软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(5):2325−2339 [doi: 10.13328/j.cnki.jos.006989]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于数据流传播路径学习的智能合约时间戳漏洞检测*

张 卓[1], 刘业鹏[2], 薛建新[3], 鄢 萌[4], 陈嘉弛[5], 毛晓光[2]

[1](广州商学院 信息技术与工程学院, 广东 广州 510700)

[2](国防科技大学 计算机学院, 湖南 长沙 410073)

[3](上海第二工业大学 计算机与信息工程学院, 上海 200127)

[4](重庆大学 大数据与软件学院, 重庆 401331)

[5](中山大学 软件工程学院, 广东 珠海 519082)

通信作者: 薛建新, E-mail: jxxue@sspu.edu.cn

摘 要: 智能合约是一种被大量部署在区块链上的去中心化的应用. 由于其具有经济属性, 智能合约漏洞会造成潜在的巨大经济和财产损失, 并破坏以太坊的稳定生态. 因此, 智能合约的漏洞检测具有十分重要的意义. 当前主流的智能合约漏洞检测方法 (诸如 Oyente 和 Securify) 采用基于人工设计的启发式算法, 在不同应用场景下的复用性较弱且耗时高, 准确率也不高. 为了提升漏洞检测效果, 针对智能合约的时间戳漏洞, 提出基于数据流传播路径学习的智能合约漏洞检测方法 Scruple. 所提方法首先获取时间戳漏洞的潜在的数据传播路径, 然后对其进行裁剪并利用融入图结构的预训练模型对传播路径进行学习, 最后对智能合约是否具有时间戳漏洞进行检测. 相比而言, Scruple 具有更强的漏洞捕捉能力和泛化能力, 传播路径学习的针对性强, 避免了对程序整体依赖图学习时造成的层次太深而无法聚焦漏洞的问题. 为了验证 Scruple 的有效性, 在真实智能合约的数据集上, 开展 Scruple 方法与 13 种主流智能合约漏洞检测方法的对比实验. 实验结果表明, Scruple 在检测时间戳漏洞上的准确率, 召回率和 $F1$ 值分别可以达到 0.96, 0.90 和 0.93, 与 13 种当前主流方法相比, 平均相对提升 59%, 46% 和 57%, 从而大幅提升时间戳漏洞的检测能力.

关键词: 智能合约; 时间戳漏洞; 漏洞检测; 数据流传播路径; 预训练

中图法分类号: TP311

中文引用格式: 张卓, 刘业鹏, 薛建新, 鄢萌, 陈嘉弛, 毛晓光. 基于数据流传播路径学习的智能合约时间戳漏洞检测. 软件学报, 2024, 35(5): 2325–2339. http://www.jos.org.cn/1000-9825/6989.htm

英文引用格式: Zhang Z, Liu YP, Xue JX, Yan M, Chen JC, Mao XG. Detection of Smart Contract Timestamp Vulnerability Based on Data-flow Path Learning. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2325–2339 (in Chinese). http://www.jos.org.cn/1000-9825/6989.htm

Detection of Smart Contract Timestamp Vulnerability Based on Data-flow Path Learning

ZHANG Zhuo[1], LIU Ye-Peng[2], XUE Jian-Xin[3], YAN Meng[4], CHEN Jia-Chi[5], MAO Xiao-Guang[2]

[1](School of Information Technology & Engineering, Guangzhou College of Commerce, Guangzhou 510700, China)

[2](College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

[3](School of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai 200127, China)

[4](School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China)

[5](School of Software Engineering, Sun Yat-sen University, Zhuhai 519082, China)

Abstract: The smart contract is a decentralized application widely deployed on the blockchain platform, e.g., Ethereum. Due to the economic attributes, the vulnerabilities in smart contracts can potentially cause huge financial losses and destroy the stable ecology of

* 基金项目: 国家重点研发计划 (2021YFB1714200); 中国博士后科学基金 (2023M732594)
收稿时间: 2022-07-03; 修改时间: 2023-02-13, 2023-04-06; 采用时间: 2023-06-27; jos 在线出版时间: 2023-11-08
CNKI 网络首发时间: 2023-11-10

Ethereum. Thus, it is crucial to detect the vulnerabilities in smart contracts before they are deployed to Ethereum. The existing smart contract vulnerability detection methods (e.g., Oyente and Secure) are mostly based on heuristic algorithms. The reusability of these methods is weak in different application scenarios. In addition, they are time-consuming and with low accuracy. In order to improve the effectiveness of vulnerability detection, this study proposes Scruple: a smart contract timestamp vulnerability detection approach based on learning data-flow path. It first obtains all possible propagation chains of timestamp vulnerabilities, then refines the propagation chains, uses a graph pre-training model to learn the relationship in the propagation chains, and finally detects whether a smart contract has timestamp vulnerabilities using the learned model. Compared with the existing detection methods, Scruple has a stronger vulnerability capture ability and generalization ability. Meanwhile, learning the propagation chain is not only well-directed but also can avoid an unnecessarily deep hierarchy of programs for the convergence of vulnerabilities. To verify the effectiveness of Scruple, this study uses real-world distinct smart contracts to compare Scruple with 13 state-of-the-art smart contract vulnerability detection methods. The experimental results show that Scruple can achieve 96% accuracy, 90% recall, and 93% *F*1-score in detecting timestamp vulnerabilities. In other words, the average improvement of Scruple over 13 methods using the three metrics is 59%, 46%, and 57% respectively. It means that Scruple has substantially improved in detecting timestamp vulnerabilities.

Key words: smart contract; timestamp vulnerability; vulnerability detection; data-flow path; pre-training

智能合约的概念最早是由 Szabo 在 20 世纪 90 年代提出, 它被定义为有效执行计算机交易协议的合同条款[1]. 然而, 由于当时缺乏可信的执行环境, 研究仅停留在理论层面. 随着 2009 年区块链技术在比特币上的首次应用[2], 智能合约也有了一个可供使用的稳定执行环境. 2013 年, Buterin[3]受比特币的启发提出了一个基于区块链的开源分布式计算平台, 即以太坊[4]. 在此基础上, 智能合约技术得到了了迅猛的发展, 已经进入到了以太坊等平台主导的新时代. 截止到 2022 年, 数以百万计的智能合约已在不同领域得到了广泛使用, 以太坊的加密货币市场资本已达到约 350 亿美元[5−7]. 区块链作为一种新技术, 自身不可避免地会出现许多漏洞. 这些漏洞给不法分子带来了可乘之机并造成了较大损失. 比较典型的因为漏洞导致的损失事件, 一是 2016 年 6 月的 DAO 漏洞事件造成了 6 000 万美元的损失, 二是整数溢出漏洞导致 BEC 活动中超过 9 亿美元瞬间蒸发[8].

智能合约与传统的应用程序不同, 它是无法更改的由代码控制的且自动运行的程序, 不受手动干扰. 此外, 智能合约需要用加密货币奖励矿工[9]. 目前, 以太坊上的代币拥有无法估算的经济价值. 智能合约部署在区块链上, 一旦执行, 几乎不可能对其进行修改. 因此, 智能合约在部署之前, 对其进行漏洞检测是必不可少的[10]. 到目前为止, 研究人员已经为智能合约开发了许多有效的漏洞检测工具, 例如: 基于程序静态分析的方法 (如: SmarkCheck[11]和 Slither[12]), 基于形式化的方法 (如: ZEUS[13]和 Securify[14]), 基于模糊测试的方法 (如: ContractFuzzer[15]) 和基于符号执行的方法 (如: Oyente[16], Osiris[17], Mythril[18]和 Manticore[19]). 这些智能合约漏洞检测方法主要基于专家知识. 这意味着在应用这些方法之前必须提前手动总结检测漏洞的规则和模式. 然而, 由于智能合约数量的爆炸式增长, 专业人员无法甚至几乎不可能总结出智能合约中存在的所有漏洞模式. 因此, 这些漏洞检测方法的应用场景受到一定限制的.

为了解决该局限性, 研究人员提出了许多新的方法, 利用大数据和机器学习技术来自动学习智能合约漏洞的特征. SmartEmbed[20]利用深度学习模型学习并计算与漏洞数据库中已有漏洞的相似度, 以此判断智能合约是否有漏洞. 该方法使用的漏洞数据库较为受限, 仅有 52 个已知的智能合约漏洞, 并且该方法在学习和计算时并没有考虑代码的结构信息和语义信息. 为融合语法和语义信息, Zhuang 等人[21]构建了一个关联关系图, 以表示智能合约的语法和语义特征, 并使用图神经网络进行学习和计算. 该方法用图表示程序的所有关系, 包含节点和边信息, 其图较为复杂, 导致其面对多种多样的智能合约泛化性不够. 在“学习”的方法中, 模型参数的初始化通常是随机的, 通过对大数据的学习训练不断调整参数, 从而达到一个较好的效果. 然而, 针对某些数据量不足的任务, 模型很难从中学习到有用的规律, 这样的现实情况使得“预训练”方法诞生. 因此, 本文试图使用预训练技术的潜力来为智能合约漏洞检测提供一个新的视角.

