基于模糊测试的以太坊智能合约漏洞检测工具

南京大学 吴秉乐 朱梓源 陈昌繁 陈鹏宇

（南京大学 计算机科学与技术系，江苏 南京 210023）

指导教师 卜磊 教授

摘　要：近年来的各项网络技术之中，区块链与分布式计算倍受关注。以太坊将去中心化货币应用与智能合约相结合，成为了全球最大的分布式应用软件开发平台。由于区块链的特性，搭载于以太坊上的智能合约必须有足够的安全性，否则数字资产风险过大，进而影响整个以太坊生态。然而，作为新兴技术，针对区块链的检测技术还未能如针对主流编程语言的技术那样成熟。鉴于对智能合约安全性的迫切需求，我们借助软件工程中的模糊测试技术，在智能合约安全性测试方面进行探索，通过模糊测试依靠变异输入覆盖路径的性质，避免静态分析的路径爆炸问题，并在此基础上开发自动化的测试工具，且兼顾用户定制化的需求。

英文摘要：Among various network technologies in recent years, blockchain and distributed computing have attracted much attention. Due to the feature of the blockchain, the smart contracts on Ethereum platform must have sufficient security, otherwise the risk of digital assets will be too high, which will affect the entire Ethereum ecosystem. We use the fuzzing technology in software engineering to develop automated testing tools and take into account the needs of user customization.

关键词：区块链；智能合约；模糊测试

一、背景简介

（一）区块链技术简介

区块链无疑是当前时代下最为炙手可热的技术之一。2008 年，第一个区块链由中本聪（Satoshi Nakamoto）《比特币：一种点对点的电子现金系统》一文概念化而出现。2009 年 1 月 3 日，随着序号 0 的创世区块诞生，基于中本聪概念的加密货币比特币正式诞生，区块链技术从此从理论走向现实，进入蓬勃生长的阶段。

区块链是一个分布式的共享账本和数据库，一个区块链包含一串使用密码学方法相关联的数据块（区块），每个数据块中包含一批次的区块链交易信息，用以验证区块链的信息有效性，类似于区块链网络内部所有人各自持有内容相同的账本，交易在所有账本上同时记录。区块链网络内部维护同一串区块的唯一与有效共识，如果某些节点对数据块的数据进行篡改，由于交易的可验证性，这一个有问题的区块将不会被大多数节点共识承认，也就不会影响整体区块链的可信性，类似于一个错误的账本内容只会被社群里其他账本纠错。因其通过分布式数据存储、点对点传输、共识机制、加密算法等技术的集成，具有去中心化、不可篡改、全程留痕、可以追溯、公开透明、集体维护等特征，可以极大

国家级大学生创新创业训练计划支持项目（项目批准号 G201910284079）

作者简介：吴秉乐（1999-），男，陕西西安人，计算机与金融工程专业，本科四年级。朱梓源（1998-），男，广西贺州人，计算机与金融工程专业，本科四年级。陈鹏宇（1999-），男，河北沧州人，计算机与金融工程专业，本科四年级。陈昌繁（1999-），男，浙江温州人，计算机与金融工程专业，本科四年级。

的推动数据的可信存储、商业协同、数据可信的交换和分享，以及随之诞生的新兴商业模式，从而构建可信交易环境，打造可信社会。[1][2]

（二）智能合约与以太坊简介

在去中心化数字货币的发展中，中本聪的区块链是第一个可靠的去中心化解决办法。现在，开发者们的注意力开始迅速地转向比特币技术的第二部分：区块链怎样应用于货币以外的领域，智能合约（smart contract）应运而生。一个智能合约是一套以数字形式定义的承诺（promises），包括合约参与方可以在上面执行这些承诺的协议。智能合约的诞生使得区块链上的交易更为智能和可信，如果说基础区块链是一个可信的记账人的话，运行于区块链之上的智能合约就是一个可信的交易中间人，使得区块链的交易从简单的转账交易跨越到依托自动执行的智能合约代码的类型丰富的交易方式。

