计算机科学
COMPUTER SCIENCE

基于深度学习和信息反馈的智能合约模糊测试方法

赵明敏, 杨秋辉, 洪玫, 蔡创

引用本文

赵明敏, 杨秋辉, 洪玫, 蔡创. 基于深度学习和信息反馈的智能合约模糊测试方法[J]. 计算机科学, 2023, 50(9): 117-122.

ZHAO Mingmin, YANG Qiuhui, HONG Mei, CAI Chuang. Smart Contract Fuzzing Based on Deep Learning and Information Feedback [J]. Computer Science, 2023, 50(9): 117-122.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800104

基于深度学习和信息反馈的智能合约模糊测试方法

赵明敏　杨秋辉　洪　玫　蔡　创
四川大学计算机学院　成都 610065
(1807452597@qq.com)

摘　要　主流区块链平台以太坊上频繁发现由不安全编程引起的智能合约安全漏洞。为了提高模糊测试对合约代码的覆盖率,以更全面地检测安全漏洞,提出了一种智能合约模糊测试方法。首先构造智能合约交易序列数据集,再基于深度学习构建智能合约交易生成模型以生成模糊测试初始种子;然后根据覆盖率和分支距离信息,对智能合约进行信息反馈引导的模糊测试,提出了特定的测试用例染色体编码方式,并设计实现了相应的交叉和变异算子。所提方法能有效覆盖智能合约的深层次状态以及严格条件守卫的分支代码。在500个智能合约上进行实验,结果表明,所提方法的代码覆盖率为93.73%,漏洞检测率为93.93%,与ILF,sFuzz,Echidna方法相比,所提方法的代码覆盖率提高了3.80%～25.49%,漏洞检测率提高了4.64%～24.02%。所提方法有助于提升以太坊智能合约安全测试的有效性,具有参考价值。

关键词:以太坊智能合约;安全测试;深度学习;模糊测试;信息反馈引导
中图法分类号　TP311.5

Smart Contract Fuzzing Based on Deep Learning and Information Feedback

ZHAO Mingmin,YANG Qiuhui,HONG Mei and CAI Chuang
School of Computer Science,Sichuan University,Chengdu 610065,China

Abstract　Vulnerabilities of smart contracts caused by insecure programming have been frequently discovered on the mainstream blockchain platform Ethereum. In order to improve the coverage of contracts by fuzzing and detect security vulnerabilities more comprehensively,this paper proposes a smart contract fuzzing. First,constructing Ethereum smart contract transaction sequence data set,then building smart contract generation model based on deep learning to generate initial seeds for fuzzing. Then,according to the information of coverage and branch distance,conduct information feedback-guided fuzzing on smart contracts,a specific chromosome encoding method for test cases is proposed,and corresponding crossover operators and mutation operators are designed and implemented. The method can effectively cover the deep state of smart contracts and branch code guarded by strict conditions. Experiments on 500 smart contracts show that the code coverage rate of this method is 93.73%,and the vulnerability detection rate is 93.93%. Compare with the ILF,sFuzz,and Echidna methods,the code coverage rate of this method increases by 3.80%～25.49%,the vulnerability detection rate increases by 4.64%～24.02%. This method helps to improve the effectiveness of Ethereum smart contract security testing and is worthy of reference for the industry.

Keywords　Ethereum smart contracts,Security testing,Deep learning,Fuzzing,Information feedback guidance

1　引言

随着区块链2.0时代的到来,主流区块链平台以太坊上频繁发现由不安全编程引起的智能合约安全漏洞,这给平台用户造成了重大的财产损失[1]。保障以太坊智能合约安全性的最有效的手段是在智能合约被部署到以太坊区块链之前,对其进行充分的安全测试,找出安全漏洞并修复[2]。

模糊测试[3]是目前以太坊智能合约最为主流的动态安全测试技术,其通过模糊器生成大量测试用例执行合约,以检测出安全漏洞。目前的智能合约模糊测试工具主要以基于变异的灰盒模糊测试为主,通常使用轻量级的程序插桩技术,利用执行测试用例得到的合约覆盖信息,来引导测试用例的变异过程。但这些方法要么难以探索智能合约深层次的状态,要么难以覆盖具有严格条件守卫的合约分支代码,因此无法全面检测漏洞。

