计算机科学
COMPUTER SCIENCE

基于双流深度学习的Dockerfile安全误配置检测方法

赵宁, 王金双, 崔帅

引用本文

赵宁, 王金双, 崔帅. 基于双流深度学习的Dockerfile安全误配置检测方法[J]. 计算机科学, 2025, 52(10): 395-403.

ZHAO Ning, WANG Jinshuang, CUI Shuai. Dual-stream Feature Fusion Approach for Dockerfile Security Misconfiguration Detection [J]. Computer Science, 2025, 52(10): 395-403.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241000014

基于双流深度学习的 Dockerfile 安全误配置检测方法

赵 宁 王金双 崔 帅
陆军工程大学指挥控制工程学院 南京 210007
(zhaonig@yeah.net)

摘 要 Dockerfile 错误配置容易引发容器安全漏洞。现有检测方法侧重于文本的结构分析和语义理解,缺乏对指令使用频率、镜像层数、代码复杂度等度量特征的关注,准确率不高。针对该问题,提出一种融合特征度量与深度语义理解的双流深度学习检测方法。该方法首先采用静态检测工具识别并标注含有安全误配置的 Dockerfile 样本;然后构建抽象语法树解析并提取代码度量特征,并使用随机森林算法进一步筛选关键安全特征;最后提取文本信息和安全特征度量信息,输入双流模型进行检测。该模型采用双向长短期记忆网络追踪指令序列前后依赖,挖掘深度语义关联;应用 Transformer 模型构建高维度量表示,捕捉度量到安全配置缺陷的复杂映射;使用卷积神经网络子层和 ReLU 激活函数融合双流信息,实现 Dockerfile 安全误配置检测。实验表明,所提方法的查准率、查全率和 F1 值分别达到了 96%,98%和 97%,其性能相较于已有检测方法有所提升,可以有效识别 Dockerfile 安全误配置。

关键词:容器安全;Dockerfile;安全误配置检测;深度学习;双流模型
中图分类号 TP391

Dual-stream Feature Fusion Approach for Dockerfile Security Misconfiguration Detection

ZHAO Ning, WANG Jinshuang and CUI Shuai
Institute of Command Control Engineering, Army Engineering University, Nanjing 210007, China

Abstract Dockerfile misconfigurations frequently lead to container security vulnerabilities. Current detection methods rely on structural analysis and semantic understanding of the text, while pay little attention to metrics such as command frequency, image layer counts, code complexity, etc. To solve this problem, a dual-stream deep learning detection approach is proposed, which integrates feature metrics with semantic comprehension. Firstly, it identifies and annotates Dockerfile samples containing security misconfigurations using static detection tools such as Hadolint and KICS. Then, by constructing abstract syntax trees, it parses and extracts code metric features and refines crucial security features using the random forest algorithm. Lastly, it extracts textual information and security feature metrics and then inputs them into a dual-stream model for detection. Bi-LSTM network is utilized to trace the forward and backward dependencies within instruction sequences, which is helpful for uncovering deep semantic associations. Transformer model is employed to create high-dimensional metric representations, which can model mappings from metric to security misconfiguration. CNN sublayers with ReLU activation functions are used to fuse information from both streams. Experimental results indicate that the proposed method achieves 96%, 98% and 97% in precision, recall, and F1-score respectively. The proposed approach can detect security misconfiguration more accurately compared to existing approaches.

Keywords Container security, Dockerfile, Security misconfiguration detection, Deep learning, Dual-stream model

1 引言

近年来,以 Docker 为代表的容器虚拟化技术被广泛应用于软件开发和运维过程[1],它能够将应用程序的代码和依赖打包到一个轻量级、独立且可移植的执行环境中[2],给人们带来了极大的便利。然而,相较于传统虚拟化技术,容器技术在提供轻量化解决方案的同时,也面临着较低隔离级别所带来的安全挑战。容器直接运行于宿主机操作系统之上,不恰当的容器配置不仅可能对容器内的应用程序构成威胁,还可能波及宿主机,进而影响整个系统的安全。