本文以时间戳漏洞作为研究对象, 提出智能合约时间戳漏洞检测方法 Scruple, 以此作为突破口改进当前漏洞检测方法. Scruple 方法针对时间戳漏洞构建出漏洞传播路径, 利用融入图结构的预训练模型对数据流传播路径进行学习, 以此检测智能合约的时间戳漏洞. Scruple 是基于学习的方法, 不需要人工定义漏洞模板, 有较强的泛化性.

Scruple 方法有两个重要特点. 一是构建了漏洞数据流传播路径. Scruple 方法与 Zhuang 等人[21]的全程序关联关系依赖图方法不同, Scruple 关注于漏洞的传播关系, 它既能获取与时间戳漏洞有关的充分信息, 又能获取复杂度和体积大幅减小的图, 拥有更好的泛化性. 二是采用预训练图模型学习漏洞数据流传播路径. 预训练模型在自然语言处理中取得很好效果, 在软件工程应用中也有很好效果[22−24]. 受此启发, 本文利用基于图学习的预训练模型, 对智能合约进行时间戳漏洞检测. 本文在智能合约漏洞大型数据集 (即 SmartBugs Wild Dataset[25]) 上, 对真实智能合约开展了时间戳漏洞检测的对比实验. 实验结果表明, Scruple 的时间戳漏洞检测准确率, 召回率和 $F1$ 值分别可以达到 0.96, 0.90 和 0.93, 与 13 种当前主流方法相比平均相对提升 57%, 45% 和 55%, 显著提升了漏洞检测结果.

本文第 1 节介绍背景知识. 第 2 节介绍基于数据流传播路径学习的智能合约漏洞检测方法 Scruple. 第 3 节介绍实验. 第 4 节介绍相关工作. 第 5 节进行总结.

1 相关背景

本节将介绍智能合约时间戳漏洞, 图预训练模型, 数据流传播路径和动机示例.

(1) 智能合约时间戳漏洞

智能合约作为一种由高级语言编写, 之后被编译为字节码并运行在区块链上的去中心化的应用, 不可避免地会遭受各种与运行环境密切关联的安全威胁[26]. 以太坊以区块链作为其基本支持技术. 它通过以太坊虚拟机 (EVM) 来支持智能合约的执行和调用[27]. 许多工作对智能合约漏洞类型进行了系统研究, 其中比较典型的有 Zhang 等人[28]的工作. 他们根据 IEEE 软件异常标准分类, 将从多个来源收集的智能合约漏洞划分为 9 类, 即数据类型漏洞, 描述类型漏洞, 环境类型漏洞, 交互类型漏洞, 逻辑类型漏洞, 性能类型漏洞, 安全类型漏洞和标准类型漏洞. 此外, 倪远东等人[29]根据智能合约的运行机制, 将智能合约漏洞进行梳理并将其自顶向下分为 3 大类: 高级语言层面, 虚拟机层面和区块链层面. 不同的工作对智能合约的漏洞分类有所不同, 这些漏洞的发生涉及诸如控制不合规, 编程语言本身的特性, 编程使用变量或关键字不合理等. 这些漏洞除了会影响正常的功能之外, 还会对金融层面产生巨大的危害. 时间戳依赖漏洞为由区块链层面引入, 主要与区块链本身的特性有关的一种较为严重的漏洞[30,29]. 时间戳依赖引入的漏洞一旦被矿工恶意利用, 可能会在金融层面造成非常严重的后果. 具体来说, 时间戳依赖指的是在智能合约代码中使用严格的区块时间戳来进行控制行为的决策. 遇到代码中存在时间戳依赖时, 矿工可以在时间戳规定的取值时间范围内构造恶意时间戳来有意图的绕过设计的限制, 从而进行一些恶意操作并造成严重的后果[30]. 图 1 为一个有时间戳依赖漏洞的智能合约例子. 在第 8 行, 时间戳 block.timestamp 赋给了变量 number; 在 28 行, 变量 winNum 依赖于时间戳 (blockhash 和 number), 29 行 winNum 作为了判断条件, 这时矿工可以提前计算出对自己有利的时间戳, 并在挖矿时将时间戳设置对自己有利的时间, 以延迟或提前用户的自毁操作. 如果矿工加速用户自毁, 用户持有的所有加密货币将被冻结, 从而造成金钱的损失.

(2) 图预训练模型

预训练首先应用在自然语言处理任务 (natural language processing, NLP) 中, 指的是在大型无监督文本语料库上进行预训练, 然后在下游任务上对模型参数进行微调. 诸如 ELMo[31], GPT[32]和 BERT[33]等预训练模型在许多自然语言处理任务中取得了理想的效果, 预训练技术已然成为当前深度学习技术处理自然语言任务的主流技术. 在软件工程领域中, NLP 中预训练模型的成功也促进了编程语言预训练模型的发展. 现有的许多研究工作[34−38]将程序源代码视为一系列单词 (token) 的集合, 并在大型代码库上进行预训练, 之后利用训练好的预训练模型在子任务上进行参数微调. 代码预训练在代码搜索[34], 代码补全[35], 代码摘要[36]和代码生成[37]等一系列任务中取得了很好效果. 它们的不足之处在于代码并不是简单的单词序列, 而是包含诸如循环, 跳转, 控制, 依赖等各种类似图结构的数据[38]. 因此, 如果能将代码的图结构融入到预训练技术中将更符合代码的特性. GraphCodeBERT[39]的应运而生则弥补了之前工作的不足. 它是图预训练模型, 即融入代码图结构的预训练编程语言模型, 采用的是代码的数据流图 (data flow graph, DFG). 数据流图是程序分析中经常使用的结构[39−41], 图中节点表示程序变量而边表示变量之间的依赖关系. 与常用的抽象语法树 (abstract syntax tree, AST) 语法级结构图相比, 数据流图减少了更

多的冗余信息, 泛化性更强并更容易训练. 同时, 对于相同功能的程序源代码, 在不同的抽象语法下 AST 是不同的, 而数据流图是相同的. 因此, 数据流图这种代码结构为代码理解提供了更为关键的代码语义信息. 本文中使用的传播路径是在数据流图的基础上, 根据时间戳依赖漏洞的关键信息找出与其有直接或间接的数据依赖关系的一种图结构信息. 数据流传播路径包含了可能触发漏洞的关键信息, 对其进行学习突出了漏洞在代码中的传播特性, 指向性更强, 训练效率更高.

```
1 contract Lottery
2 {
3       mapping (address => uint) usersBet;
4       mapping (uint => address) users;
5       uint nbUsers = 0;
6       uint totalBets = 0;
7       address owner;
8       number=block.timestamp;
9       function Lottery()
10      {
11          owner = msg.sender;
12      }
13      function Bet() public payable
14      {
15          if(msg.value > 0){
16              if(usersBet[msg.sender] == 0){
17                  users[nbUsers] = msg.sender;
18                  nbUsers += 1;}
19              usersBet[msg.sender] += msg.value;
20              totalBets = totalBets + msg.value;
21          }
22      }
23      function EndLottery() public
24      {
25          if(msg.sender == owner)
26          {
27              uint sum = 0;
28              uint winNum =uint(block.blockhash(number-1)%totalBets+1);
29              for(uint i = 0; i < nbUsers; i++)
30              {
31                sum +=usersBet[users[i]];
32                if(sum >= winNum){
33                      selfdestruct(users[i]);
34                  return;}
35              }
36          }
37      }
38}
```

图 1　一个有时间戳依赖漏洞的智能合约例子

(3) 数据流传播路径

数据流传播路径是指数据在程序内部的流向和变换过程. 数据流传播路经可以通过程序数据依赖关系进行构建, 这些关系可以为程序提供关键的代码语义信息并广泛用于程序分析和理解[39−42]. 对于智能合约来说, 存在一个或多个与导致智能合约漏洞的并和漏洞代码片段相关的数据流传播路经, 而这些数据流传播路经可以通过深度学习模型进行学习. 数据流关系在同一功能源代码的不同抽象语法下是相同的, 从而更易于深度学习模型理解和学习. 在数据流图中, 节点代表合约中的变量, 边代表的是变量之间的数据流传播关系. 例如, 在提取 $c=a+b$ 的数据流图时, Scruple 不需要关心变量 a、b 和 c 的含义, 只需要知道 c 的值来自于 a 和 b. 由数据流图构造的代码传播路径可以为深度学习模型理解和学习智能合约漏洞提供更好的泛化能力.

