信息安全研究 第8卷 第7期 2022年7月
Journal of Information Security Research Vol.8 No.7 July 2022
DOI:10.12379/j.issn.2096-1057.2022.07.01

基于符号执行的代币买卖漏洞和权限转移漏洞的检测验证方法

刘宇航　刘军杰　文伟平
(北京大学软件与微电子学院　北京　100871)
(wishucry@qq.com)

The Detection Method for Token Trading Vulnerability and Authority Transfer Vulnerability Based on Symbolic Execution

Liu Yuhang, Liu Junjie, and Wen Weiping
(*School of Software & Microelectronics, Peking University, Beijing* 100871)

Abstract　It is difficult to detect and verify comprehensively new token trading backdoor vulnerabilities and owner authority transfer vulnerabilities in smart contracts. Based on static semantic analysis and symbolic execution technology, this paper proposes a method to comprehensively and effectively detect and verify the token trading vulnerability and authority transfer vulnerability at the source code and bytecode levels. The method firstly converts contract source code into bytecode through contract collection and pre-processing. Secondly, global sensitive variables "balance" and "owner" are located through static semantic analysis. Then the state space is constructed and the transaction sequence is simulated by symbolic variables. The method performs symbolic execution on the contract, and establishes constraints through the features of vulnerability models. Finally, the method uses satisfiability modulo theories (SMT) solver to solve the constraint. The method is tested on ethereum, binance smart chain mainnet and a part of smart contract CVE vulnerability sets. The experimental results show that the method proposed in this paper can effectively detect the new token trading backdoor vulnerability as well as the owner authority transfer vulnerability.

Key words　smart contract; vulnerability detection; symbolic execution; program analysis; blockchain security

摘　要　针对智能合约中出现的新型代币买卖后门漏洞以及owner权限转移漏洞难以全面检测和验证的问题,基于静态语义分析和符号执行技术,提出了一种在源代码和字节码层面可以全面有效挖掘代币买卖漏洞和权限转移漏洞的检测和验证方法.该方法首先通过合约收集和预处理,将合约

收稿日期:2022-05-26
基金项目:国家重点研发计划项目(2020YFB1005802)
通信作者:文伟平(weipingwen@ss.pku.edu.cn)
引用格式:刘宇航, 刘军杰, 文伟平. 基于符号执行的代币买卖漏洞和权限转移漏洞的检测验证方法[J]. 信息安全研究, 2022, 8(7): 632-642

源代码转换为字节码；其次通过静态语义分析，对全局敏感变量“balance”以及“owner”进行定位；然后通过符号化变量构建状态空间，模拟交易序列，对合约进行符号执行；最后通过漏洞模型特征建立约束条件，使用约束求解器对约束进行求解.在以太坊、币安智能链主网上以及部分智能合约 CVE 漏洞集上进行测试，实验结果表明，提出的方法可以有效检测出新的代币买卖后门漏洞以及 owner 权限转移漏洞.

关键词 智能合约；漏洞检测；符号执行；程序分析；区块链安全

中图法分类号 TP309.2

随着区块链技术和应用的不断发展落地，越来越多的用户和软件工程师倾向于在区块链平台上开发、交互应用甚至配置数字资产.智能合约就是存储和运行在区块链上的应用程序，通过智能合约和区块链的 POW，POS，PBFT 等共识机制，可以在数字世界建立信任.以太坊、币安智能链等很多区块链平台都支持智能合约的运行.区块链的智能合约一般由图灵完备的编程语言写成.智能合约带给开发者和用户以无限的创造性，能够使用智能合约编程语言（例如 Solidity）定义各种规则，生成更多去中心化的应用供平台用户进行调用和交互.

这种去中心化、灵活可变、安全可靠的特性，使得智能合约被广泛地用于去中心化金融（DeFi[1]）和数字货币领域.

然而由于智能合约开发者的代码不规范、智能合约编程语言或虚拟机固有的缺陷，使得区块链上存在大量有漏洞甚至容易被利用的智能合约，造成了很多智能合约攻击事件的发生，尤其是在去中心化金融领域，很多有漏洞的智能合约上存储的数字资产被攻击者一卷而空，产生不可估量的损失.

在 DeFi 领域中最常见的攻击事件中，一类是欺诈性代币攻击事件，也被称为“貔貅盘”事件.这种攻击一般和与去中心化交易所进行交互的代币合约有关，这种代币合约由于合约开发者在编写智能合约 transfer 函数时产生了有意或者无意的代码漏洞，或者嵌入了恶意的后门代码，使得该合约中的代币买卖存在漏洞，例如只能在交易所买入，无法卖出，或者实际卖出数量与用户所期望数量不相符等.

