基于蜕变测试的区块链智能合约漏洞检测方法

陈锦富[1,2]，王震鑫[1,2]，蔡赛华[1,2]，冯乔伟[1,2]，陈宇豪[1,2]，许容天[1,2]，Patrick Kwaku Kudjo[3]

（1. 江苏大学计算机科学与通信工程学院，江苏 镇江 212013；2. 江苏省工业网络安全技术重点实验室，江苏 镇江 212013；
3. 威斯康星国际大学学院商业计算系，阿克拉 RE 00233）

摘 要：针对现有测试方法的缺陷，提出了一种基于蜕变测试的区块链智能合约漏洞检测方法，其能针对区块链智能合约中具体的功能生成针对性的测试用例，从而检测区块链智能合约中存在的漏洞。针对可能出现的安全漏洞，设计了不同的蜕变关系并进行蜕变测试。通过验证源测试用例和后续测试用例之间是否满足蜕变关系，判断智能合约是否存在相关的安全漏洞。实验结果表明，所提方法可以有效地检测出智能合约中存在的安全漏洞。

关键词：软件测试；区块链；智能合约；安全漏洞；蜕变测试

中图分类号：TP311

文献标志码：A

DOI: 10.11959/j.issn.1000−436x.2023190

Vulnerability detection method for blockchain smart contracts based on metamorphic testing

CHEN Jinfu[1,2], WANG Zhenxin[1,2], CAI Saihua[1,2], FENG Qiaowei[1,2],
CHEN Yuhao[1,2], XU Rongtian[1,2], Patrick Kwaku Kudjo[3]

1. School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China
2. Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, Zhenjiang 212013, China
3. Department of Business Computing, Wisconsin International University College, Accra RE 00233, Ghana

Abstract: Aimed at the defects of existing test methods, a vulnerability detection method for blockchain smart contracts based on metamorphic testing was proposed, which could generate test cases for specific functions in blockchain smart contracts to detect possible vulnerabilities. According to the possible security vulnerabilities, different metamorphosis relationships were designed and then metamorphic testing was performed. Through verifying whether the metamorphic relationship between the source test case and the subsequent test case was satisfied, whether the smart contract had related security vulnerabilities was judged. The experimental results show that the proposed method can effectively detect the security vulnerabilities in the smart contracts.

Keywords: software testing, blockchain, smart contract, security vulnerability, metamorphic testing

0 引言

随着区块链的发展，智能合约的使用得到了极大的普及[1]。由于不需要第三方就可以直接被调用并且自动执行交易[2-4]，智能合约的影响力已经扩散到很多方面。同时，针对智能合约的攻击造成经济

收稿日期：2023−03−17；**修回日期：**2023−09−05

通信作者：蔡赛华，caisaih@ujs.edu.cn

基金项目：国家重点研发计划基金资助项目（No.2020YFB1005501）；国家自然科学基金资助项目（No.62172194, No.62202206, No.U1836116）；江苏省自然科学基金资助项目（No.BK20220515, No.BK20202001）；中国博士后科学基金资助项目（No.2023T160275）；江苏省青蓝工程基金资助项目

Foundation Items: The National Key Research and Development Program of China (No.2020YFB1005501), The National Natural Science Foundation of China (No.62172194, No.62202206, No.U1836116), The Natural Science Foundation of Jiangsu Province (No.BK20220515, No.BK20202001), The China Postdoctoral Science Foundation (No.2023T160275), The Qinglan Project of Jiangsu Province

损失的风险也在逐渐提升，与传统的分布式应用程序平台不同的是，区块链的智能合约平台在开放的网络中运行，任何用户都可以参与其中。这就导致了任意存在恶意的节点都可能利用一些安全漏洞非法地从智能合约中获利[5]。

目前，以太坊智能合约[6]安全事故发生的频率逐年增加，每一次安全问题带来的破坏力都是巨大的，这不仅对市场造成了毁灭性的打击，还使用户对智能合约的安全性产生怀疑。TheDao 事件[7-8]是以太坊历史上最著名的安全事件，它导致了以太坊公链的硬分叉。

虽然已经发现了智能合约安全漏洞的多种类型[9]，但是这些漏洞通常在编写智能合约代码时易被忽视或者在合约分析时没有完全被检测出来，因此仍然有大量的智能合约存在安全漏洞[10-11]。到目前为止，研究人员已经提出了多款针对智能合约的测试工具[12-13]，如 SmartDec 开发的分析工具 SmartCheck[14]，SRI 系统实验室开发的 Securify 分析工具[15]，Badruddoja 等[16]开发的 Oyente 静态分析工具，Feist 等[17]提出的智能合约分析工具 Slither。

一般来说，区块链模型由数据层、网络层、共识层、合约层、应用层组成，共识层中封装了网络节点之间的共识算法以及激励机制，在合约部署的调用过程中，数据存储在节点中的顺序与优先级均服从共识机制的算法，因此调用智能合约需要额外的支出以保证各节点之间的协同和共识。以以太坊为例，调用智能合约涉及计算资源 gas 的支付，它与以太币之间存在较复杂的换算标准，因此在动态测试中获取测试预言时将会面临更大的困难[18-19]，这将导致获取预期结果消耗的资源代价过大或无法获取预期结果。

综上，本文提出了基于蜕变测试（MT, metamorphic testing）的方法用于检测智能合约的安全漏洞。通过分析多种类型的智能合约安全漏洞，设计相应的蜕变关系，将生成的测试用例分为源测试用例和后续测试用例，并分别在部署的智能合约中执行，进而验证两组测试用例执行的结果是否符合蜕变关系。

本文的主要贡献如下。

1) 提出了一种基于蜕变测试的区块链智能合约漏洞检测方法，通过收集智能合约实例验证测试结果是否符合蜕变关系，从而判断智能合约是否存在相应的安全漏洞。

2) 从区块链智能合约漏洞触发原理的角度出发，提出了面向区块链智能合约安全漏洞的蜕变关系设计方法，并根据此方法设计了 6 种适于智能合约安全漏洞测试的蜕变关系。

1　相关工作

由于区块链具有不可篡改的特性，为了在智能合约被部署前尽可能地检测智能合约是否存在安全风险，国内外研究人员对智能合约分析工具进行了相关的研究。

由 SmartDec 开发的智能合约分析工具 SmartCheck[18]通过将 Solidity 源代码直接转换为基于 XML 的中间表示，然后根据 XPath 模式检查中间表示以识别潜在的安全漏洞，这种方法缺乏准确性，因为某些错误不能表示为 XPath 模式。例如，重入攻击漏洞很难表示为 XPath 模式，因此不会被检测到。此外，由于 Smartcheck 使用 XPath 模式来检测某些 bug 的特定语法，因此即使 bug 片段语法发生细微变化，也不会与 XPath 模式匹配。由 SRI 系统实验室开发的 Securify[19]通过解析和反编译 EVM 字节码，然后将生成的代码转换为语义事实，最后将事实与预定义的模式列表相匹配以检测安全漏洞，但是 Securify 无法检测到重入、未经检查的发送、未处理的异常，还存在 TOD 误报的问题。根据研究发现，Securify 报告的误报是因为过度的近似执行[20]。此外，Grieco 等[21]提出使用模糊测试对智能合约安全漏洞进行检测的方法 Echidna，该方法使用静态分析框架 Slither 编译合约并对其进行预处理，处理并识别智能合约中的重要常量和函数，然后进行模糊测试。该方法需要检测随机交易中的违规属性，这使对违规属性的定义显得非常重要。在软件测试中，预言问题会经常伴随着测试软件类型的多样化出现，有时难以对随机的测试用例得出的结果进行判断。因此，本文将使用蜕变测试解决在区块链智能合约领域中出现的预言问题。

