基于本体推理的智能合约漏洞检测系统

陈瑞翔，焦健，王若华

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220900183

基于本体推理的智能合约漏洞检测系统

陈瑞翔[1]　焦　健[1]　王若华[2]
1 北京信息科技大学计算机学院　北京 100101
2 中国电信股份有限公司六盘水分公司　贵州 六盘水 553001
(2021020637@bistu.edu.cn)

摘　要　随着区块链的不断发展,基于以太坊的智能合约越发受到各界的广泛关注,但随之而来的是其面临着更多的安全威胁。针对以太坊智能合约的安全问题,出现了各种漏洞检测方法,如符号执行、形式化验证、深度学习等,但现有的检测方法能检测到的漏洞类型大多不全面,缺乏可解释性。针对这些问题,设计并实现了针对 Solidity 高级语言层面的基于本体推理的智能合约漏洞检测系统。该系统先把智能合约源码解析为抽象语法树,再进行合约信息抽取,利用抽取到的数据信息构建智能合约漏洞检测本体,并使用推理机进行本体推理。实验选取了其他检测工具与本系统进行对比,并使用这几种工具对 100 份智能合约样本进行检测。实验结果表明,所提系统的检测效果良好,能检测多种类型的智能合约漏洞,并能给出其漏洞的相关信息。

关键词:智能合约;漏洞检测;以太坊;区块链;本体推理
中图法分类号　TP309

Smart Contract Vulnerability Detection System Based on Ontology Reasoning

CHEN Ruixiang[1],JIAO Jian[1] and WANG Ruohua[2]
1 College of Computer Science,Beijing Information Science and Technology University,Beijing 100101,China
2 Liupanshui Company of China Telecom Co. Ltd.,Liupanshui,Guizhou 553001,China

Abstract　Withthe development of the blockchain,smart contract based on Ethereum has attracted more and more attention from all walks of life,but it has also faced more security threats. For the security problems of Ethereum smart contracts,various vulnerability detection methods have emerged,such as symbolic execution,formal verification,deep learning and other technologies. However,most of the existing methods have incomplete detection types and lack interpretability. To solve these problems,a smart contract vulnerability detection system based on ontology reasoning for Solidity high-level language level is designed and implemented. The smart contract vulnerability source code is parsed into an abstract syntax tree,and the information is extracted. The extracted information is used to construct the vulnerability detection ontology,and the reasoning engine is used to infer the ontology vulnerability. In the experiment,other detection tools are selected to compare with this system,and these tools are used to detect 100 intelligent combined source samples. The results show that the system has a good detection effect,it can detect various types of smart contract loopholes and can give the information about the cause of the vulnerability.

Keywords　Smart contract,Vulnerability detection,Ethereum,Blockchain,Ontology reasoning

1　引言

区块链是首个自带对账功能的数字记账技术实现,属于一种中心化的记录技术,其提供了在不可信环境下进行可信交易的基础环境。在此基础上,出现了一个全新开放的区块链平台——以太坊,它允许任何人在平台上建立和使用通过区块链技术运行的去中心化应用。而以太坊的核心是智能合约,智能合约现已被应用到很多领域,如去中心化的交易所、物联网、投票、医疗记录隐私、保险等。

由于区块链中的智能合约总是涉及价值数百万美元的加密货币,智能合约中的漏洞往往会导致巨大的财务损失[1]。为了解决这些安全威胁,研究人员研究出了各种漏洞检测技术,有模糊测试、形式化验证、程序分析技术和深度学习等技术。

现在大多数检测工具涵盖的漏洞类型不全面,像基于深度学习的检测方法 AME[2] 只能检测重入、时间戳依赖性和无限循环漏洞;基于静态程序分析技术的 SASC[3] 只能检测 tx.origin 漏洞、时间戳依赖漏洞;基于代码表示融合的漏洞检测方法 AFS[4] 只专注于重入漏洞的检测。这些检测工具大多都支持重入、整数溢出等漏洞的检测,因为这些安全漏洞都曾引起过重大的合约攻击事件。但有一些不常见的合约漏洞,如精度丢失,不被关注,但当这些漏洞被利用时,也会造成巨大损失。此外,现在大多数漏洞检测工具都只能检测智能

到稿日期:2022-09-18　返修日期:2023-03-06
基金项目:国家自然科学基金(61872044)
This work was supported by the National Natural Science Foundation of China(61872044).
通信作者:焦健(jiaojian@bistu.edu.cn)