另一种则是基于智能合约 owner 权限转移漏洞的攻击事件.几乎所有的智能合约中都有权限控制相关的代码逻辑，如何正确地将权限控制置于自己所开发的代码中，对于开发者来说相当重要，如果设计不当就会产生权限篡改的风险.著名的跨链互操作协议 Poly Network[2]，就曾因为权限控制代码逻辑漏洞，被攻击者修改了合约属主权限，从而在以太坊、币安智能链以及 Polygon 这 3 大区块链平台上分别被盗走 2.5 亿美元、2.7 亿美元、8 500 万美元的加密资产，损失总额高达 6.1 亿美元[3].

目前对于智能合约漏洞的研究往往仅限于一些传统的特定漏洞，例如：整数溢出、重入、delegatecall 导致的意外代码执行、合约构造函数与合约名不一致等漏洞[4].这些漏洞有一部分已经被更新版本的智能合约语言编译器消除，例如 Solidity 在 0.8 版本将默认内置整数溢出检查[5]，开发者不再为因疏忽引起的整数溢出漏洞而担惊受怕，同时在 Solidity 0.4.22 版本之后，开发者定义的合约构造函数名也不需要与合约名一致，而是统一用 constructor 代替，消除了合约构造函数与合约名不一致漏洞的产生.另外为避免一些漏洞，例如未检查返回值、重入漏洞等，智能合约 Solidity 网站也发布了简洁、可靠的安全编程规范[6-7]和框架避免写出有漏洞的合约.

可以看出很传统的智能合约漏洞模式已经逐渐被消除和有效地规避，并且在目前的研究中发现并未产生过大的危害[8].但目前各种新型的智能合约漏洞正在逐渐涌现，例如本文提到的造成欺诈性代币攻击的代币买卖漏洞和造成权限篡改攻击的智能合约 owner 权限转移漏洞，而目前的研究很少能够总结出一套全面的、有效的自动化检测方案.

鉴于此，本文提出了一种针对智能合约字节

码和源码的基于静态语义分析和符号执行的智能合约漏洞检测方法，可以用来检测智能合约的代币买卖漏洞和owner权限转移漏洞.

本文的主要贡献在于：

1）对于智能合约代币买卖漏洞和权限转移漏洞进行研究，总结了代币买卖漏洞和owner权限转移漏洞模型；

2）在智能合约字节码和源码层面，提出了针对代币买卖漏洞和owner权限转移漏洞的自动化检测工具；

3）调研和总结了目前在以太坊和币安智能链2大区块链平台上的代币买卖漏洞和owner权限转移漏洞的常见利用情况以及攻击背景和手法.

1 相关工作

在基于符号执行的智能合约自动化安全审计技术领域，Luu等人[9]提出了基于符号执行的Oyente工具来自动化检测智能合约中存在的整数溢出、时间戳依赖、交易顺序依赖以及重入等漏洞；Muller[10]提出了另外一款基于符号执行技术的工具Mythril，以检测整数溢出、代码重入等常见安全问题；Tsankov等人[11]开发了一种基于符号执行的智能合约自动化检测工具Securify，该工具可以针对给定的性质来验证智能合约的行为是否安全；Nikolic等人[12]实现了一种基于符号分析和程序验证器的工具Maian，该工具通过分析智能合约函数之间调用的字节码序列来检测可能存在的安全漏洞；Zhou等人[13]设计了一种名为SASC的静态分析工具，该工具同样基于符号执行技术，用于智能合约逻辑分析，可以生成函数之间调用流图，以帮助查找智能合约潜在的逻辑漏洞.除此之外，还有Manticore[14]，VerX[15]等智能合约自动化检测工具也是采用符号执行技术.

在基于程序静态分析的智能合约自动化安全审计技术方面，Tikhomirov等人[16]提出了一种可扩展的静态分析工具SmartCheck，将solidity源代码转换为基于XML的中间表示形式，然后根据定义的XPath进行漏洞检测；Feist等人[17]提出了一种静态分析框架Slither，它集成了大量漏洞检测模型，通过中间表示（SlithIR）可实现简单、高精度的分析，并提供一个API来轻松编写自定义合约分析；Kalra等人[18]提出了智能合约的静态分析框架ZEUS，它能够自定义用户策略，将附加上用户策略的合约源码转换为LLVM-IR的中间表示，然后结合LLVM-IR的分析工具进行代码分析和漏洞检测.该方案在进行合约源码到中间表示转换时容易失真，LLVM-IR无法完全模拟智能合约的代码和运行环境.