针对以上问题,本文提出了一种基于深度学习和信息反馈引导的智能合约模糊测试方法。首先使用符号执行方法,构造以太坊智能合约交易序列数据集,再使用神经网络模型,

到稿日期:2022-08-10　返修日期:2022-11-21
基金项目:四川省自然科学基金(23NSFSC3752);四川大学专职博士后研发基金(2022SCU12077)
This work was supported by the Natural Science Foundation of Sichuan Province,China(23NSFSC3752) and Sichuan University Postdoctoral Science Research Foundation(2022SCU12077).
通信作者:杨秋辉(yangqiuhui@scu.edu.cn)

在数据集上进行训练，构建智能合约交易生成模型，然后利用模型生成模糊测试初始种子，以提升合约覆盖率为目标对智能合约进行信息反馈引导的模糊测试，根据测试用例的执行结果，检测智能合约的安全漏洞。该方法能高度覆盖智能合约代码，并能有效检测智能合约中存在的安全漏洞，具有良好的应用效果。

2 相关工作

文献[4-7]提出了一系列智能合约静态分析测试方法，根据预定义的漏洞类型分析和检测合约中的漏洞或问题，然后通过逐一匹配现有漏洞来给出检测结果。但这类方法没有分析运行后的动态信息，致使遗漏潜在漏洞。文献[8-10]使用符号执行方法对智能合约进行逐路径推理，通过求解路径约束来获得可行输入，但是目前的技术依旧存在路径爆炸和复杂约束求解的问题[11]。因此，本文主要利用轻量级的静态分析技术来构造被测合约的控制流程图，从而获取覆盖率和分支距离信息。使用符号执行方法构造交易序列数据集，基于深度学习构建智能合约交易生成模型，在数据集上训练后的模型保留了符号执行方法探索合约状态空间的能力，同时其消耗的计算资源比符号执行方法更少。

在以太坊智能合约动态安全测试技术中，模糊测试是最为主流的方法。Jiang 等[12]提出的 ContractFuzzer 是第一个检测合约安全漏洞的黑盒模糊测试工具，但其随机生成测试用例，无法有效覆盖合约代码的深层次路径，因此可能会错过一些安全漏洞。He 等[13]提出的 ILF 是一种基于神经网络的灰盒模糊测试框架，根据被测合约的历史输入序列生成新的输入，执行并分析漏洞。Grieco 等[14-15]提出的 Echidna 是一个自动生成测试以检测断言和用户自定义属性中的违规行为的灰盒模糊工具。Nguyen 等[16]提出的 sFuzz 借鉴传统的模糊测试工具 AFL[17]并采用反馈自适应模糊策略来提高测试用例的覆盖率。但目前的灰盒模糊方法要么难以有效地变异交易序列来探索智能合约可能的状态[18]，要么难以覆盖具有严格条件守卫的智能合约分支代码。因此，本文提出了一种基于深度学习和信息反馈引导的模糊测试方法，使用能有效探索智能合约可能状态空间的测试用例作为初始种子，基于覆盖率和分支距离信息反馈引导模糊测试过程，以提升模糊测试对智能合约的代码覆盖率，并有效检测智能合约中的安全漏洞。

3 问题描述与基本思路

在智能合约测试中，一些安全漏洞需要在深层次状态下才能够被触发。图 1 给出了一个智能合约 DeepState。若要触发第 9 行的漏洞，则需要连续调用 setY(39)，copyY2X()和 Bar()这 3 种方法，或者需要在调用 Bar()之前调用 39 次 increaseX()，使合约状态 $x=39$ 满足条件以执行 assert 语句。探索合约深层次状态往往需要执行特定的交易序列。符号化执行方法能生成交易探索合约状态，但随着交易序列长度的增加，合约可能的状态数量呈指数增长，符号化执行方法的效率将大幅降低。

