文章编号:1671-4229(2019)03-0059-09

智能合约安全综述:漏洞分析

赵淦森[1a,b,c],谢智健[1a,b,c*],王欣明[1a,b,c],何嘉浩[1a,b,c],刘学枫[1a,b,c],王锡亮[1a,b,c],周子衡[1c,2],田志宏[3],谭庆丰[3],聂瑞华[1a,b,c]

(1. 华南师范大学 a. 计算机学院,b. 广州市云计算安全与测评技术重点实验室,
c. 唯链区块链技术与应用联合实验室,广东 广州 510000;2. 唯链基金会,上海 200000;
3. 广州大学 网络空间先进技术研究院,广东 广州 510006)

摘 要: 加密货币比特币的出现带动了区块链技术的蓬勃发展,智能合约技术则是区块链技术中的一个技术高地. 目前以太坊中的智能合约应用受到大量的关注,创造了海量的价值应用,同时也带来了密集的攻击活动. 随着智能合约的数量越来越多,尤其是智能合约中的代码漏洞也逐渐被许多研究人员和恶意攻击者发现,造成了一系列重大的经济损失案件. 为了对智能合约技术的稳定性发展提供理论研究基础,文章针对以太坊上已知的智能合约漏洞进行了介绍、分类和总结,并对智能合约安全漏洞进行详细的原理阐述与场景代码复现.

关键词: 区块链; 以太坊; 智能合约; 安全漏洞

中图分类号: TP 311 **文献标志码:** A

新型的去中心化加密货币——比特币[1],自2009 年面世至今,在实践上的稳定性成功地吸引了业界和学术界的广泛关注[2]. 其蕴含的去中心化、可追溯性以及不可篡改等特征的底层架区块链技术,为数据存储与价值交换提供了一种安全可靠的去中心化模式. 基于区块链技术构造去中心化应用的前景被众多学术界和企业界人士,甚至是政府部门寄予厚望[3-4]. 近年来,随着区块链技术的发展,搭载智能合约功能的区块链技术是最重要的发展趋势之一,以太坊(Ethereum)则是其中最重要的区块链平台[5]. 智能合约是一种可编程式交易,可以被互不受信任的节点通过区块链共识机制(如工作量证明[1]、权益证明[6])自动且正确的执行. 目前借助智能合约所部署的应用已经涵盖了金融衍生品及服务、实体资产以及供应链等.

由于智能合约应用便利,大量高价值数字资产利用智能合约进行存储和转移,因此,容易受到攻击者的密集活动影响. 而且,因为区块链上的智能合约具有一经部署就不可篡改的特性,如果智能合约中存在了安全性漏洞,则无法对已部署到区块链上的合约代码进行补丁更新. 即是说,以太坊智能合约的正确执行是其结果有效性的必要条件,单独的正确性不足以证明智能合约的安全性. 这使得区块链智能合约从本质上存在天然的机制漏洞,容易造成经济损失. 目前针对以太坊智能合约的攻击已经有大量的例子[7-8]. 例如,在 2016 年 6 月,恶意攻击者通过以太坊 Solidity[9] 编程语言里递归调用的漏洞对智能合约 DAO 进行了攻击[10],盗取了大约 6 000 万美元,最终导致以太坊的硬分叉(Hard-fork).

实际上,在以太坊上开发和设计智能合约很容易引入安全性漏洞,其中很大一部分是由于以太坊虚拟机(Ethereum Virtual Machine,EVM)所支持的高级语言,如 Solidity(一种与 Javascript 类似的编程语言),其语法特性与开发人员的直觉认知存在误差,导致所编写的智能合约存在一些程序上的漏洞. 此外,造成漏洞的原因还有可能是来源

收稿日期: 2019-06-20; **修回日期:** 2019-06-25

基金项目: 国家重点研发计划资助项目(2018YFB1404402);广东省科技计划资助项目(2019B010137003,2016B030305006,2018A07071702);广州市科技计划资助项目(201804010314,2012224-12);唯链基金会资助项目(SCNU-2018-01)

作者简介: 赵淦森(1977—),男,教授,博士. E-mail: gzhao@m.scnu.edu.cn

* **通信作者**. E-mail: xiezhijian@m.scnu.edu.cn

于虚拟机层面的指令以及区块链协议内部执行的机制,比如智能合约之间的依赖问题会与智能合约部署时间存在关系.