蜕变测试与其他智能合约检测方法的优势对比如表 1 所示。

2　预备工作

本文所提方法主要基于区块链智能合约漏洞分析和蜕变测试，下面针对相关概念和预备工作进行介绍。

2.1　蜕变测试分析以及蜕变关系的设计

针对区块链软件的测试普遍面临预言问题。通

表 1 蜕变测试与其他智能合约检测方法的优势对比

智能合约检测方法	存在的问题	蜕变测试的优势
Securify	无法检测到重入攻击漏洞，并存在 TOD 误报的问题	有效地检测重入攻击漏洞
Echidna	对违规属性的定义非常重要，但定义违规属性可能面临预言问题	有效地应对预言问题
SmartCheck	某些漏洞无法表示为 Xpath 模式，导致 Securify 无法检测到包括重入攻击漏洞在内的部分漏洞，缺乏准确性	可以检测重入攻击漏洞，有效地增加测试的准确性

常，在执行测试用例 x 之后，需要一种称为测试预言的系统机制来检查执行结果。如果执行结果与预期结果不一致，说明 x 是导致失败的测试用例；否则，说明 x 不是导致失败的测试用例。然而，在许多现实测试中，测试预言可能不存在，或者它可能存在但资源限制使其无法使用。当预言问题发生时，许多可靠测试集问题策略的适用性和有效性都会变得有限。

目前，蜕变测试被广泛应用于多个场景，具有代表性的是将蜕变测试应用于机器学习分类器[22]。在 Weka3.5.7 版本上进行了测试，以 k-最近邻（KNN, k-nearest neighbor）和朴素贝叶斯分类器（NBC, naive Bayesian classifier）的实现为例。通过改变算法的实现来注入漏洞，然后使用交叉验证和蜕变测试方法来检测漏洞。实验证明，一些错误很难通过交叉验证检测到，但可以通过蜕变测试检测到。

蜕变测试的核心元素是一组蜕变关系（MR, metamorphic relation）[23]，它是与多个输入及其预期输出相关的目标函数或算法的必要属性。在实现蜕变测试时，首先生成一些程序输入（称为源输入）作为原始测试用例，在此基础上使用蜕变关系生成新的输入作为后续测试用例。蜕变测试会根据相应的蜕变关系验证源测试用例和后续测试用例的输出。

基于上述的指导思想，在区块链智能合约中构建蜕变关系需要满足以下两点特性。

1) 针对区块链智能合约的蜕变测试中，蜕变关系是它的核心元素。

2) 蜕变关系的生成，需要确定多个输入及其预期输出的目标函数或算法。

针对这两点特性，需要对待测的智能合约进行处理和分析，通过处理确定目标算法，作为构建蜕变关系的依据。

2.2 智能合约整型溢出漏洞原理

使用蜕变测试方法，通过设计蜕变关系的多样性可以检测多种类型的漏洞。在此之前，需要对相应类型的漏洞原理进行分析。

在区块链中，整型溢出漏洞是最常见同样也是非常严重的安全漏洞，它导致的经济损失在区块链领域中占比很高。

在 solidity 语言中，整型变量支持的整数类型以 8 bit 为步长递增，定义的范围为 2^8～2^{256}，在定义时没有特别说明的整型变量是 256 bit。如图 1 所示，一个 uint8 类型变量可以存储的数据范围为 $[0,2^8-1]$。这意味着在智能合约的使用中，通过算数操作使变量存储的内容超过整型定义的上限或下限（变量 x_1 在算数操作后溢出到 x_2 的位置），那么将会导致合约执行的结果发生错误，对使用合约的用户都将造成巨大损失。

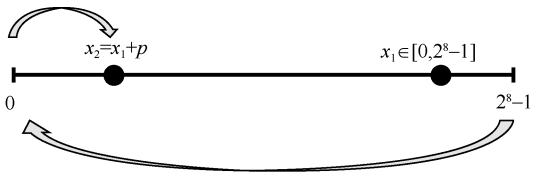


图 1 智能合约中溢出漏洞的触发原理

存在此类漏洞的智能合约通常出现在交易转账、代币发行等智能合约中。根据国家区块链漏洞库中近年来收录的有关整型溢出的漏洞案例可知，多数受到攻击的智能合约为通过代币发行的整型溢出漏洞造成项目增发无限量代币，它们通常与存储代币的参数未进行溢出漏洞的检查相关。

2.3 智能合约重入攻击漏洞原理

重入攻击是区块链系统早期最著名同时也是危害性最大的攻击方式之一[24]。近几年来，重入攻击对区块链造成了巨大的经济损失。在以太坊的智能合约中，可以声明一个不带任何参数和返回值的匿名函数，称为 fallback 函数，当对这个合约发送消息时，如果没有找到相匹配的函数就会调用 fallback 函数。如图 2 所示，攻击者调用攻击合约，在被攻击合约更新攻击者的余额之前调用 fallback 函数，fallback 函数再次调用取款方法，然后继续循环上述流程，直至攻击合约中的资源被消耗完毕或攻击合约中的余额被完全盗取。

根据 solidity 的定义，向合约发送 send、transfer、

call 消息时都会调用 fallback 函数，不同的是 send 和 transfer 有 2 300 gas 的限制，传递给 fallback 函数的 gas 只用于日志的记录。因此，如果增加其他操作则可能会超过 gas 的限制。而 call 则会把剩余的 gas 都传递给 fallback 函数，使其可能会拥有足够的 gas 进行循环调用和使用其他方法进行攻击。

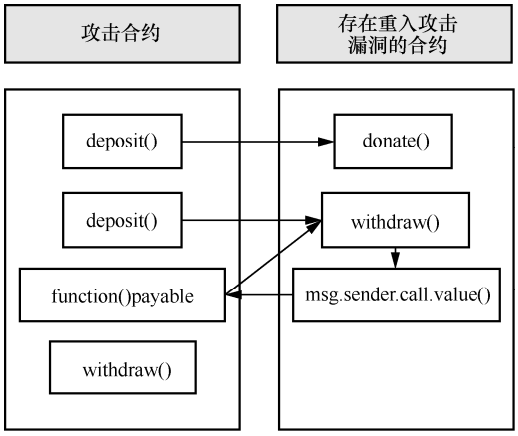


图 2　智能合约中重入攻击漏洞的触发原理

3　区块链智能合约蜕变测试方法

本节将在传统软件的蜕变测试基础上，结合区块链智能合约的特点，与传统蜕变测试的流程融合后设计出适合智能合约的蜕变测试框架。

3.1　区块链智能合约的蜕变测试特点

区块链系统中，智能合约有“公开透明”“不可篡改”的特点。随着区块链的发展，智能合约的类型更加多元，可实现的功能也更加丰富，因此带来的安全缺陷也更加多样。这意味着在针对智能合约的整个蜕变测试中，需要对智能合约进行完整的分析或预处理，包括智能合约每个方法实现的功能。根据智能合约的具体功能分析合约可能存在的潜在漏洞，这些依据对后续确定目标变量、设计蜕变关系都是至关重要的。因此，智能合约应在部署上链之前进行完备的安全漏洞测试。为了应对在区块链测试中更加频发的测试预言获取难题，本文提出了基于蜕变测试的智能合约安全漏洞检测方法。