智能合约中具有严格条件守卫的分支代码在测试时很难被覆盖，图 2 给出了一个智能合约 HardToCover。第 4 行和第 7 行的分支代码均通过严格条件守卫，若要覆盖分支代码，则测试用例必须使用特定的参数调用 foo 方法。由于变量 x 的类型为 int256，如果按均匀分布随机生成测试输入，那么只有 $1/2^{256}$ 的概率生成特定参数。对于简单条件守卫的分支代码，使用符号化执行方法分析合约能有效得到能通过条件的测试输入，但在上述例子中，由于第 3 行代码中的非线性计算，符号化执行方法受到底层约束求解器的限制可能无法求解到合适的输入。

```
1. contract DeepState {
2.     int256 private x;
3.     int256 private y;
4.     constructor() public {
5.         x=0;
6.         y=0;}
7.     function Bar() public view returns(int256) {
8.         if(x==39) {
9.             assert(false);
10.            return 1;}
11.        return 0;}
12.        function setY(int256 newY) public {
13.        y=newY;}
14.    function increaseX() public {
15.        x++;}
16.    function copyY2X() public {
17.        x=y;}
18. }
```

图 1 具有深层次状态的 Solidity 智能合约示例

Fig. 1 Example of Solidity smart contracts with deep state

```
1. contract HardToCover {
2.         function foo(int x) {
3.         int y=x*x+20;
4.         if(y==120) {
5.                                 …
6.                     }
7.                 if(y==90020) {
8.                         …
9.                     }
10.    }
11. }
```

图 2 具有严格条件守卫分支代码的 Solidity 智能合约示例

Fig. 2 Example of Solidity smart contracts for branch code with strict condition guards

为解决上述问题，本文提出了一种智能合约模糊测试方法，整体思路如图 3 所示，针对智能合约深层次状态的有效覆盖问题，本文方法首先使用符号执行方法来构造能探索合约深层次状态的交易序列数据集，然后使用神经网络构建智能合约交易生成模型，训练后的模型保留了符号执行方法探索合约状态空间的能力，并以更低的时间复杂度生成交易。使用模型构造模糊测试初始种子，提高对合约深层次状态的覆盖。针对严格条件守卫的分支代码的有效覆盖问题，本文方法在信息反馈引导的模糊测试中引入分支距离，以提升覆盖率和减小测试用例与严格条件守卫的基本块的分支距离为引导方向，更新模糊测试种子，从而提高对智能合约严格条件

守卫的分支代码的覆盖。

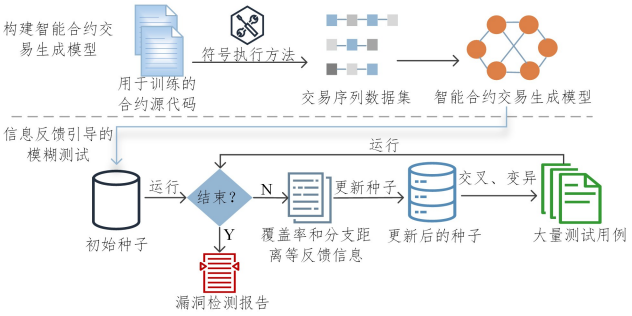

图 3 本文方法的整体思路

Fig. 3 Overall idea of the proposed method

4 基于深度学习构建智能合约交易生成模型

本文方法根据 Cadar 等[19]的研究，使用覆盖率优化搜索的启发式搜索策略的符号执行方法分析合约，构造出探索合约深层次状态的交易序列数据集。参考 He 等[13]的研究，用神经网络构建智能合约交易生成模型。

4.1 构造智能合约交易序列数据集

首先收集以太坊 Solidity 智能合约源代码，然后编译部署合约，本文在本地私有以太坊网络上成功部署了 14 609 条智能合约。然后通过轻量级的静态分析技术生成合约的控制流程图(Control Flow Graph，CFG)，并构造合约方法依赖图。在部署的每条智能合约上，应用 Cadar 等[19]提出的算法，使用覆盖率优化搜索的启发式搜索策略的符号执行方法分析合约，为每个合约生成一条能最大程度覆盖合约代码的最长的交易序列，代表对合约深层次状态的探索。

