计算机科学
COMPUTER SCIENCE

基于预训练技术和专家知识的重入漏洞检测

陈乔松, 何小阳, 许文杰, 邓欣, 王进, 朴昌浩

引用本文

陈乔松, 何小阳, 许文杰, 邓欣, 王进, 朴昌浩基于预训练技术和专家知识的重入漏洞检测[J]. 计算机科学, 2022, 49(11A): 211200182-8.

CHEN Qiao-song, HE Xiao-yang, XU Wen-jie, DENG Xin, WANG Jin, PIAO Chang-hao. Reentrancy Vulnerability Detection Based on Pre-training Technology and Expert Knowledge [J]. Computer Science, 2022, 49(11A): 211200182-8.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211200182

基于预训练技术和专家知识的重入漏洞检测

陈乔松[1] 何小阳[1] 许文杰[1] 邓 欣[1] 王 进[1] 朴昌浩[2]
1 重庆邮电大学计算机科学与技术学院数据工程与可视计算重点实验室 重庆 400065
2 重庆邮电大学自动化学院智慧能源技术研究中心 重庆 400065

摘 要 随着区块链中智能合约的安全问题日益突出,智能合约的漏洞检测任务逐渐成为研究的热点。然而,目前的智能合约重入漏洞检测技术主要是符号执行、静态分析、形式化验证和模糊测试等传统的检测方法,这些检测方法不仅存在较高的误报率和漏报率,而且检测精度较低。同时,基于深度学习的方法也有其独特的局限性。针对这些问题,文中提出了一种将预训练技术与传统的专家知识相融合的检测方法,同时将智能合约进行切片处理,以此减小无关数据对模型的影响。文中聚焦于重入漏洞的检测,在203716份合约数据集上进行实验。实验结果表明,基于预训练技术和专家知识的智能合约重入漏洞检测方法具有96.2%的精确率、97.7%的召回率以及96.9%的F1分数,检测效果均优于现有的检测方法。

关键词:区块链;智能合约;漏洞检测;预训练技术;专家知识
中图法分类号 TP311

Reentrancy Vulnerability Detection Based on Pre-training Technology and Expert Knowledge

CHEN Qiao-song[1], HE Xiao-yang[1], XU Wen-jie[1], DENG Xin[1], WANG Jin[1] and PIAO Chang-hao[2]
1 Key Laboratory of Data Engineering and Visual Computing, School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
2 Smart Energy Technology Research Center, School of Automation, Chongqing University of Posts and Telecommunications, Chongqing 400065, China

Abstract As the security issues of smart contracts in blockchain become increasingly prominent, the vulnerability detection tasks of smart contracts have gradually become a research hotspot. However, the current smart contract reentrancy vulnerability detection technologies are mainly traditional detection methods such as symbolic execution, static analysis, formal verification and fuzzing. These detection methods not only have high false positive rate and false negative rate, but also have low detection accuracy. At the same time, methods based on deep learning also have their unique limitations. In response to these problems, this paper proposes a detection method that combines pre-training technology and traditional expert knowledge, and at the same time slices smart contracts to reduce the impact of irrelevant data on the model. This paper focuses on the detection of reentrancy vulnerability and conducts experiments on 203716 contract data sets. Experimental results show that the smart contract reentrancy vulnerability detection method based on pre-training technology and expert knowledge has an accuracy rate of 96.2%, a recall rate of 97.7% and a F1 score of 96.9%, which are better than existing detection methods.

Keywords Blockchain, Smart contract, Vulnerability detection, Pre-training technology, Expert knowledge

1 引言

近年来,由于区块链本身的去中心化特性,区块链技术发展迅速,在各个领域得到了广泛的应用。区块链本质上是一个由多方根据共识协议共同维护的分布式账本,而智能合约则是区块链的关键核心,智能合约部署在区块链上,能够根据条件自动执行合约代码,满足各种应用场景的功能需求。以太坊是目前使用人数最多的区块链平台,同时以太坊上也部署了大量的智能合约。值得注意的是,以太坊上的智能合约有很多都涉及到加密货币,这些加密货币的价值可能达到数百万美元,这意味着智能合约的安全性可能会严重影响到资金的安全。2016年6月,以太坊上发生了著名的以太币盗窃(Dao[1])事件,该事件导致了价值达6000万美元的360万枚以太币被非法转移。据SlowMist Hacked[2]统计数据显示,区块链上已发生的攻击事件超过500起,造成的资金损失总计超200亿美元,其中很多都是由于智能合约的漏洞造成的,因此智能合约的安全问题引起了广泛的关注。由于区块链具有不可更改性,智能合约一旦部署就不能再修改,智能合约中存在的漏洞无法进行修复。因此,在智能合约部署之前进行漏洞检测是智能合约开发流程中必不可少的环节。

到目前为止,很多学者已经提出一些关于智能合约重入漏洞检测的方法,Oyente[3],Osiris[4],Mythril[5],Maian[6],

基金项目:国家自然科学基金(61806033);国家社会科学基金西部项目(18XGL013)
This work was supported by the National Natural Science Foundation of China(61806033) and Western Project of National Social Science Foundation of China(18XGL013).
通信作者:陈乔松(chenqs@cqupt.edu.cn)

Manticore[7]使用符号执行的方式进行漏洞检测，Securify[8]和 Zeus[9]提出形式化验证的方式进行漏洞检测。Slither[10]和 SmartCheck[11]提出使用静态分析的方式进行漏洞检测。ContractFuzzer[12]提出使用模糊测试的方式进行漏洞检测。但是这些传统的检测方法存在一系列缺陷，如检测准确率较低，存在较高的误报率和漏报率。

为了解决存在的问题，本文提出了一种基于预训练技术和专家知识的智能合约重入漏洞检测方法。在 NLP(自然语言处理)领域中，预训练技术已经得到了大量学者的使用。本文将预训练技术的优势转移到智能合约漏洞检测模型中，融合了传统的专家知识检测方法，并对智能合约进行切片处理，使合约中的无关数据对模型的影响降到最低。专家知识能够进一步降低预训练模型在智能合约漏洞检测中的误报率和漏报率，专家知识和预训练技术融合能够互相提升，并提取到深度学习方式不能学习到的特征信息。专家知识的提取不采用手动提取的方式，而是使用自动提取的方式，因此本文模型能够完全自动地进行智能合约重入漏洞检测。