在theDAO事件发生之后,出现了Oyente[11]、ContractFuzzer[12]、Teether[13]、Madmax[14]、Zeus[15]等检测智能合约安全性的工具.这些工具通过代码审计、测试、模型检测等方法,提供智能合约开发者在部署前发现漏洞的机会,以减少因为安全漏洞而导致的损失.

为了系统地总结并对智能合约漏洞问题进行分类,本文主要从以太坊上的智能合约漏洞类型进行阐述和归纳.在第一章里,本文对以太坊智能合约的程序编程模型进行介绍,第二章从Solidity编程语言的层面上对漏洞类型进行整理,第三和第四章则分别从EVM和区块链层面进行总结和归纳.第五章介绍了DoS类型的特殊漏洞.最后对智能合约的漏洞问题进行总结以及探讨智能合约的未来发展.

1 以太坊智能合约程序编程模型

以太坊上的智能合约由一个或多个合约账户所对应的合约代码以及持有Ether的用户地址所组成.以太坊上的用户需要通过发起交易的方式调用智能合约,并且执行过程和结果由矿工来打包写入到区块链中.

1.1 程序结构

以太坊上的智能合约主要是通过Solidity进行编写,Solidity是一种具有面向对象性质的弱类型语言.使用Solidity编写的智能合约主要包含状态变量的声明、函数、修饰符和构造函数的定义等部分.在以太坊上部署智能合约时,开发人员需要先将使用Solidity编写的智能合约代码编译为以太坊虚拟机可执行的二进制代码.而在编译过程中,智能合约代码的入口会插入一小段称为函数选择器(Function Selector)的代码,用以在调用函数时快速跳转到相应函数并加以执行.在编译完成后,可以通过客户端发送合约创建交易(Contract Creation Transaction),或通过其他合约执行特殊的EVM指令CREATE来部署该编译后的智能合约.

1.2 调用方式

在以太坊成功部署的智能合约,可以通过三种方式调用合约中的公共函数(External/Public).第一种方式是通过客户端发送消息调用交易(Message Call Transaction),其中包含了数据参数以及目标函数签名的哈希值.这种函数调用方式必须在交易得到确认后才能生效.另外,矿工会对该交易收取Gas来作为执行函数时所需要的代价,因此,该方式是一种写操作,即会对消息调用者的账户的余额以及合约的状态进行更改.第二种方式则通过另一个合约来间接的调用,这种方式最终可以被追溯成另一笔消息调用交易.最后一种方式是通过客户端调用view(或pure)函数,但这种方式并不会改变合约的状态,也不需要耗费Gas.

1.3 存储结构

以太坊虚拟机(Ethereum Virtual Machine,EVM)的存储方式可以分为四种:栈(Stack)、状态存储(Storage)、虚拟机内存(Memory)和只读内存.EVM是基于栈的虚拟机,栈中的每一个元素的长度是256位,基本的算术运算和逻辑运算都是使用栈完成.虚拟机内存实际上是一个连续的数组空间,用于存放如字符串等较复杂的数据类型.状态存储是key-value的存储结构,用于持久化数据.与栈和虚拟机内存不同,状态存储的值会被记录到以太坊的状态树当中.只读内存是EVM最特殊的一种存储结构,主要用于存放参数和返回值.

2 代码层漏洞

Solidity(类似于JavaScript语法)是图灵完备的高级语言,是目前智能合约编写的主要语言之一,虽然它为开发者提供便利的编程语言,但是使用不当会使得编写的智能合约异常脆弱,容易受到攻击者攻击,下面介绍及分析存在于代码层的漏洞,对这些漏洞进行案例还原.

2.1 可重入漏洞

可重入漏洞是目前以太坊系统中最著名的漏洞之一,黑客利用该漏洞在2016年成功转移超过360万个以太币,盗取了大约6 000万美元.该漏洞主要是因为智能合约调用一个未知的合约地址,攻击者可以精心构造一份智能合约,在回调函数中加入恶意代码.当智能合约向该恶意合约地址发送以太币的时候,合约上的恶意代码将会被触发,这段恶意代码通常会进行开发者意想不到的操

作. 图 1 例子根据 DAO 合约进行改造而来.

```
1   contract MyStore {
2       function depositFunds( ) public payable {
3         balances [msg. sender]  + = msg. value;
4       }
5       function withdrawFunds ( uint _wei) public {
6           require( balances [msg. sender]  > = _wei) ;
7           require( Withdraw_wei  < = withdrawalLimit) ;
8           require( msg. sender. call. value( _wei) ( ) ) ;
9           balances [msg. sender]  - = _wei;
10        lastWithdrawTime [msg. sender]  = now;
11      } }
```