通过实现区块链上的智能合约，以太坊（Ethereum）基于脚本、竞争币和链上元协议（on-chain meta-protocol）概念进行整合和提高，使得开发者能够创建任意的基于共识的、可扩展的、标准化的、特性完备的、易于开发的和协同的应用。以太坊通过建立终极的抽象的基础层——内置有图灵完备编程语言的区块链——使得任何人都能够创建合约和去中心化应用，并在其中设立他们自由定义的所有权规则、交易方式和状态转换函数。以太坊因此成为第一个、也是目前最大的开源公共智能合约区块链平台。

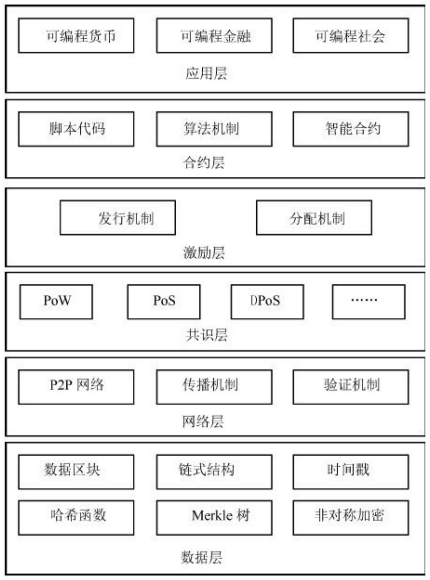

图 1 区块链架构

此后，一般的区块链系统由数据层、网络层、共识层、激励层、合约层和应用层组成。其中，数据层封装了底层数据区块以及相关的数据加密和时间戳等基础数据和基本算法；网络层则包括分布式组网机制、数据传播机制和数据验证机制等；共识层主要封装网络节点的各类共识算法；激励层将经济因素集成到区块链技术体系中来，主要包括经济激励的发行机制和分配机制等；合约层主要封装各类脚本、算法和智能合约，是区块链可编程特性的基础；应用层则封装了区块链的各种应用场景和案例。[3]

（三）智能合约的安全隐患

以太坊拥有搭建于区块链上的图灵完备的虚拟机（EVM），可以编写完整的程序。以太坊作为开源的公共区块链平台，其所有用户都可以看到基于区块链的智能合约。这就意味着包括安全漏洞在内的所有漏洞都是可见的。如果智能合约开发者疏忽或者测试不充分，造成智能合约的代码有漏洞的话，就非常容易被黑客利用并攻击。并且越是功能强大的智能合约，就越是逻辑复杂，也越是容易出现逻辑漏洞。同时，智能合约语言 Solidity 自身与合约设计都有存在漏洞的可能性。

以太坊在发展过程中已经出现过多次重大安全事件。2016 年 6 月 17 日发生了在区块链历史上沉重的一次攻击事件。由于以太坊的智能合约存在着重大缺陷,区块链业界最大的众筹项目 TheDAO 遭到攻击，导致 300 多万以太币资产被分离出 TheDAO 资产池。2017 年 7 月 21 日，智能合约编码公司 Parity 警告 1.5 版本及之后的钱包软件存在漏洞，据 Etherscan.io 的数据确认有价值 3000 万美元的 15 万以太币被盗。2017 年 11 月 8 日，以太坊 Parity 钱包再出现重大 bug，多重签名漏洞被黑客利用，导致上亿美元资金被冻结。[4]

以太坊的智能合约目前已知存在 Solidity 漏洞、短地址漏洞、交易顺序依赖、时间戳依赖、可重入攻击等漏洞。对于合约调用者，对开源甚至不开源的源码不完全熟悉，不能期待其对合约风险有完整理解，调用他人的智能合约可能会遭到隐含的漏洞的利用；对于合约创建者，由于智能合约部署后难

以更新，合约的漏洞可能造成极大的损害。[5]因此，合约安全性的检验是十分必要的。

二、研究内容

根据研究方向，项目研究划分为以下几个部分：基于合约 ABI 与运行结果生成模糊测试用例，以太坊虚拟机运行关键过程的插桩，基于插桩结果的自动化分析。研究内容也将按以上三个部分划分进行详细的说明。

（一）基于合约 ABI 与运行结果生成模糊测试用例

在本部分，我们的研究目标是：能够基于智能合约函数的应用程序二进制接口（Application Binary Interface，ABI）,有针对性地生成模糊测试用例，并借助运行过程的信息帮助调整测试用例的生成，以尽可能快速地覆盖程序分支。