在具有超过 20 万份真实智能合约源代码的 SmartBugs Wild[13]数据集上进行实验，重点关注重入漏洞的检测，实验结果显示，本文方法对智能合约重入漏洞检测具有较好的效果。

2 背景介绍

2.1 区块链

区块链是一种结合了分布式技术、密码学技术、网络技术和时间戳技术的融合技术，具有匿名性、去中心化、不可篡改、可追溯和多方共同维护等特性，这使得区块链技术在各个领域得到了广泛的应用。区块链逻辑上是一种链式结构，每一个区块都通过一个父 hash 值指向其上一个区块，区块链的结构可大致分为网络层、共识层和应用层。

2.2 智能合约和漏洞

2.2.1 智能合约

智能合约是一种可执行的代码，主要用于在满足特定条件下代码能够自动执行，智能合约已经广泛应用于数字支付和金融资产处理等领域。以太坊上的智能合约由高级编程语言 Solidity[14]编写，合约通过编译后部署在以太坊区块链上，编译后的合约字节码在 EVM(以太坊虚拟机)中运行，通过合约的地址即可进行调用。然而，由于区块链的不可更改性，智能合约一旦部署就不能再进行修改，这也导致了很多已部署的智能合约包含各种各样的漏洞，攻击者可以重复利用这些漏洞，获取巨大的非法利益，并且交易无法回退，从而造成严重的资金损失。下面介绍 3 种常见的漏洞类型。

2.2.2 重入漏洞

重入漏洞是最值得关注的一种漏洞，因为这种漏洞一旦被发现，就会造成严重的资金损失，Dao 事件就是重入漏洞之一。以太坊上的智能合约可以通过外部合约进行调用，在调用合约函数时，往往伴随着合约状态的改变，如 balance(余额)的更新。但是，如果攻击者通过一些方式暂时跳过更新状态的代码，则可能会造成重入攻击，重入攻击示例如图 1 所示。

```
1  contract SimpleDAO {                          1  contract Attacker {
2    mapping (address => uint) public balance;   2    address dao = 0x2a3...6e;
3                                                3
4    function donate() payable public {          4    function attack() public {
5      balance[msg.sender] += msg.value;         5      dao.donate.value(1 ether)();
6    }                                           6      dao.withdraw(1 ether);
7                                                7    }
8    function withdraw(uint amount) public {     8
9      if (balance[msg.sender] >= amount) {      9    function () payable public {
10       require(msg.sender.call.value(amount)()); 10     if(counter++ < 99) {
11       credit[msg.sender]  -= amount;          11       dao.withdraw(1 ether);
12     }                                         12     }
13   }                                           13   }
14 }                                            14 }
```

图 1 一个重入漏洞例子

Fig. 1 Example of reentrancy vulnerability

Attacker 合约为发起攻击的合约，SimpleDAO 合约为受害者合约。Attacker 合约第 1 步调用 SimpleDAO 合约中的 donate 函数向自己的账户存入 1 个 Ether(以太币)，Attacker 合约第 2 步调用 SimpleDAO 合约中的 withdraw 函数提取自己的 1 个 Ether。但是，Attacker 合约中有一个 fallback 函数，即 payable 函数，在使用 call. value(SimpleDao 合约第 10 行)向 Attacker 转移 Ether 时这个函数会被调用。因此，攻击者可以在 fallback 函数内部实现再次调用 SimpleDAO 合约中的 withdraw 函数。SimpleDAO 合约中 withdraw 函数执行 msg. sender. call. value(amount)时，不仅有向 Attacker 合约转移 Ether 的操作，而且会触发 Attacker 合约的 fallback 函数，即图 1 中的第 3 步。fallback 函数再调用 withdraw 函数，即图 1 中的第 4 步，此时又会重新从 withdraw 函数的开头开始执行，陷入第 3 步和第 4 步的循环中，从而跳过了 balance 数组变量的更新。最后 SimpleDAO 合约向 Attacker 合约转入了大量的 Ether，而 SimpleDAO 中的 balance 变量却未发生变化，从而引起了重入攻击。

2.2.3 时间戳漏洞

时间戳是区块中的时间属性，如果合约中使用区块的时间戳属性来判断是否满足合约代码的执行条件，则该合约可能具有时间戳漏洞。例如，一些合约使用区块的时间戳属性作为生成随机数的种子，或者使用时间戳作为关键操作的判断条件。但是区块链中的区块是经过旷工挖掘的，也就是说区块时间戳属性的值可以由旷工决定，旷工能够选择有利于自己交易的时间戳，这使得这些合约的某些执行条件能够被旷工所控制，从而造成资金损失。

2.2.4 算术溢出漏洞

算术溢出包括算术上溢和算术下溢，表示运算结果的值超出了变量能够表示的范围，使得变量表示的值并不是实际真实的值。例如，一个 uint8 类型的变量，它能够表示的整数范围是 0 到 255(2^8-1)。如果经过加法运算后它的值为 256，则会发生算术上溢，最后的结果变为一个很小的数。如果经过减法运算后它的值为 1，则会发生算术下溢，变为一个很大的数。在 2018 年，BECToken 合约就遭遇了整数上溢攻击，因为合约的开发人员没有考虑不同变量类型的溢出问题。

2.3 预训练技术

预训练技术在 NLP 领域中是在解决研究问题时经常使用的方法，包括预训练和微调两个阶段。预训练阶段是为了构建一个基于上下文嵌入的预训练模型(Pre-Trained Model，PTM)，微调阶段是根据不同的下游任务对 PTM 进行微调。