Dockerfile 作为构建 Docker 容器的核心配置文件,其编写质量直接影响容器的安全。如图 1 所示,默认情况下,Dockerfile 赋予容器 root 权限,违背了“最小权限原则”,一旦容器内部服务遭到入侵,将直接暴露系统最高权限,增加数据泄露、系统被破坏等安全风险。为缓解此问题,图 2 通过在 Dockerfile 中(第 7 行)新增 USER 指令,创建并切换到权限受限的用户来运行 Nginx 服务,可有效缩减潜在攻击面,限制恶意行为的权限范围,从而增强容器的安全。

近期研究表明,Dockerfile 普遍存在质量问题[3-4],高达 68%的镜像存在安全漏洞[5]。传统的 Dockerfile 误配置检测

到稿日期:2024-10-08 返修日期:2024-12-06
通信作者:王金双(siyezhishuang@163.com)

大多采用基于规则的方法，这需要开发人员根据最佳实践规则编写复杂的模板来实现。该方法在检测已有模板的规则时准确率较高，但也受限于规则提取与模板设计，已有检测规则模板不足以应对复杂检测任务。例如在进行 Dockerfile 构建失败预测时，使用 Hadolint[1] 检测的查全率仅为 20.6%[4]，查准率仅有 6.5%[6]。

```
1. FROM ubuntu
2. RUN apt-get update && apt-get install -y nginx
3. ENV DB_PASSWORD=123123
4. COPY public /usr/share/nginx/html
5. EXPOSE 80
6. CMD ["nginx","g","daemon off;"]
```

图 1 默认 root 权限风险示例

Fig. 1 Default root privileges risk example

```
1. FROM ubuntu:18.04
2. RUN apt-get update && apt-get install -y nginx
3. RUN useradd -r -s /bin/false nginxuser
4. ENV DB_PASSWORD=123123
5. COPY public /usr/share/nginx/html
6. EXPOSE 80
7. USER nginxuser
8. CMD ["nginx","-g","daemon off;"]
```

图 2 USER 指令权限优化示例

Fig. 2 USER instruction privilege optimization example

机器学习和深度学习技术已广泛应用于代码缺陷检测领域，它们通过学习代码特征来识别潜在的缺陷模式，展现出了良好的泛化能力。文献[7]通过机器学习算法学习 Dockerfile 语义特征，可有效预测 Docker 容器构建的成败。与传统的基于规则的方法相比，这种方法有效避免了规则挖掘和模板构建所带来的挑战。特征选择对模型性能至关重要，但其定义和提取却高度依赖于领域知识和专家经验。深度学习技术可以自动捕获相关特征，在基础设施即代码(Infrastructure as Code，IaC)的缺陷检测中取得了一定的效果[8]。

文献[9]将深度学习引入 Dockerfile 检测，提出一种基于卷积神经网络和特征金字塔网络(CNN-FPN)的 Dockerfile 风险预测方法。该方法提取 Dockerfile 中的重要指令集和控制流节点作为 Token 序列，通过 Word2Vec 模型将其转化为词向量，训练 CNN-FPN 网络作为分类器。该方法能够有效识别安全风险并对 Dockerfile 进行分类，但存在以下两方面问题：1)仅关注高层级的指令集和控制流节点，而忽视了指令内部的细节，导致无法识别隐藏在指令内部的安全风险；2)仅提取了文本特征(包括结构特征和语义特征)，缺少对度量特征的关注，导致准确率不够高。例如，在图 1 所示文本中，CNN-FPN 提取的 Token 序列为{[FROM][RUN][ENV][COPY][EXPOSE][CMD]}，无法检测到第 3 行环境变量中存在的硬编码风险，也无法识别第 1 行存在的未指定镜像版本的安全风险。提取每个指令及其参数作为文本特征，统计字符串、镜像和标签个数作为度量特征，将有助于此类安全漏洞的检测。