(4) 动机示例

本节使用一个真实的智能合约时间戳漏洞代码到漏洞传播路径的示例来对研究动机进行说明. 在图 2 中, 最左侧为一个带有时间戳漏洞的智能合约源代码. 正如上文提到的, 该代码存在时间戳漏洞的根源为: 从第 8 行时间戳赋值给 number, 而在第 32 行将对 number 有依赖的 winNum 作为判断条件, 这时矿工可以提前计算出对自己有

利的时间戳, 并在挖矿时将时间戳设置为对自己有利的时间, 以延迟或提前用户的自毁操作. 在源代码的基础上, Scruple 使用 tree-sitter[43]将之转化为语法树. 之后根据语法树可以得到带有标号的源代码变量序列. 最后可以根据变量序列和数据流关系构建数据流传播路径. 如图 2 所示, propagation paths (PP) 为源代码的数据流传播路径集合. 该集合一共有 47 个节点和 43 条边, 其中每个节点表示一个带有标号的变量, 红色的虚线边表示变量的值从哪里来, 绿色实线边表示变量值由谁计算得到. 从该传播路径集合中可以看出, 导致时间戳依赖漏洞产生的漏洞传播路径为 timestamp[29]→number[27]→number[125]→winNum[117]→winNum(if-selfdestruct)[162]. 因此, 可以通过使用基于图学习的预训练模型对传播路径集合进行学习, 从而可以捕获智能合约的时间戳漏洞关键传播特征同时避免过分关注无关的程序关联关系.

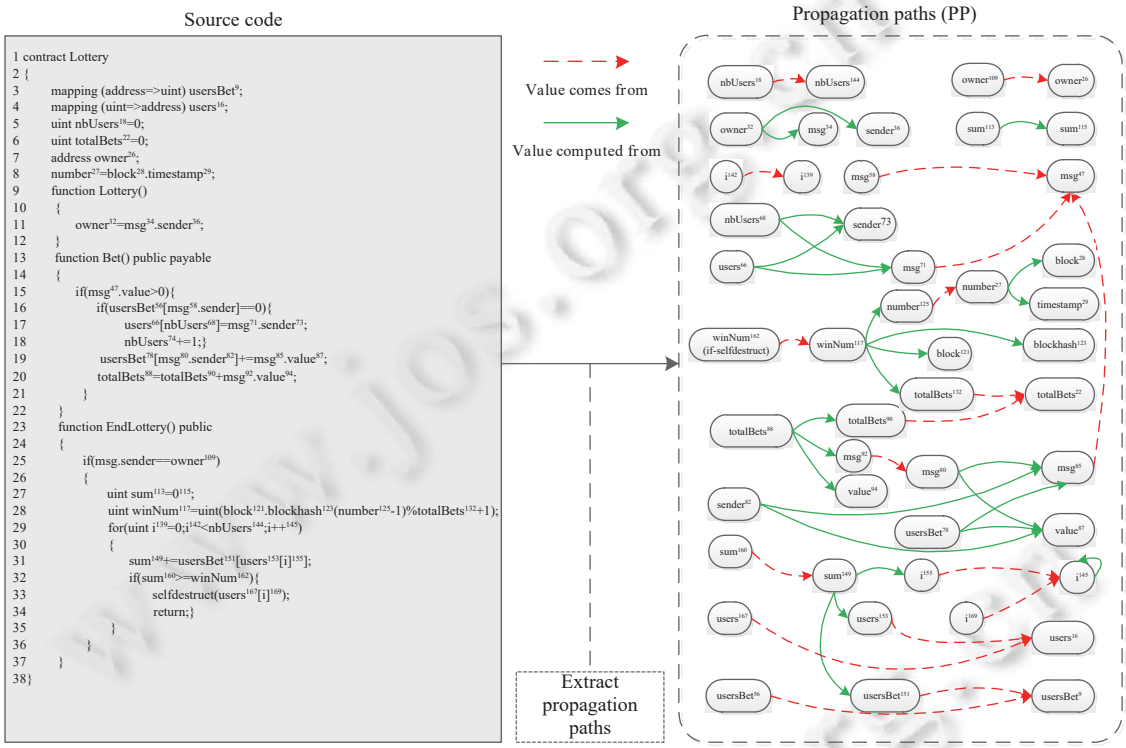


图 2 从给定源代码中提取传播路径

2 本文方法

本文方法 Scruple 并不进行预训练任务, 预训练过程由 GraphCodeBERT[39]完成. Scruple 模型以 GraphCodeBERT 为基础, 载入其预训练参数. GraphCodeBERT 在进行预训练时使用了 2.3M 个包含 6 种程序语言的函数. 其预训练时包括 3 个任务, 分别是对程序语言进行建模, 对图结构的程序数据 (数据流图) 进行边的预测和跨程序源代码和图结构的程序数据的变量对齐. 更详细的预训练设计过程可以参考研究[39]. Scruple 需要载入 GraphCodeBERT 的预训练参数, 之后在智能合约数据集上进行微调. 工作流程有两个阶段. 一是智能合约数据流传播路径生成阶段. 该阶段将源代码转换为抽象语法树 AST, 在 AST 基础上提取变量序列以生成数据流传播路径; 二是时间戳漏洞检测阶段. 该阶段采用基于图学习的预训练模型来检测智能合约时间戳漏洞. 接下来将详细介绍这两个阶段.

(1) 数据流传播路径生成

当前研究[40]表明, 程序可以转换为图形表示, 从而得以保留程序元素之间的语义关系. Zhuang 等人[21]将智能合约定制为合约图, 图中节点表示程序中的元素, 边表示元素之间的关联关系, 他们根据图中节点的重要性将边分

为 3 类. 虽然该方法利用了图的信息, 但是过于复杂的信息结构并不利于模型的泛化[39]. 因此, 本文采用更为简洁的数据流传播路径来表示图信息, 有助于模型更精准地对时间戳漏洞进行学习. 具体过程为, 首先将智能合约源代码解析为 AST, 然后从 AST 中提取变量序列以及它之间的数据流关系, 最后根据提取出的信息生成合约数据流传播路径. 图 2 展示了从给定源代码提取数据流传播路径的过程.

● 源代码解析为 AST 并确定变量序列. 假设有一个源代码 $C=\{c_1, c_2,\ldots, c_n\}$, Scruple 使用 tree-sitter 将其解析为 AST[43]. 由于 tree-sitter 不支持 Solidity 语言, 因此根据 JoranHonig 语法规则[44]对 tree-sitter 进行改进, 从而使其可以对 Solidity 语言进行解析. AST 包含了源代码的语法信息, 其叶子节点可以被用来确定变量序列. 变量序列可以表示为 $V=\{v_1, v_2,\ldots, v_k\}$. 模型根据变量在代码中出现的顺序按照号码由小到大对变量依次编号. 例如图 2 的 identify variable sequence 中, 第 3 行的变量 usersBet 在变量序列中编号为 9, 而第 16 行的 usersBet 在变量序列中编号为 56. 他们拥有着相同的变量名而在变量序列中拥有不同的变量编号, 代表着不同的变量.

● 生成数据流传播路径. 利用上一步生成的变量序列确定数据流传播路径的节点, 即变量序列中的每一个变量为传播路径中的一个节点, 而从节点 v_i 到节点 v_j 之间的边 $\varepsilon=<v_i, v_j>$ 表示变量序列中的 v_j 来自于 v_i 或者是由 v_i 计算得到. 在图 2 的 identify variable sequece 中, v_j 来自于 v_i 用红色虚线表示 (value comes from), v_j 由 v_i 计算得到用绿色实线表示 (value computed from). 以第 8 行的表达式 number^{27} = block^{28}.timestamp^{29} 为例, number^{27}, block^{28}, timestamp^{29} 均作为节点加入到传播路径中, 而 block^{28}→number^{27} 和 timestamp^{29}→number^{27} 作为 value computed from 类型的边加入传播路径. 将传播路径中的边记为 $E=\{e_1, e_2,\ldots, e_l\}$, 图 $CH(C)=(V, E)$ 表示源代码 C 的传播路径, 其中节点为 V, 边为 E.

(2) 时间戳漏洞检测

本节将详细介绍 Scruple 方法如何基于图的预训练模型来进行智能合约时间戳漏洞检测, 包括数据准备, 模型架构及训练过程.

