摘要

智能合约是运行在以太坊平台上的可执行代码，具有一旦部署就不可改变的特点。由于部署之后无法更改合约内容，如果智能合约存在安全漏洞，则攻击者可以利用漏洞攻击合约，对整个项目造成重大损失。针对Solidity智能合约的安全问题，本课题拟研究Solidity智能合约安全漏洞问题，采用基于编译的方法，针对以太坊平台上Solidity语言编写的智能合约，给出基于Antlr工具定义的Solidity子集的词法规则和语法规则，使用Antlr工具自动生成Solidity智能合约的语法分析树，然后给出整型溢出等7种经典的安全漏洞的检测模式和算法。最后基于上述研究，设计并实现了Solidity语言智能合约安全漏洞的分类检测工具，能够实现对上述7种安全漏洞的检测。

**关键词：**Solidity;Antlr;以太坊；智能合约；漏洞检测

**Abstract**

Smart contracts are executable code that runs on the Ethereum platform and has the characteristic that they cannot be changed once deployed. Since the contract content cannot be changed after deployment, if there are security holes in the smart contract, the attacker can use the holes to attack the contract, causing significant losses to the entire project. In view of the security issues of Solidity smart contracts, this topic intends to study the security vulnerabilities of Solidity smart contracts, using a compilation-based approach, and for the smart contracts written in the Solidity language on the Ethereum platform, lexical rules based on the Solidity subset defined by the Antlr tool And grammar rules, use the Antlr tool to automatically generate the grammatical analysis tree of the Solidity smart contract, and then give 7 classic security vulnerability detection modes and algorithms such as integer overflow. Finally, based on the above research, we designed and implemented a classification detection tool for the security vulnerabilities of Solidity language smart contracts, which can realize the detection of the above seven security vulnerabilities.

**Keywords:** Solidity; Antlr; Ethereum; smart contract; vulnerability detection

# 引言

## 1.1研究背景与意义

### 1.1.1智能合约与以太坊

智能合约产生于20世纪末，由杰出的计算机科学家Nick Szabo最早提出,他当时将智能合约描述一个数字协议，在这个协议中可以定义承诺，然后该承诺可以自动执行[1]。但是由于当时缺乏支持可编程合约的系统和技术，智能合约没能得到真正实现。

2008年由中本聪提出的区块链技术为智能合约的实现和推广带来契机，区块链技术具有的去中心化、过程透明、不可篡改等特点适用于智能合约[2]。虽然最初的区块链技术仅应用于金融转账，但伴随区块链技术的不断成熟，2013年末， Vitalik Buterin启动了一个能够在区块链上实现智能合约的开源系统——以太坊[3]。以太坊是第一个提供完善的智能合约开发框架的区块链，将区块链技术的应用从金融转账扩展到了任意应用上，因此它也被称为区块链2.0的代表。以太坊通过其专用[加密货币](https://baike.baidu.com/item/%E5%8A%A0%E5%AF%86%E8%B4%A7%E5%B8%81)[以太币](https://baike.baidu.com/item/%E4%BB%A5%E5%A4%AA%E5%B8%81/20857686)和提供[去中心化](https://baike.baidu.com/item/%E5%8E%BB%E4%B8%AD%E5%BF%83%E5%8C%96/8719532)的以太坊[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA/104440)来处理[点对点](https://baike.baidu.com/item/%E7%82%B9%E5%AF%B9%E7%82%B9/7452984)合约[4]。以太坊虚拟机(EVM)是以太坊平台上智能合约的运行环境。以太坊虚拟机内是完全独立的环境，智能合约代码运行在以太坊虚拟机时是没有网络，文件系统或是其他进程的，甚至会被限制访问其他的智能合约[5]，因此运行在以太坊平台上的智能合约具有一旦部署就不可改变的特点。通常智能合约的开发流程是首先使用solidlity语言编写逻辑代码，之后再通过编译器编译数据，最后再发布到以太坊平台上。

智能合约把传统的现实合约以编写代码的方式将合约规则写入计算机中，形成计算机协议，之后以信息化方式传播、验证和执行合同，允许在没有第三方的情况下进行可信交易，并且这些交易均可追踪且不可逆转[6]。与传统合约相比智能合约的成本更低，可信度更高。使用智能合约来替代人为履约，有助于信任成本的下降，改善交易行为达成效率[7]，加上以太坊平台的出现，智能合约得到迅速发展。目前主流的以太坊平台智能合约的编写语言有solidity语言、Serpent语言、Mutan语言等[8]。由于Solidity的语法类似于JavaScript，更加容易掌握和使用，因此Solidity成为了以太坊智能合约开发中的首选语言。