3.2　区块链智能合约的蜕变测试原理

由于蜕变测试中蜕变关系的特殊机制，蜕变测试拥有两组相对应的测试用例输入，即源测试用例和后续测试用例，通过判断它们的输出是否符合蜕变关系从而获得测试判定。在智能合约测试中可以绕过测试预言的获取阶段，有效地应对了测试预言获取的难题，从漏洞触发角度设计蜕变关系并用于相应智能合约的蜕变测试，保证了测试的准确率。

3.3　区块链智能合约的蜕变测试方法流程

为了将蜕变测试方法在区块链智能合约中有效实现，本文提出了一种区块链智能合约蜕变测试方法。图 3 给出该方法的实施流程，整个方法分为 4 个模块：智能合约分析模块，通过分析和预处理确定待测合约的相关变量和输入域；蜕变关系设计模块，根据智能合约存在的多类型漏洞触发逻辑构建蜕变关系；测试用例生成模块以及蜕变关系验证模块。

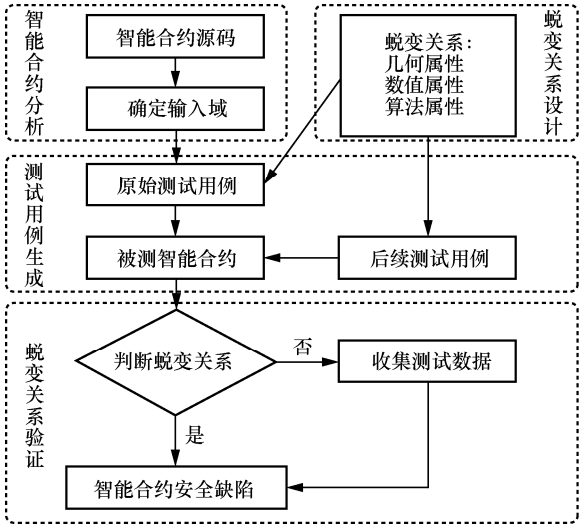


图 3　区块链智能合约蜕变测试方法

为了更全面地对区块链智能合约的蜕变测试的流程进行解释，在算法 1 中以伪代码的形式对图 3 中展示的流程进行进一步展示。其中，$k[i]$为输入域内随机生成的源测试用例，MR 为设计的蜕变关系，$k'[i]$为后续测试用例。判断输出的结果是否符合预先设计的蜕变关系，若不符合，则说明智能合约存在相应的安全漏洞。

算法 1　区块链智能合约蜕变测试算法

输入　智能合约方法 Contract()，未通过（失败）的测试用例 Test

输出　针对相应智能合约设计的蜕变关系 MRx()

1) Data=<a, b>　　//确定输入域。

2) do $k[i]$=rand()　　//随机生成源测试用例。

3) while($i \notin [a, b]$)　　//在输入域内生成源测试用例。

4) do $p[i]$=MRx($k[i]$)　　//后续测试用例生成。

5) while($i \notin [a, b]$)

6) for each element $k[i]$&&$p[i]$ do

7) MRx($k[i]$)&&MRx($p[i]$) //验证输出是否符合蜕变关系。

8) if MRx($k[i]$)!=MRx($p[i]$)

9) Test=i //若不符合蜕变关系，则说明智能合约存在相应安全漏洞。

10) return Test

3.4 区块链智能合约的蜕变测试框架

根据 3.1 节中描述的特点可知，与传统软件的蜕变测试相比，针对区块链智能合约的蜕变测试对智能合约的分析或预处理也在整体的测试流程中占有重要的作用。因此，将蜕变测试方法融入区块链智能合约中。智能合约分析、蜕变关系设计以及蜕变关系验证 3 个方面组成了整个测试流程的框架，如图 4 所示。

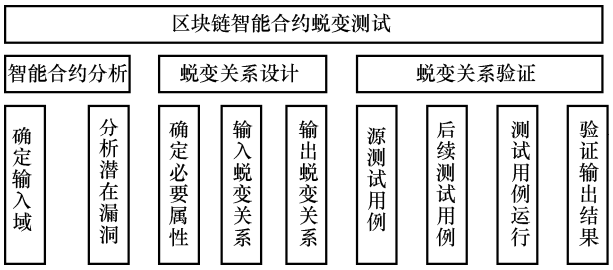


图 4 区块链智能合约蜕变测试框架

根据图 4 可以看出，除了智能合约分析，蜕变关系设计和蜕变关系验证也是整个蜕变测试的核心。因此，第 4 节将详细介绍蜕变关系的设计方法和设计过程。

4 区块链智能合约漏洞的蜕变关系设计

本节将详细展示针对第 3 节中介绍的区块链智能合约出现的漏洞的蜕变关系应如何设计。具体来说，蜕变关系的设计涉及 2 个步骤：1) 将待测的智能合约进行分析，提取可能触发漏洞的函数及目标变量，涉及蜕变关系中测试用例输入的变量，需要确定它们的输入域；2) 针对待测智能合约的功能及可能出现的漏洞类，根据潜在漏洞涉及的变量与输出之间的属性设计蜕变关系。

4.1 整型溢出漏洞蜕变关系设计——上溢

根据整型溢出的特点，将对可能存在整型溢出漏洞的智能合约进行初步分析，使生成的源测试用例和后续测试用例的蜕变关系满足触发整型溢出漏洞的必要条件。图 5 为一个存在加法溢出漏洞的智能合约的部分内容，其描述了一个银行定期储蓄的智能合约。在合约中，针对每个用户设置了一个 mapping 用于设置定期储蓄时间，值为 uint256 类型。之后的 3 个函数：deposit()为存款函数，并且储蓄时间至少为一周；increaseLockTime()为增加储蓄时间的函数，用户可以根据需求自行增加储蓄时间；withdraw()为取款函数，当用户拥有余额且当前的时间戳迟于储蓄结束后预定的时间戳后可以将储蓄余额取出。

```
1.  function increaseLockTime(uint_secondsToIncrease) public {
2.  lockTime[msg.sender] += secondsToIncrease; }
3.  function withdraw() public {
4.  require(balances[msg.sender] > 0);
5.  require(how > lockTime[msg.sender]); }
6.  function getTime() public  contant returns(uint 256){
7.   return lockTime[msg.sender] }
8.  function deposit() external payable{
9.  balances[msg.sender]+=msg.value;
10. lockTime[msg.sender]=block.timestamp+1 weeks;}
```

图 5 存在加法溢出漏洞的智能合约的部分内容

此案例为智能合约加法溢出的实例。通过分析，合约中涉及输入的主要参数为存款数和定期储蓄时间，这两类参数虽然同时影响取款操作的输出结果，但整型溢出漏洞是否触发主要在于储蓄时间 lockTime，由于设置定期储蓄时间时并未使用 SafeMath 进行溢出检测，因此通过增加定期储蓄时间，可以使储蓄时间发生上溢，使储蓄时间溢出，那么定期储蓄时间将会溢出到 0。以此跳过时间检测机制取出存款。

根据此智能合约的设计逻辑，当目标变量 x 为定期储蓄时间时，$f(x)$为取款操作执行时取出的存款，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n) \subseteq q(x_1 \vee x_2 \vee \cdots \vee x_n \vee \cdots \vee x_{2n}) \quad (1)$$

$$f(p) = f(q) \quad (2)$$

在没有触发加法溢出的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。基于此，可以设计蜕变关系 MR_1

$$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = 2x_1) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) == 0 \,\&\&\, P(x_2) > 0))))\} \quad (3)$$