首先是数据准备. Scruple 的输入数据包括 6 个部分 (见图 3), 分别为源代码的 token 集合, 变量集合, 源代码 token 的位置集合, 变量位置集合, 智能合约数据流传播路径和关键信息. 前 4 个输入单元为源代码的表示. 假设有源代码 C 的 token 集合 $CT=\{ct_1, ct_2,\ldots, ct_m\}$, 其变量集合为 $V=\{v_1, v_2,\ldots, v_k\}$. 将源代码 C 的 token 序列和变量集合 V 连接成一个序列 $I_1=\{[\text{CLS}], CT, [\text{SEP}], V\}$, 其中 [CLS] 是两个集合前面的特殊标记, [SEP] 是分割源代码 token 集合 CT 和变量集合 V 的特殊符号. 将源代码 token 的位置集合和变量位置集合链接成另一个序列 $I_2=\{[\text{CLS}], CT\_P, [\text{SEP}], V\_P\}$, CT_P 为 token 的位置集合, V_P 为变量位置集合. 将这两个序列 I_1 和 I_2 转换为输入向量 X^0 作为源代码 C 的表示向量, X^0 包括两个嵌入表示, 分别为 token 序列的表示和位置序列的表示. 第 5 个输入单元为数据流传播路径, 表示为 $CH(C)=(V, E)$, 其中 V 为变量集合$\{v_1, v_2,\ldots, v_k\}$, E 为边集合 $\{e_1, e_2,\ldots, e_l\}$, 表示变量的值从哪里来 (value comes from) 或者由谁计算而来 (value computed from). 第 6 个输入单元为从源代码 C 中获取的时间戳漏洞关键信息集合 $V'=\{v'_1, v'_2,\ldots, v'_n\}$, 其中 v'_i 表示和时间戳漏洞关键信息 (block.timestamp, block.number, now 等) 同一行的变量. 输入层根据第 5 个输入单元和第 6 个输入单元对智能合约传播路径进行精化, 具体来说, 在传播路径 $CH(C)$ 中根据 V'中的 v'_i $(i\in\{1, 2,\ldots, n\})$ 找到与之相对应的节点, 对与 v'_i 没有直接或间接关联关系的边进行裁剪, 最后留下了一个新的边集合 $E'=\{e'_1, e'_2,\ldots, e'_h\}$. 之后对孤立节点 (传播路径 $CH(C)$ 中没有 E'中的边经过的节点) 进行裁剪, 从而得到一个精简的和时间戳漏洞相关的漏洞传播路径 $CH_1(C)$.

其次是模型架构和训练过程. 如图 3 所示, Scruple 模型包括 6 大部分, 分别为输入层, 连接层, 多头注意力层, layer normalization 层, 若干 Transformer 层和线性层. 公式 (1) 和公式 (2) 为模型的训练过程.

$$H^n = LN(MHSA(X^{n-1}) + X^{n-1}) \tag{1}$$

$$X^n = LN(FFN(H^n) + H^n) \tag{2}$$

在公式 (1) 中, H 和 X 为向量, $MHSA$ 表示一个多头自注意力操作, LN 表示 layer normalization 操作; 在公式 (2) 中, FFN 表示一个双层的前向网络. Scruple 在输入层将数据准备阶段的 6 个输入单元输入到连接层. 在连接层 (join layer), 将数据准备阶段的 I_1, I_2 和 $CH_1(C)$ 转换为输入向量 X^0, 输入向量 X^0 将通过多头注意力层 (masked

multi-head attention layer), layer normalization 层, 若干 Transformer 层 (n=12), 以生成特定的上下文表示. X^n=$Transformer_n(X^{n-1})$, $n \in [1, 12]$. 其中, 如公式 (1) 所示, 每个 Transformer 层的向量 X^{n-1} 在经过一个多头注意力操作后会生成一个向量 H^n[45], 之后按照公式 (2) 的计算输出向量 X^n. 对于第 n 个 Transformer 层来说, 输出为 X^n, $MHSA(X^{n-1})$ 的计算过程如公式 (3)–公式 (6) 所示.

$$Q_i = X^{n-1}W_i^Q,\ K_i = X^{n-1}W_i^K,\ V_i = X^{n-1}W_i^v \tag{3}$$

$$head_i = Softmax\left(\frac{Q_i K_i^{\mathrm{T}}}{\sqrt{d_k}} + M\right)V_i \tag{4}$$

$$M_{ij} = \begin{cases} 0, & \text{if } q_i \in \{[\mathrm{CLS}], [\mathrm{SEP}]\} \\ & \text{or } q_i, k_j \in CT \\ & \text{or } < q_i, k_j > \in E_1 \cup E_1' \\ -\infty, & \text{otherwise} \end{cases} \tag{5}$$

$$X^n = [head_1; \ldots, head_m]W_n^Q \tag{6}$$

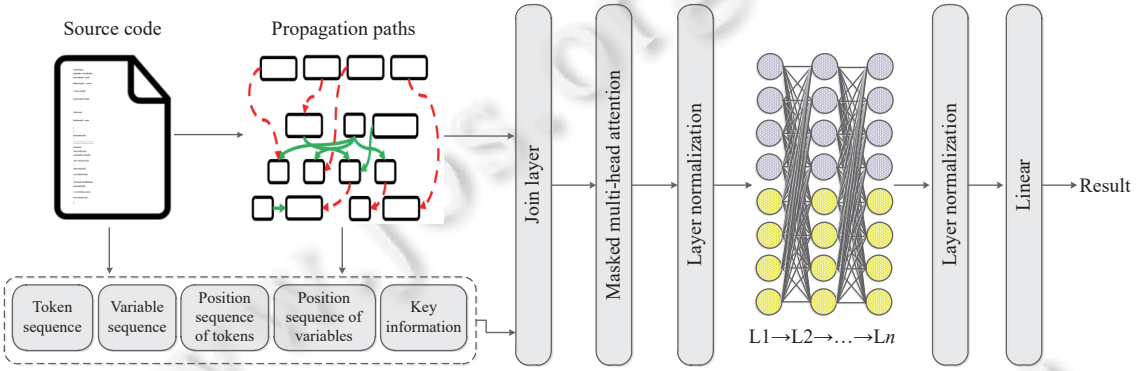


图 3 Scruple 总体架构

在公式 (3) 中, Q, K 和 V 为三元组, X, W 为向量, $X^{n-1} \in \mathbb{R}^{|I| \times d_h}$ 为 $n-1$ 个 Transformer 层的输出, 它被线性投影到一个分别由 Q_i, K_i 和 V_i 组成的三元组上, Q_i, K_i 和 V_i 分别由 W_i^Q, W_i^K, $W_i^V \in \mathbb{R}^{d_h \times d_k}$ 计算. 为了能够使 Transformer 层可以学习图结构, 使用 graph-guided masked attention 来建立智能合约中 token 间的关联关系[39]. 使用 mask 矩阵 M 来实现 graph-guided masked attention. 公式 (4) 中, $head$ 为多头注意力中的头, $Softmax$ 为函数, d_k 为头 (head) 的维度, M 为一个 mask 矩阵, $M \in \mathbb{R}^{|I| \times |I|}$, 其中如果第 i 个 token 和第 j 个 token 有关联关系则 M_{ij}=0, 否则为$-\infty$. M 的计算过程如公式 (5) 所示, [CLS] 是集合前面的特殊标记, [SEP] 是分割符, CT ={ct_1, ct_2,…, ct_m}为源代码 C 中 token 的集合, 对于数据流传播路径集合 $CH_1(C)$, E_1 为边集合 {e_1, e_2,…, e_l}, 表示变量的值从哪里来 (value comes from) 或者由谁计算而来 (value computed from), E_1' 为表示智能合约 token 与数据流传播路径中变量有关联关系的集合, $<v_i, ct_j><ct_j, v_i> \in E_1'$ 当且仅当变量 v_i 由源代码 token 中的 ct_j 确定且节点 v_i 的 query 与 c_j 的 node-key 有关联关系. 当节点 v_i 和节点 v_j 存在一条有向边 ($<v_i, v_j> \in E_1$) 或者为相同的节点 (i=j), 那么 query q_{v_j} 与 node-key k_{v_i} 有关联关系. 否则, 注意力 (attention) 值将会被赋予 $-\infty$, 经过公式 (4) 的 $Softmax$ 计算后被赋予 0. 在公式 (6) 中, m 为多头注意力中头 (head) 的数量. $W_n^O \in R^{d_h \times d_h}$ 为模型参数.

在第 n 层 Transformer 层后, 使用 layer normalization 层进行正则化, 之后使用线性层 (linear layer) 和 $Sigmoid$ 函数[46]来输出合约包含漏洞的可能性 y (如公式 (7) 所示). 之后构造损失函数 (loss) 计算输出值 y 与目标值 (智能合约包含时间戳漏洞则目标值为 1, 反之为 0) 的差值, 最后运用反向传播算法来训练网络.

$$y = Sigmoid(X^n) \tag{7}$$

整个时间戳漏洞检测任务的目标是通过对模型进行学习, 找出智能合约中存在的潜在时间戳漏洞. 在学习过

程中, Scruple 模型会将大批量的智能合约源代码和它们的传播路径, 以及相对应的标签作为输入. 微调任务的目标是识别智能合约代码中潜在的时间戳依赖漏洞. 微调任务在训练时, 经历过预训练的模型被输入了源代码和数据流传播路径信息, 以及数据集中标注的漏洞标签. 然后, 训练好的模型被输入智能合约源代码后, Scruple 可以给出合约是否包含时间戳漏洞的判断.