模糊测试是一种灰盒测试，在静态分析的时间开销非常大时，模糊测试可以更容易地达到静态分析无法到达的分支。它的基本流程是：生成模糊测试数据，执行模糊测试数据，监视运行异常，检测是否存在漏洞。[6]

在以太坊中，智能合约的函数运行所提供的参数是有指定类型的，其中既包括常见的数字类型，也包括区块链上的合约地址。因此，不能简单地像通常的模糊测试那样，对数据进行随机变异就作为模糊测试的输入。

因此，除了程序字节码之外，本项目还要求用户提供智能合约的 ABI，以快速地确定合约函数输入的数据类型，精准地产生模糊输入。

在每一次函数运行之后，会产生一些运行记录，其中记录了运行时的堆栈状态和分支覆盖情况。基于这些运行记录，本项目提供了一个分析框架，并提供了一个基础的模糊测试输入发生器，基于运行时的操作数和堆栈数据，将可能有用的数字和合约地址提取出来作为输入种子池，并基于种子池产生更多的模糊测试输入。

当然，用户更可以利用分析框架，根据自身需求定制模糊测试输入的产生算法，而不需要关心整个模糊测试运行的流程，使得相关开发工作更为简便高效。

（二）以太坊虚拟机运行关键过程的插桩

在本部分，我们的研究目标是：在以太坊虚拟机的实现代码中进行关键信息的插桩，为后续的分析提供格式化的输出日志。

在智能合约的运行过程中，并不是每一个操作的执行都需要详细地记录下来。因此主要的记录对象就是 CALL 指令和部分比较关键的指令。

CALL 指令是以太坊虚拟机的关键指令，每一个函数的调用都是通过 CALL 指令实现的，而 CALL 指令也包含了比较重要的数据，例如调用者和调用对象 CALLER、CALLEE，调用时所携带的数据 INPUT，调用时提供的燃料 GAS（智能合约需要付出资金才能进行，资金多少的衡量就是基于 GAS），等等。

除了记录 CALL 指令外，插桩过程还记录了 CALL 的层级，即可以通过插桩信息还原调用树。和一般的程序一样，智能合约中的函数也是可以调用其它函数的，因此也就有了调用树。调用树的作用，举例来说，可以发现是否存在重入漏洞，这也是一个非常著名的智能合约漏洞，曾导致了轰动一时的资产盗窃案件，致使以太坊区块链发生了硬分叉。

除了 CALL 指令，我们还监测了一些关键的指令，例如时间戳获取指令 TIMESTAMP，比较指令，算术相关指令等。例如，以太坊中并没有禁止算术溢出，而相关案例表明，溢出漏洞可能会造成严重的财产损失。因此，通过对算术指令的插桩，可以观察到在模糊测试的运行阶段是否发生过溢出，并对用户进行警示。

本项目的插桩框架具有极高的可扩展性，如果用户有相应的需求，可以非常快捷地加入针对其它指令的插桩。

```
func execute(){
    if opcode == CALL{
        记录运行参数
        Call(...)
        记录运行结果
    }
    else {
        记录opcode名称
        记录运行状态
        res, err = operation.execute(&pc, in, contract, mem, stack)
    }
}
```

图 2 插桩位置的伪代码示意

```
<call>
    <type>call</type>
    <caller>0x2159e7aAEe5f45F6493F62B7e185020C58853AFa</caller>
    <callee>0x8d5826209CC09d96550E1444ef7F19e72682e4c3</callee>
    <value>134</value>
    <gas>268412775</gas>
    <input>[39 215 135 76 0 0 0 0 0 0 0 0 0 0 0 0 0 231 200 27 11 1
    <opcodes>
    </opcodes>
    <error>invalid opcode 0xfd</error>
    <remainGas>0</remainGas>
</call>
```