符号执行方法生成的交易序列无法直接用于模型训练，需要对其进行特征提取，构造历史交易特征矩阵。特征矩阵由合约各个公有方法的语义特征向量构成，语义特征可分为两类，一是方法本身的语义特征(包括方法参数、操作码、方法名称)，二是与历史交易相关的语义特征(包括方法的返回操作码、交易数占比、覆盖率)[13]。最终加入到训练数据集中的是特征矩阵与下一个交易的二元组序列，其中的下一个交易即是模型中有监督学习任务的标签。最后将数据集按照6∶2∶2 的比例划分为训练集、验证集、测试集。

4.2 构建智能合约交易生成模型

本文方法中的智能合约交易生成模型的神经网络结构如图 4 所示。

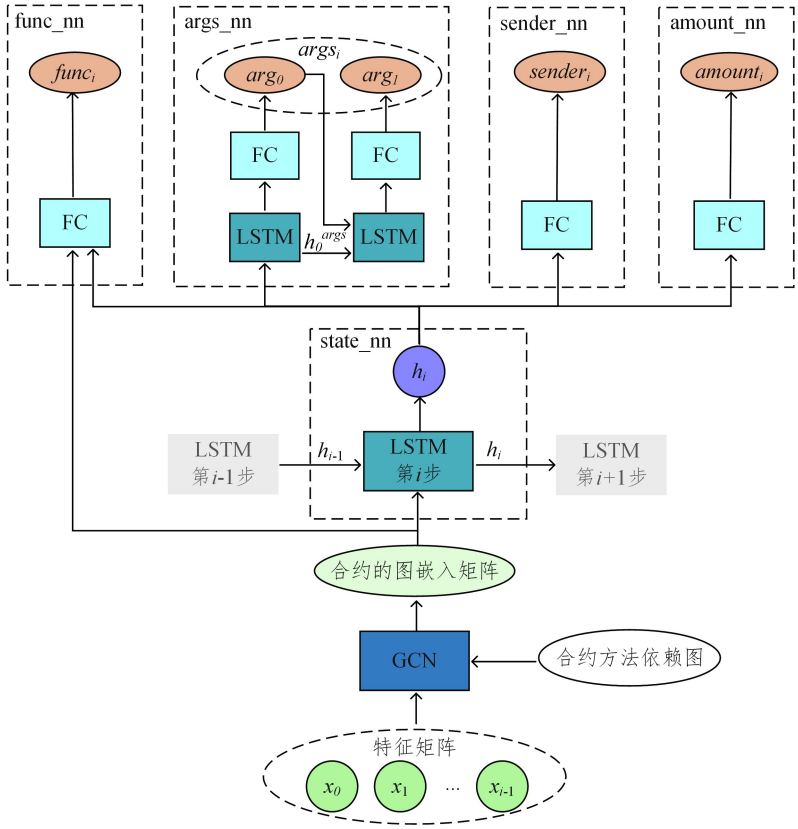

图 4 智能合约交易生成模型的神经网络结构

Fig. 4 Neural network architecture for smart contract transaction generation model

首先模型输入能表示交易序列信息的特征矩阵，然后图卷积神经网络(Graph Convolution Network，GCN)以特征矩阵与合约方法依赖图为输入，捕获合约公有方法间的依赖关系，输出合约的图嵌入矩阵，并将其作为 state_nn 和 func_nn 的输入。长短期记忆网络(Long Short-Term Memory，LSTM)是循环神经网络(Recurrent Neural Network，RNN)的变体，相比 RNN，LSTM 在处理长序列时有更好的表现[20]，本文方法在 ILF[13]模型结构的基础上，使用 LSTM 学习得到能表示合约历史交易序列信息的状态矩阵 $\boldsymbol{h}_i$，并将其作为后续 func_nn，args_nn，sender_nn 和 amount_nn 等分类