其中，x_1、x_2 分别表示目标变量（储蓄时间），对应的 $P(x_1)$、$P(x_2)$分别表示在改变了储蓄时间后进行取

款操作的结果。

MR 的公式基于已提出的一阶谓词逻辑的蜕变关系形式化模型[25]，具有精确的描述形式，同时也具备较强的兼容性，MR_2～MR_6的蜕变关系也是基于此模型构建的。根据智能合约中的定义，确定了构建蜕变关系涉及的 4 个输入变量和一个输出变量：定期储蓄时间 A、存款金额 B、增加储蓄时间 P、取款金额 M 和储蓄余额在每一次测试前后的差值 S。因此，将 MR_1 的形式化公式分解为表格形式，如表 2 所示。

表 2　　MR_1 中目标变量间的蜕变关系

变量名称	变量 1	变量2	变量 3	变量 4
Source input	A_s	B_s	P_s	M_s
Source output	ΔS_s	—	—	—
Follow input	A_f	B_f	P_f	M_f
Follow output	ΔS_f	—	—	—
Input relation	A_f=A_s	B_f=B_s	P_f=2P_s	M_s=M_f
Output relation	ΔS_s=0&&ΔS_f>0	—	—	—

在该蜕变关系中，源测试用例的输入以随机测试的方法随机生成。后续测试用例生成的蜕变关系在源测试用例的基础上倍数递增，当储蓄时间递增到发生溢出时，储蓄时间将会溢出到从 0 开始，因此每次后续测试用例执行后，将执行一次取款操作。如果该操作成功执行，即取款金额与储蓄余额的差值相等，说明检测出了储蓄时间的溢出，该合约存在整型溢出的漏洞。

根据此智能合约的设计，当目标变量 x 为定期储蓄时间时，$f(x)$为取款操作后用户的余额差值，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n)\,\&\&\,q(x_1+2^{8n} \vee x_2+2^{8n} \vee \cdots \vee x_n+2^{8n}) \tag{4}$$

$$f(p)<f(q) \tag{5}$$

在没有触发溢出漏洞的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。在表 2 所示的蜕变关系的基础上，可以进一步改进对智能合约整型溢出漏洞检测的蜕变关系 MR_2

$$\{(r,[p],rf)\,|\,(r(x_1,x_2)=(x_2=x_1+2^{8n}) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2)=(P(x_1)<P(x_2)))))\} \tag{6}$$

其中，x_1、x_2 分别表示目标变量（单次转账金额），对应的 $P(x_1)$、$P(x_2)$分别表示目标变量的两次不同的实际取值（转账操作后接收方账户的余额）。

在 solidity 中，整型变量支持的整数类型以 8 bit 为步长递增，即整型定义的范围为 2^8~2^{256}，在定义时没有特别说明的整型变量是 256 bit。在蜕变测试中以 2^8 为下限，试图在一个 uint8 的整型变量中存储数字 256，则它将会溢出变成 0。

将 MR_2 的形式化公式分解为表格形式，如表 3 所示。

表 3　　MR_2 中目标变量间的蜕变关系

变量名称	变量 1	变量 2	变量 3	变量 4
Source input	A_s	B_s	P_s	M_s
Source output	ΔS_s	—	—	—
Follow input	A_f	B_f	P_f	M_f
Follow output	ΔS_f	—	—	—
Input relation	A_f=A_s	B_f=B_s	P_f=P_s+2^{8n}	M_s=M_f
Output relation	ΔS_s<ΔS_f	—	—	—

在随机生成的源测试用例的基础上，每一次生成后续测试用例时依次增加 2^{8n} 的储蓄时间，这种蜕变关系可以大大减少测试整型溢出漏洞时的时间复杂度。

通过上述 2 种关于检测智能合约整型溢出漏洞的蜕变关系，可以在实验中针对测试具体的智能合约中触发漏洞的变量进行修改，从而验证待测合约是否存在整型溢出漏洞。

4.2　整型溢出漏洞蜕变关系设计——下溢

对于智能合约中出现的减法溢出型漏洞，通常其整型变量的赋值会超出其下限。图 6 是一个存在减法溢出漏洞的智能合约的部分内容。

```
1.  constructor() public {
2.      owner = msg.sender;
3.      blances[owner] = 2 000 * 10**8; }
4.  function distribute(address[] addresses) public onlyOwner {
5.  for (uint i = 0;i < addresses.length;i++){
6.  balances[owner] −=2 000 * 10**8;
7.  balances[addresses[i]] += 2 000 * 10**8;
8.  emit Transfer(owner,addresses[i],2 000 * 10**8); }  }
```

图 6　存在减法溢出漏洞的智能合约的部分内容

此案例为智能合约减法溢出漏洞的实例。通过分析，合约中涉及输入的主要参数为接收转账用户的代币存量，如果在对 balances[owner]的计算中未使用 SafeMath，当转出代币总量大于 owner 账户余

额时，balances[owner]产生减法溢出，变成一个极大值，造成代币增发的严重后果。

根据此智能合约的设计，当目标变量 x 为单次发起转账交易的金额时，$f(x)$为转账操作后发送方账户的余额，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n)\ \&\&$$
$$q(x_1 + 2^{8n} \vee x_2 + 2^{8n} \vee \cdots \vee x_n + 2^{8n}) \tag{7}$$
$$f(p) > f(q) \tag{8}$$

在没有触发漏洞的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。基于此和 3.1 节加法溢出中改进的蜕变关系设计思路，可以设计减法溢出的蜕变关系 MR_3

$$\{(r,[p],rf)\,|\,(r(x_1,x_2)=(x_2=x_1+2^{8n}) \xrightarrow{p(x_i)=[p](x_i)}$$
$$(rf(P(x_1),P(x_2)=(P(x_1)>P(x_2)))))\} \tag{9}$$

其中，x_1、x_2 分别表示目标变量（单次转账金额），对应的 $P(x_1)$、$P(x_2)$分别表示目标变量的两次不同的实际取值（转账操作后发送方账户的余额）。

根据智能合约中的定义，确定了构建蜕变关系涉及的 4 个参数：发送方账户 A、接收方账户的个数 B、每次给接收方账户转账的金额 M、发送方账户余额在每一次测试前后的差值 S。将 MR_3 的形式化公式分解为表格形式，如表 4 所示。

表 4　　MR_3 中目标变量间的蜕变关系

变量名称	变量 1	变量 2	变量 3
Source input	A_s	B_s	M_s
Source output	S_s	—	—
Follow input	A_f	B_f	M_f
Follow output	S_f	—	—
Input relation	B_f=(B_s+2^{8n})	—	—
Output relation	S_s>S_f	—	—

在该蜕变关系中，为了检测后续测试用例是否会触发减法溢出，将给每个接收方账户转账的金额作为变量，通过增加该变量逐步增加发送方转账的总金额，当发送方账户转账金额的总量大于发送方本身的余额时，如果触发减法溢出，发送方在转账操作后的余额将会大于转账操作前的余额。