在基于模糊测试的智能合约漏洞检测领域，Trail of Bits安全团队[19]提出了以太坊智能合约模糊测试框架Echidna，通过静态分析和模拟执行智能合约源码来自动化生成调用合约方法的交易数据.而ContractFuzzer[20]将模糊测试和漏洞检测方式结合，通过随机生成交易数据、交易发起者、交易金额和日志监测来检测有无漏洞触发.在模糊测试种子生成策略方面，ILF[21]采用基于神经网络的机器学习算法，对通过符号执行后生成的高覆盖率交易序列进行学习，从而生成更好的模糊测试策略.但是模糊测试方法相较于符号执行，依旧存在着路径覆盖不够全面的问题.

以上大多数工具都是针对传统的以太坊智能合约漏洞，如算术溢出、重入、交易顺序依赖、delegatecall导致意外代码执行[22]等，但对于其他的一些危害较为严重的新型漏洞模式则不能准确识别，比如代币买卖漏洞、owner权限转移漏洞等.

2 漏洞分析及其检测方法

2.1 代币买卖漏洞

2.1.1 漏洞产生的原因及危害

代币买卖漏洞通常发生在遵循ERC-20代币标准的智能合约的transfer函数调用流中.

ERC-20代币标准是以太坊区块链上一种通用的同质化代币标准，通过遵循ERC-20代币标准，可以让不同智能合约中发行的代币都具有相同的类型和接口.

ERC-20代币标准中定义了标准函数和标准事件，如表1所示.其中涉及到代币买卖的函数是transfer和transferFrom函数，这2种函数一般会调用同一种开发者自定义的子函数“_transfer(address sender,address recipient,uint256 amount)”，即执行sender地址向recipient地址转账amount数量的代币操作.

表 1　ERC-20 代币标准

函数或者事件签名	说明
name()	函数,返回代币名称
symbol()	函数,返回代币符号
decimals()	函数,返回代币精度
totalSupply()	函数,返回代币总供应量
balanceOf(address _owner)	函数,返回 _owner 的代币余额
transfer(address _to,uint256 _value)	函数,调用者向 _to 地址转入 _value 数量代币
transferFrom(address _from,address _to,uint256 _value)	函数,代 _from 地址向 _to 地址转入 _value 数量代币
approve(address _spender,uint256 _value)	函数,授权 _spender 地址可以代调用者地址转账 _value 额度
allowance(address _owner,address _spender)	函数,返回 _spender 可代为 _owner 转账的额度
Transfer(address indexed _from,address indexed _to,uint256 _value)	事件,记录 _from 地址向 _to 地址转入 _value 数量代币
Approval(address indexed _owner,address indexed _spender, uint256 _value)	事件,记录 _owner 地址授权 _spender 地址可代为转账 _value 额度

本文将 transfer 和 transferFrom 函数调用中的程序行为抽象为“_transfer(address from, address to,uint256 amount)”函数行为.

图 1 所示为一个代币买卖漏洞代码片段:

```
1  function _transfer(address sender,address recipient,
     uint256 amount) public virtual {
2      require(sender!=address(0));
3      require(recipient!=address(0));
4      if (sender==owner) {
5          _balances[sender]= _balances[sender]-amount;
6          _balances[recipient]+=amount;
7      } else {
8          _balances[sender]= _balances[sender]-amount;
9          _balances[recipient]+=amount/2;
10     }
11 }
```

图 1　代币买卖漏洞代码片段

代码第 4 行条件分支语句中当 sender 地址为 owner 时,执行正常的第 5 行和第 6 行转账操作;如果 sender 地址不为 owner 时,执行第 8 行和第 9 行操作,即接收方 recipient 地址余额仅增加 amount 的一半.

代币买卖漏洞是指类似上述代码的这种情况:实际转账数量与用户所期望数量不相符,收取了高额的转账手续费;只能特定账户进行转出,普通用户只能转入,不能转出等.

在 DeFi 领域,大多数代币都在去中心化交易所(DEX)中建立了交易对和流动池,在去中心化交易所中用户可以使用其他价值较高的数字货币(例如 wBTC,wETH,wBNB,USDT 等)来买入这些代币,一旦这些代币合约中出现买卖漏洞,买入后无法卖出,或者卖出数量远低于实际数额,那么用户的 wBTC,wETH,wBNB,USDT 等数字货币将会锁在合约中,对用户的数字资产造成巨额损失.

2.1.2　漏洞分析与漏洞模型

将合约源代码转换为字节码,第 5,6,8,9 行关于“_balances”的修改存储操作在字节码层面的模型特征为

SHA(memory)
→SLOAD(key)
→ADD(value,amount)
→SSTORE(key,new _value)

全局变量 _balances 在 Solidity 智能合约中是一个 address 映射到 uint256 类型的 Mapping 数据结构,其在以太坊虚拟机(EVM)底层通过 key-value 方式进行存储,value 就是映射中对应的 uint256 数值,key 值是将 address 与全局变量 _balances 的槽位拼接后进行哈希(SHA3 指令)得到的 256 位数值.SHA3 指令对 memory 中存储的值进行哈希,SLOAD 指令则通过哈希后得到的 key 值在 storage 中进行检索相应的 value.对检索到的 value 值进行加或减(SUB 指令)操作,然后将新的 value 值存储(SSTORE 指令)到 key 对应的变量 _balance 中.