子模型的输入。这些分类子模型使用全连接层(Fully Connected layers,FC)分别输出合约方法 func、参数 *args*、发送者 sender 和以太币(ETH)数量 amount 的概率分布,根据概率分布选择元素,构造出能最大覆盖合约的交易。由于合约方法可能有多个参数,因此在 args_nn 分类子模型中也使用了 LSTM,为生成第 j 个参数的概率分布,LSTM 以第 $j-1$ 个参数的输出和上一个隐藏层状态为输入,输出新的隐藏层状态,然后通过 FC 生成在参数集合中的概率分布。

5 信息反馈引导的模糊测试

本文方法使用模型构造高质量的初始种子,在迭代中不断更新种子,且通过交叉、变异得到大量新的测试用例,根据执行结果检测漏洞。

5.1 构造模糊测试初始种子

首先对被测合约进行预处理,部署合约并生成合约 CFG 和合约方法依赖图,后续根据合约 CFG 获取合约代码覆盖信息和计算测试用例到未覆盖分支之间的距离。在使用智能合约交易生成模型生成第一个交易时,合约处于初始状态,交易序列为空,此时特征矩阵中只包含合约方法本身的语义特征,与历史交易相关的语义特征均为初始值。执行交易后,根据执行信息更新特征矩阵,然后模型根据更新后的特征矩阵生成下一个交易。使用模型迭代生成交易,直至生成的交易序列达到预定义长度。最后为被测合约构造若干交易序列(即测试用例),形成模糊测试的初始种子,这些种子相比随机构造的初始种子能探索合约深层次状态,并能有效覆盖合约代码。

5.2 基于覆盖率和分支距离选择测试用例

为了覆盖智能合约中严格条件守卫的分支代码,本文方法结合覆盖率适应度函数与分支距离法[21]适应度函数对测试用例进行选择。根据智能合约的特点,覆盖率适应度函数应用了 AFL[17]中的适应度函数,即能覆盖一个新的合约基本块的测试用例具有高适应度。这种策略能快速覆盖大多数基本块,但是该策略没有充分利用在测试过程中获得的合约分支信息,在测试后期难以发现新的基本块,尤其是具有严格条件守卫的合约分支中的基本块,分支距离法适应度函数可以解决该问题。

分支距离法在计算适应度时,判断测试用例产生的合约执行路径与目标分支的偏移大小。本文方法结合 Nguyen 等[16]的工作,为了加快适应度计算,只判断测试用例与其执行后刚好未被覆盖的基本块之间的分支距离。所谓刚好未被覆盖的基本块,指测试用例在执行后会覆盖 CFG 中的节点,被覆盖节点的出边连接的节点如果没有被覆盖,则被称为刚好未被覆盖的基本块。

在高质量的初始种子的基础上,基于执行后的测试用例的反馈信息选择合适的测试用例更新种子,进一步地提高了种子质量。

5.3 交叉和变异模糊测试种子

本文方法针对智能合约交易和方法参数的特点,提出了特定的测试用例染色体编码方式,将测试用例即交易序列视为染色体,染色体中的每个基因是若干交易,基因的顺序与交易的顺序相同。每个基因中包括合约方法的应用二进制接口(Application Binary Interface,ABI)规格说明以及实际的参数值列表。

本文方法针对测试用例染色体的交叉操作发生在序列级别,即对交易序列按预定概率进行交叉,使用 crossover_1point 和 crossover_2point 两种交叉算子,crossover_1point 随机选择一个位置,将两个交易序列分别分成两段,然后交换第二个片段生成两个新序列。crossover_2point 则随机选择两个位置,将两个交易序列分别分成 3 段,然后交换中间的片段生成两个新序列。