4.3　整型溢出漏洞蜕变关系设计——乘法溢出

根据合约功能的设计，若在合约功能中出现乘法运算，则可能伴随着加减法超过整型定义范围的上下限导致乘法溢出。图 6 中智能合约中转账功能的设计就伴随着乘法溢出。

在图 6 所示的合约中，发送方账户每次可以转给不止一个接收方账户等量的金额，那么当接收方账户数量递增时，发送方每次转账的总金额也会根据乘数数量的递增而增加，从而发生溢出。

根据此功能设计，当目标变量 x 为单次发起转账交易的目标账户数时，$f(x)$为转账操作后发送方账户的余额，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n)\ \&\&$$
$$q(x_1 + 1 \vee x_2 + 1 \vee \cdots \vee x_n + 1) \tag{10}$$
$$f(p) > f(q) \tag{11}$$

在没有触发漏洞的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。基于此，可以设计乘法溢出的蜕变关系 MR_4

$$\{(r,[p],rf)\,|\,(r(x_1,x_2)=(x_2=x_1+1) \xrightarrow{p(x_i)=[p](x_i)}$$
$$(rf(P(x_1),P(x_2)=(P(x_1)>P(x_2)))))\} \tag{12}$$

其中，x_1、x_2 表示目标变量（接收方账户的数量），对应的 $P(x_1)$、$P(x_2)$分别表示目标变量的两次不同的实际取值（转账操作后发送方账户的余额）。

将 MR_4 的形式化公式分解为表格形式，如表 5 所示。

表 5　　MR_4 中目标变量间的蜕变关系

变量名称	变量 1	变量 2	变量 3
Source input	A_s	B_s	M_s
Source output	S_s	—	—
Follow input	A_f	B_f	M_f
Follow output	S_f	—	—
Input relation	B_f=(B_s+1)	—	—
Output relation	S_s>S_f	—	—

在该蜕变关系中，为了检测后续测试用例是否会触发溢出，将接收方账户的个数作为变量，通过接收方账户数量自增的方式逐步增加发送方转账的总金额，当发送方账户转账金额的总量大于发送方本身的余额时，如果触发溢出，发送方在转账操作后的余额将会大于转账操作前的余额，变为一个极大值。

4.4　针对智能合约重入攻击漏洞的蜕变关系设计

根据重入攻击漏洞的原理，将针对存在重入攻击漏洞的智能合约进行初步分析，将源测试用例和生成的后续测试用例认为是满足触发重入攻击漏洞的必

要关系。和整型溢出漏洞不同的是，触发重入攻击漏洞需要通过重入攻击操作而不是改变相关参数，因此测试用例需要在涉及相应的攻击合约的基础上生成。图 7 是一个存在重入攻击漏洞的智能合约和攻击者在触发重入攻击漏洞时调用的攻击合约的部分内容。

```
1.  function deposit() public payable{
2.  balances[msg.sender] +=msg.value; }
3.  function withdraw(uint256 amount) public payable{
4.  require(balances[msg.sender]>=amount);
5.  require(this.balance>=amount);
6.  msg.sender.call.value(amount)();
7.  balances[msg.sender]-=amount; }
8.  function balanceof(address addr) constant returns(uint256){
9.  return balances[addr];}
```

图 7　存在重入攻击漏洞的智能合约和攻击者在触发重入攻击漏洞时调用的攻击合约的部分内容

存在重入攻击漏洞的代码在 6 行和 7 行，根据 1.3 节中的描述，向合约发送 call 消息存在重入攻击的风险，攻击者会在 7 行更新用户余额之前递归地从合约中取款。在分析重入攻击漏洞的原理后，针对这两行存在重入攻击漏洞的代码，设计一个重入攻击的智能合约，在后续的实验中可以通过攻击操作作为测试用例，检测智能合约是否存在重入攻击漏洞的风险。图 8 是一个发起重入攻击的合约的部分内容。

```
1.  function attack() public{
2.      re.withdraw(1 ether);  }
3.  function() public payable{
4.      if(address(re).balance>=1 ether){
5.          re.withdraw(1 ether);  }  }
```

图 8　发起重入攻击的合约的部分内容

此案例是智能合约重入攻击漏洞的实例。根据分析，在合约需要接收 ether 时，fallback 函数必须声明为 payable，因此第 4 行函数为 fallback 函数。可以看出，fallback 函数内再次调用了 withdraw 方法，使被攻击合约再次进行新一轮的转账操作，实现重入攻击。

重入攻击的次数和攻击合约的 gas 数量由被攻击合约的具体余额决定，因此可能存在假阴性的问题（合约存在重入攻击漏洞，但是 gas 数量和余额的条件不满足二次攻击的实现，会误以为被测合约不存在重入攻击漏洞）。

根据此功能设计，当目标变量 x 为 fallback 函数每次使用 withdraw 取出的金额时，$f(x)$为使用攻击合约调用该智能合约的地址余额，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n)\ \&\&\ q(x_1-1 \vee x_2-1 \vee \cdots \vee x_n-1) \tag{13}$$

$$f(p) == f(q) \tag{14}$$

在没有触发漏洞的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。为了在测试的基础上减小假阴性出现的概率，针对蜕变关系进行了优化，设计蜕变关系 MR_5

$$\{(r,[p],rf)\,|\,(r(x_1,x_2)=(x_2=x_1-1)\xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2)=(P(x_1)==P(x_2)))))\} \tag{15}$$

其中，x_1、x_2 表示目标变量（fallback 函数中每次使用 withdraw 取出的金额），对应的 $P(x_1)$、$P(x_2)$分别表示使用攻击合约模拟重入攻击时合约地址的余额数量。

根据智能合约中的定义，确定了构建蜕变关系涉及的 4 个输入变量和一个输出变量：攻击者的钱包地址 A、攻击者发动攻击使用的合约地址 B、被攻击合约的地址 P、fallback 函数中每次攻击取出的金额 M 和攻击合约中的余额 S。因此，将 MR_5 的形式化公式分解为表格形式，如表 6 所示。

表 6　　MR_5 中目标变量间的蜕变关系

变量名称	变量 1	变量 2	变量 3	变量 4
Source input	A_s	B_s	P_s	M_s
Source output	S_s	—	—	—
Follow input	A_f	B_f	P_f	M_f
Follow output	S_f	—	—	—
Input relation	M_f=M_s−1	—	—	—
Output relation	S_s==S_f	—	—	—

在该蜕变关系中，为了准确检测是否会触发合约中可能存在的重入攻击漏洞，将测试用例设置为调用攻击合约发动的重入攻击，在源测试用例测试阶段就有可能触发重入攻击漏洞。为了避免假阴性的情况，源测试用例中的目标变量将在合约规定的范围内随机生成，如果满足蜕变关系，则说明重入攻击没有触发。后续测试用例将会在源测试用例的基础上自减并进行下一轮的重入测试。直到源测试用例自减到一个

设置的极小值，如果输出的结果仍然满足输出关系，则证明被测合约不存在重入攻击漏洞。

通过后续实验发现蜕变关系 MR_5 存在局限性：如果源测试用例和后续测试用例均能成功地触发重入攻击漏洞，那么仍然无法验证该合约是否存在重入攻击漏洞。因此，对蜕变关系进行进一步的改进。