针对以上问题，本文提出名为 DSDC(Dual-Stream Dockerfile Checker)的双流深度学习 Dockerfile 检测方法，通过融合学习度量特征和指令语义，提高对 Dockerfile 安全误配置的识别能力。首先，该方法利用现有检测工具识别并标注含有安全误配置的 Dockerfile 样本，自动生成正负样本集。随后，构建抽象语法树(Abstract Syntax Tree，AST)量化源代码属性提取度量特征，并使用随机森林(Random Forest，RF)算法筛选出关键安全特征。最后，提取安全特征度量信息并输入 Transformer 模型，捕捉特征关联，映射安全配置缺陷；提取文本信息并输入双向长短期记忆网络(Bi-directional Long Short-Term Memory，Bi-LSTM)，挖掘指令序列间的语义关联；采用卷积神经网络(Convolutional Neural Network，CNN)子层与 ReLU 激活函数融合双流信息，对 Dockerfile 安全误配置进行检测。实验结果表明，相较于已有方法，所提方法具有更高的性能。

综上，本文的贡献如下：

1)设计了一个能够自动获取 Dockerfile 特征度量信息和文本信息的工具；

2)筛选出 15 个与安全误配置紧密相关的 Dockerfile 源代码属性，并通过随机森林算法验证了它们的有效性；

3)提出了一种融合特征度量与深度语义理解的双流深度学习检测方法 DSDC，通过实验验证了其整体性能高于已有检测方法。

2 相关工作

1)Dockerfile 的实证研究。文献[10]指出，在高质量数据集中，81.3%的 Dockerfile 至少包含一种安全误配置，而在普通数据集中，这一比例高达 97.6%。文献[11]对 GitHub 中 STAR 排名较高的 10064 个 Dockerfile 项目的 30 多种主流风险类型进行安全问题分析，发现 90%以上的项目至少存在一种安全风险。文献[12]发现，有近 84%的项目受到 Dockerfile 异味(即因违反最佳实践而造成 Docker 容器安全漏洞或性能下降)的影响，且不同类型的 Dockerfile 异味有时会同时出现。文献[13]显示，至少有 48.7%的 Dockerfile 包含膨胀镜像体积的不良实践，使镜像体积平均增加了 4.6%，有时增幅甚至达到了 10%，在降低构建效率的同时还增大了安全风险点。尽管这些实证研究的侧重点不同，但它们都证实了 Dockerfile 质量检查的必要性。

2)Dockerfile 常用检测工具。为应对 Dockerfile 的质量挑战，目前已有多种检测工具，如 Hadolint，KICS[2]，Docker-bench-security[3]，Trivy[4] 和 Checkov[5] 等，它们主要通过预设规则库进行静态分析，以发现潜在问题。文献[10]对这些检测工具进行测评发现：内置安全相关规则较少，不能满足

[1] https://github.com/hadolint/hadolint

[2] https://github.com/Checkmarx/kics

[3] https://github.com/docker/docker-bench-security

[4] https://github.com/aquasecurity/trivy

[5] https://github.com/bridgecrewio/checkov

Docker 官方文档提供的最佳安全实践检测需求;安全规则在整体规则集中占比较低,导致安全告警信息被大量的非安全告警信息淹没,难以引起开发者足够的重视。Hadolint 和 KICS 两个工具,在已有工作[1,10,14-17]中被广泛应用。其中,Hadolint 是 Dockerfile 异味检测的专用工具,其检测准确率较高,可以检测 66 种 Dockerfile 异味,但涉及安全误配置检测的规则较少。KICS 主要针对 IaC 脚本的错误配置进行检测,内含 48 条 Dockerfile 相关规则和 1 条敏感信息检测规则,它采用字符串匹配技术来检测敏感信息,具有较高的误报率,需人工加以核验。

