前为止,已经有大量的漏洞自动检测方法或工具被研发,他们重点关注某种或几种漏洞类型,以设计专门的检测方法发现这些漏洞,但在整个智能合约安全保障过程中这些还并不足够。相比于传统程序,智能合约在程序开发、编程语言和代码优化等方面的发展尚不完善,一些开发人员可能并不知道如何正确修改这些漏洞代码,并且人工修复需要消耗大量资源。漏洞自动化修复技术可以实现漏洞代码修复的自动化,能解决人工维护能力不足的问题,降低人工维护的成本,提高代码质量,因此加深对智能合约漏洞自动修复技术的研究很有必要。

该文提出了一个包含漏洞识别和补丁生成两大关键步骤的智能合约部署前漏洞自动化修复流程,主要工作与贡献如下:

(1)提出了一个适应于自动化修复智能合约漏洞的框架,包括基于智能合约样本和漏洞库的输入、漏洞检测和报告生成、智能合约的修复以及修复后的验证;

(2)基于框架的关键部分,分析了典型的智能合约漏洞产生原因和漏洞识别技术,分类归纳了基于字节码和源码级别生成补丁的最新研究进展;

(3)总结了修复后合约所需满足的条件,并从有效性、成本问题、可扩展性三个方面展开讨论了补丁修复技术的性能评估需求。

1　相关背景

智能合约是对现实中的合约条款执行电子化的量化交易协议[5],区块链技术的发展为其提供了可信的执行环境。智能合约是传统合约的数字化版本,签署合约的参与者在内容上达成一致,存在形式为一段可在区块链上满足执行条件后自动执行的代码,它能实现区块链中各类数据的控制和管理,可应用于去中心化金融、医疗保健、供应链和物联网等多个方面。由于运行于区块链之上,智能合约遵循区块链的种种机制,其程序特性与传统应用程序也有所差异。

1.1　Gas 机制

区块链是一种分布式系统,设计者引入激励机制使得这类去中心化网络可以达成共识,以保证网络状态的一致性。主流的共识算法有工作量证明 PoW、权益证明 PoS、授权股份证明 DPoS 等[6],通过共识算法可以决定哪个网络节点拥有新区块的记账权。在以太坊中,网络节点主要采用 PoW 共识算法来竞争记账权,需要打包的交易都位于交易池中,拥有记账权的节点负责打包交易并记录到新区块。交易种类包括普通转账交易、创建合约交易和调用合约交易。智能合约的部署和调用,都通过交易的形式来实现。智能合约的执行和交易合法性的验证,都将占用一定的资源,因此拥有记账权的节点可以收取一定的费用作为奖励,即 Gas 费用。

智能合约每一条指令的执行都对应一定的 Gas 消耗,以太坊技术黄皮书[7]详细讲解了不同指令消耗的 Gas 数量,例如创建合约、数据存储等操作比较昂贵,需消耗更多的 Gas 数量。用户发起交易时需自行设定愿意支付的 Gas 用量上限并上交费用,执行合约代码过程中交易实际产生的费用等于实际消耗的 Gas 用量与单位 Gas 价格乘积所得的值,单位 Gas 的价格随市场变化而变化。如果交易执行完之后还有剩余 Gas,将退还给交易发送方,而一旦执行超过了 Gas 用量上限,交易将会失败导致交易回滚,但在这个过程中已经消耗的 Gas 用量并不会退还,Gas 机制的设计除了起到奖励矿工的作用,还在一定程度上防止了以太坊中一些恶意交易和资源浪费现象的发生。

1.2　智能合约运作过程

以太坊是最具代表性的智能合约应用平台,开发以太坊智能合约的编程语言主要是 Solidity,除此之外还有 Vyper、LLL 等。编码完成后的智能合约,由以太坊虚拟机(environment virtual machine,EVM)执行编译过程。EVM 是一个基于堆栈的虚拟机,为智能合约提供运行环境,将智能合约源代码编译成可在以太坊上执行的字节码。字节码由一串十六进制的数字组成,以一个字节为单位,每个字节对应一个操作码(指令)或操作数。EVM 字节码指令集中目前大约有 100 多条指令,常见的有堆栈操作指令:PUSH、POP ,跳转指令:JUMPI,数据读写操作指令:MLOAD、MSTORE、SLOAD、SSTORE,算数运算操作指令:ADD、MUL、SUB 等。

以太坊的账户分为合约账户和外部账户[8]。合约所有者充当外部账户的角色向地址 0X0 处发起一个交易,交易中的 data 域中存放合约字节码,并将交易的转账金额设为 0,通过这种方式实现合约的创建。创建合约的交易利用区块链中的共识机制实现部署,具体来说,区块链 P2P 网络中的全节点首先各自在本地执行创建合约的交易,运行合约代码并修改数据和状态,经共识算法决定出拥有记账权的矿工后,矿工打包交易更新区块,而后其他节点根据区块中的交易再执行一遍,更新本地的数据和状态达到同步,完成部署过程。

最后,每个部署之后的合约会生成唯一的地址,形成合约账户。用户使用外部账户向合约账户发起交易,同样经过区块链共识的过程对交易进行验证并执行,从而调用已经被部署在链上的智能合约。部署之