3 实 验

为了验证 Scruple 的有效性, 本文实验回答以下研究问题.

RQ1: Scruple 检测智能合约时间戳漏洞的有效性如何? 该问题回答的是与最先进的方法相比, Scruple 在智能合约时间戳漏洞检测方面的表现怎么样. 本文采用 Precision (准确率), Recall (召回率) 和 *F*1-score (*F*1 值) 这些指标对比来回答该问题.

RQ2: Scruple 不同模块对漏洞检测的贡献如何? 该问题回答的是 Scruple 不同的模块对模型的贡献度, 包括数据流传播路径和预训练模型. 本文设计了消融实验来回答该问题.

(1) 实验设置

● 数据集. 为了验证 Scruple 的有效性, 本文选取广泛使用的基于 Solidity 语言的大规模智能合约数据集 SmartBugs Wild Dataset[25]作为实验数据集. 该数据集包含了 47 398 个真实的智能合约源代码文件[47]. 每个文件包含多个智能合约. 总合约数为 20 万个, 其中不存在时间戳漏洞的合约为 18.9 万, 剩下的为存在时间戳漏洞的合约, 二者比例为 17:1.

● 对比方法. 对于 RQ1, 本文将 Scruple 与 13 种最先进的智能合约漏洞检测方法进行了对比. 其中包括 5 种传统的基于专家知识的智能合约漏洞检测方法 (Manticore[19], Osiris[17], Oyente[16], Slither[12]和 SmartCheck[11]) 和 7 种基于深度学习的智能合约漏洞检测方法 (GCN[47], Vanilla-RNN[47], LSTM[47], GRU[47], DR-GCN[21], TMP[21]和 CGE[47]). 对于 RQ2, 首先将预训练模型的参数进行重置并规范化, 在输入不变的情况下对其重新进行训练并测试, 以研究预训练模型的贡献. 其次将输入中的数据流传播路径去掉, 保持其他输入不变的情况下进行训练并测试, 以研究数据流传播路径对模型的贡献.

● 实验环境. 实验环境为一台包含英特尔 i7-9700 CPU 和 64 GB 物理内存的计算机, 包含一个 12 GB 的 NVIDIA TITAN X Pascal 的 GPU, 操作系统为 Ubuntu 18.04, 数据统计在 Matlab R2016b 上进行.

● 参数设置. Scruple 使用的是 adam 优化器并试图寻找最优的超参数设置: 学习率 (learning rate) 为 2E–5, 训练集的批次 (batch size) 为 2, 验证集的批次 (batch size) 为 32, 梯度累加步数 (gradient accumulation step) 为 1, adam epsilon 为 1E–8. 对于原始数据集的分割, 实验参照文献 [21] 随机选择 20% 作为训练集, 10% 作为验证集, 剩余 70% 作为测试集.

(2) 评价指标

本文采用智能合约漏洞检测领域广泛使用的评价指标, 即 Precision, Recall 和 *F*1 值[11,21]. 具体来说, Precision = true positive/(true positive + false positive), Recall = true positive/(true positive + false negative), *F*1 值 = Precision×Recall/2×(Precision+Recall). 在计算指标 Precision, Recall 和 *F*1 值时, 本文选择 macro 方式进行, 即对有漏洞的智能合约和无漏洞的智能合约的指标分别计算再求平均. 这种计算方式可以反映 Scruple 的总体性能.

(3) 实验结果

● RQ1: Scruple 检测智能合约时间戳漏洞的有效性如何?

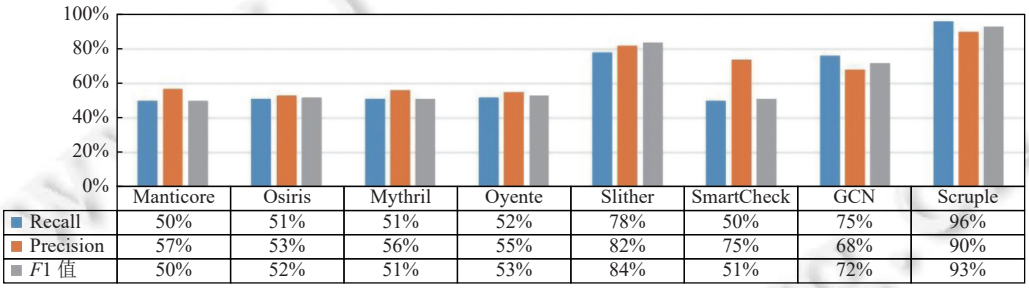
实验将 Scruple 与 13 种最先进的方法进行了比较, 它们分别 Manticore[19], Osiris[17], Mythril[18], Oyente[16], Slither[12], SmartCheck[11], GCN[47], Vanilla-RNN[47], LSTM[47], GRU[47], DR-GCN[21], TMP[21]和 CGE[47]. 表 1 给出了 Scruple 及 13 种漏洞检测方法在测试集上的 Recall, Precision 和 *F*1 值. 为了便于更直观地查看各种方法的比较结果, 图 4 给出了它们 Recall, Precision 和 *F*1 值的分布情况. 图 5 给出了 Scruple 和 13 种方法相比的改进提升情况.

从表 1 和图 4 可看出, 基于专家知识的智能合约漏洞测方法 (Manticore, Osiris, Mythril, Oyente, Slither 和

SmartCheck) 在检测时间戳漏洞时 Recall 最高为 Slither 的 0.78, 最低为 Manticore 和 SmartCheck 的 0.50, 平均为 0.55; Precision 最高为 Slither 的 0.82, 最低为 Osiris 的 0.53, 平均为 0.63; *F*1 值最高为 Slither 的 0.84, 最低为 Manticore 的 0.50; 平均为 0.57. 与它们相比, Scruple 的 Recall 最高相对提升 92%, 最低相对提升 23%; Precision 最高相对提升 70%, 最低相对提升 10%; *F*1 值最高相对提升 86%, 最低相对提升 11%.

表 1　相关方法在 Recall, Precision 和 *F*1 值的性能比较

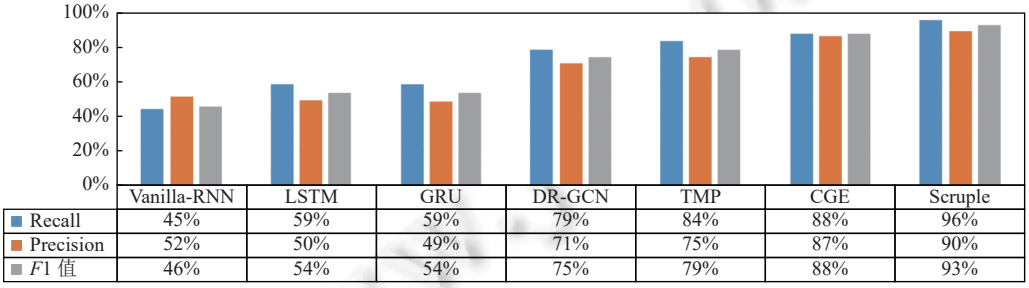
Method		Timestamp vulnerability detection		
		Recall	Precision	*F*1值
Conventional approaches	Manticore	0.50	0.57	0.50
	Osiris	0.51	0.53	0.52
	Mythril	0.51	0.56	0.51
	Oyente	0.52	0.55	0.53
	Slither	0.78	0.82	0.84
	SmartCheck	0.50	0.74	0.51
Deep-learning-based approaches	GCN	0.76	0.68	0.72
	Vanilla-RNN	0.45	0.52	0.46
	LSTM	0.59	0.50	0.54
	GRU	0.59	0.49	0.54
	DR-GCN	0.79	0.71	0.75
	TMP	0.84	0.75	0.79
	CGE	0.88	0.87	0.88
Our approach	Scruple	**0.96**	**0.90**	**0.93**



	Manticore	Osiris	Mythril	Oyente	Slither	SmartCheck	GCN	Scruple
Recall	50%	51%	51%	52%	78%	50%	75%	96%
Precision	57%	53%	56%	55%	82%	75%	68%	90%
*F*1 值	50%	52%	51%	53%	84%	51%	72%	93%

Recall Precision *F*1 值

(a) Scruple 和前 7 种方法的比较



	Vanilla-RNN	LSTM	GRU	DR-GCN	TMP	CGE	Scruple
Recall	45%	59%	59%	79%	84%	88%	96%
Precision	52%	50%	49%	71%	75%	87%	90%
*F*1 值	46%	54%	54%	75%	79%	88%	93%