3)Dockerfile 误配置检测技术。如表 1 所列,Dockerfile 误配置检测技术大致可以分为基于规则模板和基于特征学习两大类。该技术的初期研究主要集中于基于规则模板的方法上,研究人员大多围绕设计出更为严密的规则执行算法,来提高检测的准确性,典型代表有 Binnacle[3]、Hadolint 和 DRIVE[18]。其中,Binnacle 将 Dockerfile 解析为 AST,使用预定义的树关联规则模板进行检测;Hadolint 采用正则匹配进行检测;DRIVE 将 Dockerfile 转换为序列化的中间表示,并制定语义逻辑规则进行检测。这些方法的检测准确度较高,但高度依赖预置规则模板,现有规则模板难以应对复杂任务场景。鉴于此,人们开始尝试提取 Dockerfile 特征,使用机器学习或深度学习技术进行检测,如 PDBR[7] 和 DRPA[9]。PDBR 利用抽象语法树提取文本语义特征,通过多种机器学习算法来预测 Dockerfile 是否能够成功构建,其 AUC 值可达 0.72;DRPA 采用 CNN-FPN 模型,将 Dockerfile 关键节点作为输入进行风险预测,检测准确率可达 0.87。这些方法通过学习特征间的复杂非线性关系进行检测,打破了规则模板的限制,但它们提取的特征相对单一,检测准确率还有待提升。此外,该类方法需要大量带有标签的样本用于模型训练,目前缺少此类公开数据集。

表 1　Dockerfile 误配置检测方法分析

Table 1　Analysis of dockerfile misconfiguration detection methods

类别	方法	应用技术	优势	局限性
基于规则模板	Binnacle[3]	树关联规则匹配	检测预置规则的准确率高	高度依赖规则模板,现有规则模板无法应对复杂任务
	Hadolint	正则匹配		
	DRIVE[18]	自定义语义逻辑		
基于特征学习	PDBR[7]	机器学习	可通过特征学习来自动识别潜在错误模式	特征提取单一,准确率有待提升;缺少训练所需的公开数据集
	DRPA[9]	深度学习		

4)双流深度学习模型的应用。双流深度学习模型在代码检测领域应用广泛。文献[19]提取语义和度量特征,训练 GCN-Transformer 模型进行可操作代码异味检测。文献[20]提取文本信息和度量信息,训练 DeepSmell 双流模型检测代码异味。文献[21]提取代码文本特征和距离度量,训练 Bi-LSTM 模型进行特征嫉妒代码异味检测。实验结果显示,相比于单信息维度的传统模型,多信息维度的双流模型具有更好的检测性能。此外,文献[22]将多种机器学习算法与度量指标进行组合实验,发现 RF 在识别 IaC 脚本异味方面表现最优。文献[23-24]使用 IaC 脚本的关键度量特征,训练 RF 分类器进行配置安全检测,准确率达 0.78,体现出使用度量特征进行 IaC 脚本配置检测的有效性。因此,本文同时提取 Dockerfile 的文本特征和度量特征,采用双流模型进行安全误配置检测,以提高模型检测性能。

3　DSDC 方法

DSDC 方法框架如图 3 所示。首先,采用静态检测工具识别存在安全误配置的 Dockerfile 样本并进行分类标记,生成正负样本集。然后,构建抽象语法树,选取树的每个节点作为 Token,遍历节点生成文本信息,并提取关键安全特征度量信息。将文本信息与度量信息结合作为神经网络分类器的输入,分类器的预期输出为样本的标签(即文件是否存在安全误配置)。在训练集上进行迭代训练,生成分类器;在测试集上验证分类器性能,并输出检测结果。

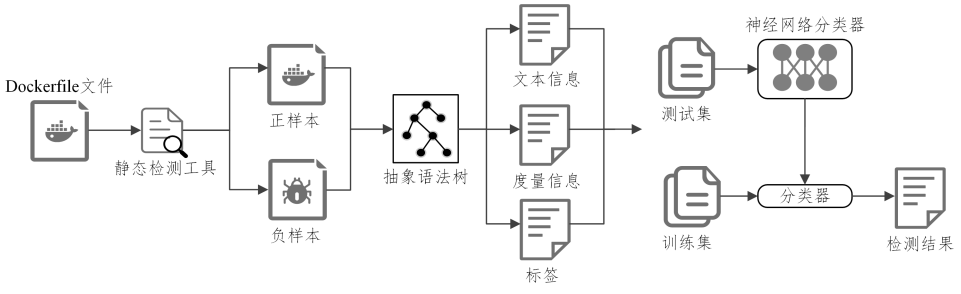

图 3　DSDC 框架

Fig. 3　Framework of DSDC

3.1　样本自动标注

深度学习需要大量带有标签的数据集对模型进行训练,而在 Dockerfile 检测研究领域,目前尚无此类公开数据集供研究者使用。在代码异味检测领域也存在类似问题,文献[25]指出,有 60%的研究借助现有检测工具对数据集进行标记,以生成模型训练集。本文借鉴该思路,利用 Hadolint 和 KICS 工具自动化识别引发 Dockerfile 安全漏洞的错误配置,并以此作为样本自动分类的基础。由于工具检测存在误差,尤其是 KICS 在进行敏感信息检测时存在较高的误报率,本文方法针对工具判断不一致的样本以及检测显示存在敏感