后的合约具有不可篡改、公开透明等特点。用户可以通过运行在 windows、linux 等操作系统上的以太坊客户端实现合约调用的操作,EVM 确认触发智能合约的条件满足后,解释执行合约代码。

1.3 智能合约与传统应用程序的区别

智能合约作为图灵完备的程序脚本,与传统应用程序有多个方面的区别,表 1 展示了智能合约应用与传统软件应用在不同角度的对比分析。

表 1 智能合约应用与传统软件应用对比分析

角度	智能合约应用	传统软件应用
系统属性	去中心化	中心化
可追溯性	可追溯	难以追溯
编程语言	Solidity、Vyper、LLL 等	c++、java、python 等
代码可见性	用户可见	用户不可见
生命周期[2]	设计、实现、部署、交易调用、监测	需求分析、设计、实现、测试、维护
开发框架[9]	稀缺	丰富
代码优化	功能扩展、安全巩固	功能扩展、安全巩固、Gas 成本

基于以上的比较,从学习成本、开发难度等方面来看,由于智能合约发展历史较短,各种开发指导文档以及测试框架都比较匮乏,Solidity 作为一种较新的编程语言,自身也存在很多局限性,而且缺少功能强大的调试器,这些因素导致智能合约开发人员的学习成本和开发难度更高;从安全需求方面来看,传统应用程序的编码通常对用户不可见,只在用户层提供一些接口,而智能合约的公开可见性,增加了受攻击的风险,又由于涉及大量资金交易,智能合约对安全性的要求比传统应用程序更严格;另外,从代码优化和漏洞修复方面来看,传统软件应用往往注重于功能扩展和安全性能的优化,而智能合约遵循 Gas 机制,使得合约代码在优化和漏洞修复过程还需着重考虑成本问题。Gas 机制是智能合约性能优化过程中,区别于传统软件应用的一个显著特征。

2 智能合约漏洞自动化修复框架

智能合约与传统应用程序存在的种种差异为智能合约的安全问题及其漏洞自动修复技术带来了不一样的挑战。当前针对智能合约漏洞检测的研究较多,而修复技术的研究相对较少,也缺乏整体认识。对于智能合约上链前的代码漏洞修复,认为其同传统软件修复技术[10-11]具有一些共性,同样存在漏洞定位、补丁生成和补丁验证等步骤,基于此,可以建立专注于智能合约的自动化修复方案。该文总结性地提出了适应于智能合约漏洞自动化修复技术的研究框架,符合目前已有的智能合约漏洞自动化修复的一般流程,如图 1 所示。

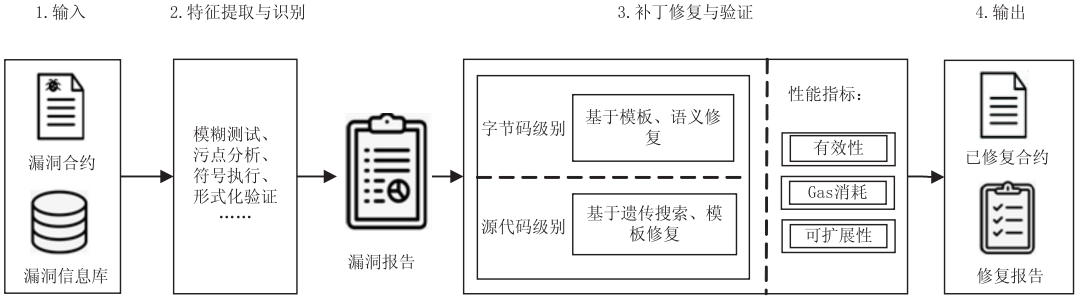

图 1 智能合约漏洞自动化修复框架

首先,将待测合约 Solidity 源码或者编译之后的字节码作为输入,利用智能合约漏洞库中相关漏洞类型和特征的记录,对合约代码进行漏洞分析,再采用漏洞检测技术识别漏洞,判断是否存在安全漏洞并生成漏洞检测结果报告。报告可以是 json 格式的文件,其中包含漏洞的类型和所在位置等重要说明。

特征提取与识别是自动修复过程的前提,其识别与定位的准确性会对后面的修复工作产生直接的影响,近年来得到了研究人员的广泛专注。主要可采取模糊测试、符号执行、污点分析以及形式化验证等不同的方法手段[12]对程序进行分析,或者嵌入现有的多种漏洞自动检测工具形成 docker 镜像提供漏洞检测的功能,为后续进行补丁修复提供有效信息。

补丁修复与验证是消除漏洞的关键步骤,依据漏洞报告产生的漏洞分析与定位描述,将通过自动修复方案产生一系列补丁修复语句或根据预先定义的修复模板,生成候选补丁。然后在漏洞位置依据候选补丁进行增加、删除和替换语句的操作,再通过软件测试方法使用测试集测试或者使用漏洞测试工具对修复代码的正确性进行检验。目前已有的自动修复技术可根据修复语言级别分为基于字节码修复与基于源码修复,具体修复方案的性能可依据漏洞修复的有效性、Gas 成本消耗、修复技术可扩展性等角度进行评估。