本文方法针对测试用例染色体的变异操作既发生在序列级别,又发生在基因级别。序列级别的变异算子包括增加交易、删除交易和交换交易 3 种。基因级别的变异是对测试用例中每个交易的方法参数和 ETH 金额进行变异,在对方法的参数进行变异时,先根据 ABI 规格说明中的参数类型将参数转为内存中实际的二进制表示,完成变异后,再将二进制表示根据参数类型解码为实际的参数。在对 ETH 金额进行变异前,首先根据方法的 ABI 规格说明判断该方法是否可以接收 ETH,如果可以,则将其转为二进制表示使用变异算子生成新值。实现的基因级别的变异算子包括 mutation_reverse_bytes 和 mutation_swap_bytes 两种,mutation_reverse_bytes 以指定的变异率翻转二进制表示中的每一个比特位,mutation_swap_bytes 以指定的变异率交换二进制表示中的两个比特位。

6 实验验证

本文根据所提方法使用 python 语言实现了原型工具 DLFFuzz(Using Deep Learning and Information Feedback Guidance to Fuzz),并选取来自以太坊上的真实的 Solidity 智能合约作为实验对象,通过实验探究本文方法对实际的智能合约的覆盖效果和对安全漏洞的检测效果。

实验时,首先训练智能合约交易生成模型,使用交叉熵损失函数计算损失,使用 Adam 梯度下降优化算法优化损失,采用等间距调整学习率的策略训练,然后在测试集上评估模型。模型在 func 分类、args 分类、sender 分类和 amount 分类上的准确率分别为 78.05%,86.64%,29.26%和 78.31%,与随机方法相比,分别提升了 71.79,84.45,9.47 和 76 个百分点,因此可使用模型构造模糊测试初始种子。

6.1 实验设计

实验对象为从以太坊收集的 500 条 Solidity 智能合约,合约的编译器版本为 0.4.x。按照合约编译后的 EVM 指令数将智能合约分为 148 个大规模合约(指令数超过 3000)和 352 个小规模合约(指令数小于 3000)。在 500 个实验对象中共有 6 类、346 个安全漏洞,其中包括 14 个锁定漏洞、25 个泄露漏洞、162 个自杀漏洞、130 个区块参数依赖漏洞、14 个未处理的异常漏洞和 1 个受到控制的代理调用漏洞。

实验使用的 CPU 为 Intel(R) Core(TM) i7-9700,内存为 16GB,GPU 为 GeForce GTX 1660 Ti,显存为 6GB,操作系统为 Ubuntu20.04 LTS。采用指令覆盖率和漏洞检测率进行结果评价,指标越高,代表其越能全面地找到安全漏洞。

指令覆盖率的计算式如式(1)所示，漏洞检测率的计算式如式(2)所示。

$$\text{指令覆盖率}=\frac{\text{被覆盖的指令数量}}{\text{指令总数量}}\times 100\% \tag{1}$$

$$\text{漏洞检测率}=\frac{\text{检测出的漏洞数量}}{\text{漏洞总数量}}\times 100\% \tag{2}$$

6.2 实验 1：本文方法的覆盖率和漏洞检测能力评估

使用 DLFFuzz 对各个实验对象进行模糊测试，设定测试时间为 2 min。

图 5 给出了 DLFFuzz 的代码覆盖结果，DLFFuzz 在 412 个合约上得到了 90%以上的指令覆盖率，在所有实验对象上平均得到了 93.73%的指令覆盖率。图 5 下方的直方图显示了执行初始种子中的测试用例得到的指令覆盖率，在所有实验对象上平均得到了 34.78%的初始指令覆盖率，DLFFuzz 在初始种子的基础上，迭代生成大量新的测试用例，得到图 5 中折线表示的最终覆盖率。

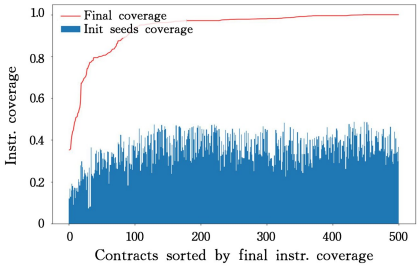

图 5 DLFFuzz 的代码覆盖结果

Fig. 5 Code coverage result of DLFFuzz

分析在不同规模合约上的覆盖率可以发现，在大规模和小规模合约上的平均指令覆盖率分别为 91.03%和 94.87%，这表明本文方法在不同规模的智能合约上均能得到高的指令覆盖率。