在 NLP 领域中的多项研究表明，预训练技术能够明显提升模型的泛化能力，使研究人员能够集中精力进行下游任务实验。Bert[15] 和 GPT[16] 是很受欢迎的 PTM，它们的底层都是基于 Transformer[17] 进行改进的，能够提取丰富的上下文特征信息。Codebert[18] 基于 Bert 增强了预训练技术在程序语言（Programming Language，PL）方面的学习能力，Bert 适用于自然语言（Natural Language，NL）数据类型，而 Codebert 引入了程序语言进行训练，适用于自然语言和程序语言（PL-NL）的双模态数据类型，在代码搜索和代码生成任务中具有很好的效果。Graphcodebert[19] 在 Codebert 的基础上进行了扩展，引入了程序语言内部变量之间的数据流关系图，保留了代码的语义信息，Graphcodebert 在克隆检测、代码搜索、代码修复和代码翻译任务中达到了最好的性能。这表明在代码智能相关任务中，代码类型的数据不适合直接被当作文本序列输入模型，而应该保留代码内部的语义信息，这正是 Graphcodebert 在许多代码智能相关任务中效果明显优于 RNN（循环神经网络）和 LSTM（长短期记忆网络）的原因。

3 方案设计

本文方法的漏洞检测原理是提取已有漏洞的特征信息，然后与测试合约的特征信息进行相似性比较，相似性越高表明该合约具有该漏洞的可能性越高。检测过程包含 6 个阶段，数据预处理阶段、提取变量数据流图阶段、构造输入数据阶段、预训练网络特征提取阶段、专家知识网络特征提取阶段和漏洞检测阶段，核心网络结构图如图 2 所示。

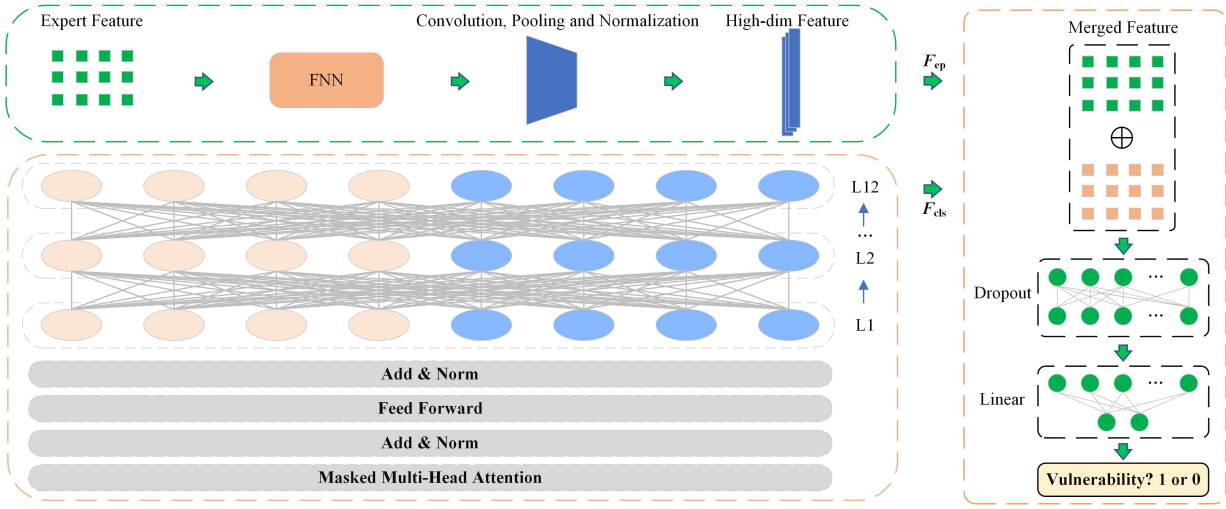

图 2 网络架构图

Fig. 2 Architecture of the proposed network

3.1 数据预处理

在软件漏洞检测领域中，SySeVR[20] 提出了使用基于程序的语法和语义获取程序漏洞的相关表示。本文数据集中的智能合约是以太坊平台上的真实合约，其中包含了大量的注释语句和空白语句，对于网络模型来说，需要控制输入模型的数据长度，以降低无关数据对模型的影响。因此，本文根据 SySeVR 的方式对智能合约数据集进行了进一步处理，提取出合约的关键切片表示。该切片根据特定于漏洞的提取方式进行提取，并遵守 Solidity 的合约结构层次规范。这使得该切片能够表示该合约特定于漏洞的完整语义信息，同时也保留了完整的结构层次，这样的方法能够更适用于 PTM 中的代码语义学习过程。合约切片能够以最少的数据长度表示尽可能多的语义信息，同时不影响模型对漏洞检测的性能。通过这种方式去除合约中的无用代码和无用函数，使合约中跟漏洞无关的数据对模型的干扰降到最低，同时保留了合约的关键语法和语义信息，数据预处理过程如图 3 所示。

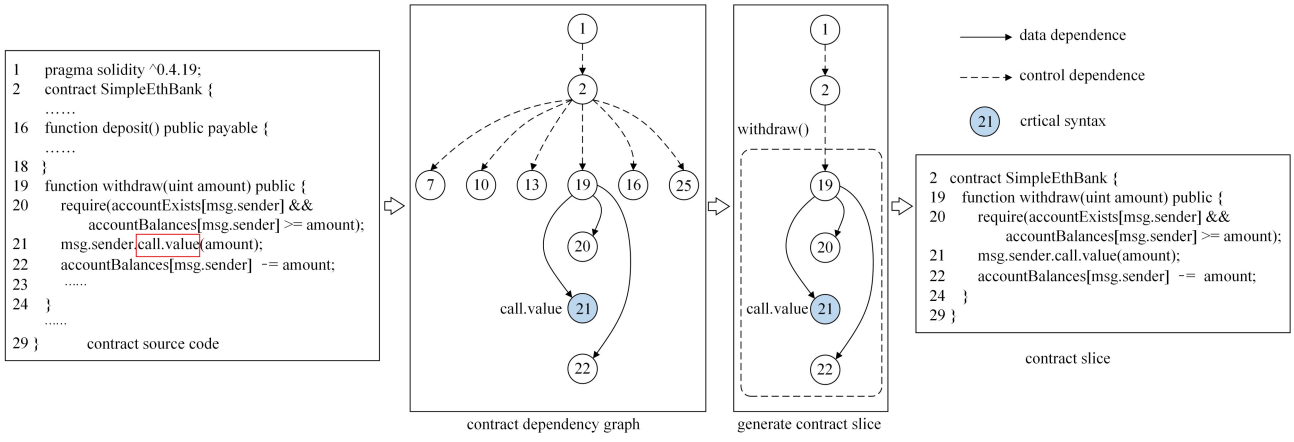

图 3 提取合约切片

Fig. 3 Extract contract slices

首先，删除合约中的空白行和代码注释内容。对于漏洞检测模型来说，需要尽可能地减少没有任何意义的数据内容，空白行和代码的注释对漏洞检测没有任何帮助，因此选择删掉这些内容。