信息的样本进行人工核验，由此生成正负样本集。

为审计 Docker 容器镜像的安全性，文献[26]开发了一组漏洞识别用例，包括 Docker 基础镜像检查、Dockerfile 配置、镜像认证 3 个方面。综合考量以上用例并结合 Dockerfile 工具检测范围，主要针对 6 类易引起安全漏洞的错误配置进行检测，如表 2 所列。

表 2 Dockerfile 安全误配置检测规则

Table 2 Dockerfile security misconfiguration detection rules

序号	检测规则	检测工具	安全危害	度量特征	
1	存在敏感信息	KICS		敏感信息泄露	字符串、RUN、ENV、ARG
2	未明确镜像版本	KICS	Hadolint	基础镜像引入安全漏洞	FROM、镜像、标签
	使“Latest”镜像	KICS	Hadolint		
	使用未注册镜像		Hadolint		
3	最后用户为“root”	KICS	Hadolint	权限过大	USER, RUN
	缺少 USER 指令	KICS			
	RUN 中使用 Sudo 命令	KICS	Hadolint		
4	暴露 22 号端口（SSH）	KICS		端口暴露	ENV
5	滥用 ADD 替代 COPY	KICS	Hadolint	自动提取功能易被恶意代码利用	ADD, COPY, Curl, Wget, URL
	未用 Curl 或 Wget 替代 ADD	KICS			
6	未在同层镜像中更新	KICS	Hadolint	安装的软件引入安全漏洞	Install, Add
	未明确软件安装版本	KICS	Hadolint		

3.2 文本信息提取

抽象语法树是一种能够体现代码语法结构和语义信息的树状抽象表示，在构建 Dockerfile 文本表示时，其节点被认为是最佳基本单元，因为它们既保留了语义信息，又保留了结构信息[9]。本文调用 dockerfile-parse[1)] 和 mvdan-sh[2)] 解析库来构建 Dockerfile 的抽象语法树，并选取它的每一个节点作为一个 Token，采用前序遍历策略生成 Token 序列作为 Dockerfile 的文本信息。不同于文献[7,9]仅选取关键节点作为 Token 的方法，全节点覆盖可以保留 Dockerfile 中的所有实现细节，增强了模型捕捉细微差异的能力。图 4 展示了一个简单的 Dockerfile 文件，其对应的 Token 序列为：{[FROM][python][3.8-slim][WORKDIR][/app][COPY][./app][RUN][pip][install][--no-cache-dir][-r][requirement.txt][ENV][FLASK_APP][app.py][EXPOSE][5000][CMD][python][app.py]}。

```
1. FROM python:3.8-slim
2. WORKDIR/app
3. COPY ./app
4. RUN pip install --no-cache-dir -r requirement.txt
5. ENV FLASK_APP=app.py
6. EXPOSE 5000
7. CMD ["python","app.py"]
```

图 4 简单 Dockerfile 示例

Fig. 4 A simple Dockerfile example

3.3 度量信息提取

综合考量 IaC 脚本度量特征提取工作[22,27-29]，初步提取一系列与 Dockerfile 安全误配置紧密相关的特征（见表 2），以及表征 Dockerfile 复杂度的特征（如代码总行数、脚本执行命令及其参数的使用情况、文件路径的设定次数、FOR 和 IF 等复杂语句个数）作为度量特征。在此基础上，借鉴 IaC 脚本度量特征提取方法[27]，进一步精选影响 Dockerfile 安全误配置检测的关键特征。首先，采用 Mann-Whitney U 检验进行初步筛选，过滤掉正负样本中差异不明显的特征；然后，应用 Cliff's Delta 方法量化剩余特征的有效性，进一步精炼关键特征；最后，构建并训练 RF 分类模型，依据模型输出的特征权重，选取对安全误配置检测贡献较大的特征。通过遍历 Dockerfile 的 AST，统计选定特征的出现频次，提炼出反映 Dockerfile 安全状况的度量信息。