漏洞点在于在 SSTORE 指令时，任意 recipient 地址（0 地址和 sender 地址这种特殊地址除外）以及 _balance 的槽位作为 key 时，其存储的 value 数值不等于原 value 数值与用户调用 transfer 函数时 amount 的加和.

2.1.3 漏洞检测方法

本文代币买卖漏洞检测方法步骤如图 2 所示：

图 2 代币买卖漏洞检测方法

图 2 中：

1）对智能合约源代码进行编译形成字节码；

2）对字节码和操作码进行静态分析，构建基本块和控制流图；

3）通过 ERC-20 标准函数签名对第 2 步静态分析处理后的 contract 对象匹配出符合 ERC-20 标准的代币合约；

4）对符合 ERC-20 标准的代币合约的 balanceOf(address _owner)进行静态语义分析，确定在执行“ _balance[_owner]”时的 _balance 变量的槽位；

5）基于符号执行技术，符号化 transfer 以及 transferFrom 函数参数，符号化 msg.sender 地址；

6）建立约束条件，求解当满足

balance[_to]==balance[_to]+amount

的 msg.sender 的值集合情况；

7）当满足模型的 msg.sender 值为有限个，则上报代币买卖漏洞.

2.2 owner 权限转移漏洞

2.2.1 漏洞产生的原因及危害

在编写智能合约的过程中，合约的开发者一般会设置一个“owner”值，该值所代表的地址拥有一些特权，如转账、函数调用等.如果对“owner”值的修改没有施加限制条件，那么攻击者能够修改该值为自己的地址，从而攻击者会利用这些特权来攻击合约并获取利益.

图 3 所示为 owner 权限转移漏洞代码片段：

```
1  contract Demo {
2    address public owner;
3    mapping(address=>uint256) public _balances;
4  uint256 totalSupply;
5    constructor() {
6      owner=msg.sender;
7    }
8    modifier onlyOwner {
9      require(msg.sender==owner);
10     _;
11   }
12   function mint(address _to,uint256 amount) onlyOwner
       public {
13     _balances[ _to]+=amount;
14     totalSupply+=amount;
15   }
16   function setOwner(address _newOwner) public {
17     owner= _newOwner;
18   }
19 }
```

图 3 owner 权限转移漏洞代码片段

图 3 中第 5 行表明在构建函数时可以将 owner 权限设置给合约创建者，同时第 12 行 mint 函数附加了修饰符 onlyOwner，第 9 行判断只有当调用者等于 owner 时才可以进行第 13，14 行的 mint 函数操作.mint 函数用于属主向某个地址铸造更多的代币.在第 16 行 setOwner 函数可以修改 owner 变量，但是未附加修饰符 onlyOwner，使得任意地址可以调用 setOwner 函数对 owner 变量进行修改，从而让属主转变为攻击者地址，再调用 mint 函数就可以铸造大量代币分发给攻击者地址.

2.2.2 漏洞分析与漏洞模型

经过调研发现，owner 所在的 storage 槽位一般

与 onlyOwner 修饰符以及构造函数中 msg.sender 相关.在构造函数中,owner 一般由 msg.sender 或者普通地址类型变量进行初始化.在 onlyOwner 修饰符中,owner 一般被用于与 msg.sender 值比较,其在字节码层面的模型特征为

EQ(owner,msg.sender)→ISZERO→JUMPI.

同时从字节码层面分析,owner 权限转移漏洞模型可以总结为:合约执行过程中存在 SSTORE 指令,并且 SSTORE 指令的 key 值与 owner 所在的 storage 槽位相等.

2.2.3 漏洞检测方法

本文 owner 权限转移漏洞检测方法的步骤如图 4 所示:

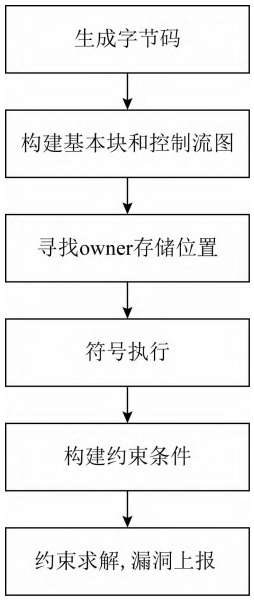


图 4 owner 权限转移漏洞检测方法

图 4 中:

1) 对智能合约源代码进行编译形成字节码.