根据此功能设计，当目标变量 x 为 fallback 函数每次使用 withdraw 取出的金额时，$f(x)$为用户直接调用智能合约进行取款操作后用户地址的余额差值，$g(x)$为用户通过调用攻击合约对该智能合约进行取款操作后用户地址的余额差值，得出蜕变关系设计依据的必要属性为

$$p(x_1 \vee x_2 \vee \cdots \vee x_n)\,\&\,\& \\ q(x_1 \vee x_2 \vee \cdots \vee x_n) \tag{16}$$

$$f(p) = g(q) \tag{17}$$

在没有触发漏洞的前提下，每次在确定的输入域中根据蜕变关系改变目标变量的输入，相应的输出就必然遵循该必要属性。基于此，得到蜕变关系 MR_6

$$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1) \xrightarrow{p(x_i)=[p](x_i)} \\ (rf(P(x_1),P(x_2) = (P(x_1) == P(x_2)))))\} \tag{18}$$

其中，x_1、x_2 表示目标变量（正常取款操作和调用攻击合约的操作取出的金额），对应的 $P(x_1)$、$P(x_2)$ 分别表示 2 种操作后取款用户钱包中的余额差值。

为了充分避免假阴性的情况出现，将攻击合约中 fallback 函数中的 withdraw 函数每次递归取款的金额设置为 1。

根据智能合约中的定义，确定了蜕变关系涉及的 4 个输入变量和一个输出变量：正常取款操作 A、调用的攻击合约触发重入攻击的操作 B、被攻击合约的地址 P、2 种操作取款的金额 M 和不同操作执行后取款用户的钱包余额差值 S。因此，将 MR_6 的形式化公式分解为表格形式，如表 7 所示。

表 7　MR_6 中目标变量间的蜕变关系

变量名称	变量 1	变量 2	变量 3	变量 4
Source input	A_s	B_s	P_s	M_s
Source output	S_s	—	—	—
Follow input	A_f	B_f	P_f	M_f
Follow output	R_f	—	—	—
Input relation	M_f=M_s	—	—	—
Output relation	S_s==R_f	—	—	—

在该蜕变关系中，将目标变量转化成 2 种不同的操作，分别是正常取款和调用攻击合约取款的操作，通过对比不同操作执行的结果是否符合输出蜕变关系，判断合约是否存在重入攻击漏洞。

5　实验分析

为验证本文提出的蜕变测试方法在检测智能合约中存在相应漏洞的有效性，设计了多个存在相关漏洞的实例进行验证。本节展示了实验的细节，主要包括实验设置、实验流程、实验数据和实验结果与分析。

5.1　实验设置

存在不同漏洞的智能合约在进行蜕变测试时需要相应的蜕变关系进行测试，利用国家安全漏洞库 CNVD、CVE 和 GitHub 中收录的漏洞实例及其他文献的数据集中收集的漏洞合约实例进行实验，收集的每个实例都有完整的源代码。使用相应的蜕变关系进行蜕变测试，验证蜕变测试的有效性。

本文具体方案设置如下：在 Remix IDE 上编译和部署待测试的智能合约，版本号为 0.4.26+commit.4563c3fc，EVM 镜像选择 default，调用合约的用户地址为 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4，测试重入攻击漏洞时调用攻击合约的用户地址为 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2。蜕变测试实验中待选的蜕变关系如表 8 所示。

表 8　蜕变测试实验中待选的蜕变关系

蜕变关系编号	蜕变关系形式化公式
MR_1	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = 2x_1) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) == 0\,\&\,\&\,P(x_2) > 0))))\}$
MR_2	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1 + 2^{8n}) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) < P(x_2)))))\}$
MR_3	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1 + 2^{8n}) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) > P(x_2)))))\}$
MR_4	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1 + 1) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) > P(x_2)))))\}$
MR_5	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1 - 1) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) == P(x_2)))))\}$
MR_6	$\{(r,[p],rf) \mid (r(x_1,x_2) = (x_2 = x_1) \xrightarrow{p(x_i)=[p](x_i)} (rf(P(x_1),P(x_2) = (P(x_1) == P(x_2)))))\}$

5.2　实验流程

第 4 节已经完成了针对蜕变关系设计的工作。

针对区块链智能合约安全漏洞的蜕变测试的整体流程介绍如下。

1) 分析智能合约的具体功能和调用方法涉及的参数变量，确定测试用例生成的目标变量与输入域，并在输入域中生成源测试用例集 E。

2) 根据目标变量中存在的函数关系确定其中的必要属性，根据必要属性设计相应的蜕变关系，包括输入蜕变关系和输出蜕变关系。

3) 使用设计的蜕变关系，基于源测试用例集生成相应的后续测试用例集 E'。

4) 将源测试用例 e 和与之对应的后续测试用例 e' 在待测智能合约中运行，获取相应的输出结果。

5) 验证源测试用例 e 和对应的后续测试用例 e' 的输出结果是否符合输出蜕变关系，如果不符合，则证明测试结果不通过。

智能合约的蜕变测试总体流程如图 9 所示。

5.3　实验数据

被测合约均来源于近年来区块链的安全事故实例以及一些经典案例，其中大量的智能合约都存在的漏洞类型包含整型（加法、减法及乘法）溢出漏洞和重入攻击漏洞。实验验证的合约实例及存在的漏洞如表 9 所示。

通过对智能合约的具体功能和相应漏洞的分析，待测的智能合约使用的蜕变关系将在第 4 节中构建的 6 个蜕变关系中选取，蜕变测试流程中所有的源测试用例生成策略采取蜕变测试常用的随机测试用例生成技术[26]，即它们都来源于目标变量输入域内的随机值，在此基础上获得的源测试用例和选取的蜕变关系进一步获得后续测试用例。根据表 9 所示的合约实例，使用蜕变测试验证在区块链智能合约上的蜕变测试有效性，并且和常用的智能合约检测方法 Slither 的检测结果进行对比。

5.4　实验结果与分析

为了评估基于蜕变测试的区块链智能合约安全漏洞检测方法的有效性，针对以下 2 个问题对实验结果进行分析：MT 是否可以检测到 MR 对应的智能合约安全漏洞？MT 是否比其他智能合约测试方法更好？

1) MT 是否可以检测到 MR 对应的智能合约安全漏洞？

为了验证这个问题，根据表 8 提供的蜕变关系，使用图 9 提出的蜕变测试流程对表 9 中的智能合约

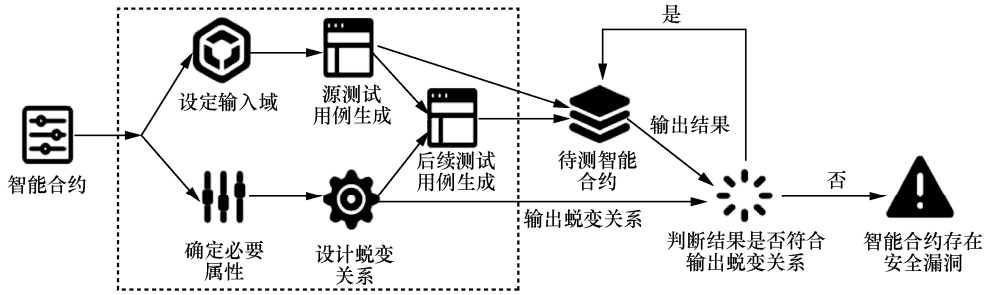


图 9　智能合约的蜕变测试总体流程

表 9　　**实验验证的合约实例与存在的漏洞**

合约编号	合约名称	漏洞类型	合约来源
1	Reentrance	重入攻击/加法溢出	Etherscan
2	Bitcoin Red	减法溢出	CVE-2018-11687
3	EtherStore	重入攻击	Etherscan
4	PolyAi	减法溢出	CVE-2018-11812
5	Internet Node Token	加法溢出	CVE-2018-11811
6	Bank	重入攻击	Etherscan
7	Beauty Ecosystem Coin	乘法溢出	CVE-2018-10299
8	Victim	重入攻击	Etherscan
9	Playkey	加法溢出	CVE-2018-11809
10	Token Example	加法溢出	Etherscan