图 1　可重入漏洞例子

Fig. 1　The example of DAO

图 1 代码有着存款取款的正常功能,但是存在可以进行可重入的漏洞,攻击者可以利用第 8 行的代码精心构造如图 2 代码进行攻击.

```
1   contract Attack {
2       function AttackMyStore( ) public payable {
3       // 开始攻击
4       myStore. withdrawFunds( 1 ether) ;
5     }
6       function ( ) payable {
7         if ( myStore. balance  > 1 ether)  {
8           myStore. withdrawFunds( 1 ether) ;
9       } }
```

图 2　可重入漏洞攻击

Fig. 2　Dao attack

当攻击者调用 Attack 合约中的 AttackMyStore 函数时,将会运行第 4 行代码,这时,会触发 MyStore 合约中的 withdrawFunds 函数,运行到第 8 行时,将会自动触发 Attack 合约的 fallback 函数,继而触发 MyStore 中的 withdrawFunds 函数,形成一个不断循环可重入的过程,攻击者将拿到原本存储在 MyStore 中的所有以太币.

Solidity 文档建议使用“检查 - 生效 - 交互”的模型编写代码以避免可重入漏洞. 若将图 1 中的第 9 行代码和第 8 行代码进行交换,则攻击者无法通过可重入的漏洞对该合约进行攻击.

2.2　危险的 DELEGATECALL

智能合约在使用 DELEGATECALL 时,会调用存在于其他智能合约中的代码,但是会保持当前的上下文关系,这种特性虽方便了开发者使用,却加大了设计安全代码库的难度,攻击者会利用保持上下文不变的特性修改原有上下文的内容从而进行攻击. 具体参考图 3.

```
1   contract Lib {
2       uint public start;
3       uint public end;
4       function set_start( uint _start)  public {
5           start  = _start;
6       }
7       function set_end( uint _end)  public {
8           end  = _end;
9       } }
```

图 3　供其他合约使用的库代码

Fig. 3　The code of Libraries

图 3 中的库代码是用来控制合约的起始时间和终止时间,下面构造是对这个代码库的使用智能合约(图 4) .

```
1   contract UseLib {
2       address public lib;
3       uint public end;
4       uint public start;
5       bytes4 constant fibSig  = bytes4( sha3( " set_start
( uint256) " ) ) ;
6       function chage_end( uint _start) {
7           lib. delegatecall( fibSig, _start) ;
8       } }
```

图 4　含有 DELEGATECALL 的合约代码

Fig. 4　DELEGATECALL in contract

这个漏洞和智能合约对于 storage 变量的存储位置有关,可以看到,在智能合约 Lib 中,第一个变量 start 存储在合约的第一个位置,即 slot [0] 中,第二个变量 end 存储在 slot [1] 中. 在合约 UseLib 中,第一变量 lib 存储在 slot [0] 中,第二个变量 end 与第三个变量 start 分别存储在 slot [1] 与 slot [2] . 当运行合约 UseLib 中的第 7 行代码的时候,会调用合约 Lib 中的 set_start 函数,由于 DELEGATECALL 中的上下文不变的特性,本来修改 slot [0] 中的内容并不是开发者预想中的变量 start,而是变成当前上下文中的变量 lib,因此,修改 lib 地址变量后,可为攻击者提供有效的攻击途径.

2017 年,恶意用户通过此漏洞攻击 Parity 钱

包,导致价值将近1.7亿美元的ETH被冻结.

2.3 算术上溢/下溢

上溢/下溢在很多程序语言中都存在,在以太坊虚拟机中,uint类型最大为256位,超过此范围会出现上下溢情况. 在2018年美链中,使用了ERC20[16]中的batchTranfer函数,在这个函数的实现中存在上溢的危险,攻击者利用这一漏洞,造成了美链的巨大的经济损失. 下面看美链的案例说明[17](图5).

```
1  contract PausableToken{
2   function batchTransfer (address [] _receivers, uint256 _
value) {
3          uint cnt = _receivers.length;
4          uint256 amount = uint256(cnt) * _value;
5          require(cnt > 0 && cnt <=20);
6          require(_value > 0);
7          require(balances[msg.sender].sub(amount));
8          for (uint i = 0; i < cnt; i++) {
9            balances[_receivers[i]].add(_value);
10           Transfer(msg.sender, _receiver[i], _value);
11           }
12           return true;
13      }}
```

图5 美链接口batchTranfer函数的具体实现

Fig. 5 BEC's code implementation

该合约虽然利用了库代码SafeMath进行安全运算,但是由于开发者的疏忽,仍然给攻击者造成上溢的机会. 攻击者调用batchTransfer函数,_value赋值57896044618658097711785492504343953926634992332820197287920039565648199 68,_receivers的值为攻击者提供的2个地址,在第8行代码中,由于算术运算发生上溢,从而导致运算的结果为0,第5、6、7行的代码无法侦测异常,最终,攻击者成功攻击,转走账户中所有的金额.