其次，提取合约关键语法切片。通过使用漏洞的语法特征去识别合约中的特定候选切片，以重入攻击为例，在有重入漏洞的合约中，造成合约有被重入攻击风险的语法特征是关键字代码 call. value。因此，需要在合约代码中提取含有 call. value 关键字代码的函数集合 $F_{\text{call. value}}=\{\text{withdraw}\}$，然后提取调用了 $F_{\text{call. value}}$ 的函数集合 F_{call}，因为重入攻击可能是由其他合约调用包含 call. value 的函数造成的。因此，根据漏洞的关键语法特征，从合约代码中提取函数集合 $F_{\text{call. value}}$ 和 F_{call} 作为关键语法切片，其余与漏洞语法特征不相关的内容不再保留。

最后，提取合约关键语义切片。在关键语法切片的基础上，对包含 call. value 关键字的代码行进行分析，提取该代码行的所有相关变量。如图 3 左侧第 21 行代码所示，其中 amount 是调用 call. value 涉及到的变量参数，提取该行涉及到的变量集合 $V_{\text{call. value}}=\{\text{amount, msg, sender}\}$。通过分析智能合约源代码生成 CFG(控制流图)，然后在合约 CFG 中根据控制依赖和数据依赖提取与 $V_{\text{call. value}}$ 有依赖关系的代码行。提取所有的代码行组成合约切片，最后得到的合约切片中只包含与该漏洞有关的关键语法和语义信息，如图 3 右侧所示，实验结果表明，该切片对于漏洞检测具有较好的效果。

经过合约切片处理后，在可能存在重入漏洞的数据集中数据的序列长度平均减小了 50%，这极大地减小了合约中跟漏洞无关的数据序列对模型的影响。

3.2 提取变量数据流图

变量之间的数据流图代表了变量之间的数据依赖关系，数据流图由多个点和多条边构成，用 $G_{df}(C)=(V,E)$ 表示智能合约 C 的数据流图，其中 $V=\{v_1,v_2,\cdots,v_m\}$ 表示智能合约中的相关变量，每个变量都可以作为图 G_{df} 中的一个节点。其中 $E=\{e_1,e_2,\cdots,e_n\}$ 表示智能合约中两个变量之间的数据依赖关系，若 $\langle v_i,v_j\rangle$ 之间存在一条边 e，则表明变量 v_j 的值来源于变量 v_i 的值。以 accountBalances[msg. sender]-= amount 为例，在数据流图中可以很容易知道 accountBalances 的值来源于 amount，或 accountBalances 的值由 amount 计算而来。但是如果仅仅给出智能合约源代码，模型并不知道智能合约中的变量语义信息，只能当作纯序列处理。因此，数据流图的变量依赖关系更能够体现智能合约的语义信息，这对于模型的智能合约源代码理解能力十分重要。提取变量数据流图的过程如图 4 所示。

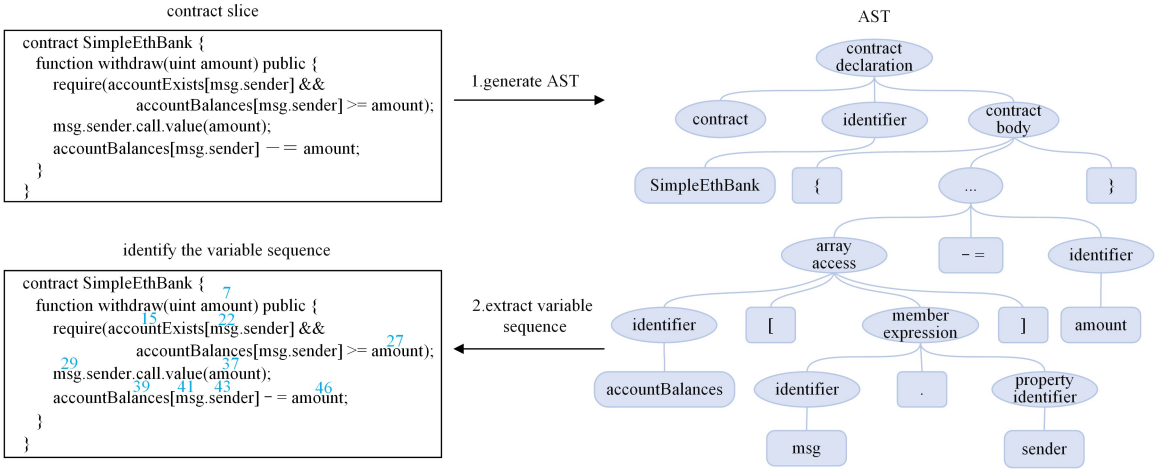

图 4 标识变量序列

Fig. 4 Identify variable sequence

首先，需要将智能合约源代码转换为 AST(抽象语法树)。使用经过 Peculiar[21] 改进的 tree-sitter 工具，该工具能够应用于 Solidity 编写的智能合约，将智能合约转换为 AST。AST 能够表示智能合约的层次结构、变量和语法关系，如 contract_declaration, contract_body, function_definition 和 function_body 等。AST 中的叶子节点表示智能合约源代码中的具体变量名或操作符等，中间节点表示定义、表达式或标识符等。