表 1 列出了 DLFFuzz 的漏洞检测结果，DLFFuzz 检测到了 500 个实验对象中的 93.93%的安全漏洞。本文方法在各类漏洞上均实现了 85%以上的漏洞检测率，并在自杀、泄露、锁定和受到控制的代理调用这 4 类漏洞上检测到了实验对象中的所有漏洞。

表 1 DLFFuzz 的漏洞检测结果

Table 1 Vulnerability detection results of DLFFuzz

漏洞类型	发现的漏洞数量	漏洞检测率/%
自杀	162	100.00
泄露	25	100.00
锁定	14	100.00
区块参数依赖	111	85.38
未处理的异常	12	85.71
受到控制的代理调用	1	100.00

综合以上分析，本文方法在对智能合约进行模糊测试时，能高度覆盖智能合约代码，并能有效检测合约中存在的安全漏洞，证明了本文方法的可行性。

6.3 实验 2：本文方法与 ILF，sFuzz，Echidna 的比较实验

通过在相同实验环境下分别比较不同工具的智能合约覆盖率和漏洞检测率，证明了本文方法的有效性。Echidna 需要手动为每个合约编写测试预言以检测安全漏洞，因此仅使用 Echidna 比较代码覆盖能力。sFuzz 使用的覆盖率标准是合约的分支覆盖率，无法与其他工具直接进行比较，因此在实验中仅使用 sFuzz 比较漏洞检测能力。

图 6 给出了 DLFFuzz，ILF 和 Echidna 的合约覆盖率结果。DLFFuzz 与 ILF，Echidna 相比，整体覆盖率分别提升了 3.80%，25.49%。图 6 中的结果表明，DLFFuzz 无论是在整体上，还是在不同规模的合约上，均比 ILF 和 Echidna 实现了更高的平均指令覆盖率。

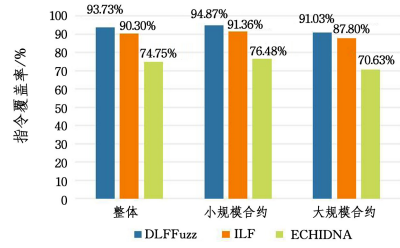

图 6 DLFFuzz，ILF 和 Echidna 的合约指令覆盖率

Fig. 6 Instruction coverage of contracts of DLFFuzz，ILF and Echidna

图 7 给出了 DLFFuzz，ILF 和 sFuzz 的漏洞检测结果。DLFFuzz 与 ILF，sFuzz 相比，漏洞平均检测率分别提升了 4.64%，24.02%。图 7 中的结果表明，DLFFuzz 发现了 ILF 所发现的所有漏洞，且检测到了更多的泄露和区块参数依赖两类漏洞。sFuzz 支持检测的漏洞类型与 DLFFuzz 重合的只有 3 类，但 DLFFuzz 比 sFuzz 检测到了更多的区块参数依赖和未处理的异常 2 类漏洞。

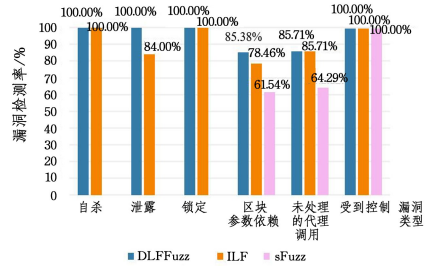

图 7 DLFFuzz，ILF 和 sFuzz 的漏洞检测结果

Fig. 7 Vulnerability detection results of DLFFuzz，ILF and sFuzz

综合以上分析，无论是对智能合约的代码覆盖能力，还是漏洞检测能力，DLFFuzz 相比其他 3 个工具均有更好的表现。

结束语 为了提高模糊测试对以太坊智能合约安全测试的有效性，本文提出了一种基于深度学习和信息反馈引导的模糊测试方法。通过实验验证，并与现有的较为先进和流行的模糊测试方法进行比较，发现本文方法有效提高了代码覆盖率，并能检测到更多的安全漏洞。