2.4 默认函数类型

目前Solidity语言是以太坊智能合约使用最广的语言,在Solidity中,默认函数的类型为public,默认变量的类型为pirvate. 由于开发者的疏忽,会使合约产生漏洞,使得攻击者能对合约进行攻击.

图6的合约是一简单的游戏,当调用者的地址的最后八位为0时即为游戏的胜利者,可以把Game合约中的金额全都拿走,但是由于开发者的使用不当,函数_sendWinnings的类型并没有显示提供,导致该函数的类型为默认类型,所有的区块链用户都能取走该智能合约中的所有金额.

```
1  contract Game{
2      function withdrawWinnings() {
3          //调用者地址最后八位都为0为游戏胜利者
4          require(uint32(msg.sender) == 0);
5          _sendWinnings();
6       }
7       function _sendWinnings() {
8           msg.sender.transfer(this.balance);
9       }}
```

图6 合约函数类型均为默认类型

Fig. 6 Default type

2.5 外部调用的返回值

在智能合约中,一般通过transfer()、send()、call()等函数进行对其他账户的转账[9],transfer()函数会自动检查转账结果,当转账失败的时候会自行抛出异常,但是send()与call()函数失败时不会自行抛出异常,而是继续往下执行剩余代码,进而攻击者可以利用此特性,故意转账失败来达到攻击目的,见图7.

```
1  contract Game{
2      bool public SendOut = false;
3      address public winner;
4      uint public reward;
5      function Send_Reward() public {
6          require(! SendOut);
7          winner.send(reward);
8          SendOut = true;
9      } }
```

图7 智能合约缺少外部调用检查

Fig. 7 Exterior appropriation check after intelligent contract default

图7中,正常的情况下执行第7行代码的时候,胜利者将会得到本次游戏的奖励,但是,由于某种原因转账失败,此时并没有检查winner.send()的返回值,代码会继续执行,在第8行代码中,将SendOut变量更改为true,从而导致该游戏中的金额永远锁在账户中无法提取.

3 虚拟机层

在以太坊系统中的智能合约必须都要依赖于

EVM 的执行,有时候由于编译问题,导致程序的执行过程不符合开发者执行的预期,最终攻击者可以利用此类漏洞进行攻击.

3.1　短地址攻击

当向智能合约发送交易的时候,根据智能合约 ABI 规范对函数参数进行编码. 在 ABI 规范中输入的地址参数必须是 20 字节,当输入的地址少于 20 个字节的时候,EVM 将用 0 补全以满足要求. 如开发者没有严格按照此规范对输入进行检查,将会使得攻击者有机会对合约发起攻击. ERC20 转账接口的定义如下:

```
function transfer( address to, uint tokens) public returns ( bool success) ;
```

在正常的转账例子里,输入的地址参数为 0xca35b7d915458ef540ade6068dfe2f44e8fa733c,输入的 tokens 参数为 100,这时候 EVM 会对 transfer( ) 的函数调用进行编码,最终的编码为 a9059cbb000000000000000000000000ca35b7d9154–58ef540ade6068dfe2f44e8fa733c0000000000000000–0000000000000000000000000000000000056bc75e2d631–00000,其中前 4 个字节(a9059cbb) 为 transfer( ) 函数,紧接着的 32 个字节为第一个 address 参数,最后的 32 个字节为第二个 uint256 参数,这时的 56bc75e2d63100000 对应着参数中的 100token.

而在短地址转账中,不足 20 字节的地址参数会出现意想不到的效果,当输入的 tokens 参数为 100,输入的地址参数为 0xca35b7d915458ef–540ade6068dfe2f44e8fa73 时(注意这里的地址参数比正常转账中少了两位) ,为了补位,EVM 会在数据后面补足 0,因此,最终编码为 a9059cbb–000000000000000000000000ca35b7d915458ef540a–de6068dfe2f44e8fa7300000000000000000000000000–0000000000000000000000056bc75e2d6310000000,其中,由于补全的原因,第二个参数这时会由原来的 56bc75e2d63100000 变成 56bc75e2d6310000000,从原来的 100tokens 变成 25600tokens,进而造成攻击. 目前该攻击在实际运用中还没发现.