然后使用 AST 标识变量序列号并提取出数据流图。通过分析 AST 中的节点类型，可以确定任意两个变量之间是否存在依赖关系。如果某两个变量之间存在依赖关系，则在这两个变量之间添加一条边，如 v_j 依赖于 v_i，则添加边 $e=\langle v_i,v_j\rangle$。通过重复这样的步骤可以得到智能合约 C 的数据流图 $G'_{df}(C)=(V',E')$。最后使用 Peculiar 提出的关键数据流图算法进一步减少 $G_{df}(C)$ 中的无用节点和无用边，只保留对漏洞检测有用的节点和边，最后得到合约 C 的数据流图 $G_{df}(C)=(V,E)$，如图 5 所示。

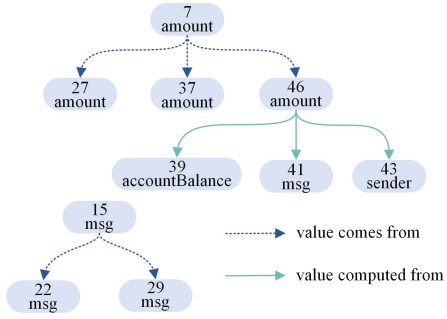

图 5 变量之间的数据流关系

Fig. 5 Data flow relationship between variables

3.3 构造预训练网络输入序列

将智能合约的代码序列和变量序列作为 Graphcodebert

（预训练模型）的输入，并不使用 Graphcodebert 模型中的注释序列，因为漏洞检测只需要考虑智能合约的源代码和变量之间的依赖关系。将智能合约 $C=\{c_1,c_2,\cdots,c_s\}$ 经过数据流图提取得到智能合约 C 的数据流图 $G_{df}(C)=(V,E)$。其中，$V=\{v_1,v_2,\cdots,v_m\}$ 是智能合约源代码中的变量节点，$E=\{e_1,e_2,\cdots,e_n\}$ 是智能合约中变量之间的依赖关系。通过把代码序列和变量序列组合在一起，构成模型的输入序列 $S=\{[CLS],C,[SEP],V\}$，其中[CLS]在序列中表示序列的开始位置，[SEP]起到分割作用，把代码序列与变量节点序列分开。对于预训练模型来说，在[CLS]和[SEP]之间的序列代表智能合约的源代码序列，在[SEP]后面的序列代表智能合约源代码之间的变量序列。

将序列 S 中的每一个序列进行词嵌入表示，将其转换为能够被神经网络处理的向量 X^0，对于 S 中的任意一个序列 S_i，S_i 的向量表示由 S_i 的词嵌入向量 E_{Si} 和 S_i 的位置编码向量 P_{Si} 构成。除了输入的合约序列和变量序列信息外，还有一个 Mask 矩阵，Mask 矩阵表示序列之间以图为基础的关系矩阵，Mask 矩阵会为每个序列之间计算一个值，这个值在多层 Transformer 网络模型中用于区分是否处理该序列之间的 attention（注意力）关系计算。构造模型输入序列如图 6 所示。

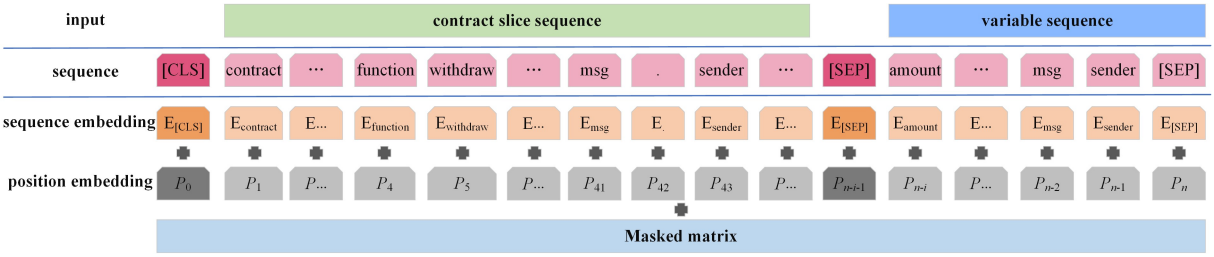

图 6　构造模型输入序列

Fig. 6　Construct model input sequence

3.4　预训练网络

如图 2 左侧下方所示，本文使用基于 Bert 的 Graphcodebert 作为网络模型的第一部分，因为在智能合约源代码中，漏洞的检测不仅需要考虑下文的关系，而且需要考虑上文的关系，而基于 Bert 的 Multi-Head Attention（多头注意力）机制非常适合这种情况。Graphcodebert 基于 Bert，不仅增加了更适用于程序语言的学习能力，而且增加了程序语言和 DFG 之间依赖关系的学习能力。Graphcodebert 将序列 S 的向量 X^0 作为输入，然后经过 N 层（$N=12$）Transformer 将向量 X^0 不断进行 Multi-Head Attention 操作来获取注意力关系表示。$X^n=transformer_n(X^{n-1})$，$n\in[1,N]$，每层 Transformer 都包括一个 Multi-Head Attention 和一个 Feed Forward 部分，如式(1)、式(2)所示：

$$G^n=LN(MATT(X^{n-1})+X^{n-1}) \tag{1}$$

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

其中，X^{n-1} 是第 $n-1$ 层 $Transformer$ 处理后的向量表示，$MATT$ 表示 Multi-Head Attention 操作，LN 表示进行标准化操作，G^n 表示向量 X^{n-1} 在经过第 n 层 Multi-Head Attention 操作后的向量表示，X^n 表示在第 n 层将 X^{n-1} 处理后的向量表示。第 n 层的 Multi-Head Attention 的输出 $\hat{G}^n$ 的计算式如下：

