基于信息融合的区块链系统隐匿安全补丁识别及迁移技术

郝偲成[1,2],魏桂鹏[1,2],肖煜铭[1,2],南雨宏[1,2],郑沛霖[1,2*],郑子彬[1,2]

(1. 中山大学软件工程学院,广东珠海 519082;2. 广东省区块链工程技术研究中心,广东珠海 519082)

摘 要: 区块链是集成了密码学、智能合约的新型分布式系统,在金融交易、版权保护等领域已取得广泛应用。目前,全球共计运行着上千条不同的链,为了兼容性和便利性,许多开发者通过分叉或复用主流区块链系统的代码进行再开发。然而,这也导致安全缺陷快速传播。与此同时,隐匿安全补丁是指开源项目中未公开披露于漏洞数据库中的安全修复。当前,区块链系统项目中的安全补丁透明度不足,存在大量隐匿安全补丁,进一步加剧了下游软件系统的修复延迟,降低了整个区块链系统生态的可靠性。因此,亟需针对涵盖多种编程语言的区块链系统生态,设计自动化的隐匿安全补丁识别方法,及时发现和修复潜在的已知安全问题。为此,本文提出 BlockPatch,首个面向区块链系统生态的通用隐匿安全补丁识别和迁移框架,通过大语言模型(Large Language Model,LLM)将多模态的变更信息进行融合,实现了多语言区块链系统安全补丁的准确识别。具体来说,BlockPatch 以代码提交为输入,提取提交文本描述、变更代码块和抽象语法树(Abstract Syntax Tree,AST)编辑行为,得到多模态的变更表示,以捕获细粒度的变更内容和过程;随后利用大语言模型的强大表征能力对三类信息进行语义嵌入,并结合神经网络实现特征融合和学习,以提升对安全补丁的识别能力。为了验证方法的有效性,本文构建了包含主流公有链和联盟链项目的补丁数据集。实验结果表明,BlockPatch 能够取得 94.02% 的精确率、94.58% 的召回率和 94.29% 的 F1 值,在 F1 分数上较现有先进方法提升了 5.03 个百分点,并在不同类型的安全补丁识别上均取得了良好效果。消融实验进一步证明了多模态信息融合的有效性。最后,BlockPatch 将识别到的安全补丁迁移至下游区块链系统中进行安全检查。基于近期比特币和以太坊仓库的代码提交,BlockPatch 识别到 16 个隐匿安全补丁,并在下游项目中检测到了 28 个未修复的安全漏洞,突出了识别并运用隐匿安全补丁的重要性。

关键词: 区块链系统;安全补丁识别;大语言模型;补丁信息融合;多模态学习;补丁迁移

基金项目: 国家自然科学基金(No.62572497,No.62302538);国家自然科学基金委员会与香港研究资助局合作研究重点项目(No.62461160332);广东省自然科学基金(No.2025A1515010279);广东珠江人才计划(No.2023QN10X561)

中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112(2026)02-0734-16

电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20251003

Identification and Migration of Silent Security Patches in Blockchain Systems via Information Fusion

HAO Sicheng[1,2], WEI Guipeng[1,2], XIAO Yuming[1,2], NAN Yuhong[1,2], ZHENG Peilin[1,2*], ZHENG Zibin[1,2]

(1. *School of Software Engineering*, *Sun Yat-sen University*, *Zhuhai*, *Guangdong* 519082, *China*;
2. *Guangdong Engineering Technology Research Center of Blockchain*, *Zhuhai*, *Guangdong* 519082, *China*)

Abstract: Blockchain is a novel distributed system integrating cryptography and smart contracts. It has been widely applied in fields such as financial transactions and copyright protection. Currently, thousands of different blockchains are running worldwide. For compatibility and convenience, many developers fork or reuse the open-source code of mainstream blockchains for further development. However, such a practice also leads to the rapid propagation of security vulnerabilities. Meanwhile, silent security patches refer to security fixes in open-source projects that are not publicly disclosed in vulnerability databases. At present, the transparency of security patches in blockchain projects is insufficient, and there are a large number of silent security patches, further exacerbating the remediation delays in downstream software systems and reducing the reliability of the entire blockchain ecosystem. Therefore, it is urgent to design an automated method for identifying silent security patches targeting the blockchain ecosystem covering multiple programming languages, to promptly detect and fix potential known security issues. To this end, this paper proposes BlockPatch, the first framework for identifying and mi-

收稿日期:2026-01-28;录用日期:2026-02-09;责任编委:崔来中

*通信作者:郑沛霖

grating general silent security patches in the blockchain ecosystem. By fusing multimodal change information through the large language model (LLM), it achieves accurate identification of security patches in multilingual blockchain systems. Specifically, taking code commits as input, BlockPatch extracts commit messages, modified code blocks, and abstract syntax tree (AST) edit actions to obtain multimodal change representations, so as to capture fine-grained change contents and processes. Subsequently, it utilizes the advanced representation capabilities of LLMs to perform semantic embedding on the three types of information and combines neural networks to achieve feature fusion and learning, thereby enhancing the identification capability of security patches. To verify the effectiveness of the method, this paper constructs a patch dataset containing mainstream public and consortium blockchain projects. The experiments show that BlockPatch can achieve a precision of 94.02%, a recall of 94.58%, and an F1-score of 94.29%, outperforming existing state-of-the-art methods by 5.03 percentage points on the F1-score, and achieves good results in identifying different types of security patches. The ablation study further demonstrates the effectiveness of multimodal information fusion. Finally, BlockPatch migrates the identified security patches to downstream blockchain systems for security checks. Based on recent code commits from Bitcoin and Ethereum repositories, BlockPatch identified 16 silent security patches and discovered 28 unpatched security vulnerabilities in downstream projects, highlighting the importance of identifying and applying silent security patches.

Keywords: blockchain system; security patch identification; large language model; patch information fusion; multimodal learning; patch migration

Foundation Item(s): National Natural Science Foundation of China (No.62572497, No.62302538); NSFC-RGC Collaborative Research (No. 62461160332); General Program of the Natural Science Foundation of Guangdong Province (No.2025A1515010279); Guangdong Zhujiang Talent Program (No.2023QN10X561)

0　引言

区块链和去中心化应用在近几年取得了蓬勃发展。作为一种新型分布式系统,区块链可以被视作集成了智能合约的去中心化账本,凭借其数据透明和不可篡改的特性,已在金融交易、版权保护等领域得到广泛应用[1-2]。当前,全球共计运行着超过1 000条不同的区块链[3]。然而由于区块链系统包含共识协议、加密算法和智能合约等一系列的复杂组件,从头实现一条完整的区块链难度较高,且难以兼容现有的工具链。因此,开发者常常会复用开源区块链的组件或实现逻辑以降低开发成本,但这同时也带来了安全漏洞传播的风险。同时,作为去中心化系统,为了避免单一系统故障,业界开发了多种编程语言的客户端和共识机制。例如,以太坊生态中同时存在Go语言(例如Go-ethereum、Erigon)、C#(Nethermind)和Java(Besu)等多语言实现的客户端。这种多样性进一步增加了理解和管理区块链系统安全缺陷的难度,尤其是隐匿安全补丁,即那些在开源软件项目未公开披露的漏洞修复[4]。

在当前的开源软件生态下,代码分叉和复用已成为推动软件快速迭代的重要机制。然而,由于时间和管理成本等因素,开发者可能不向漏洞披露平台进行报告,导致安全补丁未及时进行公开披露,形成隐匿安全补丁[5],造成软件更新的延迟,给攻击者提供了可乘之机。与Linux和Android等主流系统生态相比,区块链生态的漏洞管理体系仍显不足,缺乏统一的缺陷补丁管理,大量安全漏洞及其补丁未被提交至CVE(Common Vulnerabilities and Exposures)[6]等标准化漏洞数据库进行归档[7-8],其安全风险未能得到充分披露和传播,客观上扩大了区块链生态的攻击面。以以太坊Go-ethereum为例,截至2025年7月,Go-ethereum仓库已有超过15 000个代码提交(Commit),然而在仓库的安全说明中仅含有12个公开披露于漏洞库中的记录,存在大量未披露的安全修复。如图1所示,代码拉取请求(Pull Request,PR)#30157修复了关于区块链ID检查不足的问题,该补丁确保在读取chainID时不会意外修改原始签名数据。项目维护者评价其为一个极其可怕的漏洞。然而,该补丁并没有收录进漏洞数据库,其提交描述(Commit Message)也未显式包含漏洞关键词,因此不利于下游区块链及时修复。例如,从关联记录分析,Astria分叉链在次年5月才将这一问题进行修复。因此,亟需针对区块链生态的隐匿安全补丁进行自动化识别,并及时将其应用在相似或分叉的项目中,以保障系统的健壮性和安全性。

在隐匿安全补丁识别上,现有工作大多面向单一语言生态中的主流传统软件,例如Linux等C/C++中的大型软件[9-14],尚未针对区块链系统生态开展研究,缺乏相关的领域知识,也就难以理解和识别区块链系统中的特有漏洞补丁。从补丁的特征提取上看,由于多语言区块链客户端的广泛存在,针对特定编程语言设计的方法难以在区块链生态上进行适配。例如,GraphSPD[14]针对C/C++语言设计了补丁代码属性

图的提取机制，迁移到新编程语言上需要较高成本，无法适应区块链生态的多语言特性。除此之外，SPI[10]、PatchRNN[11]等现有方法直接采用变更前后的源代码和补丁文本描述，忽略了代码的语法特征及变更的编辑过程，无法捕获具体的代码变更细节和变更前后的关联性。例如在图1中，该代码提交直观上删除了2行代码（4~5行），并增加了2行代码（6~7行），但通过分析语法树的编辑行为，其整体结构并未发生变化，而是通过创建拷贝避免了对传入指针参数v的直接修改。从特征建模上看，以Word2Vec[15]为代表的经典文本表示方法受限于词级表征，难以有效捕捉句子层级的语义关联。面对文本和代码等不同模态信息，已有研究多采用分开处理[10]或直接特征拼接[13]等粗粒度的处理策略，这种方式忽略了提交描述文本和变更代码之间的上下文关联性，使得结果容易受到单一信息的影响。因此，在区块链生态中仅有少量安全补丁的情况下，隐匿安全补丁识别需要提取更细粒度的代码变更特征，并将不同语言的安全补丁综合利用。

core/types: don't modify signature V when reading large chainID

```
1  // deriveChainId derives the chain id from the given v parameter
2  func deriveChainId(v *big.Int) *big.Int {
3      if v.BitLen() <= 64 {...}
4  -   v.Sub(v, big.NewInt(35))
5  -   return v.Rsh(v, 1)
6  +   vCopy := new(big.Int).Sub(v, big.NewInt(35))
7  +   return vCopy.Rsh(vCopy, 1)
8  }
```