### 1.1.2智能合约安全

由于智能合约具有一旦部署就不可改变的特点，如果部署在以太坊平台上的智能合约存在安全漏洞问题，攻击者就可以通过发布包含恶意代码的“智能合约”，经过一系列操作，控制区块链网络中的所有节点，进而对项目造成毁灭性打击。2016年6月THE DAO事件中，攻击者利用简单的递归函数漏洞，造成价值2亿美元的损失；2017年7月在智能合约Parity中触发的多签名钱包合约漏洞，冻结了高达1亿美元价值的以太币；2018年4月 BEC智能合约由于存在整型溢出漏洞，损失64亿资产[9]。

NCC Group总结了在智能合约中出现频率最高的10类安全问题，包括整数溢出、代码重入、访问控制等[10]。为帮助开发安全的智能合约，针对这些安全问题，IDELBERGER等人提议使用基于逻辑的语言而不是程序语言[11],以太坊白皮书中也给出了相应的一些编程建议，如使用SafeMath库、先更新状态变量再执行写操作、标记不受信任的合约等[12]。但由于仍然有智能合约不遵守建议和规范，智能合约中依旧存在安全漏洞。Nikolic等人2018年的研究报告指出，目前的以太坊中，有约89%的智能合约的代码存在安全漏洞或隐患[13]。由于智能合约极可能存在安全漏洞，智能合约在部署前应当对合约进行安全漏洞检查，避免漏洞对智能合约造成价值损失。

常规的Solidity语言编译器无法进行漏洞检测，针对Solidity智能合约的安全问题，本课题拟研究Solidity智能合约安全漏洞的分析，采用基于编译的方法，针对以太坊平台上Solidity语言编写的智能合约，首先基于Antlr工具定义Solidity子集的词法规则和语法规则，使用Antlr工具自动生成Solidity智能合约的语法分析树；然后给出整型溢出等7种经典的安全漏洞的检测模式和算法；最后基于上述研究，设计并实现Solidity语言智能合约安全漏洞分类检测工具，该工具能够完成对上述7种漏洞的检测。该工具主要用途是为使用Solidity语言开发智能合约的开发者提供安全指导，帮助其开发出更加安全的智能合约。

## 1.2国内外研究现状

国外的智能合约审计工具中比较完善的是瑞士的Securify[14]。Tsankov等人针对智能合约的安全问题，采用反编译的方法，基于符号抽象，开发了一款轻量级的智能合约可扩展安全验证器——Securify。Securify 将安全属性的模式分为遵从性模式和违反性模式，将所有智能合约行为分为违规、警告和遵从3类。Securify从智能合约的字节码开始分析，之后分析合约的依赖关系图，进而得出合约的语义事实，最后使用语义事实进行模式检查，所有与违反性模式匹配的行为均为违规, 所有与遵从性模式匹配的行为均为为遵从, 其他行为则为警告。

国内比较成熟的智能合约的审计工具是成都链安科技公司开发的VaaS[15]。VaaS采用形式化验证，使用数学建模的方法，对审查的智能合约通过形式化验证，检测其存在的安全漏洞。使用VaaS，开发者可以在部署前检测出智能合约中存在的安全漏洞。

# 智能合约安全漏洞介绍

针对当前智能合约存在安全问题，ALHARBY等人将其分为编码问题、安全问题、隐私问题、性能问题4类[16]。

本课题研究的安全漏洞主要针对编码问题。编写智能合约的是程序设计语言，每种程序设计语言都有其优势和不足之处。将合约内容以代码形式表示，需要开发人员清晰地定义合约内容和各个部分的执行顺序，因此使用程序设计语言编写智能合约的任务繁琐且容易出错。

在Solidity语言中，回退函数是导致很多漏洞的原因，回退函数是一个特别的函数，它没有名字，没有参数，也没有返回值。但是会在两种情况下被调用，一种情况是合约调用时调用了不存在的函数名称，另一种情况是合约调用没有数据。很多漏洞的产生都和回退函数存在密不可分的的关系。

NCC Group对智能合约中出现频率高的编码问题进行了分类[10]，下面介绍整型溢出漏洞等7类经典的安全漏洞。

## 2.1整型溢出漏洞

整型溢出漏洞由整型变量引起，是智能合约中比较常见的漏洞，也是造成巨额经济损失的漏洞之一。2018年美链BEC的智能合约由于存在数据溢出漏洞，被黑客攻击后一夜之间就损失了64亿。整型溢出漏洞发生的原因是每个整型变量都有其可承受的取值范围，当一个整型变量的值超出其取值上限或者低于其取值下限时，就会发生溢出。以Solidity语言为例，在Solidity语言中，如果整型变量的取值超出其可承受的范围，会对整型变量的最高位进行丢弃，舍弃之后整型变量的取值就会发生错误。在存在整型溢出漏洞的智能合约中，如果用户输入超过系统设定最大值的转账金额，只要账户中还有余额，用户就可以把巨额以太币转走，给合约造成巨大经济损失。为了防止整型溢出漏洞，OpenZeppelin建立了一个叫做SafeMath的库，默认情况下在智能合约中使用SafeMath库的数学运算可以防止整数溢出问题。以加法溢出为例，下面是代码片段：