2) 对字节码和操作码进行静态分析,构建基本块和控制流图.

3) 确定"owner"的存储位置.详细过程为:在合约内,一般会对"owner"变量赋值为 msg.sender,即合约的创建者,msg.sender 由 CALLER 操作码压入 EVM 栈中,赋值操作由 SSTORE 操作码完成.在操作码序列中寻找 CALLER 操作码,CALLER 会将 msg.sender 的值放入栈中,对栈中该数据进行跟踪.在操作码序列中寻找 SSTORE 操作码,SSTORE 操作会从栈中获取 key 和 value.如果 value 为 msg.sender 或者经过 AND 运算后的 20 B 变量(一般为 address 类型变量),那么 key 就是"owner"在 storage 中的存储位置.同时通过第 2 步静态语义分析中寻找的 onlyOwner 修饰符,对与 msg.sender 变量比较的全局变量槽位进行捕获.将上述这 2 种方式确定的存储位置记录下来,取其交集作为 owner 变量的存储位置供后续步骤使用.

4) 符号执行,根据步骤 3)确定的存储位置,判断写操作的存储位置是否是该位置.遍历寻找 SSTORE 操作码,SSTORE 操作码会从 EVM 栈中取出 key,判断 key 是否与步骤 1)中找到的"owner"的存储位置一致.如果是,则记录该路径.

5) 对满足上述条件的路径进行约束求解,有解则报告该漏洞.根据当前的约束条件进行求解,如果有解则表明存在任意调用者可以对"owner"的值进行修改,即存在漏洞.

3 漏洞检测工具的设计与实现

3.1 总体架构

该漏洞检测模型总体架构如图 5 所示,主要包括合约收集、预处理、静态分析、符号执行和漏洞模型分析 5 个模块:

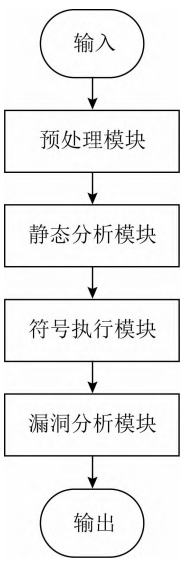


图 5 漏洞检测模型总体架构

3.2 模块设计与实现

3.2.1 合约收集模块

合约收集模块集成了以太坊 infra 节点和币

安智能链节点，可以通过各种方式对合约源代码和字节码进行收集，收集方式包括本地读取、通过etherscan收集源代码、通过合约地址收集链上字节码和etherscan源码及字节码、通过块高度区间收集链上合约源码和字节码等.

同时集成了Ethereum Signature Database[23]的API，可以通过字典查询的方式对字节码中的函数签名进行猜解.

3.2.2 预处理模块

预处理模块的主要功能是对输入的数据进行预处理，包括输入校验和数据处理.对于输入的智能合约solidity源码数据和EVM字节码数据，首先校验其合法性.如果输入的数据为solidity源码，则需要使用solc编译器编译为EVM字节码.

预处理模块执行流程如图6所示：

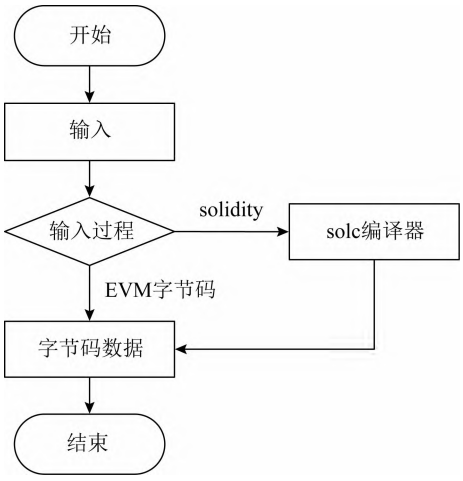


图6 预处理模块

3.2.3 静态分析模块

静态分析模块构建基本块和控制流图，基本块只包含顺序执行的指令，只有1个入口和1个出口，入口处于基本块的第1条指令，出口位于基本块的最后1条指令，中间不出现任何分叉.如果遇到跳转指令(JUMP或JUMPI)，那么结束当前基本块，将该指令作为当前基本块的最后1条指令，并分叉出1个新的基本块，将JUMPDEST指令作为新基本块的第1条指令.基本块B和基本块C之间存在1条边则构建形成基本块控制流.