3.4 双流检测模型

如图 5 所示，该模型主要由文本特征表示、度量特征表示、特征融合分类 3 个模块组成。

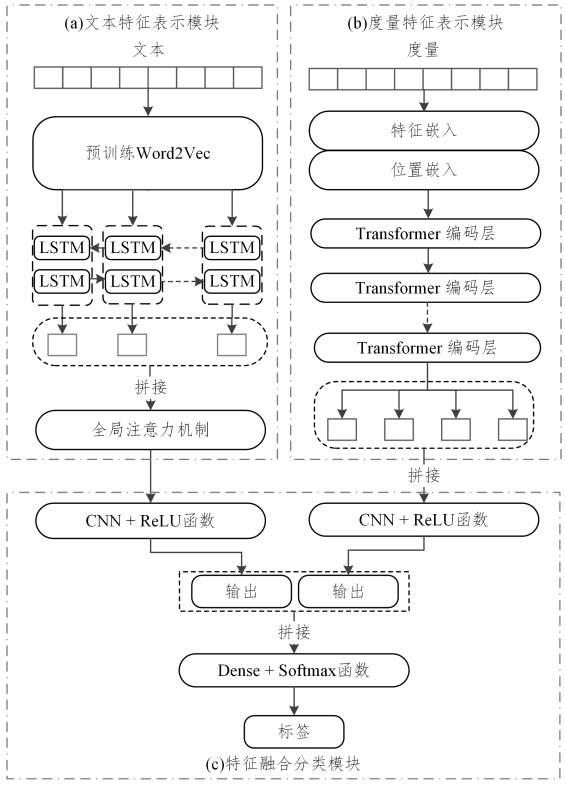

图 5 双流检测模型结构

Fig. 5 Structure of dual-stream detection model

1) https://github.com/moby/buildkit/blob/master/frontend/dockerile/parser

2) https://github.com/mvdan/sh

文本特征表示模块接收文本信息，通过 Word2Vec[30] 模型将文本信息映射为词向量，然后使用 Bi-LSTM 模型挖掘文本中的语义特征，拼接语义特征并使用全局注意力机制聚焦关键特征，生成文本特征表示流。度量特征表示模块接收度量信息，进行度量嵌入和位置嵌入，以获取度量向量和位置向量，然后使用 Transformer 模型提取度量中的数据特征并进行拼接，生成度量特征表示流。特征融合分类模块接收文本特征表示流和度量特征表示流，通过 CNN 子层和 ReLU 非线性激活函数来降低双流特征维度，然后将双流特征拼接后送入由密集层和 Softmax 激活函数组成的分类器中进行样本分类。

3.4.1　文本特征表示模块

Dockerfile 的结构具有指令间有序（指令须按序执行）而指令内命令参数无序（参数可能出现在命令前、命令后或命令中）的特性，其指令及参数富含语义信息，不同的排列组合会直接影响容器构建的结果。此外，Dockerfile 的局部细微差别很可能导致容器构建的巨大差异，甚至产生安全风险。为了能够从 Token 序列中有效捕获 Dockerfile 语义信息和结构特性，本文设计了一个文本特征表示模块。如图 5(a)所示，文本特征表示模块由 Word2Vec 词嵌入层、Bi-LSTM 层和注意力层组成。

首先，采用 Word2Vec 模型对 Token 序列进行词嵌入，获得其高维向量表示。Word2Vec 是一种通过预测邻近词来训练的词嵌入模型，可以使语义相似的 Token 在向量空间中彼此接近。该模型在代码检测领域应用广泛。