合约是否存在漏洞，缺少对漏洞成因的相关说明。

基于出现的各种问题，本文设计了一个基于本体推理的智能合约漏洞检测系统，从 Solidity 源代码层面，把源码解析为抽象语法树（Abstract Syntax Tree，AST），然后对 AST 进行数据处理，并分析了不同类型漏洞的检测方案，提取信息，建立漏洞检测本体，并制定漏洞推理规则进行漏洞推理。同时，实验还选取了开源检测工具 Slither[5] 和商业工具链安 Beosin-VaaS[6] 两种漏洞检测工具与本系统进行对比实验。实验结果表明，所提出的针对 Solidity 高级语言层面的基于本体推理的智能合约漏洞检测系统具有良好的检测效果，能检测的漏洞类型全面，不止能检测重入、整数溢出等危害较大的漏洞，还能检测未声明函数可见性、精度丢失等大多检测工具不支持的漏洞；其检测效率也优于其他两种工具。

2 相关工作

2.1 智能合约漏洞

智能合约安全威胁可以分为高级语言、虚拟机、区块链 3 个层面[7]，本系统主要针对的是 Solidity 高级语言层面，并把 11 种智能合约漏洞分为 4 类，具体分类如表 1 所列。

表 1 漏洞分类

Table 1 Vulnerability classification

漏洞分类	具体漏洞
方法误用	时间戳依赖
	基于链属性的弱随机性
	委托调用非可信合约
	利用 *tx.origin* 授权
	非预期的以太币
函数问题	错误的构造函数名
	未声明函数可见性
顺序依赖	重入漏洞
结果审查	未检查调用返回值
	整数溢出
	精度丢失

Solidity 智能合约调用其他合约函数的方式。除了直接调用函数，还可以通过 callcode，delegatecall，call 方法调用；如果想知道账户的地址，则可以利用 Solidity 的一个全局变量 *tx.origin*，它可以遍历整个调用栈并返回最初发送调用的账户的地址。方法误用就是开发者在程序开发时没有正确使用这些可能引发漏洞的特殊方法。

智能合约函数有各种特性，内部函数只能在当前合约内被使用，如在当前的代码块内以及继承的函数中；外部函数由地址和函数方法签名两部分组成，可作为外部函数调用的参数，或者由外部函数调用返回。函数问题就是智能合约中的函数没有被正确定义，如未声明函数可见性便是没有开发人员忘记设置函数的可见性，使得恶意用户能够进行未经授权的状态更改，形成漏洞。

智能合约的一些功能需要按照一定的顺序执行，如若想实现转账功能，余额比较顺序就必须在转账操作之前。顺序依赖便是智能合约中的某些操作没有按照正确的执行顺序进行，如重入漏洞是采用了先转账再修改存储变量的顺序。程序 1 是重入漏洞的关键源码，先使用了 call. value()的方式转账，再利用 balances 进行余额比较，如果转账的目标是一个带有恶意回退函数的合约，则可能导致当前账户被恶意合约再次调用进行转账，直到目标合约余额不足或 gas（交易手续费）耗尽。

程序 1 智能合约重入漏洞：

```
function withdraw(){
require(msg. sender, call. value(balances[msg. sender])
());
//转账操作
balances[msg. sender]=0;
//余额修改
}
```

智能合约在执行某些步骤后，需要对前一步执行的结果进行检测，因为如果没有审查，即使出现了异常，执行也将继续。若调用意外失败或者攻击者强制调用失败，则可能导致后续程序逻辑中的意外行为。结果审查便是当执行某些操作后，没有对其结果进行审查，若在算术运算时没有对其结果进行审查，则当算术运算达到该类型的最大或最小值时，将发生整数上溢或整数下溢。

2.2 智能合约漏洞检测方法

随着以太坊智能合约的运用越来越广泛，智能合约的检测技术也受到了更加广泛的关注。现在智能合约漏洞检测采取的方法有符号执行、形式化验证、深度学习等技术。

2.2.1 符号执行

符号执行包括动态分析和静态分析，其核心思想是把程序执行过程中不确定的输入转换为符号值，以推动程序执行与分析[8]。下面介绍了近年来基于符号执行的研究成果。

1）Maian[9] 是一种基于符号分析的智能合约分析工具，它通过长序列的合约调用过程来发现安全漏洞。Maian 只专注于 3 种类型的合约漏洞，即资产冻结漏洞、信息泄露漏洞和合约易被销毁漏洞。

2）Oyente[10] 是最早的智能合约漏洞静态检测工具之一，其以智能合约字节码和以太坊状态为输入，模拟以太坊虚拟机（ETHereum Virtual Machine，EVM）并遍历合约的不同执行路径，在合约控制流图的基础上利用符号执行的方法检测智能合约漏洞。Oyente 支持检测的漏洞类型包括重入漏洞、异常处理漏洞、交易顺序依赖漏洞等。