图1 Go-ethereum PR #30157对应的代码提交

Figure 1 Code commit corresponding to Go-ethereum PR #30157

针对以上问题，本文将代码提交的描述信息、细粒度的代码变更和语法树编辑信息相融合，提出了BlockPatch，首个面向多语言区块链系统生态的隐匿安全补丁识别和迁移框架。具体来说，BlockPatch以代码仓库中的提交作为输入，判断其是否为安全补丁，并应用于下游项目中。为了提供更加丰富的代码变更语义信息，本文设计了细粒度的代码差异表征机制，既包括对变更内容的文本描述，又通过行级差异捕捉具体修改代码，同时借助抽象语法树比对获取编辑行为信息。然后，BlockPatch利用大语言模型强大的跨语言表征能力，对多种变更信息进行联合嵌入，通过统一向量空间实现语义对齐。最后，BlockPatch通过多模态学习机制，将文本描述与代码特征通过深度神经网络进行融合，以提升安全补丁的识别准确率。对于检测到的潜在安全补丁，本文采用静态的启发式分析方法自动构建标准化检测输入，实现对下游区块链系统的安全扫描。为了验证本文提出方法的有效性，我们针对三个主流的区块链项目，构建了包含2 450个安全补丁提交的数据集。在隐匿安全补丁识别上，BlockPatch取得了94.02%的精确率、94.58%的召回率和94.29%的F1值，优于现有的相关先进方法。同时，通过消融实验，本文进一步验证了引入语法树编辑信息和多模态语义融合的有效性。最后，本文在Bitcoin和Go-ethereum的最近600个提交上应用了本方法，识别出了16个隐匿安全补丁，并在下游区块链项目中检测到了28个关联的未修复漏洞。进一步地，在本文实验分析时，下游区块链系统中仅存在3个修复实例，平均修复延迟超30天。这一结果突出了亟需针对区块链生态的隐匿安全补丁进行识别，并及时应用在下游系统中。

综上所述，本文的主要贡献如下：

（1）本文提出并实现了首个面向区块链系统的通用隐匿安全补丁识别方法，BlockPatch，它创新性地融合了文本描述、代码变更和语法树编辑信息，实现了在多语言区块链系统生态中识别隐匿安全补丁。

（2）本文从三个主流的区块链项目中构建了包含2 450个安全补丁的代码提交数据集，实验证明BlockPatch的识别能力超过现有工作，取得了94.02%的精确率、94.58%的召回率和94.29%的F1值。

（3）本文在Bitcoin和Go-ethereum上最近的600个代码提交上进行隐匿安全补丁分析，并在比特币和以太坊生态的下游项目中共计发现了28个未修复的漏洞，反映出隐匿安全补丁修复的滞后性。

（4）本文公开BlockPatch的相关数据集和源代码，以促进开展后续研究[16]。

1 相关工作

本文相关的研究工作主要包含两部分：（1）隐匿安全补丁识别技术；（2）区块链系统漏洞分析和检测。

1.1 隐匿安全补丁识别

隐匿安全补丁的识别技术用于判定代码变更是否用于安全缺陷修复，现有的研究主要面向单一语言生态开展，具体的识别方法可以分为基于规则和学习的两种模式。

基于规则的方法具有较好的可解释性，但其适用范围相对有限。这些方法通过构建具体的安全修复规则来发现隐匿补丁。例如，DAA[17]方法利用现成的静态分析工具，通过比较软件项目的连续版本，识别安全警报的消除情况，以检测漏洞修复。Early-VulnFix[18]关注污点漏洞修复，通过静态程序分析构建数据流和控制流模型，检测新增代码是否通过显式或隐式路径影响敏感操作，从而判断是否为漏洞修复。然而，这类方法需要针对具体类型的漏洞设

计相应的检测规则,尤其在安全检测工具仅覆盖少量漏洞类型的区块链生态中,难以用于通用的隐匿安全补丁识别。

由于补丁信息语义的多样性和复杂性,大量工作采用机器学习和深度学习方式进行识别,通过学习补丁的特征来判断其是否安全相关。周鹏等人[9]采用半监督的协同训练方法,从补丁的代码和描述中提取特征,训练基于支持向量机(Support Vector Machine,SVM)的分类器。VulFixMiner[19]仅分析代码变更内容,通过对CodeBERT进行微调来识别漏洞修复。Wang等人[11]开发了PatchRNN,利用循环神经网络(Recurrent Neural Network,RNN)分析代码变更和文本描述。Zhou等人[10]则使用一种结合了长短期记忆神经网络(Long Short-Term Memory network,LSTM)和卷积神经网络(Convolutional Neural Network,CNN)的混合模型。除此以外,唐建平等人[13]开发的PatchInsight,通过BERT和CodeBERT编码器提取代码变更和提交日志特征,结合双向门控循环单元(Gated Recurrent Unit,GRU)和注意力机制实现安全补丁识别。此外,为了进一步增强特征的表示能力,Wang等人[14]开发了GraphSPD,通过合并提交前后的代码属性图进行语义增强,数值化后输入图卷积神经网络(Graph Convolutional Network,GCN)和多层感知器(MultiLayer Perceptron,MLP)识别安全补丁。类似地,Han等人[12]开发了GRAPE,在构建代码属性图基础上,使用Word2Vec对节点进行嵌入,使用统计特征对边进行嵌入,共同输入图卷积神经网络进行学习。然而,在变更信息提取上,现有的方法仅采用了变更前后的代码信息,忽略了基于代码结构的语法树编辑过程等细节,使得模型对具体的变更语义表达不足。除此之外,由于区块链系统生态多语言的特性,基于复杂代码属性图等语言强相关的方法需要大量工作进行适配,且图操作具有较高的复杂度,存在计算效率低和跨语言兼容性不足的问题。

1.2　区块链系统漏洞分析

随着区块链系统在金融等领域的广泛应用,确保系统的安全性也变得愈发重要。针对区块链系统的漏洞类型,Yi等人[8]通过关键词对Bitcoin、Ethereum、Monero与Stellar 4个主流区块链系统的开源仓库进行人工分析和实证研究,总结了21种漏洞补丁的代码模式,分析了该领域的漏洞特征,提供了丰富的漏洞数据。除此之外,刘敖迪等人[20]针对区块链系统安全,从网络传输层、共识协议层和智能合约应用层3个层面,系统梳理了区块链安全技术的研究进展。基于这些实证分析,研究者们针对特定漏洞类型和历史漏洞开展了多样的安全分析。

在区块链特定漏洞检测上,研究者们针对多种区块链系统开展了多方面研究。在共识算法缺陷研究方面,Zhang等人[21]针对采用工作量证明的区块链,对其质量和抗攻击能力进行定量分析,Yang等人[22]通过多交易差分模糊测试来检测以太坊的共识协议漏洞。对于远程调用接口(Remote Procedure Call,RPC)的实现一致性,Kim等人[23]研究了不同区块链节点在提供RPC服务时的差异性,揭示了同一平台下,不同代码实现的节点在处理相同RPC请求时出现的不一致性对系统稳定性的影响。在交易执行机制上,Chord[24]通过构造冲突交易和专用检测器,识别区块链交易并行机制中的冲突处理缺陷。

在已知漏洞的传播检测上,一些研究者关注比特币及其分叉项目的补丁延迟问题。针对已知漏洞在区块链中的传播问题,CoinWatch[25]通过Simian克隆检测器对比特币项目的CVE信息及历史分叉项目的延迟进行分析。BlockScope[26]则基于比特币和以太坊的已公开漏洞,检测区块链生态中相关漏洞的修复状态。然而,由于区块链系统生态公开披露的安全漏洞信息有限,现有的漏洞传播检测仅能在很小的范围上开展,严重限制了安全检测和修复的及时性。因此,与已有的区块链安全研究不同,本文并不聚焦于单一区块链系统或漏洞类型,而是关注区块链系统多语言生态中的隐匿漏洞传播问题,核心是自动化识别区块链系统的隐匿安全补丁,并及时将其应用到下游系统的安全检测任务。

2　基于变更信息融合的安全补丁识别和迁移技术

本文的分析对象为开源区块链代码仓库中的提交,每个提交携带哈希标识的代码变动,包含提交描述信息和文件变更记录。尽管对于托管在GitHub等平台的代码仓库还涉及开发者之间用于讨论和分支合并的Issue和PR,但由于本地仓库和部分提交无对应Issue和PR,因此,在本文中,BlockPatch仅以代码提交为输入,提取其中的代码变更信息,不包含其他额外的外部信息,以增强工具的泛用性,即仅需代码提交信息即可判断其是否安全相关。为了检验隐匿安全补丁延迟修复的严重性,本文针对识别出的安全补丁构建安全补丁信息,并在主流区块链及其分叉项目中判断对应安全漏洞是否得以修复或仍然存在。

图2展示了本文所提出方法的核心框架。BlockPatch融合了提交描述信息和多粒度的代码变更信息,即行级差异和语法树编辑信息,来表示和学习区块链安全补丁,并将识别出的安全补丁应用于下游系

统。具体来说，对于每个代码提交，除基本的文本描述信息外，BlockPatch 提取代码变更的多种表示方式，在行级差异表示的基础上，引入语法树编辑信息突出变更过程行为（2.1 节）。然后，利用大语言模型进行语义建模，BlockPatch 将文本描述、行级差异表示和代码编辑信息转换为可训练的特征向量（2.2 节）。接着通过神经网络模型进行融合和学习，识别区块链系统的隐匿安全补丁（2.3 节）。最后，BlockPatch 基于识别到的安全补丁构建安全补丁信息，并应用于相似和分叉区块链系统中（2.4 节）。