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

$$head_i=softmax(\frac{Q_iK_i^{\mathrm{T}}}{\sqrt{d_k}}+M)V_i \tag{4}$$

$$\hat{G}^n=[head_1;\cdots;head_u]W_n^O \tag{5}$$

上一层的输出 $X^{n-1}\in\mathbb{R}^{|S|\times d_h}$ 分别使用 $W_i^Q,W_i^K,W_i^V\in\mathbb{R}^{d_h\times d_k}$ 线性映射到 Q_i,K_i,V_i，即 $queries,keys,values$ 的矩阵表示。u 是多头注意力中 head 的数量，d_k 是一个 head 的维度，$W_n^O\in\mathbb{R}^{d_h\times d_h}$ 是模型的参数。$M\in\mathbb{R}^{|S|\times|S|}$ 是 Mask 矩阵，表示序列 S 之间以图为基础的关系矩阵。如果序列 S 中第 i 个序列和第 j 个序列之间存在一条数据流边，或者第 i 个序列和第 j 个序列是同一节点，或者第 i 个序列和第 j 个序列在代码序列和变量序列之间存在一条边，或者是特殊的序列符号[CLS]或[SEP]，则 $M_{ij}=0$，否则 $M_{ij}=-\infty$，计算式如下：

$$M_{ij}=\begin{cases}0, & \text{if } q_i\in\{[CLS],[SEP]\} \text{ or } q_i,k_i\in C\\ & \text{or}\langle q_i,k_i\rangle\in E\cup E'\\ -\infty, & \text{otherwise}\end{cases} \tag{6}$$

其中，q_i 和 k_i 表示第 i 个序列的 $query$ 和 key，E 为智能合约数据流图的边，E 表示源代码序列和数据流图中变量节点序列之间的关系，如果变量序列中的 v_i 由代码序列中的 c_j 经过赋值或计算而来，则 $\langle v_i,c_j\rangle/\langle c_j,v_i\rangle\in E'$，PTM 会学习这种关系。

经过预训练网络后，序列 S 中的每个序列都考虑了序列本身和上下文的其他序列，然后提取预训练网络输出向量 X^n 中的[CLS]部分特征向量 $F_{cls}=\{[CLS]|[CLS]\in X^n\}$。在经过多层的 Multi-Head Attention 后，该向量融合了所有序列的特征信息，对分类任务具有较好的效果。

3.5　专家知识网络

如图 2 左侧上方所示，专家知识网络是网络模型的第二部分。传统的智能合约漏洞检测方法大多与专家知识相关，专家知识是根据漏洞的特征来定义漏洞存在的一系列规则，如果满足专家知识的规则，则存在特定的漏洞。Oyente 使用符号执行的漏洞检测，将包含 call.value 的执行路径提取出来，然后检测路径中是否有更新变量或存储变量的相关操作，如果存在向 Storage 区域写入或更新变量，则存在重入漏洞。sGuard[22] 也是使用符号执行的方式，引入控制依赖和数据依赖分析方法，判断操作码是否是某个特殊操作码，如果操作码是 SSTORE 且存在依赖关系，则可能存在重入漏洞。ContractFuzzer 通过判断是否包含 call 调用和是否满足相应的值要求来确定合约是否具有重入漏洞。通过分析，在智能合约源代码中，针对重入漏洞的专家知识可以概括为：

（1）具有 call.value 关键字。

（2）call.value 后面涉及到变量的存储或更新，一般是合约中 balance 的更新。

(3)在进行 transfer 操作前,是否包含 balance 的相关检查,如检查 balance 是否充足,这是 CGE[23] 提出的一个子模式。

通过使用 CGE 的方法对智能合约源代码进行分析,提取出特定漏洞的专家知识,然后使用 one-hot 向量编码将提取的专家知识转换为向量表示,再将专家知识向量经过前馈神经网络、卷积和池化提取出专家知识的高维特征 F_{ek}。

得到 F_{cls} 和 F_{ek} 两个特征向量之后,将 F_{cls} 和 F_{ek} 融合在一起得到融合特征 F_{vul},然后将 F_{vul} 经过全连接网络层和 sigmod 层得到最终的漏洞标签 $\hat{y}$,其中 0 表示没有重入漏洞,1 表示有重入漏洞,计算式如下:

$$F_{vul}=F_{cls}\oplus F_{ep} \tag{7}$$

$$\hat{y}=\text{sigmod}(FC(F_{vul})) \tag{8}$$

4 实验评估

本节使用部署在以太坊上的真实智能合约进行模型评估,首先介绍实验使用的数据集和环境,然后与其他的检测方法进行对比实验和消融实验,最后进行实验结果分析。

4.1 数据集及实验环境

本文使用了 Peculiar 的数据集,这个数据集是在 SmartBugs Wild 数据集的基础上经过处理得到的,包含一个大数据集 dataset-wild 和一个小数据集 dataset-vul。dataset-wild 数据集中包含 203716 份智能合约源代码数据及相应的漏洞标签,dataset-vul 数据集中包含 1668 分智能合约源代码数据及相应的漏洞标签。与 dataset-wild 数据集不同的是,dataset-vul 数据集中智能合约的源代码中都含有能够触发重入漏洞的 call.value 关键字,这个数据集更能够评估模型的漏洞检测能力。其中,包含 1197 份含有重入漏洞的智能合约源代码数据和 471 份不含有重入漏洞的智能合约源代码数据。

实验的设备配置为 Ubuntu18.04 系统、Intel Xeon Gold 5218R 处理器、32GB 内存以及 1 张 NVIDIA V100 GPU。与 Peculiar 一致,在智能合约数据集中随机选择 20%的合约数据作为训练集,将剩余 80%的合约数据作为测试集。