1：uint8 x =1;  
2：uint8 y = 255;  
3：uint8 z = x+y;

在Solidity语言中uint8代表取值范围为0-255的无符号整数。x与y相加之后的结果的值为256，超出了uint8类型的取值范围，因此在x与y的加法运算中存在整型溢出漏洞。

## 2.2重入漏洞

重入漏洞常在智能合约外部调用过程中发生，由于智能合约经常需要调用其他外部合约的代码，因此重入漏洞是智能合约中发生频率较高的漏洞之一。著名的重入漏洞有2016年在以太币组织 The DAO 中发生的盗币事件。攻击者利用两个重入漏洞创建子合约，提取了360万个以太币，造成价值2亿美元的损失。在以太坊的智能合约中，调用外部合约或者向地址发送以太币都需要调用外部合约的代码，这些外部调用可能被攻击者劫持，黑客通过劫持外部调用，迫使合约通过回退函数执行代码，从而不断重复进入以太坊智能合约，导致以太币持续消耗，直到耗尽。下面是一个重入漏洞的代码片段：

1：function withdraw () public { //回退  
2：if ( msg . sender . call . value ( balances [ msg . sender ]) ())  
3： balances [ msg . sender ] = 0;  
4：}

该代码片段目的是将智能合约体系中的代币转换为以太币。第3行中使用了call.value()进行转账，该转账函数在转账过程中可以调用账户中所有可用的以太币，由于使用该转账函数之前没有将用户代币归零，如果收款的智能合约在回退函数中再次调用付款合约的withdraw()函数，将会形成循环调用。存在漏洞的付款方智能合约会不断向收款智能合约转账，直到将账户中的代币耗尽。

## 2.3特权功能暴露漏洞

特权功能暴露漏洞由特权函数引起，多发生于对特权函数没有使用函数修饰符的智能合约中。在编写智能合约过程很容易发生此类漏洞。2017年7月在智能合约Parity中攻击者将库合约变成多重签名钱包合约，通过调用initWallet()函数，将自身变成了所有者，冻结了高达1亿美元价值的以太币。在Solidity语言中特权函数指涉及到转账，销毁合约等敏感操作的函数。用户可以任意调用没有函数修饰符的函数，如果特权函数没有函数修饰符，对特权函数的随意调用将导致产生不可控问题。下面是一个特权功能暴露漏洞的代码片段：

1：function destroycontract (address \_to){  
2： selfdestruct(\_to);  
3：}

第2行的selfdestruct()是Solidity语言中的内置函数，可以销毁当前合约，并把当前合约中的余额发送到某个地址中。此函数没有任何修饰符，可以直接调用。而函数destroycontract()也不存在函数修饰符，则用户可以任意调用该函数转移合约中的以太币并且销毁合约。

## 2.4跨合约调用漏洞

跨合约调用漏洞发生在智能合约的相互调用过程中，是常见的漏洞之一。2018年5月ATN的Token 合约就因为存在跨合约调用漏洞受到了攻击。在Solidity语言中合约之间的相互调用可以通过直接调用函数实现，在Solidity语言中提供了call(),delegatecall()和callcode()三个调用函数，这些调用函数被开发者滥用，导致了各种安全风险，以call()函数为例，当通过call()函数调用合约时，外界可以直接控制call()函数调用的参数，通过这个漏洞，攻击者可以获得所有者权限，进而控制合约的调用。下面是一个跨合约调用漏洞的代码片段：

1：function info(bytes data){

2: this.call(data);

3: }

4：function secret() public{

5: require(this==msg.sender);

6: //secret operations

7: }

在此片段中有info()和secret()两个函数，info()函数中有call()函数，外界用户为call()函数提供数据，而secret()函数是智能合约中包含隐秘内容的重要函数，原本只能由智能合约所有者进行调用，但如果用户提供的数据方法选择器指定了secret()函数，则外部用户也可以调用secret()函数，会造成安全风险。

## 2.5拒绝服务漏洞

拒绝服务指用户所需要的请求无法被系统处理，拒绝服务漏洞可能出现在智能合约所有需要其他函数结果或者判断条件执行的逻辑代码中，是智能合约中常见的漏洞之一。2016年2月在游戏KotET中发生的拒绝服务攻击，使得部分角色的补偿和未接收款项无法退回给玩家。攻击者通过消耗合约资源，使得用户短暂或永久地退出不可操作的合约，从而把以太币锁在被攻击的合约中。下面是一个拒绝服务漏洞的代码片段：

1：function bid() public payable{   
2： require(msg.value > highestBid);   
3： require(currentLeader.send(highestBid));   
4： currentLeader =msg.sender;   
5： highestBid =currentLeader;  
6：}