3.2.4 符号执行模块

经过预处理模块得到字节码数据以及静态分析处理后的基本块和控制流图，模拟EVM执行字节码，并创建当前的执行状态，包括Stack，Memory和Storage.每个操作码指令都对应1个状态，通过符号执行，可以获取每个状态下Stack，Memory和Storage所存储的内容.然后利用该状态空间的基本代码块和路径约束条件集，添加约束和路径集合形成新的符号执行控制流图.如图7所示：

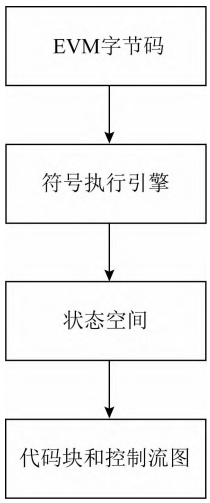


图7 符号执行模块

3.2.5 漏洞分析模块

漏洞分析模块包括代币买卖漏洞检测模块和owner权限转移漏洞模块.

漏洞分析模块通过符号执行以及静态分析暴露的接口，对EVM的stack，memory，storage，calldata，sender等进行符号化，同时由于SMT对哈希指令SHA3的支持较弱，添加SHA3的符号化组件，在符号化表达式语义层面对涉及SHA3的约束进行求解.最终利用Z3求解器对约束进行求解并上报漏洞，如图8所示：

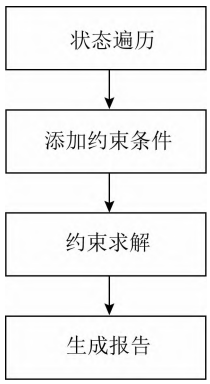


图8 漏洞分析模块

4 工具测试与实验分析

4.1 测试环境及样本类型

为验证工具的有效性以及性能等指标，本文使用以下硬件和软件环境进行测试，如表 2 和表 3 所示：

表 2 硬件环境

硬件	配置
CPU	Intel® Core™ i7-8550U CPU @ 1.80 GHz
内存/GB	8
存储/GB	512

表 3 软件环境

软件	配置
Ubuntu	20.04.3 LTS
Linux Core	5.11.0-41-generic
Python	3.8.10

本文在以太坊主网的去中心化交易所 Uniswap 以及币安智能链主网的去中心化交易所 PancakeSwap 各抽取 50 个代币合约作为代币买卖漏洞的测试样本，一共 100 个代币合约.

本文爬取了以太坊主网区块高度 13 000 000～13 001 000 中新部署的合约，共计 172 个，同时在此样本基础上，增加 12 个与 owner 权限转移漏洞相关的智能合约 CVE，作为 owner 权限转移漏洞的检测样本.

代币买卖漏洞在 CVE 库中没有相应案例，故只使用链上交易所合约样本.

4.2 测试结果与分析

4.2.1 代币买卖漏洞

对代币买卖漏洞检测工具进行测试，结果如表 4 所示：

表 4 代币买卖漏洞测试结果

指标	结果
总合约数	100
检出漏洞合约数量	79
确认有漏洞合约数量	6
运行用时	196 s/k 条指令

由表 4 可发现，代币买卖漏洞检测工具运行用时较长，误报率较高.

经过分析，运行用时长主要是符号执行耗时过长.

误报率过高主要原因有 2 个：一是由于大量代币合约在转账时设置手续费，使得实际转账金额与原数额有差别.经粗略统计，对于正常合约手续费设置基本在 25% 以内.二是因为一些合约设置了白名单和黑名单，限制了一些地址的转账权.

以代币合约 DogeKing（币安智能链地址 0x641EC142E67ab213539815f67e4276975c2f8D50）为例，其代码片段如图 9 所示：

```
1  function _transfer(
2    address from,
3    address to,
4    uint256 amount
5  ) internal override {
6    //other code here...
7    //if belongs to _isExcludedFromFee account remove the fee
8    if ( _isExcludedFromFees[from] || _isExcludedFromFees
       [to]) {
9      takeFee=false;
10   }
11   if (takeFee) {
12     uint256 fees=amount.mul(totalFees).div(100);
13     if (automatedMarketMakerPairs[to]) {
14       fees+=amount.mul(1).div(100);
15     }
16     amount=amount.sub(fees);
17     super._transfer(from,address(this),fees);
18   }
19   super._transfer(from,to,amount);
20   //other code here...
21 }
```

图 9 代币合约 DogeKing 代码片段

该合约就是收取手续费类型的代币合约.第 7～18 行，合约对不包含在 _isExcludedFromFees 映射中的地址收取转账手续费.由于该合约转账手续费不高，其用户也普遍知晓并遵从其手续费的机制，因此该合约不属于代币买卖漏洞.