2.2.2 形式化验证

形式化验证就是基于已建立的形式化规格，对系统的相关特性进行分析和验证，以评判系统是否满足期望的特性。形式化验证并不能完全确保系统性能正确无误，只能最大限度地理解和分析系统，并尽可能地发现其中的不一致性、模糊性等错误。下面介绍了近年来基于形式化验证的研究成果。

1）framework[11] 是一种形式化验证方法，通过将智能合约源码和字节码转化成函数编程语言，来分析和验证合约的安全性和功能的正确性，从而成功检测以太坊智能合约漏洞。其支持检测的漏洞类型包括重入漏洞、交易顺序依赖等。

2）ZEUS[12] 是一种静态分析工具，其利用抽象解释、符号模型检查以及约束语句来快速验证合约的安全性，支持多种智能合约漏洞的检测，例如重入漏洞、整数溢出等。

2.2.3 深度学习技术

近年来，深度学习在程序安全领域已有许多成功的实践，深度学习技术的进步促进了各种安全检测方法的诞生，对于各种安全漏洞，深度学习方法的准确率高，具有良好的适应性，但多数基于深度学习的漏洞检测工具能检测的漏洞类型

较少。下面介绍了近年来基于深度学习技术的研究成果。

1)ReChecker[13]是一个基于深度学习的智能合约重入漏洞检测工具,其通过将智能合约 Solidity 源代码转换为合约块的形式,来捕获合约中基本的语义信息和控制流依赖信息。ReChecker 利用双向长短期记忆模型和注意力机制实现了以太坊智能合约重入漏洞的自动化检测。

2)DR-GCN[14]利用合约图来检测智能合约漏洞,其将智能合约源代码转换为具有语义表示的合约图结构,并利用图卷积神经网络构建了安全漏洞检测模型。DR-GCN 能够检测重入漏洞、时间戳依赖漏洞以及死循环漏洞。

3 智能合约本体模型设计

本系统的本体模型如图 1 所示,其主要分为类(Classes)与实体(Individuals)、类与对象属性(Object Property)、实体与数据属性(Data Property)3 部分。

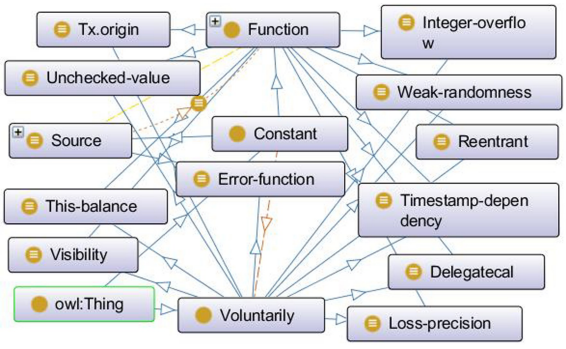

图 1 本体模型

Fig. 1 Ontology model

3.1 类与实体

类是本体的主体元素,是对本体的进一步划分。本文所设计的智能合约漏洞检测本体类的含义如表 2 所列,其主要有两个大类,即 Constant 和 Voluntarily。Constant 包含两个子类,即 Function 和 Source,其中 Source 又包含多个具体合约子类,这些子类被命名为具体的合约名;Voluntarily 包含多个漏洞子类,例如 Reentrant,Delegatecall,Visibility 等。

表 2 本体类的含义

Table 2 Significance of ontology classes

父类	子类	含义
Constant		合约类
	Function	函数名
	Source	合约名
Voluntarily		漏洞类
	Reentrant	重入漏洞
	Integer-overflow	整数溢出
	Visibility	未声明函数可见性
	Unchecked-value	未检查消息调用的返回值
	Delegatecall	委托调用非可信合约
	tx.origin	利用 *tx.origin* 授权
	Timestamp-dependency	时间戳依赖
	Error-function	错误的构造函数名
	This-balance	非预期以太币
	Weak-randomness	基于链的弱随机性
	Loss-precision	精度丢失

实体是每个类下的具体实例,每个类可包含多个实体,如 Function 可包含多个具体的合约函数实体。本体推理针对的是每个具体的实体,通过推理就可得出某个实体是否存在某种漏洞。通过对 2.1 节的程序 1 进行信息抽取,就可得到实体 withdraw,其属于 Function 类。

3.2 类与对象属性

对象属性表明了类与类之间的关系,它是多个类之间产生联系的重要属性。Constant 与 Voluntarily 之间的对象属性关系为 hasVoluntarily,表示合约可能包含各种漏洞,Source 和 Function 之间的对象属性关系为 hasFunction,表示一个合约可以有多个函数。Constant 与 Function、Source,Voluntarily 与各种漏洞类之间的对象属性关系为 hasSubclass,表示类与子类的关系:一个类可包含多个子类,子类可以继承父类的关系。