4.2 评估指标

本文使用平均 Precision、Recall 和 F1-score 作为实验的评估指标,每个指标的值是通过 TP(真阳性)、FP(假阳性)、TN(真阴性)和 FN(假阴性)计算得到的。Precision,Recall 和 F1-score 的计算式如下:

$$Precision=\frac{TP}{TP+FP} \tag{9}$$

$$Recall=\frac{TP}{TP+FN} \tag{10}$$

$$F1\text{-}score=2*\frac{Precision*Recall}{Precision+Recall} \tag{11}$$

4.3 实验结果

将本文提出的方法与其他 11 种检测方法在 dataset-wild 数据集上进行实验对比,包括 8 种传统的检测方法(Oyente, Osiris, Mythril, Manticore, Securify, Slither, SmartCheck 和 Honeybadger[24]),2 种基于深度学习的方法(DR-GCN, TMP[25])以及 1 种基于预训练模型方法 Peculiar。在 Peculiar 的基础上,通过进一步实验得到指标对比结果,如表 1 所列。从表 1 的统计可以看出,Peculiar 的检测效果是目前较好的,而本文方法相比 Peculiar 有一定的提升效果。本文方法的 Precision,Recall 和 F1-score 分别达到了 96.2%,97.7%和 96.9%,比 Peculiar 分别提高了 4.4%,5.3%和 4.8%。可以看出,本文方法能够达到非常低的误报率和漏报率。这表明融合了预训练技术和专家知识的重入漏洞检测方法在大量的数据集上有着较好的检测效果。

表 1 在 dataset-wild 数据集上的性能比较

Table 1 Performance comparison on dataset-wild

(单位:%)

Method	Reentrancy		
	Recall	*Precision*	*F1*
Honeybadger	50.5	87.2	50.9
Manticore	50.0	49.7	49.9
Mythril	51.7	50.2	49.7
Osiris	53.8	59.0	55.3
Oyente	54.1	65.6	56.4
Securify	54.8	52.6	53.4
Slither	65.4	52.0	52.6
SmartCheck	70.5	79.4	74.1
DR-GCN	80.9	72.4	76.4
TMP	82.6	74.1	78.1
Peculiar	92.4	91.8	92.1
Our Method	**97.7**	**96.2**	**96.9**

为了进一步测试不同方法的重入漏洞检测能力,将本文方法与其他 9 种方法(Oyente, Osiris, Mythril, Manticore, Securify, Slither, SmartCheck, Peculiar 和 Honeybadger)在 dataset-vul 数据集上进行实验对比,实验结果如表 2 所列。

表 2 在 dataset-vul 数据集上的性能比较

Table 2 Performance comparison on dataset-vul

(单位:%)

Method	Reentrancy		
	Recall	*Precision*	*F1*
Honeybadger	50.5	64.7	23.8
Manticore	49.9	14.6	22.5
Mythril	44.0	37.4	26.4
Osiris	46.3	41.5	27.6
Oyente	47.3	43.7	28.4
Securify	52.0	55.0	32.2
Slither	61.0	61.0	50.3
SmartCheck	54.2	53.6	48.3
Peculiar	82.6	84.1	83.3
Our Method	**88.3**	**90.3**	**89.2**

从表 2 可以看出,传统的 8 种方法(Oyente, Osiris, Mythril, Manticore, Securify, Slither, SmartCheck 和 Honeybadger)的各项指标非常低,平均 F1-score 仅有 32.4%,这表明传统的检测方法在更有针对性的数据集上的检测能力较弱,误报率和漏报率较高。这是因为传统的检测方法判断方式单一,不能很好地判断合约中有 call.value 关键字却不一定会有重入漏洞的情况,并且没有考虑代码之间的数据流关系,因此不能理解合约上下文的依赖关系。而本文方法在 dataset-vul 数据集上的 Precision,Recall 和 F1-score 分别达到了 90.3%,88.3%和 89.2%。与 Peculiar 方法相比,Precision,Recall 和 F1-score 分别提升了 6.2%,5.7%和 5.9%。这表明,正是因为使用了预训练技术、专家知识和合约切片,所以使得本文方法考虑了上下文语义关系和变量之间的依赖关系,能够很好地判断合约中有 call.value 关键字却不一定会有重入漏洞的情况。因此,本文方法在具有针对性的数据集上有更好的检测能力。

为了比较数据预处理模块和专家知识模块对模型检测能力的贡献效果,进行了消融实验对比。首先对仅有专家知识模块的模型进行实验,然后对包含专家知识模块和合约切片数据预处理模块的模型进行实验。本文以 Peculiar 为实验的基线,实验结果如表 3 和表 4 所列。

表 3 在 dataset-wild 数据集上消融研究的结果

Table 3 Results of ablation study on dataset-wild

(单位:%)

Method	Reentrancy		
	Recall	*Precision*	*F*1
Peculiar	92.4	91.8	92.1
Peculiar+ek	96.2	95.0	95.6
Peculiar+ek+cs	**97.7**	**96.2**	**96.9**

表 4 在 dataset-vul 数据集上消融研究的结果

Table 4 Results of ablation study on dataset-vul dataset

(单位:%)

Method	Reentrancy		
	Recall	*Precision*	*F*1
Peculiar	82.6	84.1	83.3
Peculiar+ek	84.0	85.4	84.6
Peculiar+ek+cs	**88.3**	**90.3**	**89.2**

与基线相比,仅有专家知识(ek)模块的模型在 dataset-wild 数据集上的 Precision,Recall 和 F1-score 分别提升了 3.2%,3.8%和 3.5%,在 dataset-vul 数据集上的 Precision,Recall 和 F1-score 分别提升了 1.3%,1.4%和 1.3%。这表明专家知识能够进一步降低漏洞检测的误报率和漏报率,提升模型的检测效果。而包含专家知识(ek)模块和合约切片数据预处理(cs)模块的模型在 dataset-wild 数据集上的 Precision,Recall 和 F1-score 分别提升了 4.4%,5.3%和 4.8%,在 dataset-vul 数据集上的 Precision,Recall 和 F1-score 分别提升了 6.2%,5.7%和 5.9%。这表明使用合约切片的方法不仅减少了数据序列的长度,而且减少了无关数据对模型的干扰。可以得出,在预训练技术的基础上,进一步融合专家知识和合约切片能够更有效地提升模型的重入漏洞检测效果。