已修复合约和修复报告是通过漏洞修复技术生成

的结果输出。修复报告包含已修复漏洞类型的说明,并记载修复所在位置、具体操作、时间消耗等详细信息,可供安全人员或开发人员查阅,有效减少合约开发过程产生的潜在漏洞,提高安全性。

针对智能合约漏洞自动化修复流程所依赖的关键部分,下面将展开分析与阐述智能合约漏洞特征识别和补丁生成相关的技术手段。

3　智能合约漏洞分析与识别

3.1　智能合约典型漏洞分析

识别智能合约漏洞需要提取代码特征并分析特征,因此需要大量详细的漏洞信息数据的支持,表 2 展示了现有的记录智能合约漏洞信息的资源库。漏洞资源库对智能合约开发过程中常见的安全漏洞类型及其代码实例做了描述,可为开发人员提供指导信息,也为安全审计的漏洞自动修复过程提供漏洞特征信息。

以太坊智能合约的安全漏洞种类众多,根据漏洞发生的层次,可以分为高级语言、虚拟机和区块链三个层面[13]。高级语言层面引入的安全漏洞主要与编程语言设计、用户程序的编写等存在缺陷有关,如整数溢出、未检查外部调用返回值等,虚拟机层面的安全漏洞则与 EVM 的实现及其字节码规范有关,如重入漏洞,而区块链层面的安全漏洞则主要与区块链本身的特性相关,如伪随机数漏洞。下面主要介绍 6 种典型智能合约漏洞类型的产生原因以及相应的修复策略。

表 2　智能合约漏洞信息库

名称	来源	描述	地址
SWC	Smart Contract Security	提供 37 种以太坊 Solidity 智能合约安全漏洞的分类,漏洞描述,以及相关测试用例集	https://swcregistry.io/
DASP	NCC 团队	由 NCC 团队发起的开放协作项目,列举 10 类影响最大的智能合约漏洞类型	https://dasp.co/
CVE	MITRE 公司	公认的安全漏洞收录平台,当前已记载 531 条智能合约相关漏洞的条例	https://cve.mitre.org/

(1)重入漏洞。

智能合约在执行外部调用的交互过程中,某个合约调用另一个合约但被再次进入,或形成递归调用导致将此合约账户的资源消耗完的过程,称为重入攻击[14]。重入攻击是智能合约中典型的攻击手段之一。它攻击成功的原因是合约中的代码缺陷,即状态变量的赋值操作在调用外部函数的语句之后发生。修复重入攻击可以使用互斥锁,保证事务执行的原子性,防止重新进入函数,或者将合约中的状态变量赋值操作移动到在外部调用之前来避免漏洞发生。

(2)整数溢出。

一个整数变量在以太坊虚拟机中会被指定为固定大小的数据类型,因此只能存储一定范围内的数字,例如 uint8 类型的整数变量只能存储 0 到 255 之间的数字,值为 255 时,如果加 1 将导致每一位数发生翻转从而产生整数上溢,同理值为 0 时,减 1 会导致整数下溢。整数溢出漏洞也常出现在其他的编程语言当中。针对整数溢出的修复,可以通过使用 OpenZeppelin 维护的 SafeMath 库来处理算术逻辑[15],它封装了算数运算接口,可以避免产生溢出漏洞。

(3)权限控制。

权限控制问题主要是合约中函数或变量的越权访问,例如智能合约的 Solidity 编程语言与其他高级语言类似,也有 public、private 等函数属性,这些属性控制着访问函数的方式[16]。没有被正确设置访问权限的函数,会增加它们被非法调用的风险。预防和修复这类漏洞,需要在代码编写过程中充分检查并设置严谨的函数访问权限,某些有访问控制要求的变量可以使用函数修饰符 modifier 进行修饰。

(4)伪随机数。

由于区块链分布式共识的特征,以太坊的智能合约无法支持真正的随机数。很多区块链上的开奖类游戏采用随机数逻辑作为开奖依据,利用区块打包时间、区块打包难度等信息作为计算随机数的种子,这种方式产生的数是伪随机数,有可能被预测到。针对随机数攻击的修复建议可以是使用链下方式生成随机数,然后利用预言机 oracle 传递到链上供使用[17]。第二种方式,可以使用当前区块的哈希值作为随机数种子[18],由于当前区块在打包之前是不可被预测的,用户提出交易请求后,合约记录执行此交易所在区块的区块哈希值,此值无法被预测,可以促使产生随机结果。

(5)Tx.origin 漏洞。

Tx.origin 是智能合约中的一个全局变量,返回最初调用合约的账户地址。如果受害合约内通过此变量对调用合约者的身份进行验证[19],容易产生 Tx.origin 漏洞。某些恶意的中间合约可能会利用 Tx.origin 变量等于最初调用受害合约的账户地址这一特征,实现越权操作。为避免这类攻击事件的发生,在一些权限验证的过程中尽量不要使用 Tx.origin,而改用 msg.