3.2　Tx. Origin 漏洞

在智能合约中存在一个全局变量 tx. origin,它返回发起本次交易的调用者,因此,攻击者能利用此漏洞创建一个类似于陷阱的合约,对调用该智能合约的用户造成攻击. 图 8 是简单的取款操作.

```
1   contract MyStore{
2       address public owner;
3       constructor ( address _owner) {
4           owner = _owner;
5       }
6       function withdrawAll( address _recipient) public {
7           require( tx. origin = = owner) ;
8           _recipient. transfer( this. balance) ;
9       }
10  }
```

图 8　含有 tx. origin 的智能合约

Fig. 8　The contract with tx. origin

攻击者可以利用图 8 的 tx. origin 漏洞构造一份相应的智能合约,见图 9.

```
1   contract Attacker {
2       MyStore myStore;
3       address attacker;
4       constructor ( MyStore _ myStore, address _attac) {
5           myStore = _ myStore;
6           attacker = _attac;
7       }
8       function ( ) payable {
9           myStore. withdrawAll( attacker) ;
10      } }
```

图 9　对 tx. origin 漏洞进行攻击

Fig. 9　Use tx. origin to attack

当使用者被骗后,用自己的账户向智能合约 Attacker 发起转账的时候,将会触发 fallback 函数,该函数会调用合约 MyStore 中的 withdrawAll 函数,因为 tx. origin 的缘故,导致无法检测出异常,这时,发起调用的账户中的金额将会全部转到 attacker 的账户里.

4　区块链层

目前市面上有多种区块链系统,最知名的是比特币与以太坊系统,不同的系统会有着不同的特性,而对于竞争记账权或者块产生的规则大不相同,由于这些细微的差别,从理论上来说有机会让攻击者能利用这些差别进行攻击.

4.1　打包交易的顺序

由于区块链通过例如 PoW [1] 或者 PoS [18] 等共识算法竞争记账权的,交易打包的顺序由竞争到记账权的节点决定,不同的节点打包交易的顺序

不一定相同,因此,交易顺序不同往往会引发不同的结果.

图10中的合约是简单的维护商品价格和售卖商品的流程,假设A用户频繁地调用updatePrice()更改商品价格,B用户调用buy()购买商品,当这些交易在几乎相同的时间进行提交,这时由于交易打包顺序由矿工节点决定,B用户可能买到商品的价格由于频繁的价格更新而与购买时价格不符,从而导致B用户的经济损失.

```
1   contract Market{
2       uint public price;
3       uint public stock;
4       /…/
5       function updatePrice (uint _price) {
6           price = _price;
7       }
8       function buy (uint quant) returns (uint) {
9           stock - = quant
10      }}
```

图10 维护商品价格的智能合约

Fig. 10 A contract to update price

4.2 基于时间戳的随机变量漏洞

基于时间戳进行随机数选取是大多数程序中常见的操作,但是由于时间戳在块产生的时候由打包节点决定,可以在大约900 s时在误差范围内调整时间戳的值[19],因此,攻击者可以利用这一特性生成有利于自己的时间戳进行攻击,见图11.

```
1   contract Game {
2       uint public pastBlockTime;
3       function () public payable {
4           require(msg.value = = 10 ether);
5           require(now ! = pastBlockTime);
6           pastBlockTime = now;
7           if(now % 15 = = 0) {
8               msg.sender.transfer(this.balance);
9           }
10      }}
```

图11 合约基于时间戳进行随机数选择

Fig. 11 Use timestamp to generate number

图11中的合约就是一个碰运气的游戏,参与者每次提供10个以太币参加,有1/15的可能性获得该合约账户中所有的金额,从上述提到的原理可知,矿工节点可以控制块的时间戳的具体值,矿工可以一直利用时间戳导致now%15永远不为0,当累计到一定程度后再对账户里的金额进行提取.