3.3 实体与数据属性

实体的数据属性是本体推理的一个重要参数,是每个实体区别于其他实体的特征,它表明了每个实体的特性。其中每个实体可以包含多个数据属性,每个数据属性可以包含不同的属性值。图 2 是 3 个实体与其数据属性的关系图,不同的实体可以有相同的数据属性,如 msg 和 public 等,也有不同于其他实体的数据属性,如 delegatecall 和 block 等,正是多个不同的数据属性与属性值的组合,如实体 withdraw 的数据属性 call 和其属性值 3、数据属性 msg 和其属性值 5 等;以及实体 Other 的数据属性 public 和其属性值 0、数据属性 block 和其属性值 7 等。表明了一个实体与其他实体的不同之处。

在智能合约漏洞推理本体中,函数实体的数据属性为每个对应函数下的具体谓词信息、结构特征,这些特征信息是本体推理的重要依据。通过对 2.1 节的程序 1 进行信息抽取,可得到如图 2 所示的实体 withdraw 和其对应的数据属性。withdraw 实体包含了 call,value,balances 等数据属性,每个实体的数据属性都有其对应的属性值 1,3,5 等,其中 1,3,5 等属性值表示其对应实体的数据属性在文中的位置,如 call 为 3,value 为 3,balances 为 5,3,5 等属性值表示对应实体的数据属性在合约中的位置关系。这些信息说明 withdraw 实体使用了 call.value()进行转账,使用了 balances 进行余额比较;通过比较实体数据属性值 5 大于 3,可知余额比较在转账之后。

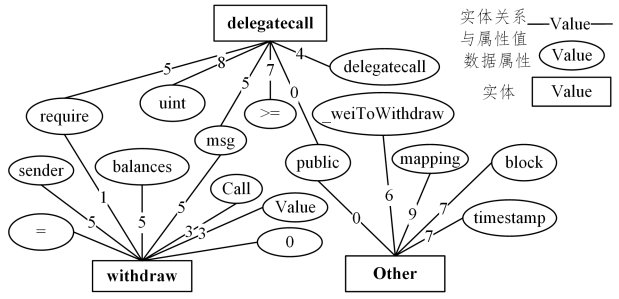

图 2 实体与数据属性

Fig. 2 Entities and data properties

4 系统结构与关键技术

4.1 系统整体结构

本文设计的智能合约漏洞检测系统如图 3 所示,主要分为 3 个模块。

1）利用抽象语法树解析工具 AST explorer[15] 的 Solidity-parser-diligence 解析器，把智能合约源码解析为 AST，并存为 AST.json 文件，便于下一步对源码漏洞信息的抽取。

2）使用 python 开发工具，利用 JsonPath 为 Json 文件提供的解析能力，对 AST 进行结构信息抽取，并利用 Pandas 扩展程序库对数据进行分析，提取所需的漏洞特征信息，导出包含智能合约漏洞特征的数据，保存在 Excel 表中。

3）利用本体开发工具 Protégé[16] 进行本体建模。先构建好本体的类与对象属性，然后利用 Cellfie[17] 把 Excel 表中的漏洞特征导入本体的相应位置，函数为实体，谓词、结构属性为对应实体数据属性。最后定义漏洞推理规则，并使用推理机 HermiT 进行推理，检测智能合约是否存在漏洞，若有漏洞，则根据推理规则和实体属性给出相应漏洞的形成原因。

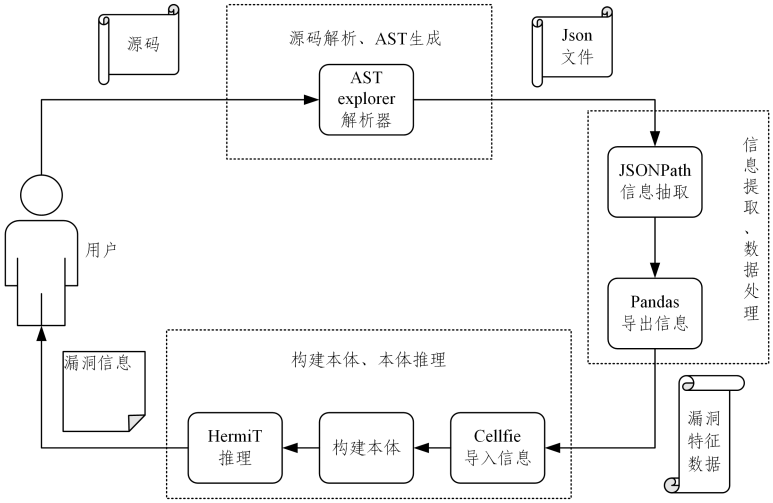

图 3 系统整体结构图

Fig. 3 Overall structure of system

4.2 合约信息抽取

智能合约 AST 的生成过程如下：首先读取基于 Solidity 的智能合约源代码文件中的字符流，然后通过词法分析生成 token，最后通过语法分析生成 AST。解析后的智能合约 AST 被保存为 Json 格式文件。其中 AST 信息提取过程如算法 1 所示，使用 JsonPath 抽取 AST 中的合约名、函数名，并以每个函数为约束条件，把合约分割，分别抽取每个函数下的具体信息，抽取的信息包括谓词名、谓词位置信息等。