sender,msg. sender 返回的是当前调用合约的账户地址,可以阻止中间合约通过身份验证。

(6)未检查 call 返回值。

Solidity 中的一些底层调用函数,如 call( )、callcode( )、delegatcall( )和 send( ),在执行结束之后仅返回布尔值 true 或 false 来表示是否执行正确,执行过程中若发生错误并不会抛出异常,而会直接继续执行后面的代码。所以如果对这类函数不进行返回值的检查,将有可能致使意外发生[20]。处理这类漏洞最好是在调用这些函数时检查其返回值,并对可能发生的异常预先设置解决方案。

3.2 漏洞检测方法

近年来学术界对智能合约漏洞检测进行了广泛和深入的研究[21-22],主要方法包含模糊测试、污点分析、符号执行、形式化验证等。

模糊测试是向程序提供大量非预期的输入,在运行时监测异常以发现漏洞的方法。对智能合约进行漏洞检测时,模糊测试通过迭代生成随机并且不同的交易,探索合约是否存在异常的执行状态。ReGuard[23]是一种专用于检测可重入漏洞的模糊分析器,其利用模糊器不断生成随机输入,执行运行时跟踪然后传递给检测器检测漏洞是否存在。Contractfuzzer[24]也是一款基于模糊测试的漏洞检测框架,但扩展了检测的范围,可检测包括重入漏洞、时间戳依赖等 7 种漏洞。

污点分析将输入数据标记为污点作为源,在智能合约运行过程中进行传播,追踪污点传播路径并判断数据的流向,从而发现可疑的数据操作来识别漏洞。Sereum 工具[14]利用污点跟踪技术监控智能合约执行过程中的数据流,自动检测异常状态,实现了对 3 种不同的重入攻击模式的检测。已有的一些漏洞检测方案通常将其与符号执行技术融合,例如 Osiris[25]为检测智能合约整数溢出,首先通过符号分析确定程序路径分支,将路径中的执行指令传递给污点分析器,然后利用污点分析器在堆栈(Stack)、内存(Memory)、存储(Storage)等位置引入污点并传播污点,最后通过检查已执行的指令中数据是否被整数溢出错误污染。

符号执行是一种程序分析方法,对智能合约进行分析时,使用符号值代替合约中的变量值,不断解析指令形成可执行路径,然后利用约束求解器求解满足约束的输入,检测输入的符号值是否存在异常问题。这种方法无需模拟以太坊上的执行环境动态地进行检测,减少了不确定性,有良好的代码覆盖率,已经有多种基于符号执行开发的针对智能合约漏洞的静态分析工具[26-27]。

形式化验证的方法是利用形式化语言对智能合约进行建模,构建推理系统来验证合约代码的正确性,文献[28]中总结了形式化验证在智能合约方面的研究,并将已有的基于形式化验证智能合约的方法分为 8 类进行阐述,分别是基于定理证明的形式化方法、基于符号执行的形式化方法、基于模型检测的形式化方法、基于规约语言的形式化方法、基于形式化建模的形式化方法、基于有限状态机的形式化方法、基于着色 Petri 网的形式化方法、基于运行时验证的形式化方法。

除了这几种主流方法,近年来机器学习的进步已经充分体现了其技术优势,研究人员也积极尝试结合机器学习算法[29-30],对智能合约代码进行特征提取,建立检测模型来识别易受攻击的智能合约,并得到了良好的性能。

4 智能合约漏洞补丁生成与验证

4.1 补丁生成的相关技术

目前对智能合约进行补丁修复的技术主要采用了基于模板、基于遗传搜索等,表 3 列出了已有的智能合约漏洞自动化修复工具。而从补丁修复的语言层面来看,可分为基于源代码和基于字节码的修复。源码级别的修复对使用者来说具有更高的可读性,字节码的修复可独立于源码语言,避免版本原因造成不正确修复等问题。下面将分别对这两类补丁修复工具所涉及的技术进行梳理和介绍。

表 3 智能合约漏洞自动化补丁修复工具

工具名称	修复级别	修复漏洞类型	时间
SMARTSHIELD[31]	字节码	重入、整数溢出、未检查 call 返回值	2020
EVMpatch[32]	字节码	整数溢出、访问权限控制漏洞	2020
ELYSIUM[33]	字节码	整数溢出、重入、未处理的异常、自杀合约、浪子合约、不安全的 delegatecall 调用、Tx. origin 漏洞	2021
sGUARD[34]	源码	整数溢出、重入、Tx. origin 漏洞	2021
SCRepair[35]	源码	异常处理漏洞、整数溢出、交易顺序依赖、重入	2020

(1)基于源码。

智能合约源码级别的漏洞修复直接对产生漏洞处的 Solidity 源码进行程序分析,然后采用不同的修复技术产生 Solidity 源码补丁进行自动修复。例如图 2

(a)中代码有典型的重入缺陷,结合漏洞类型及其产生的原因,sGUARD 工具应用预定义的修复模板,给 withdraw( )取款函数添加一个 nonReentrant_修饰器,应用了此修饰器的 withdraw( )函数会首先执行图 2(b)中第 2 ~ 3 行的语句,为函数中提取资金的进程上锁,实现事务原子性,修复缺陷。此外 sGUARD 还构建了修复整数溢出和 Tx. origin 漏洞的补丁模板。基于模板的修复技术由于较高的针对性,其修复质量比较高,但其局限性在于只能用于修复特定的场景和有限的漏洞种类。