这是一个竞标函数，函数内容为谁持有的以太币多，谁就当选为新的持有者。新的竞标者进行竞标时，执行到第3行时将会触发调用竞标者的回退函数，如果竞标者的回退函数中是一个循环调用函数，则第3行的require()语句将会一直返回false，一直占用此函数，导致其他竞标者无法参与竞标，最终以较少的以太币数量获得竞标。

## 2.6短地址攻击漏洞

短地址攻击漏洞是由用户输入的智能合约地址位数低于规范位数引起。在以太坊中，如果输入的地址位数少于规范编码规定的位数，以太坊将在处理过程中对地址自动补0，直到地址长度符合要求，这样就会导致以太币被转到其他账户，造成经济损失。下面是短地址攻击漏洞的代码片段：

1：function sendCoin(0xiofa8d97756as7df5sd8f75g8675ds8gsdg)

2:{//sendCoin operations}

在转账函数中给出的地址位数少于规范地址位数，因此以太坊检查后就会补0，此时转账地址就发生了错误，会将以太币转移到其他地址中。

## 2.7tx.origin漏洞

tx.origin漏洞由tx.origin变量引起。tx.origin是Solidity中的一个全局变量，它能够遍历调用栈并返回最初发送调用的账户的地址，是整个交易过程中最初的合约所有者，当中间人进行调用合约时，如果使用tx.origin变量进行身份验证，调用者身份就会变成最初的合约所有者，中间人就拥有了其本不该拥有的权限，从而避开权限检查，对智能合约进行攻击。 下面是tx.origin漏洞的代码片段：

1：function withdrawAll(address \_recipient) public {  
2： require(tx.origin == owner);  
3： \_recipient.transfer(this.balance);  
4：}

在此函数中，如果攻击者诱使原合约发送以太币到攻击合约中，之后调用攻击者函数，在攻击者函数中进而调用原合约的withdraw()函数，则所有者就是原合约，第2行的require()条件得到满足，就会执行以太币转账，将原合约地址中的以太币转移带攻击者地址中。

# 基于编译的Solidity智能合约漏洞检测方法研究

本课题基于Solidity语言不同的词法和语法的常见度选取了Solidity语言的词法子集和语法子集，并基于Antlr语言进行了定义。Antlr((ANother Tool for Language Recognition)是一种语言工具，可以根据用户定义的词法和语法规则，自动生成词法语法分析器以及树分析器。由于Antlr使用上下文无关文法描述语言，因此需要自己使用Antlr语言定义Solidity语言的词法和语法规则。

完成词法和语法分析之后，将进行智能合约的漏洞检测，要完成漏洞检测算法，首先需要提取出各类漏洞的检测规则，之后才能根据检测规则实现检测算法。

## 3.1词法规则定义

选取了Solidity常用子集的词法，对标识符，字面量，和关键字进行了Antlr定义。

### 3.1.1标识符定义：

Solidity语言的标识符，由字母、下划线或者美元的“$”符开头，后面可跟若干的字母、下划线、“$”符和数字。对标识符的Antlr定义如下：

Identifier : IdentifierStart IdentifierPart\* ;

IdentifierStart : [a-zA-Z$\_] ;

IdentifierPart : [a-zA-Z0-9$\_] ;

标识符（Identifier）由标识符开头（IdentifierStart）和若干的标识符内容（IdentifierPart）组成。标识符必须有一个开头，但可以匹配标识符内容零次或多次。标识符开头（IdentifierStart）规定为大小写字母、下划线或者“$”符号，标识符内容规定为字母、下划线、“$”符和数字。

### 3.1.2字面量定义：

在Solidity中定义了数字，布尔类型，字符串以及版本的字面量。

对数字字面量Antlr定义如下：

numberLiteral  
  
 : (DecimalNumber | HexNumber) NumberUnit ?;

数字字面量的组成部分可以是数字单位(NumberUnit)，或者是使用十进制数(DecimalNumber)或者十六进制数(HexNumber)表示的数字单位。

对布尔字面量Antlr定义如下：

BooleanLiteral  
 : 'true' | 'false' ;

**布尔类型的字面量取值只能为true或false。**

**对字符串字面量Antlr定义如下：**

StringLiteral  
 : '"' DoubleQuotedStringCharacter\* '"'  
 | '\'' SingleQuotedStringCharacter\* '\'' ;  
fragment  
DoubleQuotedStringCharacter  
 : ~["\r\n\\] | ('\\' .) ;  
fragment  
SingleQuotedStringCharacter  
 : ~['\r\n\\] | ('\\' .) ;

字符字面量由若干的双引号字符串字符或单引号字符串字符组成，双引号字符串是双引号之内除换行回车之外的内容，单引号字符串是单引号之内除换行回车之外的内容。

### 3.1.3关键字定义

Solidity的关键字包括标识类型的关键字，标识控制流的关键字，函数权限关键字等。下面是对部分关键字的Antlr定义：

AnonymousKeyword : 'anonymous' ;  
BreakKeyword : 'break' ;  
ConstantKeyword : 'constant' ;  
ContinueKeyword : 'continue' ;  
ExternalKeyword : 'external' ;  
IndexedKeyword : 'indexed' ;  
InternalKeyword : 'internal' ;  
PayableKeyword : 'payable' ;  
PrivateKeyword : 'private' ;  
PublicKeyword : 'public' ;  
PureKeyword : 'pure' ;  
ViewKeyword : 'view' ;