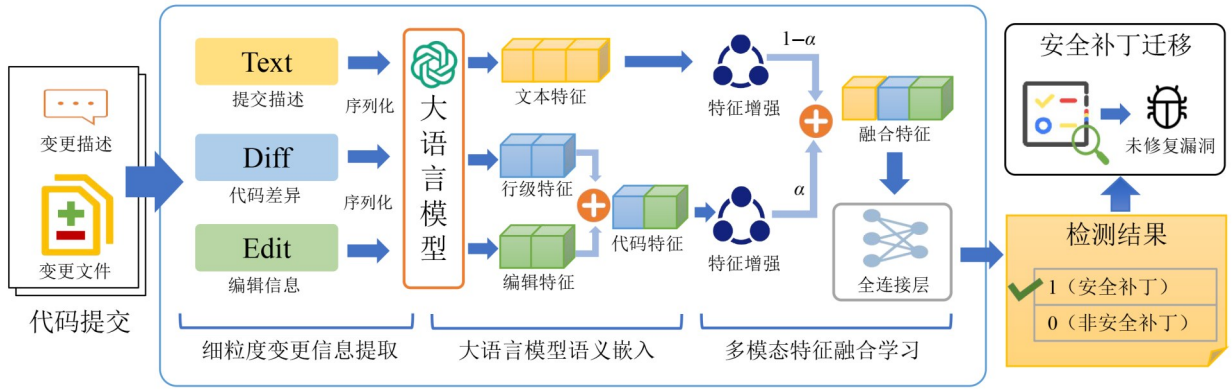


图 2 基于多模态代码变更信息融合的安全补丁识别和迁移框架

Figure 2 Framework for security patch identification and migration based on multimodal code change information fusion

2.1 代码提交信息提取

隐匿安全补丁识别的第一步是从代码提交中提取有效的信息。由于区块链系统安全补丁数量较少，本文充分挖掘现有安全补丁的多模态信息，在变更描述和变更代码块的基础上，引入语法树上的编辑信息，以反映代码的语法结构和变更过程。

代码提交包含的通用信息为提交描述文本和一系列文件中的变更代码块。这两类信息在代码托管平台和现有的安全补丁识别工作[10]中被广泛使用。然而，变更代码块通常按行进行比较，该信息仅反映了行文本的增减关系，缺乏代码结构信息和上下文，以及具体的修改过程。为此，本文引入语法树编辑操作，通过描述变更前后语法树节点的变化，来为安全补丁识别提供更加丰富的语义信息，更全面地刻画变更的行为和意图。具体来说，本文保留了文本模态的提交描述和代码模态的行级代码差异，同时通过对变更前后抽象语法树进行差异分析，获取语法树编辑序列。为此，本文基于软件仓库解析代码提交，提取描述文本、变更代码块和变更代码文件，并通过文本处理和语法树比较分析，得到三种模态（Text、Diff、Edit）的信息。三类信息的具体含义和表示方法如下，并通过辅以图 1 中的代码提交为例加以说明。

代码变更描述（Text）。代码变更描述是一段自然语言文本，用于说明所提交代码的作用和影响。由于本文的分析对象是代码仓库中的一系列提交，因此本文将提交信息作为变更描述。在示例中，代码变更描述为“core/types：don’t modify signature V when reading large chainID”。

行级差异表示（Diff）。行级差异计算通过按行对比两个文件内容生成差异信息，目前已在 Git 等版本控制系统广泛使用，可用于追踪文件的修改、生成补丁以及辅助代码审查。在本文中，我们计算代码提交前后文件的行级差异，并以统一差异的形式表示。例如，对于上述代码片段，其行级差异为第 6~9 行的内容，包含行开头的增减符号，以突出当前行的变化。相较于代码变更描述，具体的代码变更可能涉及多个代码片段。对于这种情况，BlockPatch 将多个描述进行合并。

语法树编辑信息（Edit）。语法树编辑信息指进行代码变更的操作序列。不同于纯文本的编辑操作，本文针对抽象语法树（AST）节点来计算代码的编辑信息，遵守相应语法规则的限制。由于不涉及语义分析，借助编程语言的语法分析器，可使代码编辑信息提取快速迁移到不同编程语言。例如，Gumtree[27]和 Diffsitter[28]设计了计算抽象语法树最小编辑序列的算法，仅需提供编程语言的语法信息即可高效生成编辑信息。对于提交前后的代码，编辑操作包括四类：增加、删除、更新、移动。操作的元素为抽象语法树上的树或节点。因此，编辑行为实际上是一系列对语法树节点的调整，包含了操作动作和修改的相关节点类型和位置。例如，对于在函数体中插入表达式的操作，一种常见的表示方法为{action：insert-tree，tree：expression［起始偏移，结束偏移］，parent：function_definition［起始偏移，结束偏移］}。

为了能够将多模态信息通过大语言模型进行统一嵌入，BlockPatch 需要将一系列的编辑操作表示为

序列形式。然而,上述示例中以偏移量表示节点位置的方式不利于模型理解具体内容,在不同编辑操作下格式也会发生改变。因此,为了便于模型处理和理解不同的编辑行为,本文采用统一的序列化表示格式,保留了编辑操作和节点类型,并将偏移量替换为对应的代码片段。最终,单个编辑行为的通用表示方式为"[ACTION]{action}:{src}[TO]{dst}"。具体来说,每一个编辑信息都以相同的标识开头,并衔接具体的编辑操作。{src}部分会描述在源文件上的修改的节点类型和值,而{dst}会指出修改节点在新文件中的值和位置。这种方式也为行级代码差异补充了修改片段所在语法树节点的上下文信息。对于删除操作,由于新文件中不存在对应值,仅保留描述的前半部分。将多个编辑操作拼接起来,则可得到序列化的语法树编辑信息。对于图1中的示例,其序列化后的编辑信息如图3所示。

```
[ACTION]insert-node: short_var_declaration vCopy := new(big.Int).Sub(v, big.NewInt(35)) [TO] block at 1
[ACTION]insert-tree: expression_list vCopy [TO] short_var_declaration at 0
[ACTION]insert-node: := := := [TO] short_var_declaration at 1
[ACTION]insert-node: expression_list new(big.Int).Sub(v, big.NewInt(35)) [TO] short_var_declaration at 2
[ACTION]move-tree: call_expr v.Sub(v, big.NewInt(35)) [TO] expr_list new(big.Int).Sub(v, big.NewInt(35)) at 0
[ACTION]insert-tree: call_expression new(big.Int) [TO] selector_expression at 0
[ACTION]update-node: identifier: v [TO] vCopy
[ACTION]update-node: identifier: v [TO] vCopy
[ACTION]delete-node: identifier: v
[ACTION]delete-node: expression_statement v.Sub(v, big.NewInt(35))
```

图3　图1中所示代码变更对应的语法树编辑序列

Figure 3　AST edit sequence for the code changes shown in figure 1

2.2　大语言模型语义嵌入

在获取到代码变更信息之后,BlockPatch通过序列化和嵌入将上述的三种模态信息转化为特征向量。嵌入技术通过将输入映射到向量空间实现特征表示。为了捕获不同模态信息之间的关联性,本文利用大语言模型分别对这三类信息进行语义嵌入,然后将代码相关的信息融合,以便于后续的特征学习。

过去的工作常将文本和代码信息分开处理,例如PatchInsight[13]采用Bert和CodeBERT分别处理提交描述文本和代码信息。然而,在代码提交中,自然语言描述与程序代码在关键词层面存在语义相似性特征,这种做法忽略了文本和代码之间的联系和词法层面的相似性,例如变更描述常常会提及代码中的信息。同时,这种做法也容易受到单一信息的影响,例如过度依赖文本或代码单一信息。

随着大语言模型的发展,其跨语言理解能力逐步提升,得益于丰富的训练语料,大语言模型能够应用于多编程语言环境中,并关联文本描述和代码实现。ChatGPT、DeepSeek和Qwen等预训练大语言模型不仅擅长理解自然语言,也在代码生成、理解和调试等任务上表现优秀。同时大语言嵌入模型拥有更长的输入序列长度,也能够有效避免裁剪等情况的出现。然而,大语言模型存在生成内容的幻觉、预设知识冗杂等问题,在长文本序列上这些现象更明显且伴随高成本问题[29]。这导致大语言模型的端到端应用难以保证达到识别任务的最优化目标。因此,本文将其作为嵌入工具,利用大语言模型的语义理解能力,对上述多模态的代码提交信息采用统一的向量空间建模方法,以构建可解释的表征层。这种方式既保留了大语言模型在语义理解上的优势,又通过特征空间约束降低了模型对噪声数据的敏感性。同时,由于大语言嵌入模型在海量数据上完成预训练,有限的文本和代码信息被嵌入到了一个更大的向量空间,降低了模型对特定训练数据的依赖,减少了从头训练带来的过拟合风险。

在本文中,我们采用基于大语言模型的联合嵌入框架,将2.1节得到的提交信息转化为可训练的特征向量。对于三种模态的输入,本文首先使用分词器将其序列化为X_t,其中s_t为上述的三类信息,即$t\in\{\text{Text, Diff, Edit}\}$,然后利用大语言嵌入模型计算嵌入向量$\boldsymbol{e}_t$。嵌入空间的向量均通过L2范数进行归一化处理。

$$\begin{aligned} &X_t=\text{LLMTokenizer}(s_t),t\in\{\text{Text, Diff, Edit}\}\\ &\boldsymbol{e}_t=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{H}_i\in\mathbb{R}^d,\text{其中}\ \boldsymbol{H}=\text{LLM}(X_t)\in\mathbb{R}^{n\times d}\\ &\boldsymbol{e}_t=\frac{\boldsymbol{e}_t}{\|\boldsymbol{e}_t\|_2}\ (\text{L2归一化}) \end{aligned} \tag{1}$$

其中,n为序列化之后的token长度,d为嵌入维度。

在完成向量嵌入后,本文首先将代码相关信息进行融合。对于识别模型,仅分析代码编辑序列缺少完整的变更前后代码块,增加了理解难度。另一方面,如图3所示,编辑信息本身也能为行级差异提供必要的上下文补充。因此,我们将代码相关的信息,即$\boldsymbol{e}_{\text{Diff}}$和$\boldsymbol{e}_{\text{Edit}}$进行融合表示,以实现相互促进。综上所述,BlockPatch通过两个分支,对代码提交信息进行向量化嵌入处理:(1)对于变更描述文本,直接进行嵌入得到向量$\boldsymbol{e}_{\text{Text}}$;(2)对于代码变更信息,分别对行级差异表示和代码编辑信息进行嵌入,相加后得到向量$\boldsymbol{e}_{\text{Code}}$,即$\boldsymbol{e}_{\text{Code}}=\boldsymbol{e}_{\text{Diff}}+\boldsymbol{e}_{\text{Edit}}$。

2.3　多模态特征学习

本文将上述获得的代码提交嵌入向量进行增强和融合,然后利用深度神经网络判断隐匿安全补丁。具体来说,对于得到的嵌入向量,BlockPatch构建联合嵌入空间以实现多模态表征的有机融合。首先,对于描述嵌入$\boldsymbol{e}_{\text{Text}}$和代码嵌入$\boldsymbol{e}_{\text{Code}}$,本文引入双向长短时