然后，利用深度学习模型进一步挖掘 Token 序列中的依赖关系，获得文本特征表示。Transformer 和 LSTM 模型都擅长处理序列数据，但均不能满足 Dockerfile 的特性需求。Transformer 能够有效应对指令的有序性和参数的无序性，但不擅长捕获局部微小顺序差异。LSTM 能够有效应对指令的有序性，捕获局部微小顺序差异，但它只能利用先前的信息来更新当前的状态，缺乏对未来上下文的考量，无法有效应对参数的无序性。因此，本文选用 Bi-LSTM 挖掘 Token 序列的依赖关系。Bi-LSTM 是一种双向 LSTM 结构，相较于传统的单向 LSTM，它在捕捉文本序列中的双向语义信息方面更具优势。该模型新增了一个反向传播的 LSTM 层，可以对输入的 Token 序列进行双向训练，既能通过其循环结构保持对历史信息的记忆，又能结合双向传播机制同时考虑指令序列的前序依赖和后续影响，产生多于 LSTM 两倍的特征，确保了对指令逻辑结构和依赖关系的全面理解。

最后，将文本特征拼接后送入注意力层，提取关键特征，生成文本特征表示流。注意力机制在序列的不同位置赋予不同的权重，从而突出关键信息。该过程可以描述为：

$$\boldsymbol{e}_i=\tanh(\boldsymbol{h}_i*\boldsymbol{w}_i+b) \tag{1}$$

$$\boldsymbol{\alpha}_i=\exp(\boldsymbol{e}_i)/\sum_{j=1}^{n}\exp(\boldsymbol{e}_j) \tag{2}$$

$$\boldsymbol{c}_i=\sum_{i=1}^{n}\boldsymbol{\alpha}_i\boldsymbol{h}_j \tag{3}$$

其中，$\boldsymbol{h}_i$ 表示 Bi-LSTM 在时间步 i 的隐藏状态向量；$\boldsymbol{w}_i$ 和 b 分别是权重参数和偏置项；$\boldsymbol{e}_i$ 是计算注意力权重的中间值；$\boldsymbol{\alpha}_i$ 是在时间步 i 上的注意力权重；$\boldsymbol{c}_i$ 是经过加权求和后得到的上下文向量，用于后续的任务处理。

3.4.2　度量特征表示模块

Dockerfile 的度量特征可以间接反映容器安全。例如，频繁使用 ADD 而非 COPY 指令会增加存档文件漏洞及文件下载遭遇中间人攻击的风险；大量使用 RUN 指令堆砌操作则预示镜像构建过程缺乏优化，增加了攻击面。这些特征种类多样，某些特征的样本值可能为零，特征之间的关系复杂，具有高维稀疏和全局依赖等特性。为了有效捕捉和处理这些度量特征，设计了一个度量特征表示模块。如图 5(b)所示，该模块由多个堆叠的 Transformer 编码器层组成。

Transformer 是一种鲁棒的序列建模架构，尤其擅长处理序列数据。其自注意力机制能够捕捉特征之间的全局依赖关系，动态调整特征权重，突出关键特征并减少噪声影响；其多层编码可以逐步提取更高层次的特征表示，实现从特征到目标任务的映射。

本文将 Dockerfile 特征度量视为序列数据，序列中的每个度量指标作为一个 Token 映射到高维向量矩阵。首先，使用随机初始化的矩阵对度量序列进行嵌入，获取高维度量嵌入向量。然后，通过位置嵌入给序列中的每个度量分配位置信息，确保相同数值的度量在序列中能够被区分。最后，将这些带有位置编码的数据输入 Transformer 编码器，生成与输入相同大小的输出特征，并将这些特征向量拼接，以生成度量特征表示流。

如图 6 所示，Transformer 层不仅能够自动计算每个度量与其他度量之间的相关性，挑选最相关度量；还能生成全局度量信息，反映所有度量之间的相互关系。通过级联多个 Transformer 编码层，可以逐步提取更高层次的度量特征，从而实现从度量到 Dockerfile 安全误配置的复杂映射。