对每种关键字具体给出关键字内容，建立起一对一的关系，对智能合约进行词法分析时就可以对其一一对应。

### 3.1.4类型定义

选取Solidity语言中常用的类型，包括地址，整型，字符串类型，字节类型等，下面是对类型的Antlr定义：

elementaryTypeName  
 : 'address' | 'bool' | 'string' | 'var' | Int | Uint | 'byte' | Byte ;

给出每种类型准确的标识符，之后这些标识符就用于表示各种类型。

## 3.2语法规则定义

选取了Solidity语言的语法子集语言版本，引入文件和合同内容。

### 3.2.1语言版本

为了确保Solidity语言的运行，一般使用Solidity语言编写的智能合约需要在文件开始处定义Solidity版本，以此确定编译器版本。使用Antlr对语言版本的定义如下：

pragmaDirective  
 : 'pragma' pragmaName pragmaValue ';' ;

对语言的版本定义由关键字pragma，语言名称和版本名称组成。

### 3.2.2引入文件

引入文件有三种形式，分别为import as、import as from、import from。对引入文件的Antlr语言定义如下：

importDirective  
 : 'import' StringLiteral ('as' identifier)? ';'  
 | 'import' ('\*' | identifier) ('as' identifier)? 'from' StringLiteral ';'  
 | 'import' '{' importDeclaration ( ',' importDeclaration )\* '}' 'from' StringLiteral ';' ;

importDeclaration  
 : identifier ('as' identifier)? ;

定义引入文件的三种类型，import as由 import，字符串字面量，as和标识符组成；import as from由import,标识符，from和字符串字面量组成；import from由import, 变更名字后的标识符名称，from和字符串字面量组成。

### 3.2.3合同内容

合同内容主要由表达式，状态变量，函数，结构体等部分组成。

#### 3.2.3.1表达式

选取在智能合约中比较常用的一些Solidity表达式，使用Antlr定义如下：

expression  
 : 'new' typeName #new  
 | expression '(' functionCallArguments ')'#bracketcircle  
 | expression '.' identifier #attribute  
 | '(' expression ')' #circle  
 | expression (MULTIPLY | DIVIDE | '%') expression #mulity  
 | expression (PLUS | MINUS) expression #addsub  
 | expression ('<' | '>' | '<=' | '>=') expression #compare  
 | expression ('==' | '!=') expression #compare1  
 | expression '&&' expression # yuyu  
 | expression '||' expression #huohuo  
 | expression ('=' | '|=' | '^=' | '&=' | '<<=' | '>>=' | PLUSEQUAL | MINUSEQUAL | '\*=' | '/=' | '%=') expression #other  
 | primaryExpression # primary;

为了便于编码为每种表达式类型在后面打了标签。以加减法表达式(addsub)为例进行介绍。expression (PLUS | MINUS) expression表示加减法表达式由两个表达式和加法或减号构成。

#### 3.2.3.2状态变量

状态变量由状态变量的修饰符，标识符和表达式组成。对状态变量的Antlr定义如下：

stateVariableDeclaration  
 : typeName  
 ( PublicKeyword | InternalKeyword | PrivateKeyword | ConstantKeyword )\*  
 identifier ('=' expression)? ';' ;

状态变量可以有0个或多个修饰符，其中的标识符可以初始化为一个表达式，也可以不初始化。

#### 3.2.3.3结构体

结构体由结构体名称和结构体内容组成，结构体内容可以包含整型、字符串、数组、结构体、映射等多种数据类型，但不可以以自身类型作为内容。对结构体的Antlr定义如下：

structDefinition  
 : 'struct' identifier  
 '{' ( variableDeclaration ';' (variableDeclaration ';')\* )? '}' ;

结构体内容中可以有多个变量声明，但是结构体中不能嵌套结构体，即结构体内容定义只能有一次。

#### 3.2.3.4函数

函数由函数权限符，函数修饰符，函数返回值和函数体组成。函数修饰符分为四种：public，private，internal和external。对函数的Antlr定义如下：

functionDefinition  
 : 'function' identifier? parameterList modifierList returnParameters? ( ';' | block ) ;

returnParameters  
 : 'returns' parameterList ;  
modifierList  
 : ( modifierInvocation | stateMutability | ExternalKeyword  
 | PublicKeyword | InternalKeyword | PrivateKeyword )\* ;

block  
 : '{' statement\* '}' ;