记忆网络(Bidirectional LSTM, Bi-LSTM)分别对其进行序列建模,以增强上下文表达能力。

$$\text{Bi}-\text{LSTM}(\boldsymbol{x})=[\overrightarrow{\boldsymbol{h}_T};\ \overleftarrow{\boldsymbol{h}_1}]$$

$$\overrightarrow{\boldsymbol{h}_t}=\text{LSTM}_{\text{forward}}(\boldsymbol{x}_t,\overrightarrow{\boldsymbol{h}_{t-1}},\overrightarrow{\boldsymbol{c}_{t-1}}),\quad t=1,2,\cdots,T \tag{2}$$

$$\overleftarrow{\boldsymbol{h}_t}=\text{LSTM}_{\text{backward}}(\boldsymbol{x}_t,\overleftarrow{\boldsymbol{h}_{t+1}},\overleftarrow{\boldsymbol{c}_{t+1}}),\quad t=T,T-1,\cdots,1$$

其中:t表示时间步索引;T为序列长度;$\boldsymbol{x}_t$表示时间步t的输入;$\overrightarrow{\boldsymbol{h}_t}$和$\overleftarrow{\boldsymbol{h}_t}$分别表示前向和后向LSTM的隐藏状态向量;$\overrightarrow{\boldsymbol{c}_t}$和$\overleftarrow{\boldsymbol{c}_t}$分别表示相应的细胞状态向量。该网络通过连接两个LSTM网络前后向传播的最终隐藏状态$\overrightarrow{\boldsymbol{h}_T}$和$\overleftarrow{\boldsymbol{h}_1}$,捕捉序列中的依赖关系,生成包含双向上下文信息的增强特征表示,从而提升了序列特征的表达能力。

进一步地,本文综合利用代码信息与描述信息之间的互补性,提升对区块链系统中的安全相关提交与其他类型的判别能力。为了能够动态调整文本特征和代码特征的贡献比例,模型引入可训练参数$\alpha\in(0,1)$,对不同模态的表征进行自适应加权融合,形成最终输出的表征向量$\boldsymbol{e}$。

$$\boldsymbol{e}=\alpha\cdot\text{Bi}-\text{LSTM}\left(\boldsymbol{e}_{\text{Code}}\right)+(1-\alpha)\cdot\text{Bi}-\text{LSTM}\left(\boldsymbol{e}_{\text{Text}}\right) \tag{3}$$

在得到$\boldsymbol{e}$后,本文通过前馈神经网络(Feed-Forward Neural Network, FFN)进行拟合,判别该提交是否属于区块链安全补丁。FFN通过非线性变换,将$\boldsymbol{e}$映射到适于判别安全补丁的概率空间P,计算提交属于安全补丁的概率。在训练阶段,我们引入监督对比学习损失[30]对模型进行优化,该损失函数通过最小化同类样本特征间的距离、最大化异类样本距离,增强特征表征的判别性。最终,模型结合交叉熵损失和对比损失函数进行联合优化,为了平衡这两种损失,我们引入了权重参数c_{weight}来调节对比损失,并通过网格搜索获取最优的权重值为0.6。对于最终的预测类别结果$\hat{y}$,BlockPatch接收FFN的输出结果,通过选择最大概率对应的类别获得。

$$\hat{y}=\underset{i\in\{0,1\}}{\arg\max}\ P(y=i|\boldsymbol{e}) \tag{4}$$

2.4 安全补丁迁移

本文进一步将识别结果在区块链系统的下游项目中进行迁移,采用启发式的方式自动构建补丁信息,用于检验隐匿安全补丁是否得到及时修复。为了实现这一点,本文集成了基于补丁的缺陷检测方法。具体来说,本文采用BlockScope[26]作为缺陷传播检测组件,因其已在区块链系统中被证明效果优于其他方法,且能适用于多种编程语言。该方法基于关键变量和表达式进行已知缺陷检测,并判断特定缺陷的存在状态及其修复情况。然而,从代码提交到具体的补丁信息还需要额外处理,以排除无关的代码片段。例如,BlockScope要求人工预先给定具体的补丁信息。虽然人工提取具有较高的可靠性,但效率低下。为了构建全自动化地识别和迁移方案,本文采取启发式的方法来构建补丁信息,并在下游系统中进行传播检测。

在本文中,补丁信息包括代码提交哈希、涉及文件路径、增删代码行号以及上下文依赖关系。通过分析现有人工构建的补丁信息,我们发现其主要策略是选择变更范围较大的代码块,以尽量包含可供分析的关键变量和表达式,同时根据变更情况判断是否需要上下文信息。基于该观察,对于每个选定的代码提交,本文设计了以下自动化补丁信息构建策略:(1)首先获取提交的基本信息,并基于文件路径过滤掉测试相关代码。(2)然后选取影响范围最大的变更代码块并提取其增删行号。(3)最后综合确定上下文内容需求,包括其他变更代码块的相对位置,仅新增代码的上文内容和仅删除代码的下文内容需求。当安全补丁涉及多个代码文件时,该方法会采取充分覆盖的策略,生成多组相应的信息,以尽可能发现潜在的未修复缺陷。

3 实验结果与分析

本节首先阐述了本文实验部分的数据集构建和实验设置,然后分别回答下述三个研究问题,并对有效性进行讨论。为了评估本文提出的区块链系统安全补丁识别方法的有效性和实用性,本文对以下三个研究问题进行实验分析,以验证所提出方法和模块的有效性,以及在保障区块链生态可靠性上的有效性。

(1)RQ1(有效性评估):BlockPatch识别区块链系统生态隐匿安全补丁的效果如何?对于不同漏洞类型和陌生项目中的安全补丁,BlockPatch能否有效进行识别?

(2)RQ2(消融实验分析):引入语法树节点编辑信息和多模态特征融合的效果如何?

(3)RQ3(安全补丁迁移):BlockPatch能否将安全补丁识别结果应用于下游项目的漏洞检测?

3.1 数据集构建

本文首先需要采集区块链系统生态的补丁数据集。截至目前,区块链系统生态并不存在可直接使用的包含安全和非安全补丁的数据集。尽管在以Linux为代表的C/C++领域,PatchDB[31]和SPI-DB[10]数据库积累了一定数量的补丁记录,但缺少区块链系统在分布式架构、共识算法、交易执行等方面的安全知识。因此,本文选取了主流公链项目Bitcoin和Go-ethereum,以及联盟链项目Hyperledger Fabric作为数据采集的对象。由于区块链生态在漏洞数据库中披

露的安全缺陷十分有限,为了筛选出足够数量的区块链系统生态中的隐匿安全补丁,本文基于过去针对区块链系统漏洞的实证研究所采用的方法,从历史提交中提取安全补丁(正样本)和非安全补丁(负样本)作为实验数据,构建实验原始数据集,共包含2 450个安全补丁和3 675个非安全补丁提交。具体来说,两类补丁的采集方法如下。

安全补丁(正样本)。本文基于已有的区块链漏洞实证研究[8]所提供的基本信息进行安全补丁的判断、采集及分类。该研究使用关键词过滤出与安全漏洞相关的问题(Issue)及其关联的提交,并通过人工审核筛选得到安全漏洞和对应补丁,然后分析了频率最高的20种漏洞类型。为了尽可能搜集足够数量的安全补丁,本文首先提取了该数据源中Bitcoin和Go-ethereum的漏洞补丁信息,然后对其进行了人工扩充,重点增加了Hyperledger Fabric项目的安全漏洞补丁。最后,我们提取各个项目中安全补丁相关的历史提交作为正样本数据集。本文累计构建了2 450个安全补丁样本,其中1 230个补丁来自Bitcoin,1 078个来自Go-ethereum,142个来自Fabric。

本文所采集的安全补丁覆盖了区块链系统生态中多种常见漏洞类型。由于漏洞类型的高度多样性,本文沿用了上述研究中所总结的20种区块链系统常见漏洞类型。在此基础上,为了简化后续关于漏洞类型的阐述和分析,本文根据漏洞对区块链系统的影响方式和作用范围,对上述漏洞类型进行了合并。最终,本文将区块链系统漏洞划分为五大类。表1展示了漏洞类型的名称及其覆盖范围,囊括了上述20种常见漏洞类型。对于部分难以明确归入某一类别的安全漏洞,本文统一将其纳入到其他漏洞类型中。图4展示了不同漏洞类型对应补丁的分布情况。从整体上看,各类漏洞均有一定数量的对应的补丁,表明本文所采集的数据在漏洞类型维度上具有较好的完整性,且在不同类型上均有一定的样本量。其中,区块链特有组件相关漏洞占比达27.4%,这也反映了针对区块链系统生态漏洞开展研究的重要性。

表1　安全补丁对应的漏洞类别及其范围

Table 1　Vulnerability categories and affected scopes of security patches

漏洞类别名称	覆盖范围
并发与同步相关	竞争条件;死锁;拒绝服务
内存与资源管理	资源泄漏;未初始化读取;越界访问;Off-by-One错误;段错误;内存池错误;空指针解引用;Go语言Panic
输入与边界检查	检查和验证;完整性检查;溢出
区块链特有组件	交易相关问题;区块相关问题;对等节点相关问题;钱包密钥/密码相关;RPC相关;数据库损坏相关
其他漏洞类型	无法归类至上述类别中的其他安全漏洞

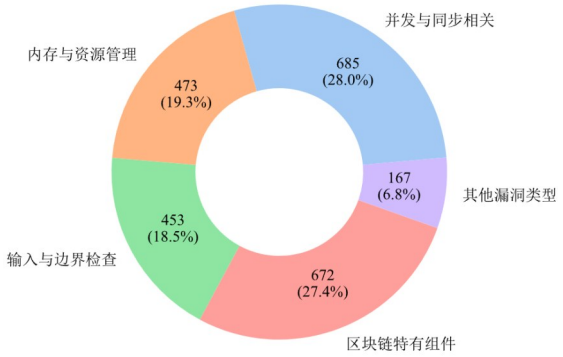


图4　不同类型安全补丁的分布图

Figure 4　Distribution of different types of security patches

非安全补丁(负样本)。为了满足负样本要求,我们从相应项目中,筛选不属于安全补丁的提交。筛选标准包括:(1)该提交不包含在漏洞数据集中,且存在变更行为;(2)该提交以及关联的Issue和PR的标题、标签和描述信息中不包含与安全漏洞相关的关键词,例如security、bug、secure、exploit、vulnerability、injection、dos;(3)安全相关的Issue关联的其他Issue或PR也被视作潜在的安全补丁,与之关联的commit也被排除。为避免负样本过多导致的偏差,同时考虑到非安全补丁中存在非代码变更的情形,本文负样本数量略多于正样本,正负样本比例接近2∶3。