Yu 等人[35]采用多任务遗传算法实现了自动修复工具 SCRepair,他们提出的搜索技术使用移动、插入、替换三种变异算子将大量小的语句变化植入漏洞合约,通过了所有测试集实例的补丁成为合约代码的候选补丁。最终实现了异常处理漏洞、整数溢出、交易顺序依赖、重入等问题的修复。为提升搜索效率,SCRepair 将搜索空间分割为较小的互斥搜索空间,并使用并行搜索,同时通过公式计算候选补丁的 Gas 消耗筛选高质量补丁。这类方法的搜索深度往往有限,产生的补丁质量和修复速度也有待提升。

```
1   function withdraw(uint256 _amount) public payable {
2       require(balances[msg.sender] >= _amount);
3       require(this.balance >= _amount);
4
5       msg.sender.call.value(_amount)();
6       balances[msg.sender] -= _amount;
7   }
8
```

(a)重入漏洞代码

```
1   modifier nonReentrant_() {
2       require( ! locked_);
3       locked_ = true;
4       _;
5       locked_ = false;
6   }
7
8   function withdraw(uint256 _amount) public payable nonReentrant_{
9       require(balances[msg.sender] >= _amount);
10      require(this.balance >= _amount);
11
12      msg.sender.call.value( _amount)();
13      balances[msg.sender] -= _amount;
14  }
15
```

(b)重入漏洞修复后代码

图 2　重入漏洞代码修复示例

(2)基于字节码。

智能合约编译之后形成字节码,基于字节码的修复往往先使用静态分析技术分析程序的控制依赖关系、数据依赖关系,然后获得字节码级别的语义信息以识别不正确的数据流或控制流[31],这个过程也可以直接使用当前已有的静态漏洞检测工具来实现。收集到错误的信息流之后,利用字节码重定位技术,将字节码补丁应用到相应位置,从而实现修复。

由于智能合约的运行环境 EVM 是基于堆栈的体系结构,程序跳转指令 JUMPI 的目的地址(如 0x000a)都会通过 PUSH 指令存放在堆栈中。基于字节码的修复在代码地址空间中插入任何语句时,原先的空间布局都可能会改变,因此必须更新所有相关的地址引用,以确保修改后的程序能调用正确的地址引用,调整地址引用的过程称为字节码重定位。字节码重定位是基于字节码修复的技术重点同时也是技术难点。

为了解决这一挑战,Rodler 等人[32]实现的自动修复工具 EVMpatch 在利用静态分析技术得到程序的控制流图之后,将需要修复的指令所在的基本块复制到代码地址空间的末尾空闲处,然后对此基本块使用预定义的字节码补丁模板进行修正,再用 JUMPI 指令重定向到此基本块,替换原始的基本块。这样就避免了初始的代码地址空间被修改。但这种基本块级别的重定位方式,在末尾处占据了与原始基本块相同大小的地址空间,增加了整体的代码量。

针对 EVMpatch 在字节码重定位方面存在的不足,Ferreira 等人[33]采用在预定义的模板中添加语义信息创建修复补丁,并在代码漏洞之处做出直接修改的方式来修复漏洞。他们在修改之前先创建一个地址副本,用于记录原始合约所有指令的地址。扫描程序控制流图进行修复工作,如图 3 中,插入虚线框内的修复语句之后,会更新地址副本中所有修复语句之后的所有地址指令。最后对于跳转地址的更新,分两步进行:第一步,找到修复后的地址空间中,所有不同于原始地址的 JUMPDEST 指令,记载它当前所在的地址,如图 3 中的 JUMPDEST 地址由 0x00c3 变为 0x00d2;第二步,查找 PUSH 值等于原始 JUMPDEST 指令地址的 PUSH 指令,使用地址副本中记录的新值替换 PUSH 值,如图 3 中等于原始 JUMPDEST 指令地址 0x00c3 的 PUSH 值应当替换为 0x00d2。这种字节码重定位的方式无需占用多余的地址空间,节省了一定的资源。

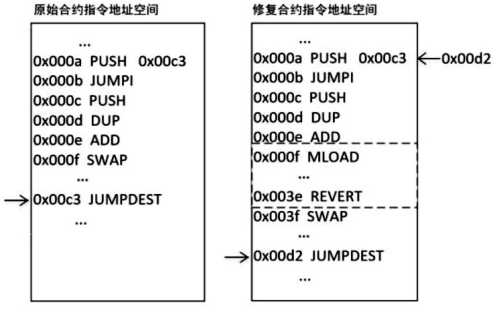

图 3　字节码重定位

4.2　补丁的性能验证

考虑智能合约与传统应用程序的差异,验证智能

合约补丁修复方案性能评估指标所涉及的内容也有所不同,修复后的合约应该满足以下 3 个先决条件:

(1)通过所有测试集中的测试用例,不存在漏洞报告中的漏洞;