5 拒绝服务

拒绝服务攻击是一种针对以太坊常见的攻击类型,其目的是使以太坊网络资源和系统资源耗尽,从而无法对外提供正常的服务.为了保证整个以太坊网络中各个节点的一致性,智能合约的运行需要利用网络同步到以太坊中的各个节点,因此,在以太坊中部署和执行智能合约都需要很大的资源开销.虽然EVM通过设置gas机制(用户需要为部署和执行智能合约而付出代价,而这个代价就是以太币)来防止某些恶意用户浪费以太坊系统中的系统和网络资源,但是,智能合约依然是以太坊面临DOS攻击时较为脆弱的一个环节,攻击者可以在智能合约中插入消耗系统资源高的代码,使得以太坊系统忙于执行这些恶意代码而无暇对外界正常提供服务.除此之外,开发人员编写智能合约也会引入DOS的漏洞,攻击者通过利用这些漏洞使智能合约陷入拒绝服务的状态当中.称前者为主动式DOS,后者为被动式DOS.

5.1 主动式DoS

主动式DOS攻击,主要是指通过部署和执行拥有大量消耗系统资源高但所需gas低廉的EVM指令的智能合约,使攻击者可以在低成本的条件下让以太坊整个网络陷入拒绝服务的状态[20].2016年,兼有系统消耗高和gas低廉的指令EXCODESIZE和SUICIDE被攻击者发现,并利用其对以太坊网络进行攻击,导致以太坊网络交易速率下降.虽然以太坊通过修改gas机制阻止这种攻击的再次出现[21-22],但严重影响了公众对以太坊等区块链系统的信心.

5.1.1 EXCODESIZE DOS攻击[23]

EXCODESIZE是EVM用于查寻某个合约账户代码规模的指令[5],会对磁盘I/O造成很大的负担,普通用户需要消耗大量资源来执行EXCODESIZE指令.同时EXCODESIZE拥有很低的调用成本(在1.3.5的geth之前的版本仅仅需要20 gas).为此,攻击者可以通过部署和调用有大量的EXCODESIZE指令的智能合约,造成以太坊节点频繁读写磁盘,降低以太坊网络交易吞吐量.在

geth 1.6 的版本之后,EXCODESIZE 指令被上调至 700 gas 的开销,相比之前 20 gas 的开销,增长了 34 倍[22]. 因此,使用大量 EXCODESIZE 进行 DOS 攻击的成本上升,此类的攻击才得到了初步的遏制.

5.1.2　空账户 DOS 攻击

空账户是指没有代码,没有以太币的以太坊账户,这些账户不存在任何功能但需要存储在以太坊的状态树中. 在 2016 年 10 月,攻击者通过创建大量的空账户对以太坊网络进行攻击,导致以太坊浪费了大量的存储资源,增加同步时间,甚至导致了 11 月份针对修复此类攻击的"Spurious Dragon"分叉[21]. 攻击者通过在母合约中创建子合约,并且循环调用子合约的 SUICIDE[5],把子合约的以太币(实际并无以太币)发送到指定的账户. 这里攻击者所指定的账户在以太坊中是不存在的,因此,以太坊会创建账户地址并纪录在状态树当中. 通过这种方式创造一个新账户仅需要 90 gas,但以太坊实际的系统开销是极其巨大的. 在 2016 年 10 月的 DOS 攻击中,攻击者通过这样的方式创造了超过 1 900 万的空账户,给以太坊的存储和网络同步造成了很沉重的负担. "Spuerious Dragon"分叉后,以太坊通过规定 SUICIDE 的 gas 消耗为 5 000,若在执行时创建了新账户,其消耗应该是 25 000 gas,并且节点可以删除之前攻击所产生的僵尸账户,以减少空账户所带来的负面影响.

5.2　被动式 DOS

被动式 DOS 主要是指由于智能合约在编写过程中所引入的漏洞,使得智能合约在部署后陷入无法对其他用户服务的状态[14]. 被动式 DOS 与其说是一种攻击,不如说是一种漏洞,因为开发者在编写过程中的考虑不周而导致智能合约在部署和运行一段时间后处于拒绝服务的状态.