此外,考虑到本文实验部分会引入大量区块链系统,为了便于读者查找,表2列出了这些系统对应的仓库链接。

3.2　实验设置

3.2.1　实现细节

BlockPatch采用Python 3.11实现,并基于先进工程资源确保相应部分的实现效果达到最优,包括使用稍作改动的Gumtree提取编辑信息,使用PyTorch框架进行训练。在特征嵌入部分,基于MTEB大规模文本嵌入基准[32]的评估结果,本文选用Qwen3-Embedding模型系列,出于实验设备和计算效率考量,采用了Qwen3-Embedding-4B模型,输出2 560维度的嵌入向量。在特征学习部分,嵌入向量在特征压缩后通过双向LSTM实现特征增强,其中LSTM网络隐藏层的维度为256,线性层的输出维度为512。在前馈神经网络中,线性层的输出维度依次为128、32和2维。为了降

表2 本文涉及的区块链项目的代码仓库链接

Table 2 Repository links of the blockchain projects in this paper

项目名称	相关代码链接
Bitcoin	https://github.com/bitcoin/bitcoin
Go-ethereum	https://github.com/ethereum/go-ethereum
Hyperledger Fabric	https://github.com/hyperledger/fabric
Nethermind	https://github.com/NethermindEth/nethermind
Erigon	https://github.com/erigontech/erigon
Besu	https://github.com/besu-eth/besu
FISCO-BCOS	https://github.com/FISCO-BCOS/FISCO-BCOS
Dogecoin	https://github.com/dogecoin/dogecoin
Bitcoin-abc	https://github.com/Bitcoin-ABC/bitcoin-abc
Dash	https://github.com/dashpay/dash
Litecoin	https://github.com/litecoin-project/litecoin
Qtum	https://github.com/qtumproject/qtum
Bnb smart chain	https://github.com/bnb-chain/bsc
Polygon	https://github.com/maticnetwork/bor
Celo-blockchain	https://github.com/celo-org/celo-blockchain
Offchainlabs	https://github.com/OffchainLabs/nitro
Optimism	https://github.com/ethereum-optimism/optimism
Sidrachain	https://github.com/SidraChain/go-ethereum
Subnet-evm	https://github.com/ava-labs/subnet-evm

低过拟合风险,本文通过在网络中添加归一化层和丢弃(Dropout)层,以提升训练稳定性并引入正则化效应,提高模型的泛化能力。模型训练过程中,本文采用了Adam优化器来提升训练效率。

在实验评估阶段,计算平台为24 GB显存的NVIDIA GeForce RTX 3090,批处理大小设置为32,学习率为1×10^{-4}。本文将数据集按照7∶1∶2的比例划分为训练集、验证集和测试集。对于数据集的划分,本文采用分层抽样来保证正负样本分布的均衡性,并根据模型在验证集上的效果来确定最终参数,进一步降低潜在的过拟合风险。同时,由于实证研究中的补丁以Issue为粒度判断是否为安全相关,而一个Issue可能关联一个或多个提交,尽管提交是独立的,但为了避免潜在的数据泄露问题,本文将同一Issue关联的安全补丁进行绑定,在划分时确保位于同一集合中。对文中所有方法,我们进行5轮实验,对模型收敛时的性能指标取平均值。对于本文模型中的可变参数,训练完成后,上文中代码模态的加权权重参数α为0.310 9。这一结果说明,在模型训练的过程中,文本信息特征的相对贡献更高,对安全补丁识别的结果有导向作用。在消融实验(3.4节)中,我们进一步分析了不同模态信息的交互机制和贡献度。

3.2.2 评估指标

本文将隐匿安全补丁的识别视为二分类的决策任务,即识别结果包括真阳性(TP)、假阴性(FN)、真阴性(TN)和假阳性(FP)。其中,TP表示正确识别安全补丁,FN为未识别出安全补丁,TN为正确识别非安全补丁,FP为将非安全补丁误判为安全相关。在有效性分析的基准对比中,为了评估不同模型在数据集上的表现,本文采用精确率(Precision)、召回率(Recall)和F1值(F1-score)三个指标进行评估。在评估维度上,本文从正(安全补丁)、负(非安全补丁)样本两个维度对模型效果进行评估。

3.2.3 基准方法

本文选取了VFCFinder[33]、SPI框架[10]、PatchRNN框架[11]、PatchInsight框架[13]作为对比的基准方法,这些方法能够直接支持多语言的安全补丁识别。例如GraphSPD[14]等针对C/C++单一语言设计,难以适用于多编程语言场景,未被纳入对比范围。为确保实验的公平性,我们利用前述实验数据对上述方法在相同环境进行了重新训练。同时,由于VFCFinder提供了原始模型参数,本文也保留了其原始参数版本,用以说明领域知识对识别隐匿安全补丁的重要性。对比方法的基本信息如下。

(1)VFCFinder。以代码提交中的文件为粒度,将描述消息和代码变更进行拼接,通过CodeBERT和全连接网络判断变更文件是否安全相关,然后统计所有文件的识别结果,计算平均值作为预测概率。

(2)SPI。采用双分支设计,文本分支通过Word2Vec进行词嵌入,通过LSTM和CNN捕捉语义关联,输出安全补丁预测概率;代码分支首先进行语法解析,随后经过Word2Vec嵌入和LSTM+CNN拟合,输出预测概率。两个分支的输出等比例加权,得到综合预测结果。

(3)PatchRNN。针对源代码和文本描述的预处理方式和SPI框架基本相同。模型结构上,PatchRNN通过RNN提取文本描述的语义特征;对于代码差异,采用双向LSTM结构进行表征学习。两部分的输出向量进行相加后,通过MLP进行拟合预测。

(4)PatchInsight。采用双重编码器融合设计,文本分支通过BERT提取提交日志的语义特征,代码分支通过CodeBERT对代码更改进行嵌入处理。两部分特征向量经过拼接后,输入到Bi-GRU网络并结合词级别注意力机制进行学习,最终通过MLP输出识别结果。

3.3 RQ1:有效性评估

3.3.1 总体效果和基准对比

表3展示了BlockPatch和对比方法在区块链系统隐匿安全补丁识别上的性能表现。本文所提出的BlockPatch总体优于现有基准对比方法。在安全补丁识别上,BlockPatch取得了94.02%的精确率和94.58%

的召回率,F1值达到94.29%。特别地,与基于RNN的方法和基于Bi-GRU的方法相比,BlockPatch在安全补丁识别的F1分数分别上涨约7.88个百分点和5.03个百分点。同时,在非安全补丁的识别上,BlockPatch上取得95.85%的精确率、95.41%的召回率和95.63%的F1值。实验结果显示,BlockPatch在所有性能指标上均取得最优,能够较为准确地识别区块链系统中的安全补丁。这充分验证了本文所构建的多模态表征和学习方法的有效性,能够在有限的学习样本中,捕捉文本描述和代码差异中的有效信息。

表3　区块链安全补丁识别性能比较　　单位:%

Table 3　Performance of security patch identification　　unit:%

方法	安全补丁			非安全补丁		
	Precision	Recall	F1	Precision	Recall	F1
原始VFCFinder	57.31	38.93	46.36	67.58	81.45	73.87
重训VFCFinder	90.28	36.72	52.20	62.26	96.35	75.64
SPI	88.43	85.13	86.75	86.14	89.24	87.66
PatchRNN	87.82	85.04	86.41	85.98	88.60	87.27
PatchInsight	92.14	86.56	89.26	88.11	93.09	90.53
BlockPatch	**94.02**	**94.58**	**94.29**	**95.85**	**95.41**	**95.63**

注:加粗数据为最优结果。

同时,我们也注意到,领域特定知识对于识别隐匿安全补丁是至关重要的。以VFCFinder在安全补丁识别上的效果为例,重新训练后的版本显著高于基于原始漏洞库数据集的版本,在安全补丁识别上提升了超32个百分点的精确率和5个百分点的F1值。这一结果说明,区块链领域的漏洞知识对于有效判断补丁是否与安全问题相关是极其重要的。这一现象要求研究者对区块链生态开展更全面的安全补丁分析,构建覆盖更多项目的漏洞数据库,以提升基于已知安全补丁的检测方法的效果。

3.3.2　不同类别补丁识别能力

为了进一步验证本文所提出方法在识别不同类型安全补丁上的有效性,我们统计了模型对不同类型安全补丁的识别结果。由于此处仅讨论能否有效识别某一类型的安全补丁,分析数据中仅包含安全补丁,不含非安全补丁。我们参照近期在C/C++领域的相关工作[34],使用准确率来衡量识别的有效性,代表在该类别中成功识别出的安全补丁数量的占比。

本文在包含381个安全补丁的测试数据集上进行了实验分析,涵盖了上表中所述的5类漏洞。表4展示了不同漏洞类型的占比,以及在不同类别上的识别准确率。BlockPatch表现出良好的泛用性,能够有效识别各类漏洞补丁。其中,在输入与边界检查上取得了最好效果(98.11%),在区块链特有组件、内存与资源管理类、并发与同步相关漏洞上同样保持较高准确率,分别达到97.44%、94.79%和91.75%。相比之下,其他漏洞类型的识别准确率相对较低,由于该类漏洞的覆盖范围更广而训练数据量却更少,这是符合预期的。同时,从样本的来源分布来看,模型在Bitcoin、Go-ethereum、Fabric上针对安全补丁的准确率分别达到96.08%、92.57%和89.66%。实验结果证明,本文所提出方法具有较好的泛用性,能够有效识别不同项目中多种类型漏洞的安全补丁。

表4　各漏洞类别补丁的识别准确率　　单位:%

Table 4　Accuracy of patch identification across vulnerability categories　　unit:%

所属漏洞类别	占比	准确率
并发与同步相关	25.46	91.75
内存与资源管理	25.20	94.79
输入与边界检查	13.91	98.11
区块链特有组件	30.71	97.44
其他漏洞类型	4.72	72.22