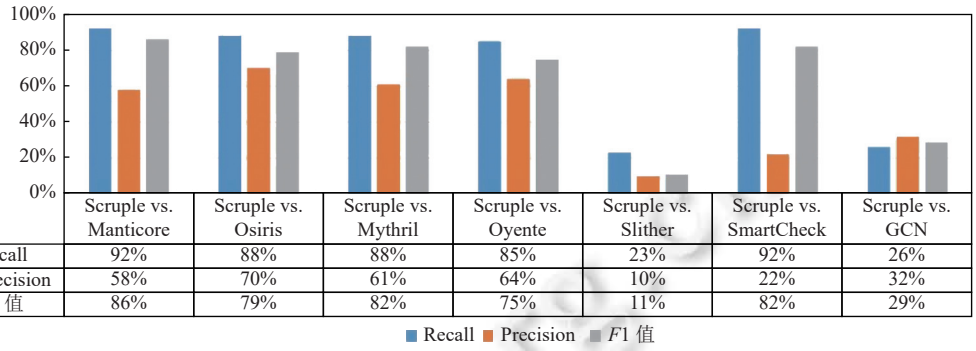
Recall Precision *F*1 值

(b) Scruple 和后 6 种方法的比较

图 4　Scruple 和 13 种方法的 Recall, Precision 和 *F*1 值的分布情况

基于深度学习的智能合约漏洞测方法 (GCN, Vanilla-RNN, LSTM, GRU, DR-GCN, TMP 和 CGE) 在检测时间戳漏洞时 Recall 最高为 CGE 的 0.88, 最低为 Vanilla-RNN 的 0.45, 平均为 0.70; Precision 最高为 CGE 的 0.87, 最

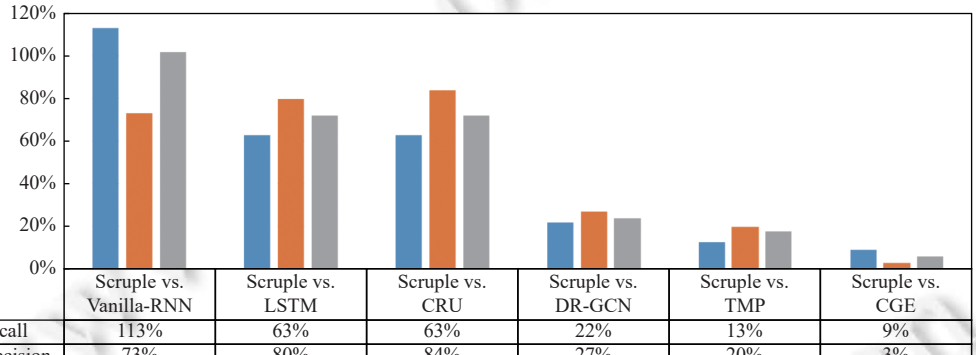
低为 GRU 的 0.49, 平均为 0.65; $F1$ 值最高为 CGE 的 0.88, 最低为 Vanilla-RNN 的 0.46; 平均为 0.67. 和基于深度学习的智能合约漏洞检测方法相比, Scruple 的 Recall 最高相对提升 113%, 最低相对提升 9%, 平均相对提升 44%; Precision 最高相对提升 84%, 最低相对提升 3%, 平均相对提升 46%; $F1$ 值最高相对提升 102%, 最低相对提升 6%, 平均相对提升 46%.



	Scruple vs. Manticore	Scruple vs. Osiris	Scruple vs. Mythril	Scruple vs. Oyente	Scruple vs. Slither	Scruple vs. SmartCheck	Scruple vs. GCN
Recall	92%	88%	88%	85%	23%	92%	26%
Precision	58%	70%	61%	64%	10%	22%	32%
$F1$ 值	86%	79%	82%	75%	11%	82%	29%

Recall Precision $F1$ 值

(a) Scruple 和前 7 种方法相比的提升情况



	Scruple vs. Vanilla-RNN	Scruple vs. LSTM	Scruple vs. CRU	Scruple vs. DR-GCN	Scruple vs. TMP	Scruple vs. CGE
Recall	113%	63%	63%	22%	13%	9%
Precision	73%	80%	84%	27%	20%	3%
$F1$ 值	102%	72%	72%	24%	18%	6%

Recall Precision $F1$ 值

(b) Scruple 和后 6 种方法相比的提升情况

图 5 Scruple 和 13 种方法的 Recall, Precision 和 $F1$ 值的提升情况

总体来说, Scruple 和 13 种智能合约漏洞检测方法相比, Recall 最高相对提升 113%, 最低相对提升 9%, 平均相对提升 59%; Precision 最高相对提升 84%, 最低相对提升 3%, 平均相对提升 46%; $F1$ 值最高相对提升 102%, 最低相对提升 6%, 平均相对提升 57%. 可以看出, 上述有些方法误报率和漏报率较高, 原因可能在于一是部分技术对专家知识过于依赖从而导致低扩展性. 虽然专家知识对提高漏洞检测能力带来了一定的帮助, 但是对于许多新出现的复杂情况, 专家定义规则往往需要耗费很大的精力, 并且容易出错; 二是对代码的语法语义信息利用不够, 或是表示太复杂不利于模型学习泛化.

● RQ2: Scruple 不同模块对漏洞检测的贡献如何?

为了研究预训练对 Scruple 的贡献度, 实验在不载入预训练参数 (GraphCodeBERT[39]) 的情况下, 使用相同的输入对模型进行训练. 训练后进行测试的结果 Recall 为 0.43, Precision 为 0.47, $F1$ 值为 0.46. 因此, 实验得出结论: 预训练技术在 Scruple 进行智能合约时间戳漏洞检测中有重要贡献.

为了研究数据流传播路径对 Scruple 的贡献度, 实验将输入中的传播路径去掉, 保持其他输入不变的情况下进行训练并测试. 即根据第 2.2 节的描述, 实验只取输入 6 个单元的前 4 个 (源代码的 token 集合, 变量集合, 源代码 token 的位置集合, 变量位置集合) 作为输入, 保持模型其他不变的情况下进行了实验. 表 2 为实验结果, 其中

Scruple-rmPC 表示去除数据流传播路经的 Scruple 在测试集的表现. 从实验结果可以看出, 在没有数据流传播路径作为输入的情况下, Scruple 的召回率, 准确率和 *F*1 值均大幅下降. 这说明数据流传播路径是触发漏洞的关键信息. 它代表了与其有直接或间接的数据依赖关系的信息. 对传播路径进行学习突出了漏洞在代码中的传播特性, 指向性更强. 因此, 实验得出结论: 数据流传播路经在 Scruple 进行智能合约时间戳漏洞检测中也有重要贡献.

表 2　Scruple 和 Scruple-rmPC 在 Recall, Precision 和 *F*1 值的性能比较

Method	Recall	Precision	*F*1值
Scruple-rmPC	0.71	0.80	0.78
Scruple	**0.96**	**0.90**	**0.93**

(3) 有效性威胁

本文实验有效性威胁主要如下.

实验选取广泛应用于智能合约漏洞检测的典型数据集 SmartBugs Wild Dataset[25]. 该数据集包含 4 7000 多个真实的智能合约. 该数据集及其标签进行了在线公布[47]. 然而, 现实中还存在许多未知因素, 本文方法不一定能覆盖与适用于现实中的所有情况. 因此, 未来工作将使用更多的真实智能合约程序来验证 Scruple 的有效性. 在实验数据分割上, 实验根据文献 [21,48] 将所有数据打乱并随机抽取其中的 20% 作为训练集, 10% 作为验证集, 70% 作为测试集, 实验结果中的 Precision/Recall 等指标是对所有 70% 测试集进行分析之后的结果. 模型无论是在训练集, 验证集还是测试集上均获得了较好的表现, 没有出现过拟合问题. 本文模型采用有监督学习, 数据集中的智能合约是否包含漏洞均由人工标注, 人工标注可能存在误报漏报的风险. 为了减轻该有效性威胁, 我们联合多名研究人员对时间戳漏洞的发生原因、发生条件和可能出现的误导因素经过反复研究讨论和交叉检查. 因此这类有效性威胁可以忽略.

实验在实现不同对比方法和 Scruple 时可能会包含潜在的错误, 本文根据公开的源代码和之前的研究仔细地实现了它们, 然后用测试用例测试这些方法的正确性. 为了减轻该有效性威胁, 我们团队的几名成员严格地检查了实验的代码实现.

实验采用了召回率, 准确率和 *F*1 值来评估各类时间戳漏洞检测方法的有效性, 鉴于评价标准使用的广泛性, 这类有效性威胁可以忽略.