5.2.1　无界循环 DOS 攻击

无界循环 DOS 是指智能合约的循环体结束条件处于开发者不可控制的状态当中,完全由用户的输入所确定. 因此,智能合约运行的 gas 开销也完全被输入所决定,很多由 out-of-gas 异常而产生的问题有很大的概率并没有被开发人员所考虑到,其中就包括 DOS 情况的出现. 一个很常见的带有无界循环 DOS 漏洞的代码如图 12,当用户数量达到一定程度的时候,执行图 11 代码所需要的 gas 变得极其巨大,超过调用者所限定的 gas-limited 甚至是 block-limited. 这就造成了一种情况,用户每次调用这段代码时都会出现 out-of-gas 的异常并回滚至调用前的状态,也就是处于拒绝服务的状态.

```
1  for (uint i = 0; i < balance.lengh; i++){
2        balances[i] = balances[i] * 1.1;
3  }
```

图 12　DOS 例子 1

Fig. 12　Example One of DOS

5.2.2　Gas-less send DOS 攻击

为了避免 reentrancy 漏洞,send() 函数的 gas 被强制指定为 2 300,一旦在转账过程中执行 fallback 函数所耗费的 gas 超过 2 300 就会触发 out-of-gas 的异常并触发以太坊的回滚. 考虑图 13 代码,代码所实现的功能较为简单: 向所有的投资者发放分红. 但考虑这样的一种情况,当中某些恶意投资者在自己的 fallback 函数带有了 gas 消耗较大的代码,而上述代码的运行过程必定会执行这个 fallback 函数,造成 out-of-gas 异常的出现. 因此,这段代码就被这个恶意投资者锁定,无法再对外提供应有的服务.

```
1  for (uint i = 0; i < investor_addrs.length; i++){
2        require(invertor_addrs[i].send(benefit));
3  }
```

图 13　DOS 例子 2

Fig. 13　Example two of DOS

5.2.3　Overflow DOS

Solidity 提供的数值类型较多,在使用过程中稍有不慎就会造成数值上溢. 若在循环结构中错误使用 Solidity 类型,也会造成 DOS 情况的出现. 考虑图 14 代码,在 Solidity 中 uint 在执行特定类型时,一般是指 8 位的无符号数,其表达的范围是 0 ~ 255. 当用户数量少于 256 时,该代码能够正常对所有用户提供服务,但是一旦用户数量超过 256 时,因为 i 在循环递增中会发生溢出,处于后面的用户就不会得到和前面用户相同的服务. 换言之,大于 255 的编号的用户被拒绝访问.

```
1  for (uint i = 0; i < users.length; i++){
2     service(users[i]);
3  }
```

图 14　DOS 例子 3

Fig. 14　Example three of DOS

6 总结与展望

智能合约作为以太坊的执行代码,有着至关重要的作用,因此,智能合约的安全与漏洞自然而然成为研究的关注点,本文分别从以太坊的代码层、虚拟机层、区块链层分层地介绍和总结了目前以太坊智能合约所存在的漏洞及其实例,而由于拒绝服务在目前的研究中存在有非常多的变种,而且在执行此类攻击时会涉及到上面三个层级,本文把它作为特殊的漏洞类型单独进行详述. 通过本文对目前以太坊存在的漏洞的分析,希望能为后续的研究工作提供一些总结,并帮助智能合约开发者们尽量避免这些已知的漏洞,从而提高智能合约的编码的规范性和有效性. 同时,也可以为目前以太坊的开发者修复这些与区块链平台相关的漏洞,促进以太坊智能合约的不断更新和完善.

参考文献:

[1] Nakamoto S, Bitcoin: A peer-to-peer electronic cash system [EB/OL]. [2019-07-05]. http://www.bitcoin.org.

[2] Bonneau J, Miller A, Clark J, et al. SoK: research perspectives and challenges for bitcoin and cryptocurrencies [C] //2015 IEEE Symposium on Security and Privacy, San Jose: CA, 2015: 104-121.

[3] 工业和信息化部信息中心. 2018 年中国区块链产业白皮书 [EB/OL]. [2019-07-05]. http://www.miit.gov.cn/n1146290/n1146402/n1146445/c6180238/-part/6180297.pdf.

[4] UK Government Chief Scientific Adviser. Distributed ledger technology: Beyond blockchain [EB/OL]. [2019-07-05], https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

[5] Wood G. Ethereum: A secure decentralised generalised transaction ledger [EB/OL]. [2019-07-05]. https://ethereum.github.io/yellowpaper/paper.pdf.

[6] Cachin C, Vukolić M. Blockchain consensus protocols in the wild [C] //The International Symposium on DIStributed Computing, Austria: Vienna, 2017: 1-16.

[7] Atzei N, Bartoletti M, Cimoli T. A survey of attacks on ethereum smart contracts (SoK) [C] //Principles of Security and Trust, Berlin Heidelberg: Springer, 2017: 164-186.