图 3 输出结果，以 xml 格式记录

（三）基于插桩结果的自动化分析

在本部分，我们的研究目标是：实现基于插桩结果的自动化分析工具，在提供基本漏洞检测的功能的基础上，提供漏洞分析框架，实现漏洞检测算法的可定制性。

本项目的自动化分析工具是基于测试预言（Oracle）的。也就是说，当调用树及其内部包含的运行结果符合预先设置的预言后，就认为存在漏洞。当分析结束后，会生成漏洞报告，并提示用户解决的办法。

下面举一些例子来说明如何基于插桩信息进行自动化分析。以最知名的重入漏洞为例，包含漏洞的程序通常存在这样的代码结构：

```
func withdraw(uint amount){
    if amount < balance[caller]{
        send(caller, amount)
        balance[caller] -= amount
    }
}
```

图 4 一个典型的重入漏洞代码

这段代码先将指定数量的金额发送给调用者，再从合约本身储存的余额记录中扣除已取出的金额。但是，根据以太坊合约的运行机制，在发送的过程中，会调用对方合约的回调函数。此时，恶意合约可以在回调函数中再次调用取款函数，此时合约内的扣除余额操作并未进行，因此该合约将持续向恶意合约转账，直到合约将所有人存储的金额都转出，因为余额不足而停止执行。

当重入漏洞存在时，调用树中会呈现这样的特征：存在一个节点是另一个节点的后代，但是这两个节点的 CALL 特征完全相同。如果我们在调用树中发现了这样的特征，就认为存在重入漏洞。

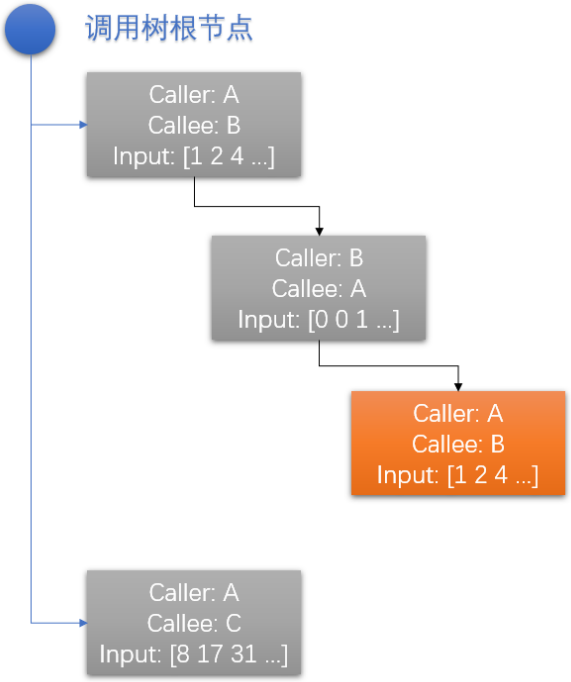

图 5 重入漏洞的调用树示例

以下是本项目提供的基础测试预言[5]：

- Timestamp/Blockstamp 依赖：如果合约执行过程中有 timestamp/blockstamp 语句，并且是 Send 函数（gas 限制为 2300），则存在 Timestamp/Blockstamp 依赖。这个依赖可能导致假随机并被恶意利用。
- 重入漏洞：如果存在调用后代与自身完全相同，则存在重入漏洞。
- 溢出漏洞：如果两个算术操作数进行运算会导致溢出，则存在溢出漏洞。
- GaslessSend 漏洞：若 Send 函数返回 out of gas 错误，则存在 GaslessSend 漏洞。这个漏洞可能导致在转账未成功的情况下完成了转账时才会发生的行为。
- 调用链异常漏洞：以太坊的机制导致调用失败时会回退，但是低级的 call 方法不会导致上级函数失效，而只是返回一个 False，导致用户无法得知调用链出错。因此，当调用链出现错误，但是根调用没有报告异常，则认为出现调用链异常漏洞。

- 危险的 DelegateCall：DelegateCall 可以让让其他合约在本合约上下文执行，但如果让用户自行指定输入，那么就可能可以使当前合约执行其他合约的任意功能。因此本测试预言检查 call 的类型是否为 DelegateCall，并检查输入是否为用户自己指定的。

除了已有的这些测试预言，本项目还为测试预言提供了接口，用户在使用自动化分析工具时，可以根据自己的需要，自定义测试预言的规则，程序在分析调用树时会自动与所有测试预言进行比对。