以检出的漏洞合约(币安智能链地址 0xe9E3666f64c699529c9d3f9e2c506FF13fDe0E61)为例,对漏洞样本进行分析,由于该合约未开源,其字节码反编译后的代码片段如图 10 所示:

```
1  def transfer(address _to,uint256 _value): # not payable
2    if not unknown1e445a90:
3      stor8[caller]-= _value
4      stor8[_to]+= _value
5      log 0xfeddf252: _value,caller, _to
6      return 1
7    if stor7[caller]:
8      stor8[caller]-= _value
9      stor8[_to]+= _value
10     log 0xfeddf252: _value,caller, _to
11     return 1
12   if unknown1b355427Address== _to:
13     stor8[caller]-= _value
14     stor8[_to]+= _value
15     log 0xfeddf252: _value,caller, _to
16     return 1
17   if caller==unknown1b355427Address:
18     stor8[caller]-= _value
19     stor8[_to]+= _value
20     log 0xfeddf252: _value,caller, _to
21     return 1
```

图 10 0xe9 漏洞合约反编译后的代码片段

在第 2～6 行、第 7～11 行、第 12～21 行,均对用户转账行为进行了限制.第 2 行的变量 unknown1e445a90 是一个全局开关,值由属主控制,当为 false 时,所有普通用户才可以转账.第 7 行是一个特权数组,在数组中的地址才可以进行转账.第 12～21 行,全局变量 unknown1b355427Address 存储属主地址,如果转账用户为属主才允许转账.在币安智能链上观测该合约的交易可以发现,该合约代币 approve 函数同样存在上述类似的恶意限制,balanceOf 函数进行了恶意改写,balanceOf 函数代码片段如图 11 所示.

在满足全局开关变量 unknown1e445a90 为 false(第 2 行所示)、查询余额的地址为属主(第 3 行所示)、查询余额的地址属于特权数组(第 4 行所示)3 个条件之一的情况下,才会返回真实余额(第 6 行所示),否则所有用户将返回 1 个固定值的全局变量(第 5 行所示).恶意 balanceOf 函数让所有普通用户观察到自己的地址内有大量代币,诱使他们通过去中心化交易所进行投资买入,却无法卖出.该合约属于有后门的恶意合约.

```
1  def balanceOf(address _owner): # not payable
2    if unknown1e445a90:
3      if unknown1b355427Address!= _owner:
4        if not stor7[addr(_owner)]:
5          return stor6
6    return stor8[addr(_owner)]
```

图 11 balanceOf 函数代码片段

4.2.2 owner 权限转移漏洞

对 owner 权限转移漏洞检测工具进行测试,结果如表 5 所示:

表 5 owner 权限转移漏洞测试结果

指标	结果
总合约数	184
检出漏洞合约数量	15
确认有漏洞合约数量	6
运行用时	320 s/k 条指令

由表 5 可知,本文工具运行用时较长,误报率较低,检出率低.

经过分析,运行用时长主要是符号执行耗时过长,并且由于符号执行中对所有的 SSTORE 都尝试进行约束求解,使得时间消耗相较于代币买卖漏洞的检测用时高.检出率低和误报率低原因一是因为模型针对性强,准确度高,其次是因为 OpenZepplin 区块链应用标准中规范了访问控制权限的开发模式,帮助开发者避免一些容易疏忽的权限漏洞.

以检出的 CVE-2021-34273 漏洞合约为例,对漏洞样本进行分析,漏洞合约代码片段如图 12 所示.

该合约为 ERC-20 代币合约 BTC2X(B2X)的一部分,用来初始化合约中的 owner 值,以及定义限定修饰符 onlyOwner,并且可以将 ownership 转移给其他地址.

在漏洞合约中使用 owner 变量来存储合约拥

有者地址,但由于构造函数名的错误,合约允许任何人调用 owned()函数来修改合约的所有者,存在 owner 权限漏洞.第 4 行代码中,由于 owned()函数访问修饰符为 public,从而任何人可以调用该函数修改 owner 值为自己账户的地址.当恶意攻击者调用该函数修改 owner 值为自己账户的地址后,便可以调用代币合约中的其他函数来获利.

```
1  pragma solidity 0.4.24;
2  contract Owned {
3      address public owner;
4      function owned() public {
5          owner=msg.sender;
6      }
7      modifier onlyOwner {
8          require(msg.sender==owner);
9          _;
10     }
11     function transferOwnership(address newOwner) onlyOwner
12         public {
13         owner=newOwner;
14     }
15 }
```

图 12　检出的 CVE-2021-34273 漏洞合约代码片段

5　结　　论

针对代币买卖的后门漏洞以及 owner 权限转移漏洞的检测问题,本文提出了一种源代码和字节码层面的自动化检测方法.通过静态语义分析与符号执行相结合的技术,对漏洞点进行检测,并且通过符号执行自动化形成利用路径.通过在以太坊和币安智能链上主网合约进行测试,发现了新的代币买卖漏洞合约,通过实验,与 owner 权限转移漏洞相关的 CVE 也得到自动化的检测和复现.