算法 1 智能合约漏洞信息提取

输入：抽象语法树 Json 文件

输出：包含智能合约漏洞特征的 Excel 文件

```
1. for fileName in files do
2.     ContractName←jsonpath(AST.json)//提取合约名
3.     FunctionName←jsonpath(AST.json)//提取函数名
4.     for function in FunctionName do
5.         feature←jsonpath(AST.json)//提取谓词
6.         for i in feature do
7.             i.range←jsonpath(AST.json)//提取谓词位置信息，为[a,
               b]形式
8.             predicate.list←i
9.             location.list←range[0]
10.        end for
11. func.list←function
12. end for
13. ContractVersion←jsonpath(AST.json)
14. data←(predicate.list,filocation.list,func.list,ContractVersion)//
    数据写入
15.    Excel←pd.DataFrame.fromdict(date)
16. end for
```

抽取到的合约信息将用于本体构建。在完成本体模型设计后，需要利用本体开发工具 Protégé[16] 的内置模块 Cellfie[17] 向本体中已定义好的各个类导入相关的实体与数据属性，构建完整的智能合约漏洞检测本体，构建完成后便可对本体进行合约漏洞推理。

4.3 合约漏洞推理

智能合约漏洞推理的总体规则如程序 2 所示。其中 condition 表示漏洞的判定条件，可以根据不同漏洞种类而改变 condition，operator 是不同判定条件之间的连接符，包括 and 和 or 等多种类型。

程序 2 漏洞推理的总体规则

```
Function and(
(condition)operator
(condition)operator
…)
```

若想利用推理机 HermiT 实现本体推理，就需要对漏洞规则进行定义，而每种智能合约漏洞都有不同的特征，因此需要在对每种漏洞研究分析的基础上，设计了针对表 1 中不同类型漏洞的推理规则。

方法误用是智能合约中使用了可能引发漏洞的特殊方法，因此在检查这类漏洞时就要关注其使用的方法。如时间戳依赖要关注 block.timestamp 和 block.number 的使用方式，委托调用要关注 delegatecall 的使用方式。condition 形式化表达为(method some xsd:integer)，其中 method 为方法误用中使用的各种特殊方法，如 block.timestamp 和 delegatecall 等。

函数问题是智能合约中的函数在创建时没有被正确定义，因此在检查此类漏洞时需要关注合约函数，如未声明函数

可见性。智能合约中的变量可被指定为 public, internal, external 或 private,没有指定可见性类型的函数,其可见性默认为 public,在检查未声明函数可见性漏洞时便要关注其函数定义。其中 condition 形式化表达为(statementsome xsd:integer),statement 为对函数的声明,如 public 和 internal 等。

顺序依赖是智能合约中的某些操作未按照一定顺序进行,检查此类漏洞时,就要关注其执行顺序,如重入漏洞,在检测时就要关注其转账和余额比较顺序。其中 condition 形式化表达为(variable some xsd:integer location),其中 variable 为对函数的声明,如 call, value, balances 等,此外还应比较其 location 的顺序关系。

结果审查是当执行某些操作后未对其结果进行审查,检查此类漏洞方法为先查找相应操作,再检查是否对相应操作进行审查。如整数溢出漏洞要关注算数运算后是否对其结果进行审查。其中 condition 形式化表达为(variable some xsd:integer location),variable 为算数运算的符号与数字、函数返回值等,此外还应比较其 location 的顺序关系。

以重入漏洞推理规则为例,如程序 3 所示,condition 为 call some xsd:integer[<= n];value some xsd:integer, balances some xsd:integer[>= n]。其中 call, value, balances 是从合约源码抽取的信息,被导入到本体模型中作为实体的数据属性;some xsd:integer 是推理机的固定规则,表示某实体具有一些 integer 类型的数据属性;[<= n],[>= n]是实体的属性值比较,代表其结构信息,表示数据 call 在 balances 之前。

程序 3 重入漏洞推理规则

```
Function and(
(call some xsd:integer[<=n]) and
(value some xsd:integer) and
(balances some xsd:integer[>=n])
)
```