实例进行了完整的蜕变测试，结果如表 10 所示，实验按照智能合约编号顺序依次进行检测。其中，“√”表示检测出相应的漏洞，“×”表示未检测出相应的漏洞。

表 10　　蜕变测试方法的检测结果

合约编号	合约名称	重入攻击	整型溢出	使用蜕变关系
1	Reentrance	√	√	MR_6
2	Bitcoin Red	×	×	MR_3
3	EtherStore	√	×	MR_5
4	PolyAi	×	×	MR_3
5	Internet Node Token	×	√	MR_2
6	Bank	√	×	MR_6
7	Beauty Ecosystem Coin	×	√	MR_4
8	Victim	√	×	MR_6
9	Playkey	×	√	MR_2
10	Token Example	×	√	MR_2

根据表 10 的实验结果可以看出，针对区块链智能合约的蜕变测试方法的有效性得到了验证，蜕变关系使后续测试用例的生成具有较强的针对性。显然，蜕变测试开展的基础就是蜕变关系的正确性，即需要事先确定蜕变关系一定符合程序的必要关系。如果测试输出不满足蜕变关系，则相应的测试用例中一定存在失效，即程序一定存在故障。基于这个原则，参考没有通过蜕变关系验证的输出结果（失败的测试用例），得出相应的智能合约存在漏洞的结果。

如果测试输出满足蜕变关系，则并不意味着相应的测试用例执行结果一定是正确的。由于现有的源测试用例的生成策略具有随机性，定量的测试用例可能导致测试结果存在假阴性的概率，使编号 2 和编号 4 的智能合约实例中存在的漏洞未能检测出。因此有限类型和数量的蜕变关系对不同智能合约检测仍然存在局限性。

综上，蜕变测试在区块链智能合约中的应用是可行的，并且具有有效性。

2) MT 是否比其他智能合约测试方法更好?

为了验证这个问题，使用常用的智能合约检测方法 Slither 对表 9 中的智能合约实例进行测试。Slither 方法的结果针对合约中的方法设置相应指标和潜在漏洞的置信度，具体的潜在漏洞以置信度评级的方式显示，在此将置信度不高的测试指标结果认定为 Slither 检出的漏洞。根据 Slither 的检测结果和蜕变测试的检测结果进行对比，实验结果如表 11 和图 10 所示。其中，“Ø”表示未能检测出任何漏洞。

表 11　　Slither 测试方法的测试结果

合约编号	合约名称	重入攻击	整型溢出	其他
1	Reentrance	√	×	—
2	Bitcoin Red	×	×	—
3	EtherStore	√	×	—
4	PolyAi	×	×	—
5	Internet Node Token	×	×	—
6	Bank	√	×	—
7	Beauty Ecosystem Coin	×	×	—
8	Victim	√	×	—
9	Playkey	×	×	—
10	Token Example	×	×	Ø

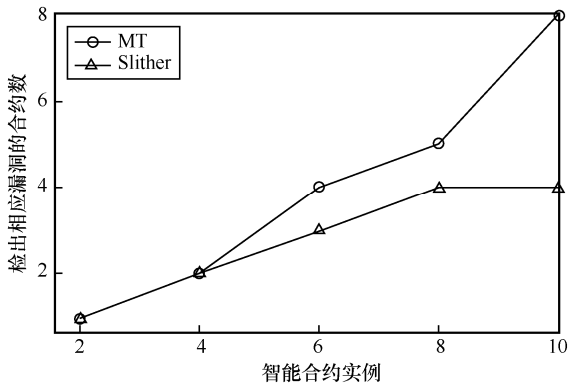


图 10　两种检测方法的检测结果

从表 11 和图 10 可以看出，蜕变测试在检测整型溢出漏洞方面明显优于 Slither 方法。在智能合约中，整型溢出漏洞是最常见的智能合约安全漏洞，但是在检测整型溢出漏洞的各种方法中，蜕变测试可以通过触发漏洞的形式以检测是否存在具体的漏洞，在准确率上占有明显优势。

相比于 Slither 方法，蜕变测试有较大优势，并且随着蜕变关系进一步的完善，这种优势将扩散到更多的漏洞类型和更广泛的智能合约应用场景中。

5.5　实验说明

根据文献[27]可知，Slither 作为一款常用且精确度较高的测试工具，由于自身局限性，截止到该文献发表日期，它还无法检测与整型溢出相关的智能合约漏洞。之所以选择 Slither 作为对比方法，是因为 Slither 工具在提高测试者对合约的了解、协助代码审查方面具有优秀的表现。在未来的工作中，将参考 Slither 的相关功能尝试为区块链智能合约的蜕变测试方法

增加预处理机制，这样可以为测试时蜕变关系的选择以及蜕变关系的设计阶段提供更客观全面的依据。

此外，蜕变关系作为蜕变测试中重要的元素组成部分，蜕变关系的识别是影响测试结果的重要因素。蜕变测试在整型溢出漏洞和重入攻击漏洞的检测中最明显的优势在于通过蜕变关系优化了动态测试面临的测试预言难以获取导致影响测试准确率的问题，由于这类漏洞的目标变量较固定，在面临不同功能的智能合约测试都有较好的泛化性，在动态测试的过程中通过触发漏洞从而进行检测也更加实用。因此蜕变测试在针对这两类漏洞的测试中有着良好的表现。本文根据已知漏洞的触发原理设计蜕变关系，但此类方法存在部分局限性，主要在于智能合约的常见漏洞并不局限于整型溢出漏洞和重入攻击漏洞，还有时间戳依赖漏洞和交易顺序依赖漏洞等，这些漏洞的触发条件也与区块链网络层的安全熟悉相关，因此暂时还无法仅通过合约层的漏洞触发原理构建蜕变关系，随着区块链智能合约的发展，漏洞类型也将进一步增加，笔者将基于此方法分析更多类型的漏洞并设计蜕变关系，使蜕变测试的适用范围更广。此外，对于未知漏洞，笔者仍然需要从输入域和输出域的关系入手，识别科学的蜕变关系。与蜕变测试在其他的场景中应用类似，此方法最重要的内容是蜕变关系的识别，因此该方法的有效性在很大程度上取决于蜕变关系的选择，这是此类方法应用到区块链智能合约领域中依然存在的挑战。在后续的工作中，寻找有效的蜕变关系识别方法[28]并应用到区块链智能合约领域将作为笔者改进此方法的一部分。

6　结束语

在区块链智能合约方面，本文提出了一种基于蜕变测试的智能合约安全漏洞检测方法，通过对实例合约的测试，检测智能合约中存在的整型溢出和重入攻击漏洞。考虑到区块链的不可篡改性和 gas 的消耗问题，蜕变测试具有明显的优势。一方面，蜕变测试避免了获取测试预言可能存在的大量资源消耗和获取困难的情况，缓解了动态测试中的 Oracle 问题。另一方面，对具体的智能合约漏洞设计了相应的蜕变关系，使蜕变测试中测试用例的生成具有更强的针对性，提升了测试过程中触发与检测相应漏洞的概率，降低了假阳性的概率。

未来的工作主要有：1) 为了覆盖更广泛的智能合约安全漏洞类型，将进一步设计更多的蜕变关系，在现有的智能合约功能和已发现的漏洞的基础上，完善关于智能合约的蜕变关系库；2) 针对整体测试框架中智能合约分析的部分，将结合 Slither 智能合约分析工具，对蜕变关系的设计和选择提供更合理的指标支撑并提高蜕变测试的自动化程度。

参考文献：

[1] CLACK C D, BAKSHI V A, BRAINE L. Smart contract templates: foundations, design landscape and research directions[J]. arXiv Preprint, arXiv: 1608.00771, 2016.