三、研究成果

经过一年多的探索和开发，我们成功实现了一个面向以太坊智能合约的模糊测试工具套件。工具套件包含自动化测试调用器、修改后的插桩测试 Geth 客户端、测试分析与用例生成器、报告分析与生成器。套件具有完成基本模糊测试、对以太坊智能合约提供非预期输入并监视异常结果的功能。

（一）自动化测试调用器

这个部分是默认情况下工具套件主要操作部分。使用者只需要调整参数使得调用器可以链接到部署着待测试合约的以太坊区块链上，再填写目标合约部署的地址，准备调用必须的 abi 文件，启动工具即可对该合约开始自动化的模糊测试。

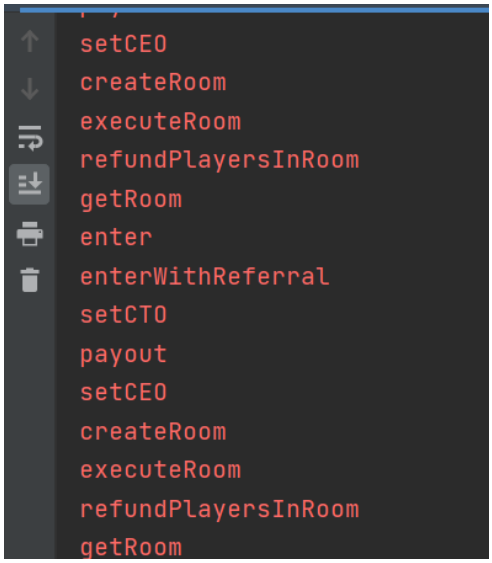

图 6 多轮逐个测试函数

（二）修改后的插桩测试 Geth 客户端

为获取插桩输出的数据，使用者应该使用该份基于以太坊官方 Go 语言客户端 Go-Ethereum 即 Geth 修改之后的客户端启动区块链节点，从而此后的每笔运算会经过修改过的 EVM 虚拟机，使得插桩代码工作并输出信息。在启动测试后，合约调用的交易信息会反映在控制台输出。

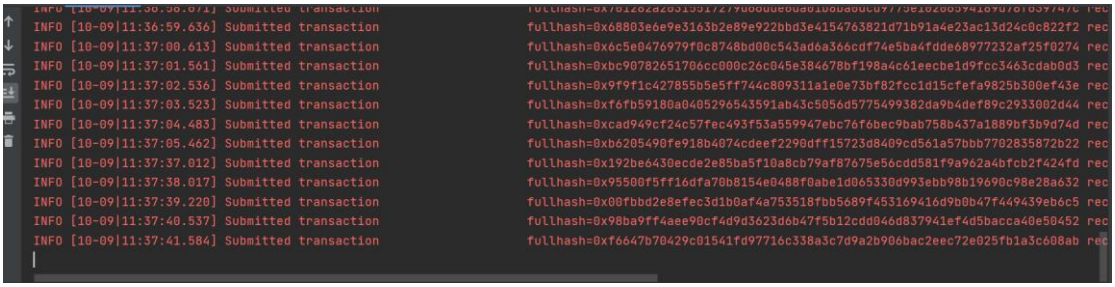

图 7 成功的测试调用

```
4   <JUMPI></JUMPI>
5   <JUMPI></JUMPI>
6   <call>
7   <type>call</type>
8   <caller>0x07621E6d14C2990421cAB82F88C1f74bBaC3085f</caller>
9   <callee>0xbE6B04c4bd1b5BB95cc3FB0D39021cCC45E1e25F</callee>
10  <value>15</value>
11  <gas>268414055</gas>
12  <input>[11 126 156 68 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13  <opcodes>
14  </opcodes><error>invalid opcode 0xfd</error>
15  <remainGas>0</remainGas>
16  </call>
17  <LT></LT>
18  <JUMPI></JUMPI>
19  <EQ></EQ>
20  <JUMPI></JUMPI>
21  <JUMPI></JUMPI>
```