函数中函数的修饰符可以有0个或多个，函数体内容由0个或多个程序语句组成。

## 3.3漏洞检测规则定义

针对第2部分中7种典型漏洞的特点，下面给出各种漏洞检测规则。

### 3.3.1整型溢出漏洞检测规则

以整型加法运算为例，如果两个整数相加的结果小于其中任何一个加数，则存在整型溢出漏洞。使用一阶逻辑表示为：

P(a,b):a+b；F(y):y发生整型溢出

((p(a+b)<=a) V (p(a+b)<=b))->F(a+b)

### 3.3.2重入漏洞检测规则

如果智能合约外部调用使用call.value(),transfer(),send()等对状态的写函数，并且在这些函数之后存在更新状态变量的行为，则智能合约存在重入漏洞。使用一阶逻辑表示为：

P(x):x是写操作函数； Q(y):y表示对状态更新； G(x,y):先完成x,后完成y；

F(x):x存在重入漏洞

G(P(x),Q(y))->F(G(P(x),Q(y)))

### 3.3.3特权功能暴露漏洞检测规则

如果智能合约中涉及转账，自毁合约这类敏感操作的函数没有权限修饰符，则智能合约存在特权功能暴露漏洞。使用一阶逻辑表示为：

P(x):x是敏感操作函数； Q(y):函数y没有权限修饰符；

F(x):x存在特权功能暴露漏洞

(P(x) ∧ Q(x)) -> F(P(x) ∧ Q(x))

### 3.3.4跨合约调用漏洞检测规则

如果智能合约使用了call(),delegatecall(),callcode()这3个调用函数，则智能合约中存在跨合约调用漏洞 。使用一阶逻辑表示为：

P(x):x是call(),delegatecall(),callcode()函数之一；

F(x):x存在跨合约调用漏洞

P(x)->F(P(x))

### 3.3.5拒绝服务漏洞检测规则

分2种情况，第1种是如果智能合约判断语句中存在函数内容是循环调用的函数则智能合约中存在拒绝服务漏洞；第2种是如果智能合约判断语句中存在send()此类的转账函数，则智能合约中存在拒绝服务漏洞。使用一阶逻辑表示为：

P(x):x是判断语句；Q(y): 函数y内容存在循环调用；G(a):a是转账类函数；

F(x):x存在拒绝服务漏洞

P(Q(y)) V P(G(a)) -> F(P(x))

### 3.3.6短地址攻击漏洞检测规则

如果合约的地址不够规定的地址位数，则智能合约中存在短地址攻击漏洞。使用一阶逻辑表示为：

P(x):地址x不够20字节；F(x):x存在短地址攻击漏洞

P(x)->F(P(x))

### 3.3.7tx.origin漏洞检测规则

如果智能合约中存在tx.origin变量，则智能合约存在tx.origin漏洞。使用一阶逻辑表示为：

P(x):合约x中存在tx.origin变量；F(x):x存在tx.origin漏洞

P(x)->F(P(x))

# Solidity智能合约漏洞检测工具原型的设计与实现

## 4.1体系结构

整个系统由词法模块，语法模块和漏洞检测模块组成。整个系统的体系结构图如图1所示。

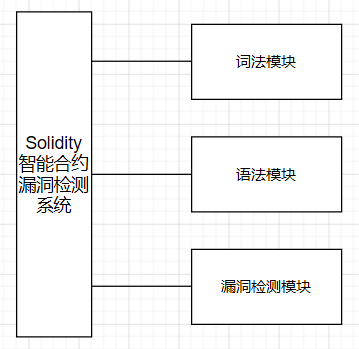


图1系统体系结构图

Figure1 System architecture diagram

词法模块中，词法分析工具使用Antlr语言定义词法，该模块可以得到Solidity智能合约的单词；语法模块中，语法分析工具使用Antlr定义语法，该模块可以得到Solidity智能合约的语法片段；漏洞检测模块使用Java语言编写，通过漏洞检测模块可以检测智能合约中存在的整型溢出等7类漏洞并给出修改建议。

## 4.2漏洞检测算法

### 4.2.1整型溢出漏洞检测算法

在变量的初始化过程中，使用哈希地图存储标识符和其对应的值。以占内存8位的无符号整型加法为例，取两边标识符存储在哈希地图中的数值，之后进行加法运算，相加结果取其内存8位的值，如果该值小于其中任何一个加数，则发生了整型溢出。伪代码如下：

Begin  
输入 z=x+y  
IF z<=x 或 z<=y  
则 加法运算溢出  
Print 整型溢出漏洞  
End

### 4.2.2重入漏洞检测算法

如果智能合约函数中存在call.value(),transfer(),send()等对状态的写函数，并且在写函数之后对状态变量进行了更新，则智能合约中存在重入漏洞。伪代码如下：