[8] Li X, Jiang P, Chen T, et al. A survey on the security of blockchain systems [C] //Future Generation Computer Systems, 2017: 1-13.

[9] Ethereum. Solidity documentation [EB/OL]. [2019-07-05]. https://solidity.readthedocs.io.

[10] Foundation E. CRITICAL UPDATE Re: DAO vulnerability [EB/OL]. [2019-07-05]. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/.

[11] 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-CCS'16, Austria: Vienna, 2016: 254-269.

[12] Jiang B, Liu Y, Chan W K. ContractFuzzer: fuzzing smart contracts for vulnerability detection [C] //Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering-ASE 2018, France: Montpellier, 2018: 259-269.

[13] Krupp J, Rossow C. Teether: Gnawing at ethereum to automatically exploit smart contracts [C] //27th USENIX Security Symposium, USA: Baltimore, MD, 2018: 1317-1333.

[14] Grech N, Kong M, Jurisevic A, et al, MadMax: surviving out-of-gas conditions in Ethereum smart contracts [C] //Proceedings of the ACM on Programming Languages, New York: ACM, 2018: 1-27.

[15] Kalra S, Goel S, Dhawan M, et al. ZEUS: Analyzing safety of smart contracts [C] //Proceedings 2018 Network and Distributed System Security Symposium, San Diego: CA, 2018.

[16] Fabian Vogelsteller and Vitalik Buterin. EIPs/eip-20.md at master · ethereum/EIPs [EB/OL]. [2019-07-05]. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[17] BeautyChain. BecToken smart contract [EB/OL]. [2019-07-05]. https://etherscan.io/address/0xc5d105e63711398af9bbff092d4b6769c82f793d#contracts.

[18] Bitcoin Forum. Topic: Proof of stake instead of proof of work [EB/OL]. [2019-07-05]. https://bitcointalk.org/index.

php? topic = 27787.0.

[19] Ethereum Foundation. Block validation algorithm [EB/OL]. [2019-07-05]. https://github.com/ethereum/wiki/wiki/BlockProtocol-2:0#block-validation-algorithm.

[20] Chen T, Li X, Wang Y, et al. An adaptive gas cost mechanism for ethereum to defend against under-priced DoS attacks [C] // International Conference on Information Security Practice and Experience, Cham: Springer, 2017: 3-24.

[21] Fabian Vogelsteller and Vitalik Buterin. EIPs/eip-161.md at master · ethereum/EIPs [EB/OL]. [2019-07-05]. https://github.com/ethereum/EIPs/blob/ master/EIPS/eip-161.md.

[22] Fabian Vogelsteller and Vitalik Buterin. EIPs/eip-150.md at master · ethereum/EIPs [EB/OL]. [2019-07-05]. https://github.com/ethereum/EIPs/blob/ master/EIPS/eip-150.md.

[23] Vitalik Buterin. Transaction spam attack: Next steps [EB/OL]. [2019-07-05]. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.

A survey on smart contract: Vulnerability analysis

ZHAO Gan-sen[1a,b,c], *XIE Zhi-jian*[1a,b,c], *WANG Xin-ming*[1a,b,c], *HE Jia-hao*[1a,b,c], *LIU Xue-feng*[1a,b,c], *WANG Xi-liang*[1a,b,c], *ZHOU Zi-heng*[1c,2], *TIAN Zhi-hong*[3], *TAN Qing-feng*[3], *NIE Rui-hua*[1a,b,c]

(1 a. School of Computer Science, b. Guangzhou Key Laboratory of Cloud Computing Security and Assessment Technology, c. VeChain blockchain technology and application joint laboratory, South China Normal University, Guangzhou 510000, China; 2. VeChain Foundation, Shanghai 200000, China; 3. Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510000, China)

Abstract: The emergence of cryptocurrency bitcoin has driven the vigorous development of blockchain technology, and smart contract technology is a technical highland of blockchain technology. At present, smart contract applications in ethereum have received a lot of attention, creating a lot of value applications, and at the same time bringing intensive attack activities. With the increasing number of intelligent contracts, especially the code loopholes in intelligent contracts have been gradually discovered by many researchers and malicious attackers, which has caused a series of serious economic losses. In order to provide theoretical research basis for the stable development of intelligent contract technology, this paper introduces, classifies and summarizes the known intelligent contract vulnerabilities in ethereum, and elaborates the principle and scene code of intelligent contract security vulnerabilities in detail.

Key words: blockchain; ethereum; smart contract; security vulnerabilities

【责任编辑: 周　全】