4　相关工作

智能合约的安全性是研究热点[10], 学术界涌现出了许多的智能合约漏洞检测技术, 包括基于程序分析的技术, 基于形式化验证的技术, 基于模糊测试的技术, 基于机器学习的技术, 基于符号执行的技术和基于污点分析的技术等. 程序分析是一种通用的计算机技术, 旨在通过自动化地对程序进行分析来获得程序的特征和属性. 比较有代表性的基于程序分析地智能合约漏洞检测技术有 SmartCheck[11], SASC[49]和 Slither[12]等. 形式化验证技术是验证程序是否符合预期设计属性和安全规范的有效方法. 典型的基于形式化验证的智能合约漏洞检测技术包括 ZEUS[13], Securify[14]和 VerX[50]等. 模糊测试是一种强大的软件分析技术. 其核心思想是为程序提供大量的测试用例, 以监控其在执行过程中的异常行为, 从而发现程序漏洞. 典型的基于模糊测试的工具或框架有 Echidna[51], Contract-Fuzzer[15], ILF[52]和 Harvey[53]等, 本文的出发点是在智能合约部署前对其进行漏洞检测, 因此对比的基线方法均为基于静态分析的方法. 和基于模糊测试的方法进行对比也是我们将来的工作之一. 污点分析技术是一种特殊的精准化的程序分析技术, 它的原理是通过对程序执行过程中的关键数据进行标记来追踪关键信息的流向, 从而可以找到影响程序操作的关键点所在, 进而可以挖掘程序漏洞. 利用污点分析技术进行智能合约漏洞检测的代表性技术有 Sereum[54]. 符号执行是一种传统的程序漏洞自动挖掘技术, 现在也广泛应用于智能合约漏洞检测. 该技术通过对程序的输入进行取值不固定的符号值的抽象来不断求解约束路径, 从而探索程序分支. 比较有代表性的基于符号执行的智能合约漏洞检测技术有 Oyente[16], Osiris[17], Mythril[18], Manticore[19]和 Honeybadger[55]等. 已有研究较多为基于规则的判定技术, 将基于时间戳的数据流传播路径知识融入规则可能也会有一定的帮助. 但是, 基于规则

的智能合约漏洞检测技术对专家知识较为依赖. 例如: ContractFuzzer 需要首先根据漏洞特征定义测试预言, Securify 需要专家预先定义安全模式, SmartCheck 需要专家知识来获取漏洞特征, 而 Slither 也需要进行预先定义的分析. 不可否认的, 基于规则的判定技术在一定程度上提高了漏洞检测的能力. 随着智能合约数量的爆炸式增长, 研究人员提出了利用代码大数据的方法来自动学习智能合约漏洞的特征, 从而可以辅助进行漏洞检测. SmartEmbed[20]利用深度学习模型计算与智能合约中已知漏洞的相似度来检测是否存在漏洞. Sgram[56]利用 Oyente[16]进行标记之后结合 Ngram 语言建模和轻量级静态语义标记来预测漏洞. Huang 等人[57]基于人工标记的数据集, 首先将智能合约字节码转换为 RGB 颜色, 之后使用卷积神经网络来训练和预测智能合约安全漏洞. Tann 等人[58]利用 MAIAN 对智能合约安全问题进行标记并使用 LSTM 来预测潜在的漏洞.

除此之外, 在程序缺陷检测领域, 还有许多基于深度学习的检测方法. DeepBugs[59]利用 Word2Vec 对源代码进行表征进而检测基于命名的程序漏洞. CNN-FL[46]利用卷积神经网络对程序动态执行覆盖信息进行学习从而对程序缺陷进行定位. Li 等人[60]利用深度神经网络学习各类缺陷定位方法的特征来进行程序缺陷的检测. Zhang 等人[61]综合利用多种深度神经网络对程序谱就行学习, 进而辅助进行真实大型程序的缺陷检测. Lam 等人[62]利用深度学习技术对 IR 进行学习来预测可能包含潜在漏洞的程序文件. DeepRL4FL[63]利用深度学习中应用于计算机视觉的技术, 将整体程序覆盖信息作为特征进行学习, 从而完成缺陷检测. 虽然关注对象有所不同, 这些基于深度学习的程序缺陷检测工作可以为智能合约漏洞检测提供借鉴.

5 结 论

本文提出了一种基于数据流传播路经学习的智能合约漏洞检测方法 Scruple. 该方法通过对程序数据流传播路经进行学习, 不仅能够将合约源代码作为特征进行学习, 更关注于与智能合约漏洞密切相关的程序传播路径特征. 此外, 本文还探讨了使用预训练模型进行漏洞检测的可行性. 实验结果表明, 本文方法明显优于 13 种主流智能合约漏洞检测方法, 大幅提升了时间戳漏洞检测性能. 未来工作包括方法优化和在更多类型的智能合约漏洞进行实验.

References:

[1] Szabo N. Smart contracts: Building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought, 1996, (16): 18.

[2] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf

[3] Buterin V. Understanding serenity, Part 2: Casper. 2013. https://blog.ethereum.org/2015/12/28/understanding-serenity-part-2-casper

[4] Dannen C. Introducing Ethereum and Solidity. Apress: Springer, 2017. [doi: 10.1007/978-1-4842-2535-6]

[5] Chen WL, Ma MJ, Ye YJ, Zheng ZB, Zhou YR. IoT service based on jointcloud blockchain: The case study of smart traveling. In: Proc. of the 2018 IEEE Symp. on Service-oriented System Engineering (SOSE). Bamberg: IEEE, 2018. 216–221. [doi: 10.1109/SOSE.2018.00036]

[6] Velner Y, Teutsch J, Luu L. Smart contracts make bitcoin mining pools vulnerable. In: Proc. of the 2017 Int'l Conf. on Financial Cryptography and Data Security. Sliema: Springer, 2017. 298–316. [doi: 10.1007/978-3-319-70278-0_19]

[7] Chen JC, Xia X, Lo D, Grundy J, Luo XP, Chen T. Defining smart contract defects on Ethereum. IEEE Trans. on Software Engineering, 2022, 48(2): 327–345. [doi: 10.1109/TSE.2020.2989002]

[8] del Castillo M. The DAO attacked: Code issue leads to $60 million ether theft. Saatavissa (viitattu 13.2. 2017). 2016. https://github.com/jaswalabhijeet/Documents-Blockchain/blob/master/The%20DAO%20Attacked:%20Code%20Issue%20Leads%20to%20%2460%20Million%20Ether%20Theft%20-%20CoinDesk.pdf

[9] Nabilou H. How to regulate bitcoin? decentralized regulation for a decentralized cryptocurrency. Int'l Journal of Law and Information Technology, 2019, 27(3): 266–291. [doi: 10.1093/ijlit/eaz008]

[10] Atzei N, Bartoletti M, Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In: Proc. of the 6th Int'l Conf. on Principles of Security and Trust. Uppsala: Springer, 2017. 164–186. [doi: 10.1007/978-3-662-54455-6_8]

[11] Tikhomirov S, Voskresenskaya E, Ivanitskiy I, Takhaviev R, Marchenko E, Alexandrov Y. SmartCheck: Static analysis of Ethereum smart contracts. In: Proc. of the 1st IEEE/ACM Int'l Workshop on Emerging Trends in Software Engineering for Blockchain. Gothenburg: IEEE, 2018. 9–16.

[12] Feist J, Grieco G, Groce A. Slither: A static analysis framework for smart contracts. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on

Emerging Trends in Software Engineering for Blockchain (WETSEB). Montreal: IEEE, 2019. 8–15. [doi: 10.1109/WETSEB.2019.00008]

[13] Kalra S, Goel S, Dhawan M, Sharma S. ZEUS: Analyzing safety of smart contracts. In: Proc. of the 2018 Network and Distributed Systems Security (NDSS) Symp. San Diego, 2018. 1–12. [doi: 10.14722/ndss.2018.23082]

[14] Tsankov P, Dan A, Drachsler-Cohen D, Gervais A, Bünzli F, Vechev M. Securify: Practical security analysis of smart contracts. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 67–82. [doi: 10.1145/3243734.3243780]

[15] Jiang B, Liu Y, Chan WK. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 259–269. [doi: 10.1145/3238147.3238177]

[16] Luu L, Chu D H, Olickel H, Saxena P, Hobor A. Making smart contracts smarter. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 254–269. [doi: 10.1145/2976749.2978309]

[17] Torres C F, Schütte J, State R. Osiris: Hunting for integer bugs in Ethereum smart contracts. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 664–676. [doi: 10.1145/3274694.3274737]

[18] Mueller B. Mythril-reversing and bug hunting framework for the Ethereum blockchain. 2017. https://pypi.org/project/mythril/0.8.2

[19] Mossberg M, Manzano F, Hennenfent E, Groce A, Grieco G, Feist J, Brunson T, Dinaburg A. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1186–1189. [doi: 10.1109/ASE.2019.00133]

[20] Gao ZP, Jayasundara V, Jiang LX, Xia X, Lo D, Grundy J. SmartEmbed: A tool for clone and bug detection in smart contracts through structural code embedding. In: Proc. of the 2019 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Cleveland: IEEE, 2019. 394–397. [doi: 10.1109/ICSME.2019.00067]

[21] Zhuang Y, Liu ZG, Qian P, Liu Q, Wang X, He Q. Smart contract vulnerability detection using graph neural networks. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence. Yokohama: Unknown Publishers, 2020. 454.