Begin  
输入 智能合约  
IF 智能合约在写操作函数之后存在更新状态的操作  
则 智能合约存在重入漏洞  
Print 重入漏洞  
End

### 4.2.3特权功能暴露漏洞检测算法

如果智能合约中涉及转账，自毁合约这类的敏感函数没有权限修饰符，则智能合约存在特权功能暴露漏洞。伪代码如下：

Begin  
输入 智能合约  
IF 智能合约中敏感操作函数没有权限修饰符  
则 智能合约存在特权功能暴露漏洞  
Print 特权功能暴露漏洞  
End

### 4.2.4跨合约调用漏洞检测算法

如果智能合约在对其他合约进行调用时使用了call(),delegatecall(),callcode()这3个函数之一，则智能合约中存在跨合约调用漏洞，给出建议改为直接指向给定地址的合约来调用函数或者使用高级函数如transfer()实现转账功能。伪代码如下：

Begin  
输入 智能合约  
IF 智能合约中使用call(),delegatecall(),callcode()调用其他合约  
则 智能合约存在跨合约调用漏洞  
Print 跨合约调用漏洞  
End

### 4.2.5拒绝服务漏洞检测算法

如果智能合约判断语句内容中存在函数内容是循环调用的函数，或者判断语句中存在转账操作，则智能合约中存在拒绝服务漏洞。伪代码如下：

Begin  
输入 智能合约  
IF 智能合约判断语句中包含内容为循环调用的函数 或 智能合约判断语句中涉及到send()转账操作  
则 智能合约存在拒绝服务漏洞  
Print 拒绝服务漏洞  
End

### 4.2.6短地址攻击漏洞检测算法

如果用户输入的智能合约地址不足20字节，则智能合约中存在短地址攻击漏洞。伪代码如下：

Begin  
输入 智能合约地址  
IF 地址不够20字节  
则 智能合约存在短地址攻击漏洞  
Print 短地址攻击漏洞  
End

### 4.2.7tx.origin漏洞检测算法

如果智能合约中使用了tx.origin变量，则智能合约中存在tx.origin漏洞。伪代码如下：

Begin  
输入 智能合约  
IF 智能合约使用了tx.origin变量  
则 智能合约存在tx.origin漏洞  
Print tx.origin漏洞  
End

# 测试与结果分析

## 5.1测试用例设计

对7种典型漏洞，每种漏洞设计一个测试用例，具体内容如表1所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 漏洞类型 | 输入数据 | 预期输出 |
| 1 | 整型溢出漏洞 | 1：uint8 x =1; 2：uint8 y = 255; 3：uint8 z = x+y; | x+y  存在加法溢出 |
| 2 | 重入漏洞 | 1：function withdraw () public { //回退 2：if ( msg . sender . call . value ( balances [ msg . sender ]) ()) 3：balances [ msg . sender ] = 0; 4：} | 2：存在重入漏洞 |
| 3 | 特权功能暴露漏洞 | 1：function destroycontract (address \_to){ 2：selfdestruct(\_to); 3：} | 1：存在特权功能暴露漏洞 |
| 4 | 跨合约调用漏洞 | 1:function info(bytes data){  2: this.call(data);  3:}  4:Function secret() public{  5:require(this==msg.sender);  6://secret operations} | 2:存在跨合约调用漏洞 |
| 5 | 拒绝服务漏洞 | 1:function bid() public payable{  2:require(msg.value > highestBid); 3:require(currentLeader.send(highestBid)); 4:currentLeader =msg.sender; 5:highestBid =currentLeader; 6:} | 3:存在拒绝服务漏洞 |
| 6 | 短地址攻击漏洞 | 1:function sendCoin(0xiofa8d97756as7df5sd8f75g8675ds8gsdg)  2:{//sendCoin operations} | 1:存在短地址攻击漏洞 |
| 7 | tx.origin漏洞 | 1:function withdrawAll(address \_recipient) public { 2: require(tx.origin == owner); 3:\_recipient.transfer(this.balance); 4:} | 2:存在tx.origin漏洞 |

表1测试用例表

Table 1 Test case table

## 5.2测试与结果分析

针对每种漏洞的测试用例的测试结果如表2所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 漏洞类型 | 预期输出 | 实际输出 |
| 1 | 整型溢出漏洞 | 3:存在加法溢出 | 3:存在加法溢出 |
| 2 | 重入漏洞 | 2：存在重入漏洞 | 2：存在重入漏洞 |
| 3 | 特权功能暴露漏洞 | 1：存在特权功能暴露漏洞 | 1：存在特权功能暴露漏洞 |
| 4 | 跨合约调用漏洞 | 2:存在跨合约调用漏洞 | 2:存在跨合约调用漏洞 |
| 5 | 拒绝服务漏洞 | 3:存在拒绝服务漏洞 | 3:存在拒绝服务漏洞 |
| 6 | 短地址攻击漏洞 | 1:存在短地址攻击漏洞 | 1:存在短地址攻击漏洞 |
| 7 | tx.origin漏洞 | 2:存在tx.origin漏洞 | 2:存在tx.origin漏洞 |