3.3.3　陌生项目补丁识别能力

为验证BlockPatch在训练集之外的区块链项目上的适用性,本文选取了四个由不同编程语言编写且具有代表性的区块链系统,即Nethermind(C#)、Erigon(Go)、Besu(Java)和FISCO BCOS(C++),覆盖现实中的公有链和联盟链。它们由不同编程语言编写且非数据集中项目的分叉链,确保了补丁类型的多样性,并且不可能出现在训练集中。安全补丁的筛选遵循以下两个标准:(1)补丁代码提交时间在最近一年内;(2)补丁关联的Issue或PR明确被开发者指明为漏洞或打上漏洞相关标签。我们共计采集了15个包含多种安全类型的补丁。通过将上述训练所得的模型应用于这些安全补丁,BlockPatch在15个测试案例中正确识别出了12个。表5展示了完整的补丁信息来源、安全缺陷类型和识别结果。实验结果证明BlockPatch能够识别新项目中的安全补丁,具备一定的跨项目隐匿安全补丁识别能力。

3.4　RQ2:消融实验分析

为验证关键模块的有效性,本文从两个方面设计消融实验。首先,本文首次将代码的语法树编辑信息引入到隐匿安全补丁的识别任务中(2.1节),用于丰富行级代码差异,因此有必要对编辑信息的有效性进行验证。之后,对于2.3节,BlockPatch融合了文本和代码特征,我们通过保留其中之一,以反映多模态特征的有效性。为了简化表述,消融实验中Model 1为完整模型,Model 2(w/o Edit)为模型移除了编辑信息,Model 3(w/o Text)为模型去除描述信息,Model 4(w/o Code)为模型去除了代码相关信息。

测试结果表现。表6展示了消融实验在测试集

表5 陌生区块链系统安全补丁识别结果(按项目分类)

Table 5 Security patch identification results for unseen blockchain systems (grouped by projects)

所属项目	问题来源	缺陷类型描述	提交哈希	正确识别
Nethermind (C#)	Issue #8797	会话处理逻辑	db134d	√
	Issue #8479	异常处理逻辑	8cac9b	×
	Issue #8223	创世区块逻辑	3f6a9c	√
	Issue #8124	兼容性检查	7bdf74	√
	Issue #7751	日志索引计算	ecc387	√
Erigon (Go)	Issue #13488	空指针引用	7e3384	√
	Issue #12248	Gas值计算	1ca05e	√
Besu (Java)	Issue #8449	读写格式冲突	db6ac7	√
	Issue #7661	RPC返回错误	35d1a7	√
	Issue #7635	空元素异常	bc7555	√
	Issue #7608	返回格式错误	37e861	×
	Issue #7585	空指针引用	c5abc3	√
FISCO BCOS (C++)	Issue #4884	EVM状态逻辑	fe489d	×
	Issue #4835	竞态条件漏洞	3c2931	√
	Issue #4662	交易池逻辑	d17c39	√

上的结果,即不同模态特征对模型性能的影响。表中最优性能指标值以粗体表示,次优值添加了下划线。整体来看,融合了三类信息的完整模型(Model 1)在安全补丁识别任务上表现最佳,其F1值达到94.29%,优于其他单一或部分特征缺失的消融模型,证明了多模态信息融合的有效性。

表6 各消融模型在测试集上的效果 单位:%

Table 6 Performance of different ablation models on the test set unit:%

方法	安全补丁			非安全补丁		
	Precision	Recall	F1	Precision	Recall	F1
Model 1	**94.02**	**94.58**	**94.29**	**95.85**	95.41	**95.63**
Model 2	93.31	92.47	92.89	94.31	94.95	94.63
Model 3	95.93	85.24	90.27	89.64	**97.25**	93.29
Model 4	92.08	94.58	93.31	95.78	93.81	94.79

通过对比模型各项性能指标的差异,本文进一步分析了各模块在安全补丁识别上的贡献度及交互机制。

(1)语法树编辑信息能有效提升综合性能。对于完整模型Model 1与Model 2(w/o Edit)的性能差异,可以观察到引入语法树编辑信息后,模型所有的性能指标都有一定的提升,在F1值上提升了1.4个百分点。这一结果进一步证明了细粒度的语法树编辑操作(如节点的增删、移动)及相关语法结构信息和上下文,能够帮助模型理解补丁的功能,从而提高整体性能。

(2)代码描述信息具有更强的导向作用。从F1值来看,仅采用描述信息的Model 4(w/o Code)达到了次优值(93.31%),说明了文本信息在安全补丁识别中的重要性。这实际也呼应了特征权重参数α在训练后偏向文本信息的结果。这是合理的,因为在多数情况下,开发者所编写的提交信息实际上已经包含了变更的主要内容。另一方面,Model 4的良好性能同样也验证了大语言嵌入模型在语义表征上的优势。

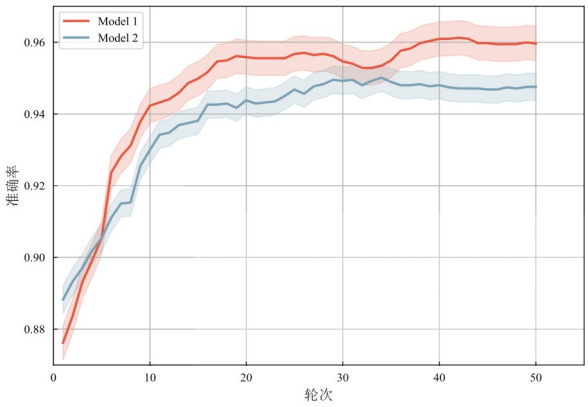
(3)文本和代码特征具有明显互补效应。直观来说,Model 3(w/o Text)取得了次优的精确率(95.93%),而仅有文本信息的Model 4取得了次优的召回率(94.58%)。从补丁的角度来看,去除补丁文本描述后,模型倾向于仅将具备明显安全特征的代码变更判定为正例,从而提升了精确率,但由于缺乏描述信息中关于漏洞的语义暗示,导致大量修改特征隐晦的安全补丁被漏报,损害了召回率。通过将文本描述引入,模型获得了补丁的意图信息,弥补仅依靠代码特征时的语义鸿沟,能够有效提升安全补丁的召回率。反过来,代码信息也缓解了仅依靠文本信息导致的识别不准确。

验证过程表现。为了评估不同特征模块对模型收敛性及性能的影响,本文记录了训练过程中上述各模型在验证集上的表现。图5展示了消融实验中各个模型在验证集上准确率变化曲线。

图5(a)聚焦于语法树编辑信息的作用,即完整模型Model 1(红线)和去除编辑信息的模型Model 2(蓝线)的比较。在训练过程中,Model 1准确率攀升更快,约在第5轮后便一直保持着领先,并且能够更快达到较高准确率的稳态。这表明编辑信息有助于模型捕捉代码变更的重要特征,提升训练效率和模型准确率。图5(b)进一步分析了文本与代码特征的贡献,对比了Model 1、Model 3(绿线)和Model 4(黄线)。与其他两个模型相比,Model 1依然保持最优的准确率曲线的训练效率。值得注意的是,去除描述信息的Model 3表现最差,初期上升迟缓且收敛点最低,这一趋势揭示了提交描述在早期训练中提供了关键的语义先验,若缺失该信息,模型难以仅凭代码特征快速建立准确的分类边界。Model 1的优势证实了多模态信息融合机制的有效性,实现准确率与收敛速度的双重提升。

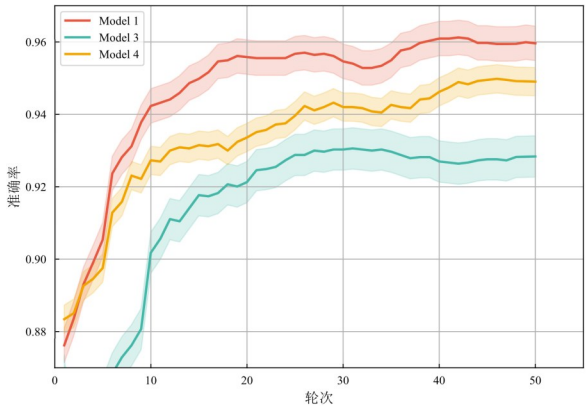
3.5 RQ3:安全补丁迁移

基于安全补丁识别结果,我们构建补丁信息并开展跨项目漏洞检测工作,以验证自动化安全补丁迁移在区块链生态的有效性。鉴于Bitcoin和Go-ethereum项目在区块链生态中的广泛流行度及其组件在下游项目中的高度复用性,我们选取这两个区块链项目作为基础,在采用相同编程语言的下游项目中进行安全补丁迁移验证。从Bitcoin和Go-ethereum项目中,我们首先拉取最近600个代码提交,通过BlockPatch进



(a) 语法树编辑信息消融结果

(a) Ablation results on AST edit information



(b) 描述信息和代码信息消融结果

(b) Ablation results on text and code information

图5　训练过程中各消融模型在验证集上的表现

Figure 5　Validation performance of ablation models during training

行安全补丁识别。针对比特币生态,我们选取了5个相关项目:Dogecoin、Bitcoin-abc、Dash、Litecoin、Qtum;针对以太坊生态,选取了7个相关项目:BSC、Polygon、Celo-blockchain、OffchainLabs、Optimism、SidraChain和Subnet-evm。用于漏洞检测的代码仓库均为2025年7月15日的主分支版本。

表7展示了近期代码提交中的隐匿安全补丁,及其对下游应用的影响。基于BlockPatch和少量的人工验证,我们共计发现了16个隐匿安全补丁。具体来说,在比特币生态,基于Bitcoin项目的10个修复提交,在下游项目中共计发现了18个相关漏洞;在以太坊生态,基于Go-ethereum项目的6个修复提交,在下游项目中发现了10个相关漏洞。从修复的情况来看,Bitcoin相关的下游项目中无一进行了漏洞的修复。在Go-ethereum生态中,我们发现仅BSC和Polygon中包含已进行补丁的安全缺陷。其中,BSC包含了2个已修复的实例,然而依然有平均22天的修复延迟。这一结果更加突出了及时报告隐匿安全补丁并进行传播检测和应用的重要性。

表7　基于安全补丁的下游项目漏洞检测结果

Table 7　Vulnerability detection results for downstream projects based on security patches

区块链生态	隐匿补丁	下游项目	未修复	已修复	平均延迟/天
比特币	10	Dogecoin	2	0	—
		Bitcoin-abc	5	0	—
		Litecoin	3	0	—
		Dash	4	0	—
		Qtum	4	0	—
以太坊	6	BSC	1	2	22
		Optimism	1	0	—
		Polygon	2	1	56
		Subnet-evm	1	0	—
		Celo-blockchain	2	0	—
		SidraChain	2	0	—
		OffchainLabs	1	0	—