根据第 3 节所述,实体 withdraw 属于 Function 类,其包含了 call, value, balances 等数据属性,对应的属性值分别为 3,3 和 5,3 和 5 等属性值表示对应实体的数据属性在合约中的位置关系,通过比较实体数据属性值 5 大于 3,表明余额比较在转账之后。其符合了顺序依赖中重入漏洞规则,因此可以判定实体 withdraw 具有重入漏洞。推理出实体 withdraw 具有重入漏洞后,系统能列出其漏洞形成原因的相关信息:call, value, balances 等重入漏洞的特征属性,并标明属性值 3,3,5,其表示余额比较在转账之后;并给出所定义的重入漏洞推理规则。

5 实验验证与结果分析

5.1 评估方法

本节主要介绍了实验的评估方法,设定了评估标准。本实验的评估标准有准确率、漏报率。

准确率(Accuracy)的计算式如下:

$$Accuracy=\frac{TP+TN}{TP+FP+FN+TN}$$

漏报率(FPR)的计算式如下:

$$FPR=\frac{FN}{TP+FN}$$

其中,FP 表示实际为负但被预测为正的样本数量,TN 表示实际为负被预测为负的样本数量,TP 表示实际为正被预测为正的样本数量,FN 表示实际为正但被预测为负的样本数量。$TP+TN$ 表示所有被预测正确的样本数量,$TP+FN$ 表示所有实际为正的样本数量。

为了检测设计的系统能否检测文中所述的 11 类智能合约漏洞,选取了包含委托调用非可信合约、利用 *tx.origin* 授权、精度丢失等这 11 种漏洞的智能合约样本作为检测对象。

本文选取开源检测工具 Slither 和商业工具链安 Beosin-VaaS 进行对比实验,实验随机选取 100 个已部署在以太坊的智能合约作为检测样本,使用这 3 种工具对这 100 份智能合约样本进行检测,比较其准确率、漏报率。同时,将本系统与 2.2 节所述漏洞检测工具在 Solidity 高级语言层面上进行了漏洞检测能力比较,并与现有的符号执行、形式化验证、深度学习技术等检测方法进行了对比,说明了本文方法的独特优势。

5.2 结果分析

首先分析本系统能否检测文中所述的 11 种智能合约漏洞,并与 2.2 节所述漏洞检测工具进行比较,然后对其准确率、漏报率进行实验对比,并对实验结果进行分析讨论。

5.2.1 漏洞检测

3 种工具能检测的漏洞类型如表 3 所列,其中 Onsol 表示本系统,√表示漏洞检测工具可以检测到该类型漏洞。实验结果显示,本系统能检测文中所述的 11 种智能合约漏洞。

表 3 Solidity 源码层检测漏洞

Table 3 Vulnerability detection based on Solidity source code level

检测工具	委托调用非可信合约	利用 *tx.origin* 授权	未检查调用返回值	重入	时间戳依赖	基于链的弱随机性	整数溢出	未声明函数可见性	错误的构造函数名	非预期以太币	精度丢失
Onsol	√	√	√	√	√	√	√	√	√	√	√
Slither[5]		√	√	√						√	√
Beosin-VaaS[6]		√	√				√		√	√	
Maian[9]				√							
Oyente[10]			√	√	√		√				
framework[11]	√		√	√	√	√					
ZEUS[12]				√	√	√	√				
ReChecker[13]				√							
DR-GCN[14]				√	√						

表 3 列出了 2.2 节所述漏洞检测工具能检测的合约漏洞,大多数检测工具都支持重入漏洞的检测,这是因为智能

合约历史上最著名的“TheDAO”事件[18]是由重入漏洞引起的，因此研究者大多会关注此类漏洞。另外，很多检测工具也支持整数溢出、时间戳依赖等漏洞的检测，因为这些安全漏洞都曾引起过重大的合约攻击事件，例如美链 BEC 合约整数溢出事件[19]。针对未声明函数可见性、精度丢失等漏洞，能检测的工具较少，因为这些漏洞大多尚未产生过巨大的危害，但这些漏洞若被攻击者利用，会造成巨大的财产损失。

通过实验对比发现，本系统既能检测出造成过巨大财产损失的重入、整数溢出等大多工具能检测的漏洞，也能检测一些出现较少、大多检测工具不支持的漏洞，如未声明函数可见性、精度丢失等，能检查的漏洞类型较为全面。