图 8 插桩获取漏洞相关信息

（三）测试分析与用例生成器

该部分为了提高模糊测试的测试效率，避免测试沦为随机测试，可以基于 EVM 虚拟机内插桩获得的堆栈数据，将可能有用的数字和合约地址提取出来作为输入种子池，并基于种子池产生更多的模糊测试输入，以测试逻辑条件分支以及运算限制的边界状况各侧的智能合约运行情况。

```
1745    <data>
1746    <mem>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1747    <stack>[192846916]</stack>
1748    </data>
1749    <data>
1750    <mem>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1751    <stack>[192846916 134]</stack>
1752    </data>
1753    <data>
1754    <mem>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1755    <stack>[192846916 134 134]</stack>
1756    </data>
1757    <data>
1758    <mem>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1759    <stack>[192846916 134 0]</stack>
1760    </data>
1761    <data>
1762    <mem>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1763    <stack>[192846916 134 0 167]</stack>
1764    </data>
1765    <data>
```

图 9 用于调整用例生成的堆栈数据

（四）报告分析与生成器

在模糊测试已经基本完成，插桩信息已经成功输出的情况下，最终借助分析漏洞相关信息并总结测试覆盖率，生成针对目标智能合约的本次模糊测试的最终报告。

图 10 分析报告样例

四、总结

经过一年多的探索和开发，加上疫情期间结题被迫搁置，本项目组终于得以结题，并成功实现了一个面向智能合约的模糊测试工具。

事实上，智能合约一直都在受到安全领域的关注，然而区块链技术毕竟与以往的程序模型存在不同，所需要关注的问题也不同。这使得现有的漏洞检测技术在移植到以太坊智能合约这样的平台时，需要有针对性地在模型与方法上做出一些调整。在这一过程中，包括静态分析等技术率先得到人们的关注和研究，但是以模糊测试为代表的灰盒测试依然是一个亟待开发的领域。本项目的意义不仅仅是实现了智能合约的模糊测试，更重要的是从输入到分析，制作了一套方便快捷的测试工具，使得用户不只是使用一个预先制作好的工具，而是可以基于工具提供的框架完成进一步的定制化测试。

在项目的开发过程中，我们接触到了各方面的知识，例如软件工程领域的各种测试技术，区块链与分布式账本的概念，以太坊智能合约的开发，以太坊虚拟机的底层原理及实现方法等等，这让我们极大地拓宽了我们的知识视野，加深了对测试技术与智能合约交叉领域的了解，在导师与指导学长的帮助下对以太坊领域有了更深入和清晰的认知，充分地掌握了智能合约背后的原理。

在团队合作方面，大家在项目的进程中都十分积极主动，除了在线上实时汇报进展情况，在仓库中提交自己的学习笔记外，还定时在图书馆组织线下交流，总结阶段性成果，确定下一阶段的目标。当组员们遇到问题，大家都会尽可能地提出解决的办法，互帮互助，让每个人都能对项目有深刻和清晰的理解，实现更为完善的功能。

总之，本创新项目在智能合约安全领域实现了基于模糊测试技术的漏洞检测工具，在实现相应技术的同时也充分地做到了高可扩展性，对用户的合约开发与测试业务有着极大的助力。对于项目组成员来说，我们不仅个人的视野和能力都得到了进步，团队合作的意识也大大地加强，每个人都受益非凡。

最后特别感谢卜磊老师和沈思远学长给予我们的指导和帮助！感谢在项目开展过程中所有给予我们帮助支持和鼓励的同学们！

参考文献：

[1]百度百科.区块链[EB/OL].https://baike.baidu.com/item/区块链/13465666.2020-10-22

[2]李拯.人民日报人民时评：区块链，换道超车的突破口[EB/OL].
http://opinion.people.com.cn/n1/2019/1104/c1003-31434810.html.2019-11-04

[3]百度百科.共识层[EB/OL].https://baike.baidu.com/item/共识层/22448420.2018-08-17

[4]伍旭川,秦谊.The DAO事件，区块链征途上的一场暴风雨[EB/OL].清华金融评论.2016-06-27

[5]Bo Jiang,Ye Liu,W. K. Chan. ContractFuzzer: fuzzing smart contracts for vulnerability detection[P]. Automated Software Engineering,2018.

[6]张雄,李舟军.模糊测试技术研究综述[J].计算机科学,2016,43(05):1-8+26.