该实验结果表明BlockPatch能够有效识别真实的隐匿安全补丁,并通过集成基于补丁的静态检测技术,实现下游项目的潜在漏洞检测。相较于传统人工分析方法,BlockPatch有助于构建端到端的自动化工作流,能够更高效地实现安全补丁的迁移。针对实验过程中发现的安全漏洞,我们向相关区块链项目的代码仓库提交了反馈,目前已收到6个积极确认回复并已修复。

案例分析。我们通过两个由BlockPatch发现并向项目方进行报告的隐匿安全补丁,来阐释在区块链生态中的软件供应链风险。这两个安全补丁均影响了下游分叉或相似的区块链系统。

案例1,Go-ethereum提交0feb999。图6展示了该提交的相关内容,其修复了bn256包的MulScalar函数错误,将原本错误使用接收者字段改为正确使用输入参数a的字段,确保了正确执行椭圆曲线上的标量乘法运算。相较于Go-ethereum,其他复用了相似代码的区块链系统的修复时间则更长,甚至至今仍未修复。例如,同样是以太坊重要客户端之一的Erigon延后2个月进行修复,而SidraChain则仍未修复。事实上,该漏洞源于Google与Cloudflare提供的Golang加密算法库中存在的错误,并分别在2019年和2023年得到修复。这一案例突出了识别隐匿安全补丁的重要性,亟需拓展至更多区块链生态及其上游组件。

案例2,Go-ethereum提交05e1994。图7展示了该提交的相关内容,其修复了一个典型的空引用问题。具体来说,当调用GetKey函数获取结果时,预映像

crypto/bn256: fix MulScalar (#30974). The 'a' parameter should be used in the 'MulScalar' function. The upstream cloudflare and google repos have already merged fixes.

```
1  func (e *gfP12) MulScalar(a *gfP12, b *gfP6) *gfP12 {
2  -    e.x.Mul(&e.x, b)
3  -    e.y.Mul(&e.y, b)
4  +    e.x.Mul(&a.x, b)
5  +    e.y.Mul(&a.y, b)
6       return es
7  }
```

图6 案例1中Go-ethereum提交0feb999相关内容

Figure 6 Details of commit 0feb999 in Go-ethereum of case 1

(preimage)的缺失会导致返回值为空,进而在后续哈希过程中发生错误。该提交通过添加对nil的检查来跳过无效的存储条目,确保账户存储的正确性。Go-ethereum在6月接收了这一安全补丁,但通过BlockPatch我们发现,在Polygen、BSC、Celo-blockchain、SidraChain和Subnet-evm等区块链系统中存在类似实现且未被修复。

fix: skip storage entries with missing preimage keys (#32051). When 'GetKey' is called, a missing preimage can cause the function to return a 'nil' key. This makes 'account.Storage' persist incorrect value.

```
1  for storageIt.Next() {
2      _, content, _, err := rlp.Split(storageIt.Value)
3      if err != nil {...}
4  -   account.Storage[common.BytesToHash(
5  -           s.trie.GetKey(storageIt.Key))] = common.Bytes2Hex(content)
6  +   key := s.trie.GetKey(storageIt.Key)
7  +   if key == nil { continue }
8  +   account.Storage[common.BytesToHash(key)] = common.Bytes2Hex(content)
```

图7 案例2中Go-ethereum提交05e1994相关内容

Figure 7 Details of commit 05e1994 in Go-ethereum of case 2

3.6 讨论

本文从误报分析和有效性威胁两个方面展开讨论,并据此阐述未来工作的主要方向。

3.6.1 安全补丁误报分析

在上述实验分析中,本文讨论了所提出的方法在漏洞补丁识别上的有效性。我们对实验结果中误报的补丁样本进行了抽样分析。选取的错误识别案例覆盖了实验中的不同区块链系统,且具有明显的提交信息差异,共包含10个误判(FP)和10个漏判(FN)案例。直观来说,导致错误识别的核心在于漏洞和缺陷之间的关联和差异,即部分缺陷实际上不会导致安全问题,即非安全缺陷,但影响系统效率和资源利用率等。对于人类专家而言,正确区分这两者同样具有难度,这些变更与安全修复有很多相似之处,需要理解补丁所发挥的功能才能准确识别。本节的后续部分对误判和漏判进行了归类分析。

误判(FP)分析。误判主要源于重构与代码优化(50%)、良性功能性修复(30%)和安全无关的日志与文档修复(20%)。

(1)重构与代码优化是导致模型误判的首要原因。当开发者对系统安全相关模块进行较大规模的代码重写或结构调整时,例如交易验证或内存池管理模块,尽管未涉及逻辑漏洞修复,但其代码变更模式(即代码差异和编辑信息)与安全修复高度相似,从而影响了模型判断。例如Bitcoin b9cec7尽管修改了验证逻辑,但其核心是优化交易验证性能。

(2)良性功能性修复也容易被误标记。这类提交通常是对功能或配置进行调整,虽然确实修复了某些缺陷,但并非安全相关漏洞,例如Go-ethereum 3e4fbc更改了握手协议重发送逻辑。然而,由于这类提交信息中包含"handshake""nonce"等高敏感的区块链系统相关安全术语,模型容易将其错误关联为安全修复。

(3)与安全无关的日志或文档的修复也会导致一定的误判。当提交的核心目的是修改日志或文本记录的内容时,若包含了"fix""verify"等常用于描述安全检查的词汇,模型容易误判为安全相关的修复。

漏判(FN)分析。漏判主要源于修复位于非源码文件(40%)、预防性加固(30%)、修复逻辑隐蔽(20%)以及安全相关的运维和可用性问题(10%)。

(1)模型的漏判主要集中在安全修复位于非源码文件的变更上,例如基础设施配置(如CI/CD脚本中的安全检测规则)或依赖库版本的升级。这些修改有时会包含关键的安全加固,但由于其不涉及核心业务源码,或代码层面的差异较小,模型在分析时容易忽略其中所蕴含的安全修复语义。

(2)值得注意的是,预防性加固是另一大漏判来源,其通过增加额外的防护逻辑来修复漏洞,例如增加资源限制和统计功能以防止DoS。这类提交通常更侧重于描述"新功能实现"而非"漏洞修复",其代码也更为复杂多样,导致模型难以识别代码变更的意图。

(3)修复逻辑过于隐蔽也会干扰模型判断。在一些提交中,核心的安全修复逻辑仅占极少行数,例如边界检查和条件修正,而伴随的安全无关改动和测试代码占据了绝大篇幅,导致模型的注意力分散,关键的修复逻辑容易被模型忽略。

(4)与安全相关的运维和可用性改动也是导致漏判的一种情况,例如补充或修正影响安全分析的错误消息。这类补丁通常处于安全修复与可用性改进的边界,部分开发者会将其标注为漏洞,但由于缺乏明显的安全修复特征,增加了模型识别的难度。

3.6.2 有效性威胁

内部威胁。内部威胁主要来源于所构建的数据集的局限性。区块链生态尚缺乏成熟且公开的漏洞数据库,在CVE等漏洞数据库中也仅有少量的公开

披露信息。本文的数据仅涵盖了部分项目,样本总量相对有限,一定程度上制约了模型对安全补丁特征的学习能力。为此,我们从多个代表性的公链和联盟链项目中搜集多类别的安全补丁。在方法上,本文采用了一系列方法缓解潜在的过拟合风险,包括基于预训练大语言嵌入模型进行高维语义表示,使用L2归一化限制参数大小,在网络模型中引入Dropout层,在数据划分上采用分层抽样,通过独立验证集来确定最优模型,以提升模型的泛化性。另一方面,由于漏洞补丁的多样性和复杂性,训练数据的标签难以完全准确,从而引入少量标签噪声,影响模型效果。另一个有效性威胁源于本文所集成的漏洞识别方法,利用启发式方法构建补丁信息可能存在准确率不足的问题。由于本文的核心是验证隐匿安全补丁在下游项目中修复的滞后性,从实验分析来看,这种方式已然能在下游项目中开展自动化的识别。目前,基于补丁的漏洞检测依然是一个开放问题,修复文件和函数的定位[35]、补丁应用前后的代码检测[7]等一系列问题仍需后续研究。

外部威胁。外部威胁主要来源于基准方法的复现偏差。本文采用的对比方法存在实现细节披露不足的情况,尽管实验遵循原论文的模块设计和参数配置,但可能存在训练过程中的效果误差。为最大限度消除此偏差,我们对基准模型进行了参数调整,并在统一的数据集上重新训练,以尽量匹配原论文的实验结果,确保实验对比的公平性与可靠性。除此之外,由于部分安全补丁的数据采集和漏洞分类依赖于现有研究[8],其潜在的数据质量问题也难免会传递到本文的实验分析中。

3.6.3　未来工作

基于上述错误识别案例和有效性威胁分析,未来工作的主要方向是提升数据集质量和增强模型对补丁语义理解能力,以进一步提升区块链系统生态的安全补丁识别效果。在补丁数据构建上,我们可以扩展数据来源,纳入更多区块链系统项目以丰富安全和非安全补丁样本的多样性。同时,通过建立更严谨的安全漏洞类型界定标准,降低模糊缺陷带来的分类偏差,进一步提升数据集的质量和覆盖面。

另一方面,当前特征提取部分所采用的变更信息序列化方式,尽管能够高效适用于不同的编程语言中,但在复杂上下文和结构化语义信息的表示上存在一定的局限性,可能导致模型对补丁语义的理解不够完整,尤其是针对难以区分的功能性重构与安全修复的问题。为此,我们可以探索融合智能体的跨语言的补丁语义分析方法,通过结合语法树结构和大语言模型,优先分析代码变更的核心文件和相关上下文。一方面,这种做法可以为模型训练提供更充足和聚焦的上下文。另一方面,基于大模型的理解能力,可以理解代码变更背后的功能意图,进而判断是否有实质性的漏洞修复。这种方式也使模型能提供更详细的补丁信息,以便更好地在下游生态中开展补丁识别和迁移。除此之外,针对配置文件及依赖项变更易被遗漏的问题,可以通过结合专家知识,以覆盖非源码层面安全加固行为的识别。