(2)合约如需部署上链,执行消耗的 Gas 不超过其 Gas 用量上限;

(3)不影响原始合约的正常功能。

通过分析上述条件,从有效性、Gas 消耗、可扩展性三个方面讨论智能合约补丁修复的性能评估需求。

(1)有效性。

修改后的智能合约程序要求通过所有测试实例,且合约的正常功能不被破坏,保持与原始合约一致,才被认定是有效的修复。因此,有效性包含正确性和功能一致性两方面。为了验证智能合约修复的正确性,可利用软件测试方法,使其通过测试用例集进行验证,但这种方法的覆盖率有限,另外还可以利用现有的漏洞检测工具如 slither[36] 对修复合约进行漏洞测试,判断漏洞是否已被修复,这种方式的验证效果依赖于检测工具本身的准确性。为了验证功能一致性,可重现合约历史交易。即通过区块链浏览器如 etherscan 查询大量的原始合约正常历史交易记录,分别对修复合约与原始合约模拟重现这些交易,判定两者的功能是否一致。

(2)Gas 消耗。

运行于区块链系统上的智能合约,计算资源受到限制,Gas 消耗问题需要着重考虑。代码修复会改变代码本身的大小,增加的计算会增加额外的 Gas 消耗,也因此在求解补丁修复语句时,在保证正确修复漏洞的情况下,应当追求更少的代码改动和 Gas 消耗。一些智能合约漏洞补丁修复工具(如 sGUARD、SMARTSHIELD 等)通过重现合约交易的方式,比较修复合约与原始合约的 Gas 及时间消耗上的差异,来衡量他们的平均修复成本以及修复质量。为了优化补丁的质量,SCRepair 综合考虑会影响成本的参数,将智能合约的 Gas 消耗量化为一个计算公式,在产生大量候选的修复补丁时,通过公式计算 Gas 消耗并排序,择优进行修复。

(3)可扩展性。

应用程序的功能需求可能会面临更新,应用程序的扩展性表现为功能模块之间的依赖与耦合程度。拥有良好扩展性的应用程序意味着可以更好地支持功能修改和拓展。对于智能合约漏洞的自动化补丁修复工具而言,如果实现模块化的可修复漏洞种类增加、检测功能与补丁修复功能分离,将可以提高其扩展性。E-LYSIUM、EVMpatch 等工具均直接利用了现有的漏洞检测工具在补丁修复前定位合约的漏洞,并在补丁修复后验证修复效果。

5 结束语

如前所述,已有部分研究人员在智能合约自动化修复问题方面做出了贡献,并取得了一定的成果。但与传统程序自动修复技术相对比,智能合约自动修复技术研究时间较短,仍有许多问题尚未得到解决,笔者认为未来在如下几个方面值得关注:

(1)在漏洞识别阶段,基于动态的运行测试方法,其检测准确率依赖于测试用例,不能保证代码覆盖率,提高动态检测方法的准确率需要有良好的测试用例集。而基于静态的分析方法可能无法检测一些运行时的异常行为,因而存在一定的误报率。漏洞识别的准确性将直接关系到后续补丁修复工作,如何将几种主流的方法相结合,融合各自的优势以期找到更具适应性和更高准确率的智能合约漏洞检测方案,是进一步研究的方向。

(2)目前已实现的补丁生成方案采用的技术方法比较单一,主要是基于模板或结合语义的方法。原因是前期程序分析阶段过程中,利用代码覆盖率高的静态分析技术可以方便地获取程序语义信息,另外基于模板的补丁修复可以利用漏洞特征预先定义修复模式,修复效率较高。但基于模板修复的方式,其修复能力和范围受限于预先定义的模板数量和质量,可修复漏洞的种类有限,难以广泛应用,如何结合人工智能技术和智能合约应用场景语义以探索具有场景语义的修复模板值得探讨。传统软件缺陷自动修复方面,已有基于搜索[37]、基于语义[38]、基于模板[39]以及基于统计分析[40]等多种方法,在未来的方案探索中,可进一步结合这些方法建立适用于智能合约部署前的自动化修复技术,填补当前的研究空缺。

(3)在漏洞补丁修复能力方面,已有的几种补丁修复工具仅能修复 2 ~ 7 种漏洞,且这些工具关注更多的是高级语言或虚拟机层面的漏洞,而与区块链本身特性有关的漏洞诸如时间戳依赖、条件竞争等很少被研究。随着智能合约应用规模的发展,研究智能合约更深层次漏洞的自动修复技术显得尤为重要,研究者需要加强对不同层面漏洞的关注,并不断扩大可修复的漏洞类型。另外当前的一些自动化修复工具误报率仍然较高,修复过程产生的额外成本也是一直有待改进的问题。

(4)智能合约上链之前的漏洞自动修复不是一个一蹴而就的工作,智能合约漏洞问题也不会因此完全消失,存有漏洞的合约部署上链之后暴露在公众视野种将会存在更大的风险,链上合约安全问题同样值得关注。智能合约升级作为一种修复验证过程中遗漏错

误的补救方案,可以根据地址重定向替换原先存在于链上的漏洞合约[41]。另外,一些研究提出了通过监测链上交易执行,及时阻止危险交易发生的方案来提高安全性[42]。合约升级和交易保护措施为解决链上合约安全提供了新的思路。

基于区块链技术的智能合约上链之后难以修改,安全性需求高。该文以典型的以太坊平台为例,介绍了智能合约的运行过程和开发特性,探讨了其上链前安全审计过程中的漏洞自动修复技术,指出了当前技术方案在漏洞修复准确率、修复代价、修复覆盖度等方面的不足,并提出了未来进一步研究的方向。

参考文献:

[1] PUTHAL D, MALIK N, MOHANTY S P, et al. Everything you wanted to know about the blockchain: its promise, components, processes, and problems[J]. IEEE Consumer Electronics Magazine, 2018, 7(4): 6-14.