3 种工具都可检测的漏洞为利用 *tx.origin* 授权、未检查调用返回值、非预期以太币，检测到的每种具体漏洞数量如图 4 所示。其中 Slither 报告的未检查调用返回值漏洞明显多于本系统，这是因为 Slither 只检测是否存在低级消息调用语句，Beosin-VaaS 对未检查调用返回值漏洞的检测逻辑与之类似。本系统对未检查调用返回值漏洞的检测依据不仅需要找到低级调用语句，还需要确认该语句的返回值是否用于判断，不存在对应的判断语句时才会报告漏洞。由于 Slither 与 Beosin-VaaS 均未对此类漏洞进行深入甄别，与本系统对此类漏洞的检测目标和检测逻辑不同，因此选择非预期以太币漏洞与利用 *tx.origin* 授权漏洞进行实验。

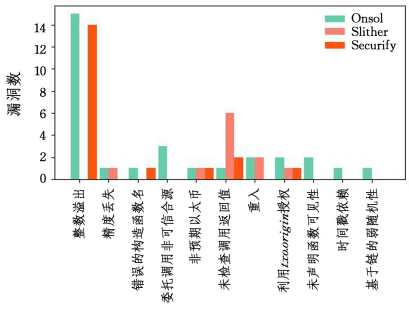

图 4 具体漏洞数

Fig. 4 Number of vulnerabilities

此外，本系统还能对检测出的合约漏洞给出其漏洞形成原因的相关信息，如图 5 所示。系统利用本体推理机 Hermi 对本体进行推理，检测出某实体具有所定义的重入漏洞的所有特征，于是判定该智能合约存在漏洞，并给出漏洞相关信息：函数 Function 类下的函数 *withdraw* 有重入漏洞，其中 call，value，balances 等数据属性对应的属性值为 3，3，5，符合定义的重入漏洞推理规则。

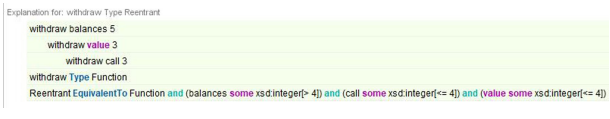

图 5 重入漏洞形成原因的解释

Fig. 5 Explanation of reentrancy vulnerability

5.2.2 效率对比

实验随机选取 100 个已部署在以太坊的智能合约作为检测样本。根据上文所述，选择非预期以太币漏洞和利用 *tx.origin* 授权漏洞进行实验。经核实，在 100 个智能合约中，非预期以太币漏洞与利用 *tx.origin* 授权漏洞的数量均为零，但是 Beosin-VaaS 与 Slither 对这两类漏洞的检查结果均存在误报，而本系统的检测结果中未见这两类漏洞。通过分析发现，Beosin-VaaS 利用 *tx.origin* 授权漏洞误报的情况是，智能合约源代码使用了 *tx.origin* 判断调用者是否为一个合约，以此实现一个避免钓鱼攻击的函数修饰符，而针对利用 *tx.origin* 授权漏洞的攻击方式正是钓鱼攻击，因此此处不存在该漏洞。Slither 对非预期以太币漏洞误报的情况是，智能合约源代码的判断语句是用于检查所有者是否为零地址，并非判断合约余额的值等于某数，因此此处不存在该漏洞。

由于非预期以太币漏洞与利用 *tx.origin* 授权漏洞出现在普通智能合约中的频率过低，实验结果不足以作为评价依据，因此选择整数溢出漏洞作为本系统与 Beosin-VaaS 的对比评价依据，选择精度丢失漏洞作为本系统与 Slither 的对比评价依据。实验结果如表 4 所列。

表 4 3 种工具检测的结果

Table 4 Detection results of three tools

漏洞	检测工具	正确数	漏报数	误报数
整数溢出	Onsol	61	4	9
	Beosin-VaaS	41	2	41
精度丢失	Onsol	9	1	1
	Slither	1	2	7

3 个工具的误报率与漏报率如图 6 所示。本系统对整数溢出漏洞检测的准确率为 87.1%，漏报率为 6.1%；对比工具 Beosin-VaaS 对整数溢出漏洞检测的准确率为 50%，漏报率为 4.7%。本系统对精度丢失漏洞检测的准确率为 90%，漏报率为 10%；对比工具 Slither 对精度丢失漏洞检测的准确率为 12.5%，漏报率为 66.7%。

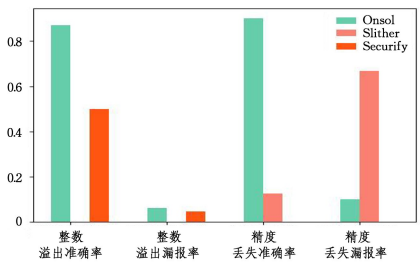

图 6 检测效率对比

Fig. 6 Comparison of detection efficiency

对检测结果进行整理后可以发现，Beosin-VaaS 对整数溢出漏洞产生误报主要有以下 3 种情况：1）一些条件语句中存在算术运算；2）一些条件语句过长，并分为多行；3）构造函数中存在未检查参数的算术运算。Slither 对精度丢失漏洞的检测准确率较低，主要原因是除数为常数的情况比重较大。当除数为非零常数时不需对除数进行判断，但是工具无法准确获取这一信息，从而导致大量误报。由于本系统在信息提取、数据处理模块充分考虑到了这些情况，因此提高了生成谓词的准确度，在很大程度上减少了误报的产生。