4　结论

本文提出了BlockPatch,首个面向多语言区块链系统生态的隐匿安全补丁识别和迁移框架。该框架融合了代码提交中文本描述信息和细粒度代码差异信息,包括行级和语法树编辑信息,实现了更丰富的变更语义提取。同时,该框架利用大语言模型的强大的表征能力,将文本描述与代码特征联合嵌入和融合,进一步提升了深度神经网络模型对安全补丁的识别能力。对于识别结果,本文通过启发式方案自动构建分析的检测输入,实现了对下游区块链系统的安全检查。为了验证方法的有效性,本文构建了包含2 450个安全补丁的数据集。实验结果显示,BlockPatch在安全补丁识别上取得了94.02%的精确率、94.58%的召回率和94.29%的F1值,在F1值上优于对比方法5.03个百分点,且能够适用于多种类型的安全补丁识别。消融实验进一步验证了核心模块的重要性,显示出在准确率和稳定性上的优势。基于安全补丁识别结果,我们还在比特币和以太坊生态的下游系统中共计检测出了28个未修复的安全漏洞。

参考文献

[1] 盖珂珂, 陈思源, 祝烈煌. 基于区块链的可审计隐私保护机密交易[J]. 电子学报, 2025, 53(2): 460-473.
Gai Keke, Chen Siyuan, Zhu Liehuang. Blockchain-based privacy-preserving auditable confidential transaction scheme[J]. Acta Electronica Sinica, 2025, 53(2): 460-473. (in Chinese)

[2] 陆琪鹏, 刘亚丽, 刘长庚, 等. 基于区块链的RFID供应链产品所有权转移方案[J]. 电子学报, 2025, 53(2): 451-459.
Lu Qipeng, Liu Yali, Liu Changgeng, et al. Product ownership transfer scheme of RFID-enabled supply chain based on blockchain[J]. Acta Electronica Sinica, 2025, 53(2): 451-459. (in Chinese)

[3] Munir S. How many blockchain networks are there in 2025 [EB/OL]. [2026-03-01]. https://coinweb.com/trends/how-many-blockchain-networks-are-there/.

[4] Wang Xinda, Sun Kun, Batcheller A, et al. Detecting "0-day" Vulnerability: An empirical study of secret security patch in OSS[C]//2019 49th Annual IEEE/IFIP Internation-

al Conference on Dependable Systems and Networks. Piscataway: IEEE, 2019: 485-492.

[5] Dong Jialiang, Chen Xinzhang, Susilo W, et al. What lies beneath: An empirical study of silent vulnerability fixes in open-source software[C]//2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway: IEEE, 2025: 345-357.

[6] Cve. CVE[EB/OL]. [2026-03-01]. https://www.cve.org/.

[7] Lin Ruyan, Fu Yulong, Yi Wei, et al. Vulnerabilities and security patches detection in OSS: A survey[J]. ACM Computing Surveys, 2025, 57(1): 1-37.

[8] Yi Xiao, Wu Daoyuan, Jiang Lingxiao, et al. An empirical study of blockchain system vulnerabilities: Modules, types, and patterns[C]//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2022: 709-721.

[9] 周鹏, 武延军, 赵琛. 一种Linux安全漏洞修复补丁自动识别方法[J]. 计算机研究与发展, 2022, 59(1): 197-208.
Zhou Peng, Wu Yanjun, Zhao Chen. Identify linux security vulnerability fix patches automatically[J]. Journal of Computer Research and Development, 2022, 59(1): 197-208. (in Chinese)

[10] Zhou Yaqin, Siow J K, Wang Chenyu, et al. SPI: Automated identification of security patches via commits[J]. ACM Transactions on Software Engineering and Methodology, 2022, 31(1): 1-27.

[11] Wang Xinda, Wang Shu, Feng Pengbin, et al. PatchRNN: A deep learning-based system for security patch identification[C]//MILCOM 2021 - 2021 IEEE Military Communications Conference. Piscataway: IEEE, 2021: 595-600.

[12] Han Mei, Wang Lulu, Chang Jianming, et al. Learning graph-based patch representations for identifying and assessing silent vulnerability fixes[C]//2024 IEEE 35th International Symposium on Software Reliability Engineering. Piscataway: IEEE, 2024: 120-131.

[13] 唐建平, 魏书宁, 王植, 等. 融合双重编码器和词注意力机制的安全补丁识别模型[J]. 小型微型计算机系统, 2025, 46(12): 3055-3062.
Tang Jianping, Wei Shuning, Wang Zhi, et al. Secure patch identification model integrating dual encoders and word attention mechanism[J]. Journal of Chinese Computer Systems, 2025, 46(12): 3055-3062. (in Chinese)

[14] Wang Shu, Wang Xinda, Sun Kun, et al. GraphSPD: Graph-based security patch detection with enriched code semantics[C]//2023 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2023: 2409-2426.

[15] Mikolov T, Sutskever I, Chen Kai, et al. Distributed representations of words and phrases and their compositionality[C]//Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. New York: ACM, 2013: 3111-3119.

[16] Blockpatch. BlockPatch[EB/OL]. [2026-03-01]. https://github.com/0x0FOG/BlockPatch.

[17] Dunlap T, Thorn S, Enck W, et al. Finding fixed vulnerabilities with off-the-shelf static analysis[C]//2023 IEEE 8th European Symposium on Security and Privacy. Piscataway: IEEE, 2023: 489-505.

[18] Wen Zhongzhen, Zhou Jiayuan, Pan Minxue, et al. Silent taint-style vulnerability fixes identification[C]//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2024: 428-439.

[19] Zhou Jiayuan, Pacheco M, Wan Zhiyuan, et al. Finding a needle in a haystack: Automated mining of silent vulnerability fixes[C]//2021 36th IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE, 2021: 705-716.

[20] 刘敖迪, 杜学绘, 王娜, 等. 区块链系统安全防护技术研究进展[J]. 计算机学报, 2024, 47(3): 608-646.
Liu Aodi, Du Xuehui, Wang Na, et al. Research progress on blockchain system security technology[J]. Chinese Journal of Computers, 2024, 47(3): 608-646. (in Chinese)

[21] Zhang Ren, Preneel B. Lay down the common metrics: Evaluating proof-of-work consensus protocols' security[C]//2019 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2019: 175-192.

[22] Yang Y, Kim T, Chun B. Finding Consensus Bugs in Ethereum via Multi-transaction Differential Fuzzing. [C/OL]//15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021. 2021: 349-365. https://www.usenix.org/conference/osdi21/presentation/yang.

[23] Kim S, Hwang S. EtherDiffer: Differential testing on RPC services of ethereum nodes[C]//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2023: 1333-1344.

[24] Zhou Yuanhang, Yan Zhen, Chen Yuanliang, et al. Chord: Towards a unified detection of blockchain transaction parallelism bugs[C]//2025 IEEE/ACM 47th International Conference on Software Engineering. Piscataway: IEEE, 2025: 3022-3034.

[25] Qingze Hum, Tan W J, Tey S Y, et al. CoinWatch: A clone-based approach for detecting vulnerabilities in cryptocurrencies[C]//2020 IEEE International Conference on Blockchain. Piscataway: IEEE, 2020: 17-25.
[26] Yi Xiao, Fang Yuzhou, Wu Daoyuan, et al. BlockScope: Detecting and investigating propagated vulnerabilities in forked blockchain projects[C]//Proceedings 2023 Network and Distributed System Security Symposium. Internet Society, 2023.
[27] Falleri J R, Martinez M. Fine-grained, accurate and scalable source differencing[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. New York: ACM, 2024: 3639148.
[28] Afnanenayet. Diffsitter[EB/OL]. [2026-03-01]. https://github.com/afnanenayet/diffsitter.
[29] Zhao W X, Zhou Kun, Li Junyi, et al. A survey of large language models[PP/OL]. V19. arXiv (2026-03-18)[2026-01-28]. https://doi.org/10.48550/arXiv.2303.18223.
[30] Khosla P, Teterwak P, Wang Chen, et al. Supervised contrastive learning[C]//Proceedings of the 34th International Conference on Neural Information Processing Systems. New York: ACM, 2020: 18661-18673.
[31] Wang Xinda, Wang Shu, Feng Pengbin, et al. PatchDB: A large-scale security patch dataset[C]//2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway: IEEE, 2021: 149-160.
[32] Muennighoff N, Tazi N, Magne L, et al. MTEB: Massive text embedding benchmark[PP/OL]. V3. arXiv (2023-03-19)[2026-01-28]. https://doi.org/10.48550/arXiv.2210.07316.
[33] Dunlap T, Lin E, Enck W, et al. VFCFinder: Pairing security advisories and patches[C]//Proceedings of the 19th ACM Asia Conference on Computer and Communications Security. New York: ACM, 2024: 1128-1142.
[34] Wen Xincheng, Lin Zirun, Gao Cuiyun, et al. Repository-level graph representation learning for enhanced security patch detection[C]//2025 IEEE/ACM 47th International Conference on Software Engineering. Piscataway: IEEE, 2025: 00121.
[35] Chen Zhaoling, Tang R, Deng Gangda, et al. LocAgent: graph-guided LLM agents for code localization[C/OL]//Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025. https://aclanthology.org/2025.acl-long.426/.

作者简介

郝偲成　男,1998年生。2021年获四川大学工学学士学位,现为中山大学博士研究生。主要研究方向为区块链和智能合约安全。
E-mail: haosch@mail2.sysu.edu.cn

魏桂鹏　男,2002年生。2025年获中山大学工学学士学位,现为中山大学硕士研究生。主要研究方向为软件安全、区块链、数据挖掘。
E-mail: weigp5@mail2.sysu.edu.cn

肖煜铭　男,2002年生。2025年获中山大学学士学位,现为中山大学软件工程学院博士研究生。主要研究方向为区块链安全与关键技术。
E-mail: xiaoym23@mail2.sysu.edu.cn

南雨宏　男,1990年生。2018年于复旦大学获得博士学位,曾任美国普渡大学博士后研究员。现为中山大学软件工程学院副教授、博士生导师。主要研究方向为系统安全及隐私保护。
E-mail: nanyh@mail.sysu.edu.cn

郑沛霖　男,1994年生。分别于2017年、2022年获中山大学学士、博士学位。现为中山大学软件工程学院博士后。主要研究方向为区块链及智能合约可靠性。
E-mail: zhengplin@mail.sysu.edu.cn

郑子彬　男,1982年生。现为中山大学软件工程学院院长、中山大学人工智能研究院副院长、IEEE Fellow、IET Fellow、ACM杰出科学家、国家数字家庭工程技术研究中心副主任、广东省区块链工程技术研究中心主任。主要研究方向为区块链智能合约、软件可靠性和可信大模型。
E-mail: zhzibin@mail.sysu.edu.cn