表2测试结果表

Table 2 Test result table

对比每个测试用例中包含的漏洞和漏洞所在的行数，均与运行结果一致。因此该系统可以检测出智能合约中存在的这7种经典漏洞并输出漏洞所在行数。

# 总结与展望

本课题采用基于编译的方法，设计并实现了Solidity智能合约漏洞检测工具。针对以太坊平台上Solidity语言编写的智能合约，该工具可以完成对智能合约中存在的7种典型漏洞进行检测。该工具可以为使用Solidity语言进行开发的工作者提供安全提醒，帮助其开发更安全的智能合约。

虽然整个系统能够基本实现对7种典型漏洞的检测，但是仍然存在很多需要改进的地方。比如各种漏洞的检测方法还不够完善，存在误判和漏判的情况；以及检测的漏洞类型还不够全面，可以进一步增加新的漏洞类型等。

# 参考文献

1. 马昂,潘晓,吴雷,等.区块链技术基础及应用研究综述[J].信息安全研究, 2017, 3(11):968-983.
2. NAKAMOTO S.Bitcoin:A peer-to-peer Electronic Cash System[EB/OL].

https://www.coindesk.com/bitcoin-peer-topeer-electronic-cash-system, 2018-5-11.

1. BUTERIN V.A Next-generation Smart Contract and Decentralized Application Platform[EB/OL]. https://github.com/ethereum/wiki/wiki/White-Paper, 2018-5-11.
2. 范吉立,李晓华,聂铁铮,于戈.区块链系统中智能合约技术综述[J].计 算机科学,2019, 46 (11): 1-10.(Fan J L,Li X H,Nie T Z,Yu G. Survey on Smart Contract based on Blockchain System[J].Computer Science,2019, 46 (11): 1-10.)
3. BUTERINV.[https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform- 1390528211/ . 2014](https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform-%201390528211/%20.%202014)
4. 王化群,张帆,李甜,等.智能合约中的安全与隐私保护技术[J].南京邮 电大学学报：自然科学版,2019,39(4):63-71.(WANG H Q,ZHANG F,LI T, et al. Security and privacy protection technologies in smart contract[J]. Journal of Nanjing University of Posts and Telecommunications,2019,39(4):63-71.)
5. Wright C, Serguieva A. Sustainable blockchain-enabled services: Smart contracts[C]. 2017 IEEE International Conference on Big Data. Boston:IEEE, 2017: 4255-4264.
6. Grishchenko I, Maffei M, Schneidewind C. A semantic framework for the security analysis of ethereum smart contracts[C]. ICPOST 2018: International Conference on Principles of Security and Trust. Cham:Springer,2018: 243-269.
7. 付梦琳,吴礼发,洪征,冯文博.智能合约安全漏洞挖掘技术研究[J].计 算机应用,2019,39(07):1959-1966.(Fu M L,Wu L F,Hong Z,Feng W B. Research on Smart Contracts Vulnerability Mining Technique[J].Journal of Computer

Applications,2019,39(07):1959-1966.)

1. NCC Group.Decentralized application security project top 10 of 2018 [EB/OL]. (2018- 07- 08) [2018- 09- 08].https://www.dasp.co/index.html.
2. IDELBERGER F, GOVERNATORI G, RIVERET R, et al.Evaluation of Logic-based Smart Contracts for Blockchain Systems[C]//Springer.International Symposium on Rules and Rule Markup Languages for the Semantic Web, July 6-9, 2016, Stony Brook, NY, USA.Heidelberg:Springer, 2016:167-183.
3. BUTERIN V.Ethereum:a next-generation smart contract and decentralized application platform [EB/OL]. (2014- 01- 23) [2018- 09- 08]

.https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform- 1390528211/.

1. Nikolic I , Kolluri A , Sergey I , et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale[J]. 2018.
2. Tsankov P,Dan A,Drachsler-Cohen D,et al.Securify:Practical Security Analysis of Smart Contracts[C]. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.ACM,2018:67~82.
3. CSDN Research and Development Technology.Formal verification is a sharp weapon for smart contracts safety [EB/OL]. (2018-06-12) [2018-09-

08].https://blog.csdn.net /CDLianan/article/details/80665163.

1. ALHARBY M, MOORSEL A V.Blockchain-based Smart Contracts:A Systematic Mapping Study[EB/OL]. http://cn.arxiv.org/ftp/arxiv/papers/1710/1710.06372.pdf, 2018-5-11.

# 致谢

首先要感谢的是我的指导老师王颖老师，王颖老师对学生认真负责，对我存在的问题总是很细致耐心的讲解，在我整个毕业设计的完成过程中，给与了我很多帮助。

接下来要感谢大学四年来我的所有老师们，他们的精心教导，使我在大学四年的时光中收获颇丰。

还要感谢我的同学们，感谢他们大学四年的陪伴，尤其是室友们在4年时光里的相互扶持和帮助。

最后感谢母校，感谢华农对我四年的培育，也希望母校越来越好。