5 相关工作

早期工作一般使用传统方法检测智能合约的漏洞,Oyente,Osiris,Mythril,Maian,Manticore 使用符号执行的方式进行漏洞检测,在合约字节码级别上进行工作,使用符号值代替具体值的方式,能够标识出含有漏洞的字节码位置信息。Securify 和 Zeus 使用形式化验证方法通过建模、推理和证明来实现漏洞检测。Slither 和 SmartCheck 提出使用静态分析的方式进行漏洞检测。ContractFuzzer 提出使用模糊测试的方式进行漏洞检测,该方式能够构造大量非预期的输入数据并监视是否有漏洞发生的情况。

近年来,机器学习和深度学习发展迅速,在各个领域得到了广泛应用,一些学者提出了机器学习相关方法。Xing 等[26]提出使用字节码切片矩阵提取漏洞特征,使用机器学习的方法来检测智能合约是否有短地址攻击、贪婪合约和算术溢出漏洞。Tann 等[27]使用 LSTM 的方法来实现合约漏洞的序列学习,能够检测合约是否具有自杀漏洞、挥霍漏洞和贪婪漏洞。还有一些学者提出了使用深度学习来进行漏洞检测的方法,NarayanaL 等[28]提出使用自动编码器的方法来检测漏洞,Zhuang 等提出使用图神经网络的检测方法,将合约中的控制流和数据流语义转化为图结构然后进行特征学习。Detect[29]和 Peculiar 利用在 NLP 中广泛应用的预训练技术来检测漏洞,在引入了注意力机制的情况下,进一步提升了模型的泛化能力。

结束语 本文提出了一种基于预训练技术和专家知识的智能合约重入漏洞检测方法,与已有的工作相比,本文方法充分利用了预训练技术的优势,融合传统的专家知识,进一步降低了漏洞检测的误报率和漏报率,并使用关键语法语义进行合约切片,有效降低了合约中与跟漏洞无关的数据对模型的影响。实验结果表明,本文方法的 *Precision*,*Recall* 和 *F*1-*score* 分别达到了 96.2%,97.7%和 96.9%。本文是首次提出将预训练技术与传统的专家知识融合进行智能合约漏洞检测。目前仅考虑了重入漏洞的检测,在未来,将进一步提升模型的漏洞检测性能,同时整理其他类型漏洞的数据集,进一步研究其他类型漏洞的检测方法。

参 考 文 献

[1] MEHAR M I,SHIER C L,GIAMBATTISTA A,et al. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack[J]. Journal of Cases on Information Technology(JCIT),2019,21(1):19-32.

[2] SLOWMIST HACKED[OL]. https://hacked.slowmi-st.io/en/

[3] 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. New York: Association for Computing Machinery,2016:254-269.

[4] 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: Association for Computing Machinery,2018:664-676.

[5] MUELLER B,HONIG J,PARASARAM N,et al. ConsenSys/mythril [OL]. https://github.com/ConsenSys/mythril.

[6] 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. New York: Association for Computing Machinery,2018:653-663.

[7] MOSSBERG M,MANZANO F,HENNENFENT E,et al. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts[C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering(ASE). New York: IEEE Press,2019:1186-1189.

[8] 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: Association for Computing Machinery,2018:67-82.

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

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

[11] 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.

[12] JIANG B, LIU Y, CHAN W K. Contractfuzzer: Fuzzing smart contracts for vulnerability detection[C]// 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). New York: IEEE Press, 2018: 259-269.

[13] FERREIRA J F, CRUZ P, DURIEUX T, et al. SmartBugs: a framework to analyze solidity smart contracts[C]// Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York: Association for Computing Machinery, 2020: 1349-1352.

[14] DANNEN C. Solidity programming[M]// Introducing Ethereum and Solidity. Apress, Berkeley, CA, 2017: 69-88.

[15] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv: 1810. 04805, 2018.

[16] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training [OL]. https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsuper-vised/language_understanding_paper. pdf.

[17] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017, 30.

[18] FENG Z, GUO D, TANG D, et al. Codebert: A pre-trained model for programming and natural languages[J]. arXiv: 2002. 08155, 2020.

[19] GUO D, REN S, LU S, et al. Graphcodebert: Pre-training code representations with data flow[J]. arXiv: 2009. 08366, 2020.

[20] LI Z, ZOU D, XU S, et al. SySeVR: A framework for using deep learning to detect software vulnerabilities [J]. arXiv: 1807. 06756, 2021.

[21] WU H, ZHANG Z, WANG S, et al. Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques[C]// 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). New York: IEEE Press, 378-389.

[22] NGUYEN T D, PHAM L H, SUN J. sGUARD: Towards Fixing Vulnerable Smart Contracts Automatically[J]. arXiv: 2101. 01917, 2021.

[23] LIU Z, QIAN P, WANG X, et al. Combining graph neural networks with expert knowledge for smart contract vulnerability detection[J]. arXiv: 2107. 11598, 2021.

[24] TORRES C F, STEICHEN M. The art of the scam: Demystifying honeypots in ethereum smart contracts[C]// 28th {USENIX} security symposium({USENIX} security 19). Santa Clara: USENIX Association, 2019: 1591-1607.

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

[26] XING C, CHEN Z, CHEN L, et al. A new scheme of vulnerability analysis in smart contract with machine learning[J]. Wireless Networks, 2020: 1-10.

[27] TANN W J W, HAN X J, GUPTA S S, et al. Towards safer smart contracts: A sequence learning approach to detecting security threats[J]. arXiv: 1811. 06632, 2018.

[28] NARAYANA K L, SATHIYAMURTHY K. Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning[OL]. https://www. sciencedirect. com/science/article/pii/S2214785321030273.

[29] JEON S, LEE G, KIM H, et al. SmartConDetect: Highly Accurate Smart Contract Code Vulnerability Detection Mechanism using BERT [OL]. https://seclab. skku. edu/wp-content/uploads/2021/08/PLP _ 7 _ SmartConDetect _-Highly-Accurate-Smart-Contract-Code-Vulnerability-Detection-Mechanism-using-BERT-Sowon-Jeon. pdf.

CHEN Qiao-song, born in 1978, Ph. D, associate professor, is a member of China Computer Federation. His main research interests include blockchain, data mining and deep learning.