本文方法还存在一些不足，未来值得进一步研究，包括但不局限于：未来可以探索不同的变异算子对模糊测试检测效果的影响，并可以考虑在设计变异算子时利用测试用例执行结果的反馈信息，来进一步提高模糊测试的效率；未来还可进一步扩充本文方法的测试预言，以支持检测更多类型的智能合约安全漏洞。

参 考 文 献

[1] TIAN G H，HU Y H，CHEN X F. Research progress of Block

Chain System Attack and defense technology [J]. Journal of Software, 2021, 32(5): 1495-1525.
[2] HU T Y, LI Z C, LI B X, et al. Contract security and privacy security of smart contracts [J]. Chinese Journal of Computers, 2021, 44(12): 2485-2514.
[3] MILLER B P, FREDRIKSEN L, SO B. An Empirical Study of the Reliability of UNIX Utilities [J]. Communications of the ACM, 1990, 33(12): 32-44.
[4] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. SmartCheck: Static Analysis of Ethereum Smart Contracts [C]// Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. New York: Association for Computing Machinery, 2018: 9-16.
[5] LIU H, LIU C, ZHAO W, et al. S-Gram: Towards Semantic-Aware Security Auditing for Ethereum Smart Contracts [C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier: ACM, 2018: 814-819.
[6] GRECH N, KONG M, JURISEVIC A, et al. MadMax: Surviving out-of-Gas Conditions in Ethereum Smart Contracts [J]. Proceedings of the ACM on Programming Languages, 2018, 2(OOPSLA): 116: 1-116: 27.
[7] YE J, MA M, LIN Y, et al. Clairvoyance: Cross-Contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts [C]// Proceedings of the ACM/IEEE 42nd International Confe- rence on Software Engineering: Companion Proceedings. New York: Association for Computing Machinery, 2020: 274-275.
[8] 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. Vienna: ACM, 2016: 254-269.
[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. San Juan: ACM, 2018: 653-663.
[10] TORRES C F, SCHÜTTE J, STATE R. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts [C]// Proceedings of the 34th Annual Computer Security Applications Conference. New York: ACM, 2018: 664-676.
[11] HUANG S, DU J H, WANG X Y, et al. A survey of fuzzy testing technology for Ethereum Smart Contract [J]. Computer Science, 2022, 49(8): 294-305.
[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. Montpellier: ACM, 2018: 259-269.
[13] HE J, BALUNOVIĆ M, AMBROLADZE N, et al. Learning to Fuzz from Symbolic Execution with Application to Smart Contracts [C]// Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2019: 531-548.
[14] 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, 2020: 557-560.
[15] GROCE A, GRIECO G. Echidna-Parade: A Tool for Diverse Multicore Smart Contract Fuzzing [C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2021: 658-661.
[16] NGUYEN T D, PHAM L H, SUN J, et al. SFuzz: An Efficient Adaptive Fuzzer for Solidity Smart Contracts [C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. New York: ACM, 2020: 778-788.
[17] ZALEWSKI M. American Fuzzy Lop [EB/OL]. [2022-01-15]. https://lcamtuf. coredump. cx/afl/.
[18] WÜSTHOLZ V, CHRISTAKIS M. Harvey: A Greybox Fuzzer for Smart Contracts [C]// Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2020: 1398-1409.
[19] CADAR C, DUNBAR D, ENGLER D R, et al. Klee: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs [C]// Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. California: ACM, 2008: 209-224.
[20] ILYA S, ORIOL V, QUOC V L, et al. Sequence to sequence learning with neural networks [C]// Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14). Montreal: MIT Press, 2014: 3104-3112.
[21] BAO X A, XIONG Z J, ZHANG W, et al. A Path Test Case Generation Method Based on Improved Genetic Algorithm [J]. Computer Science, 2018, 45(8): 174-178, 190.

ZHAO Mingmin, born in 1999, postgraduate, is a student member of China Computer Federation. Her main research interests include software quality assurance and testing.

YANG Qiuhui, born in 1970, Ph.D, associate professor. Her main research interests include software automation testing, and software project management.

(责任编辑:喻藜)