分析以上实验可以看出，和符号执行、形式化验证等方法相比，本文所提出的技术方案在漏洞检测类型方面更加全面，且准确率和漏报率等性能较好。

结束语 为了检测智能合约漏洞，本文设计了一个基于本体推理的智能合约漏洞检测系统。该系统把智能合约源码

抽象为 AST，再对 AST 进行漏洞信息抽取，利用抽取到的信息建立漏洞本体，进行智能合约漏洞推理，并给出漏洞形成原因。实验部分与开源检测工具 Slither 和商业工具链安 Beosin-VaaS 两种漏洞检测工具进行对比，结果表明种系统具有良好的检测效果。本系统检测漏洞全面，能检测 11 种智能合约漏洞；检测效率高于其余两个检测工具，其中整数溢出漏洞的准确率为 87.1%；此外，相比其他漏洞检测工具，本系统能在检测出相关漏洞的同时，给出漏洞的相关信息。

本系统是针对 Solidity 源码层面的漏洞检测，在检测如虚拟机层面的某些类型的漏洞或非 Solidity 语言的漏洞时有局限性；其次，本系统自动化程度不高，在面对大量的漏洞检测时，需要较多的人工参与。在未来的研究中，可以使用该方法对其他非 Solidity 智能合约类语言做扩展应用和研究；同时，可以向知识图谱方面迁移，未来向人工智能方面发展，朝着智能合约漏洞检测更自动化和智能化的方向探索。

参 考 文 献

[1] WU H, ZHANG Z, WANG S, et al. Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques[C]// 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2021: 378-389.

[2] LIU Z, QIAN P, WANG X, et al. Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion[J]. arXiv: 2106.09282, 2021.

[3] ZHOU E, HUA S, PI B, et al. Security assurance for smart contract[C]// 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2018: 1-5.

[4] WANG B, CHU H, ZHANG P, et al. Smart Contract Vulnerability Detection Using Code Representation Fusion[C]// 2021 28th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2021: 564-565.

[5] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]// 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019: 8-15.

[6] BEOSIN-VAAS. Smart Contract Formal Verification Platform [EB/OL]. [2022-06-17]. https://vaas.beosin.com/#/home.

[7] NI Y D, ZHANG C, YIN T T. A Review of Smart Contract Security Vulnerability Research[J]. Journal of Cyber Security, 2020, 5(3): 78-99.

[8] FU M L, WU L F, HONG Z, et al. Research on vulnerability mining technique for smart contracts[J]. Journal of Computer Applications, 2019, 39(7): 1959-1966.

[9] NIKOLIĆ I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018: 653-663.

[10] 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. 2016: 254-269.

[11] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. A semantic framework for the security analysis of ethereum smart contracts[C]// International Conference on Principles of Security and Trust. Cham: Springer, 2018: 243-269.

[12] KALRA S, GOEL S, DHAWAN M, et al. Zeus: analyzing safety of smart contracts[C]// NDSS. 2018: 1-12.

[13] QIAN P, LIU Z, HE Q, et al. Towards automated reentrancy detection for smart contracts based on sequential models[J]. IEEE Access, 2020, 8: 19685-19695.

[14] ZHUANG Y, LIU Z, QIAN P, et al. Smart Contract Vulnerability Detection using Graph Neural Network[C]// IJCAI. 2020: 3283-3290.

[15] AST Explorer online tools [EB/OL]. [2022-06-23]. https://astexplorer.net/.

[16] Protégé official website [EB/OL]. [2022-07-21]. https://protege.stanford.edu/products.php.

[17] Protégé cellfie-plugin [EB/OL]. [2022-07-21]. https://github.com/protegeproject/cellfie-plugin.

[18] NIKOLIĆ I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018: 653-663.

[19] Beauty Chain Integer Overflow [EB/OL]. [2022-06-27]. https://www.36kr.com/p/1722463027201.

CHEN Ruixiang, born in 1997, postgraduate, is a member of China Computer Federation. His main research interests include network security and blockchain.

JIAO Jian, born in 1978, Ph.D, professor, is a member of China Computer Federation. His main research interests include network security and blockchain.

（责任编辑：喻藜）