[2] HUANG Y, BIAN Y, LI R, et al. Smart contract security: a software lifecycle perspective[J]. IEEE Access, 2019, 7: 150184-150202.

[3] GAO J, LIU H, LIU C, et al. Easyflow: keep Ethereum away from overflow[C]//2019 international conference on software engineering: companion proceedings (ICSE-Companion). Montreal: IEEE, 2019: 23-26.

[4] YANG M. A summary of the attack on Lendf. Me on April 19, 2020[DB/OL]. 2020-04-20. https://medium.com/dforcenet/a-summary-of-the-attack-on-lendf-me-on-april-19-2020-e2f1c5d96640.

[5] SZABO N. Smart contracts: building blocks for digital markets[EB/OL]. 1996. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTWinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[6] 刘懿中, 刘建伟, 张宗洋, 等. 区块链共识机制研究综述[J]. 密码学报, 2019, 6(4): 395-432.

[7] WOOD G. Ethereum: a secure decentralised generalised transaction ledger[J]. Ethereum Project Yellow Paper, 2014, 151: 1-32.

[8] 孟　博, 刘加兵, 刘　琴, 等. 智能合约安全综述[J]. 网络与信息安全学报, 2020, 6(3): 1-13.

[9] ZOU W, LO D, KOCHHAR P S, et al. Smart contract development: challenges and opportunities[J]. IEEE Transactions on Software Engineering, 2019, 47(10): 2084-2106.

[10] 李　斌, 贺也平, 马恒太. 程序自动修复: 关键问题及技术[J]. 软件学报, 2019, 30(2): 244-265.

[11] 姜佳君, 陈俊洁, 熊英飞. 软件缺陷自动修复技术综述[J]. 软件学报, 2021, 32(9): 2665-2690.

[12] 付梦琳, 吴礼发, 洪　征, 等. 智能合约安全漏洞挖掘技术研究[J]. 计算机应用, 2019, 39(7): 1959-1966.

[13] 倪远东, 张　超, 殷婷婷. 智能合约安全漏洞研究综述[J]. 信息安全学报, 2020, 5(3): 78-99.

[14] RODLER M, LI W, KARAME G O, et al. Sereum: protecting existing smart contracts against reentrancy attacks[J]. arXiv: 1812.05934, 2018.

[15] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. Smartcheck: static analysis of Ethereum smart contracts[C]//2018 international workshop on emerging trends in software engineering for blockchain (WETSEB). Gothenburg: ACM, 2018: 9-16.

[16] SAYEED S, MARCO-GISBERT H, CAIRA T. Smart contract: attacks and protections[J]. IEEE Access, 2020, 8: 24416-24427.

[17] 郑忠斌, 王朝栋, 蔡佳浩. 智能合约的安全研究现状与检测方法分析综述[J]. 信息安全与通信保密, 2020(7): 93-105.

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

[19] BRENT L, JURISEVIC A, KONG M, et al. Vandal: a scalable security analysis framework for smart contracts[J]. arXiv: 1809.03981, 2018.

[20] 胡甜媛, 李泽成, 李必信, 等. 智能合约的合约安全和隐私安全研究综述[J]. 计算机学报, 2021, 44(12): 2485-2514.

[21] VIVAR A L, OROZCO A L S, VILLALBA L J G. A security framework for Ethereum smart contracts[J]. Computer Communications, 2021, 172: 119-129.

[22] PRAITHEESHAN P, PAN L, YU J, et al. Security analysis methods on Ethereum smart contract vulnerabilities: a survey[J]. arXiv: 1908.08605, 2019.

[23] LIU C, LIU H, CAO Z, et al. Reguard: finding reentrancy bugs in smart contracts[C]//2018 international conference on software engineering: companion (ICSE-Companion). Gothenburg: IEEE, 2018: 65-68.

[24] JIANG B, LIU Y, CHAN W K. Contractfuzzer: fuzzing smart contracts for vulnerability detection[C]//2018 international conference on automated software engineering (ASE). Montpellier: IEEE, 2018: 259-269.

[25] TORRES C F, SCHÜTTE J, STATE R. Osiris: hunting for integer bugs in Ethereum smart contracts[C]//2018 international conference on annual computer security applications. San Juan: ACM, 2018: 664-676.

[26] LUU L, CHU D H, OLICKEL H, et al. Making smart contracts smarter[C]//2016 international conference on computer and communications security (CCS). Vienna: ACM, 2016: 254-269.