参　考　文　献

[1] Ethereum. DeFi [EB/OL]. [2022-02-15]. https://ethereum.org/en/defi/

[2] PolyNetwork. PolyNetwork [EB/OL]. [2022-02-16]. https://poly.network/

[3] SlowMist. Poly network attacked [EB/OL]. (2021-08-11) [2022-05-26]. https://www.freebuf.com/vuls/284340.html

[4] SmartContractSecurity. SWC vulnerability [EB/OL]. [2022-02-16]. https://swcregistry.io/

[5] Ethereum. Solidity 0.8 breaking changes [EB/OL]. (2021-04-06) [2022-05-26]. https://docs.soliditylang.org/en/v0.8.11/080-breaking-changes.html

[6] Khan Academy. Solidity examples: Sending ether [EB/OL]. [2022-02-16]. https://solidity-by-example.org/sending-ether/

[7] Khan Academy. Solidity examples: Reentrancy [EB/OL]. [2022-02-16]. https://solidity-by-example.org/hacks/re-entrancy/

[8] Perez D, Livshits B. Smart contract vulnerabilities: Vulnerable does not imply exploited [C] //Proc of USENIX Security Symp. Berkeley, CA: USENIX, 2021

[9] Luu L, Chu D H, Olickel H, et al. Making smart contracts smarter [C] //Proc of the 2016 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 254-269

[10] Mueller B. Smashing ethereum smart contracts for fun and ACTUAL profit [EB/OL]. [2022-02-16]. https://conference.hitb.org/hitbsecconf2018ams/sessions/smashing-ethereum-smart-contracts-for-fun-and-actual-profit/

[11] Tsankov P, Dan A, Drachsler-Cohen D, et al. Securify: Practical security analysis of smart contracts [C] //Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2018: 67-82

[12] Nikolić I, Kolluri A, Sergey I, et al. Finding the greedy, prodigal, and suicidal contracts at scale [C] //Proc of the 34th Annual Computer Security Applications Conf. Piscataway, NJ: IEEE, 2018: 653-663

[13] Zhou E, Hua S, Pi B, et al. Security assurance for smart contract [C] //Proc of the 9th IFIP Int Conf on New Technologies, Mobility and Security (NTMS). Piscataway, NJ: IEEE, 2018: 1-5

[14] Mossberg M, Manzano F, Hennenfent E, et al. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts [C] //Proc of the 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). New York: ACM, 2019: 1186-1189

[15] Permenev A, Dimitrov D, Tsankov P, et al. VerX: Safety verification of smart contracts [C] //Proc of 2020 IEEE Symp on Security and Privacy (SP). Piscataway, NJ: IEEE, 2020: 1661-1677

[16] Tikhomirov S, Voskresenskaya E, Ivanitskiy I, et al. SmartCheck: Static analysis of ethereum smart contracts [C] //Proc of the 1st IEEE/ACM Int Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). New York: ACM, 2018: 9-16

[17] Feist J, Greico G, Groce A. Slither: A static analysis framework for smart contracts [C] //Proc of the 2nd Int Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Piscataway, NJ: IEEE, 8-15

[18] Kalra S, Goel S, Dhawan M, et al. ZEUS: Analyzing safety of smart contracts [C] //Proc of ISOC Network and Distributed System Security Symp. San Diego, USA: ISOC, 2018

[19] Trail of Bits. Echidna: A smart fuzzer for ethereum [EB/OL]. (2018-05-09) [2022-05-26]. https://blog.trailofbits.com/2018/03/09/echidna-a-smart-fuzzer-for-ethereum/

[20] Jiang B, Liu Y, Chan W K. ContractFuzzer: Fuzzing smart contracts for vulnerability detection [C] //Proc of the 33rd ACM/IEEE Int Conf on Automated Software Engineering. New York: ACM, 2018: 259-269

[21] He J, Balunović M, Ambroladze N, et al. Learning to fuzz from symbolic execution with application to smart contracts [C] //Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 531-548

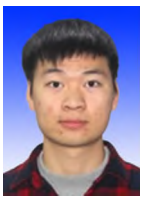
[22] 黄凯峰, 张胜利, 金石. 区块链智能合约安全研究[J]. 信息安全研究, 2019, 5(3): 192-206

[23] Piper Merriam. Ethereum signature database [EB/OL]. [2022-02-16]. https://www.4byte.directory/

刘宇航

硕士研究生.主要研究方向为软件安全、区块链安全.

wishucry@qq.com



刘军杰

硕士研究生.主要研究方向为区块链安全、智能合约安全.

jj_liu@stu.pku.edu.cn

文伟平

博士,教授,博士生导师.主要研究方向为系统与网络安全、大数据与云安全、智能计算安全.

weipingwen@pku.edu.cn