[22] Zhang T, Xu BW, Thung F, Haryono SA, Lo D, Jiang LX. Sentiment analysis for software engineering: How far can pre-trained transformer models go? In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 70–80. [doi: 10.1109/ICSME46990.2020.00017]

[23] Liu F, Li G, Zhao YF, Jin Z. Multi-task learning based pre-trained language model for code completion. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 473–485.

[24] Robbes R, Janes A. Leveraging small software engineering data sets with pre-trained neural networks. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). Montreal: IEEE, 2019. 29–32. [doi: 10.1109/ICSE-NIER.2019.00016]

[25] Ferreira JF, Cruz P, Durieux T, Abreu R. Smartbugs: A framework to analyze Solidity smart contracts. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 1349–1352.

[26] Wang W, Song JJ, Xu GQ, Li YD, Wang H, Su CH. ContractWard: Automated vulnerability detection models for Ethereum smart contracts. IEEE Trans. on Network Science and Engineering, 2021, 8(2): 1133–1144. [doi: 10.1109/TNSE.2020.2968505]

[27] Wood G. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014. https://cryptodeep.ru/doc/paper.pdf

[28] Zhang PC, Xiao F, Luo XP. A framework and dataset for bugs in Ethereum smart contracts. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 139–150. [doi: 10.1109/ICSME46990.2020.00023]

[29] Ni YD, Zhang C, Yin TT. A survey of smart contract vulnerability research. Journal of Cyber Security, 2020, 5(3): 78–99 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.07]

[30] Bach LM, Mihaljevic B, Zagar M. Comparative analysis of blockchain consensus algorithms. In: Proc. of the 41st Int'l Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija: IEEE, 2018. 1545–1550. [doi: 10.23919/MIPRO.2018.8400278]

[31] Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L. Deep contextualized word representations. In: Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans: ACL, 2018. 2227–2237. [doi: 10.18653/v1/N18-1202]

[32] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training. 2018. https://paperswithcode.com/paper/improving-language-understanding-by

[33] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

Minneapolis: ACL, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[34] Kanade A, Maniatis P, Balakrishnan G, Shi K. Learning and evaluating contextual embedding of source code. arXiv:2001.00059, 2019.

[35] Karampatsis RM, Sutton C. SCELMo: Source code embeddings from language models. arXiv:2004.13214, 2020.

[36] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[37] Svyatkovskiy A, Deng SK, Fu SY, Sundaresan N. Intellicode compose: Code generation using transformer. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1433–1443. [doi: 10.1145/3368089.3417058]

[38] Buratti L, Pujar S, Bornea M, McCarley S, Zheng YH, Rossiello G, Morari A, Laredo J, Thost V, Zhuang YF, Domeniconi G. Exploring software naturalness through neural language models. arXiv:2006.12641, 2020.

[39] Guo DY, Ren S, Lu S, Feng ZY, Tang DY, Liu SJ, Zhou L, Duan N, Svyatkovskiy A, Fu SY, Tufano M, Deng SK, Clement CB, Drain D, Sundaresan N, Yin J, Jiang DX, Zhou M. GraphCodeBERT: Pre-training code representations with data flow. In: Proc. of the 9th Int'l Conf. on Learning Representations. ICLR, 2021.

[40] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: ICLR, 2018.

[41] Hellendoorn VJ, Sutton C, Singh R, Maniatis P, Bieber D. Global relational models of source code. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: ICLR, 2019.

[42] Guo AB, Mao XG, Yang DH, Wang SW. An empirical study on the effect of dynamic slicing on automated program repair efficiency. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 554–558. [doi: 10.1109/ICSME.2018.00066]

[43] Lua T. tree-sitter. 2023. https://tree-sitter.github.io/tree-sitter/

[44] Honig J. tree-sitter-solidity. 2023. https://pypi.org/project/tree-sitter-solidity/

[45] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[46] Zhang Z, Lei Y, Mao XG, Li PP. CNN-FL: An effective approach for localizing faults using convolutional neural networks. In: Proc. of the 26th Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Hangzhou: IEEE, 2019. 445–455. [doi: 10.1109/SANER.2019.8668002]

[47] Liu ZG, Qian P, Wang XY, Zhuang Y, Qiu L, Wang X. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(2): 1296–1310. [doi: 10.1109/TKDE.2021.3095196]

[48] Zhang Z, Lei Y, Yan M, Yu Y, Chen JC, Wang SW, Mao XG. Reentrancy vulnerability detection and localization: A deep learning based two-phase approach. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2022. 1–13.

[49] Zhou EC, Hua S, Pi BF, Sun J, Nomura Y, Yamashita K, Kurihara H. Security assurance for smart contract. In: Proc. of the 9th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). Paris: IEEE, 2018. 1–5. [doi: 10.1109/NTMS.2018.8328743]

[50] Permenev A, Dimitrov D, Tsankov P, Drachsler-Cohen D, Vechev M. VerX: Safety verification of smart contracts. In: Proc. of the 2020 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2020. 1661–1677. [doi: 10.1109/SP40000.2020.00024]

[51] Grieco G, Song W, Cygan A, Feist J, Groce A. Echidna: Effective, usable, and fast fuzzing for smart contracts. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 557–560. [doi: 10.1145/3395363.3404366]

[52] He JX, Balunović M, Ambroladze N, Tsankov P, Vechev M. Learning to fuzz from symbolic execution with application to smart contracts. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 531–548. [doi: 10.1145/3319535.3363230]

[53] Wustholz V, Christakis M. Harvey: A greybox fuzzer for smart contracts. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. ACM, 2020. 1398–1409. [doi: 10.1145/3368089.3417064]

[54] Rodler M, Li WT, Karame GO, Davi L. Sereum: Protecting existing smart contracts against re-entrancy attacks. In: Proc. of the 26th Annual Network and Distributed System Security Symp. San Diego: NDSS, 2018.

[55] Torres CF, Steichen M, State R. The art of the scam: Demystifying honeypots in Ethereum smart contracts. In: Proc. of the 28th USENIX Conf. on Security Symp. Santa Clara: USENIX Association, 2019. 1591–1607.

[56] Liu H, Liu C, Zhao WQ, Jiang Y, Sun JG. S-gram: Towards semantic-aware security auditing for Ethereum smart contracts. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 814–819. [doi: 10.1145/3238147.

3240728]
[57] Huang TTHD. Hunting the Ethereum smart contract: Color-inspired inspection of potential attacks. arXiv:1807.01868, 2018.
[58] Tann WJW, Han XJ, Gupta SS, Ong YS. Towards safer smart contracts: A sequence learning approach to detecting security threats. arXiv:1811.06632, 2018.
[59] Pradel M, Sen K. DeepBugs: A learning approach to name-based bug detection. Proc. of the ACM on Programming Languages, 2018, 2: 147. [doi: 10.1145/3276517]
[60] Li X, Li W, Zhang YQ, Zhang LM. DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 169–180. [doi: 10.1145/3293882.3330574]
[61] Zhang Z, Lei Y, Mao XG, Yan M, Xu L, Zhang XH. A study of effectiveness of deep learning in locating real faults. Information and Software Technology, 2021, 131: 106486. [doi: 10.1016/j.infsof.2020.106486]
[62] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports (N). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 476–481. [doi: 10.1109/ASE.2015.73]
[63] Li Y, Wang SH, Nguyen T. Fault localization with code coverage representation learning. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 661–673. [doi: 10.1109/ICSE43902.2021.00067]

附中文参考文献:
[29] 倪远东, 张超, 殷婷婷. 智能合约安全漏洞研究综述. 信息安全学报, 2020, 5(3): 78–99. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.05.07]

张卓(1984－), 男, 博士, 主要研究领域为软件错误定位, 软件自动修复, 智能软件工程.

鄢萌(1989－), 男, 博士, 研究员, CCF 专业会员, 主要研究领域为智能软件工程.

刘业鹏(1989－), 男, 助理工程师, 主要研究领域为智能软件工程.

陈嘉弛(1994－), 男, 博士, 助理教授, CCF 专业会员, 主要研究领域为智能合约可靠性分析, 软件工程.

薛建新(1980－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为并发理论, 程序分析.

毛晓光(1970－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为可信软件, 软件维护与演化.