[27] KRUPP J, ROSSOW C. Teether: gnawing at Ethereum to automatically exploit smart contracts[C]//2018 USENIX security symposium. Baltimore: USENIX Association, 2018: 1317-1333.

[28] 朱　健, 胡　凯, 张伯钧. 智能合约的形式化验证方法研究综述[J]. 电子学报, 2021, 49(4): 792-804.

[29] WANG W, SONG J, XU G, et al. Contractward: automated

vulnerability detection models for Ethereum smart contracts [J]. IEEE Transactions on Network Science and Engineering, 2020, 8(2): 1133-1144.

[30] ESHGHIE M, ARTHO C, GUROV D. Dynamic vulnerability detection on smart contracts using machine learning[C]//Evaluation and assessment in software engineering. Trondheim: ACM, 2021: 305-312.

[31] ZHANG Y, MA S, LI J, et al. Smartshield: automatic smart contract protection made easy[C]//2020 international conference on software analysis, evolution and reengineering (SANER). London: IEEE, 2020: 23-34.

[32] RODLER M, LI W, KARAME G O, et al. EVMPatch: timely and automated patching of Ethereum smart contracts[C]//2021 USENIX security symposium (USENIX security). Berkeley: USENIX Association, 2021: 1289-1306.

[33] TORRES C F, JONKER H, STATE R. Elysium: automagically healing vulnerable smart contracts using context-aware patching[J]. arXiv: 2108.10071, 2021.

[34] NGUYEN T D, PHAM L H, SUN J. sGUARD: towards fixing vulnerable smart contracts automatically [J]. arXiv: 2101.01917, 2021.

[35] YU X L, AL-BATAINEH O, LO D, et al. Smart contract repair[J]. ACM Transactions on Software Engineering and Methodology, 2020, 29(4): 1-32.

[36] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]//2019 international workshop on emerging trends in software engineering for blockchain (WETSEB). Montreal: IEEE, 2019: 8-15.

[37] WEIMER W, NGUYEN T V, LE GOUES C, et al. Automatically finding patches using genetic programming[C]//2009 international conference on software engineering (ICSE). Vancouver: IEEE, 2009: 364-374.

[38] NGUYEN H D T, QI D, ROYCHOUDHURY A, et al. Semfix: program repair via semantic analysis[C]//2013 international conference on software engineering (ICSE). San Francisco: IEEE, 2013: 772-781.

[39] LIU K, KOYUNCU A, KIM D, et al. Tbar: revisiting template-based automated program repair[C]//2019 international symposium on software testing and analysis. New York: ACM, 2019: 31-42.

[40] BADER J, SCOTT A, PRADEL M, et al. Getafix: learning to fix bugs automatically[J]. Proceedings of the ACM on Programming Languages, 2019, 3: 1-27.

[41] MANDRYKIN M, O'SHANNESSY J, PAYNE J, et al. Formal specification of a security framework for smart contracts[C]//2019 international symposium on formal methods. [s. l.]: Springer, 2019: 392-403.

[42] MA F, FU Y, REN M, et al. EVM*: from offline detection to online reinforcement for Ethereum virtual machine[C]//2019 international conference on software analysis, evolution and reengineering (SANER). Hangzhou: IEEE, 2019: 554-558.

(上接第 109 页)

[8] WIRIAATMADJA D T, CHOI K W. Hybrid random access and data transmission protocol for machine-to-machine communications in cellular networks[J]. IEEE Transactions on Wireless Communications, 2015, 14(1): 33-46.

[9] SUN Y, FENG G, QIN S, et al. The SMART hand-off policy for millimeter wave heterogeneous cellular networks[J]. IEEE Transactions on Mobile Computing, 2018, 17(6): 1456-1468.

[10] MOON Jihun, LIM Yujin. Access control of MTC devices using reinforcement learning approach[C]//2017 international conference on information networking. Danang: IEEE, 2017: 641-643.

[11] MOHAMMED A H, KHWAJA A S, ANPALAGAN A, et al. Base station selection in M2M communication using Q-learning algorithm in LTE-a networks[C]//2015 IEEE 29th international conference on advance information networking and applications. Gwangju: IEEE, 2015: 17-22.

[12] TSOUKANERI G, WU Shangbin, WANG Yue. Probabilistic preamble selection with reinforcement learning for massive machine type communication devices[C]//2019 IEEE 30th annual international symposium on personal, indoor and mobile radio communications. Istanbul: IEEE, 2019: 1-6.

[13] SHARMA S K, WANG Xianbin. Collaborative distributed Q-learning for RACH congestion minimization in cellular IoT networks[J]. IEEE Communications Letters, 2019, 23(4): 600-603.

[14] ZHANG Dong, LIU Jianlong. ACB scheme based on reinforcement learning in M2M communication [C]//IEEE global communications conference 2020. Taiwan, China: IEEE, 2020: 1-6.

[15] HAMEED E, ELSAYED K. Q-learning approach for machine-type communication random access in LTE-advanced [J]. Telecommunication Systems, 2019, 71(3): 397-413.