[2] 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.

[3] ATZEI N, BARTOLETTI M, CIMOLI T. A survey of attacks on ethereum smart contracts SoK[C]//Proceedings of the 6th International Conference on Principles of Security and Trust. New York: ACM Press, 2017: 164-186.

[4] 邵奇峰, 金澈清, 张召, 等. 区块链技术: 架构及进展[J]. 计算机学报, 2018, 41(5): 969-988.
SHAO Q F, JIN C Q, ZHANG Z, et al. Blockchain: architecture and research progress[J]. Chinese Journal of Computers, 2018, 41(5): 969-988.

[5] CECCHETTI E, YAO S Q, NI H B, et al. Compositional security for reentrant applications[C]//Proceedings of 2021 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2021: 1249-1267.

[6] ALBERT E, CORREAS J, GORDILLO P, et al. GASOL: gas analysis and optimization for Ethereum smart contracts[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Cham: Springer, 2020: 118-125.

[7] ZHANG Y Y, MA S Q, LI J R, et al. SMARTSHIELD: automatic smart contract protection made easy[C]//Proceedings of 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). Piscataway: IEEE Press, 2020: 23-34.

[8] RODLER M, LI W T, KARAME G O, et al. Sereum: protecting existing smart contracts against re-entrancy attacks[C]//Proceedings of 2019 Network and Distributed System Security Symposium. Reston: Internet Society, 2019: 24-27.

[9] 倪远东, 张超, 殷婷婷. 智能合约安全漏洞研究综述[J]. 信息安全学报, 2020, 5(3): 78-99.
NI Y D, ZHANG C, YIN T T. A survey of smart contract vulnerability research[J]. Journal of Cyber Security, 2020, 5(3): 78-99.

[10] KALRA S, GOEL S, DHAWAN M, et al. ZEUS: analyzing safety of smart contracts[C]//Proceedings of 2018 Network and Distributed System Security Symposium. Reston: Internet Society, 2018: 1-15.

[11] GROCE A, FEIST J, GRIECO G, et al. What are the actual flaws in important smart contracts (and how can we find them)? [C]// International Conference on Financial Cryptography and Data Security. Cham: Springer, 2020: 634-653.

[12] 杨慧文, 崔展齐, 陈翔, 等. 基于软件度量的 Solidity 智能合约缺陷预测方法[J]. 软件学报, 2022, 33(5): 1587-1611.
YANG H W, CUI Z Q, CHEN X, et al. Defect prediction for solidity smart contracts based on software measurement[J]. Journal of Soft-

ware, 2022, 33(5): 1587-1611.
[13] 钱鹏, 刘振广, 何钦铭, 等. 智能合约安全漏洞检测技术研究综述[J]. 软件学报, 2021, 33(8): 3059-3085.
QIAN P, LIU Z G, HE Q M, et al. Smart contract vulnerability detection technique: a survey[J]. Journal of Software, 2021, 33(8): 3059-3085.
[14] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. SmartCheck: static analysis of Ethereum smart contracts[C]//Proceedings of 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Piscataway: IEEE Press, 2018: 9-16.
[15] 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. New York: ACM Press, 2018: 67-82.
[16] BADRUDDOJA S, DANTU R, HE Y Y, et al. Making smart contracts smarter[C]//Proceedings of 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Piscataway: IEEE Press, 2021: 1-3.
[17] FEIST J, GRIECO G, GROCE A. Slither: a static analysis framework for smart contracts[C]//Proceedings of 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Piscataway: IEEE Press, 2019: 8-15.
[18] ANAND S, BURKE E K, CHEN T Y, et al. An orchestrated survey of methodologies for automated software test case generation[J]. Journal of Systems and Software, 2013, 86(8): 1978-2001.
[19] CHEN T Y, KUO F C, LIU H, et al. Metamorphic testing: a review of challenges and opportunities[J]. ACM Computing Surveys, 2018, 51(1): 1-27.
[20] FENG Y, TORLAK E, BODIK R. Precise attack synthesis for smart contracts[J]. arXiv Preprint, arXiv: 1902.06067, 2019.
[21] GRIECO G, SONG W, CYGAN A, et al. Echidna: effective, usable, and fast fuzzing for smart contracts[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2020: 557-560.
[22] XIE X Y, JOSHUA W K H, CHEN. T. Y. Testing and validating machine learning classifiers by metamorphic testing[J]. Journal of Systems and Software, 2011, 84(4): 544-558.
[23] CHEN T Y, CHEUNG S C, YIU S M. Metamorphic testing: a new approach for generating next test cases[J]. arXiv Preprint, arXiv: 2002.12543, 2020.
[24] CHINEN Y, YANAI N, CRUZ J P, et al. RA: hunting for re-entrancy attacks in Ethereum smart contracts via static analysis[C]//Proceedings of 2020 IEEE International Conference on Blockchain (Blockchain). Piscataway: IEEE Press, 2020: 327-336.
[25] 惠战伟. 蜕变测试技术研究[D]. 南京: 解放军理工大学, 2015.
HUI Z W. Metamorphic testing techniques research[D]. Nanjing: PLA University of Science and Technology, 2015.
[26] CHEN T Y, YU Y T. On the relationship between partition and random testing[J]. IEEE Transactions on Software Engineering, 1994, 20(12): 977-980.
[27] REN M, YIN Z J, MA F C, et al. Empirical evaluation of smart contract testing: what is the best choice? [C]//Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2021: 566-579.
[28] SUN C A, FU A, POON P L, et al. METRIC+: a metamorphic relation identification technique based on input plus output domains[J]. IEEE Transactions on Software Engineering, 2021, 47(9):1764-1785.

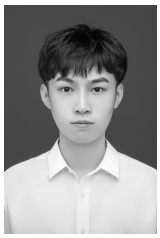
[作者简介]

陈锦富（1978− ），男，江西赣州人，博士，江苏大学教授、博士生导师，主要研究方向为软件测试、软件安全和可信软件。

王震鑫（1997− ），男，河南南阳人，江苏大学硕士生，主要研究方向为区块链系统安全、漏洞检测等。

蔡赛华（1990− ），男，江苏南通人，博士，江苏大学讲师、硕士生导师，主要研究方向为恶意流量检测、异常数据检测、软件安全测试。

冯乔伟（1998− ），男，江苏扬州人，江苏大学硕士生，主要研究方向为区块链漏洞检测、软件安全测试。



陈宇豪（1999− ），男，江苏无锡人，江苏大学硕士生，主要研究方向为区块链漏洞检测、软件安全测试。

许容天（1997− ），男，江苏无锡人，江苏大学硕士生，主要研究方向为区块链漏洞检测、软件安全测试。

Patrick Kwaku Kudjo（1981− ），男，博士，威斯康星国际大学学院副教授，主要研究方向为软件安全、漏洞分析和预测、智能软件安全和网络安全等。