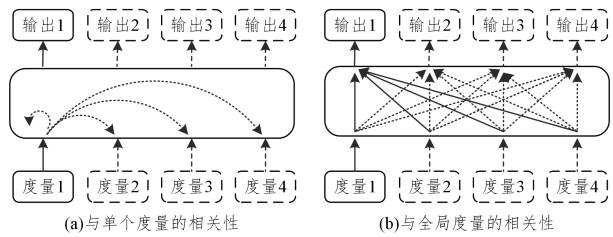

图 6　Transformer 度量相关性示意图

Fig. 6　Transformer metric correlation diagram

3.4.3　特征融合分类模块

为了克服度量特征序列和文本特征序列在对齐和交互上的困难，设计了一个特征融合分类模块。该模块接收由 Transformer 生成的度量特征表示流和由 Bi-LSTM 生成的文本特征表示流。首先，将双流分别送入由 CNN 子层和 ReLU 非线性激活函数组成的输出层中，以降低双流特征向量维度并保持统一。然后，将拼接输出的双流特征送至由密集层和 Softmax 函数组成的分类器中。分类器使用 Focal Loss[31] 作为损失函数，以解决样本不平衡问题。输出层和损失函数的定义如下：

$$output=ReLU((f_1\parallel f_2\parallel\cdots\parallel f_n)W),W\in\mathbb{R}^{nd\times d'} \tag{4}$$

$$Loss=\frac{\alpha}{N}\sum_{1}^{N}(1-\mathrm{e}^{-BCE})^{\gamma}\cdot BCE \tag{5}$$

4 实验验证

首先，通过度量特征优化实验来筛选对检测任务贡献较大的关键度量特征，并使用 RF 模型证明其有效性。然后，将所提模型与现有模型以及传统检测方法进行对比，证明了所提模型的显著优势。最后，进行消融实验，评估模型中关键组件的贡献，进一步证明了所提双流模型的优越性。

4.1 实验环境及超参数

实验在一台搭载了 Ubuntu Server 20.04 LTS 64 位操作系统的服务器上进行，其配备了 Intel Xeon 2.1 GHz 80 核 CPU，NVIDIA Corporation 3090 显卡和 128 GB RAM。所有模型均基于 PyTorch 框架实现，并通过随机欠采样方法来平衡数据集，深度学习模型采用 Focal Loss 函数来强化学习难分类样本。重复 3 轮实验，取平均值作为最终结果。模型具体参数如表 3 所列。

表 3 模型参数设置

Table 3 Model parameter settings

模型	超参数	值
Bi-LSTM	令牌嵌入维度	128
	学习率	0.001
	训练轮次	100
	批大小	64
Transformer	编码层数	3

4.2 数据集

实验主要使用了 DRIVE 提供的公开数据集，内含 1 761 个高质量的 Dockerfile 文件(D1)，以及 12 066 个在 github 上采集的原始 Dockerfile 文件(D2)。其中，D1 用于训练 Word2Vec 模型，D2 用于生成检测所需的正负样本集。

4.3 基线模型

选取 RF，DNN，CNN-FPN 和 Bi-LSTM 作为基准模型。

RF 使用度量特征来训练决策树作为子分类器，再通过投票机制做出最终分类结果。这是一种基于度量的检测方法，不考虑代码语义。文献[22]提取 IaC 脚本的关键度量特征，使用 RF 进行配置安全检测。本实验将 Dockerfile 度量序列作为 RF 的输入，输出二值分类结果，用于验证所选度量特征的有效性。

DNN 主要由全连接层堆叠而成，也是一种基于度量的检测方法。文献[32]将其用于代码异味检测，来有效识别长方法气味。本实验将 DNN 应用于 Dockerfile 安全误配置检测，模型接收 Dockerfile 度量序列作为输入，输出二值分类结果。

CNN-FPN 模型[9]使用 FPN 融合多层特征，用于 Dockerfile 风险预测。与前两者不同，CNN-FPN 是一种基于文本特征的检测方法，接收 Dockerfile 的文本序列作为输入，输出二值分类结果。

Bi-LSTM 模型是一种同时利用文本特征和度量特征的双流检测方法。文献[28]将类名和方法名作为文本特征，将距离度量作为度量特征，然后使用基于注意力的 Bi-LSTM 来识别代码中的特征嫉妒。如图 7 所示，该方法与所提方法最为相似，均是基于双流架构。本文将其与 Dockerfile 安全误配置检测进行适配，模型接收文本序列和特征度量作为输入数据，输出二值分类结果。

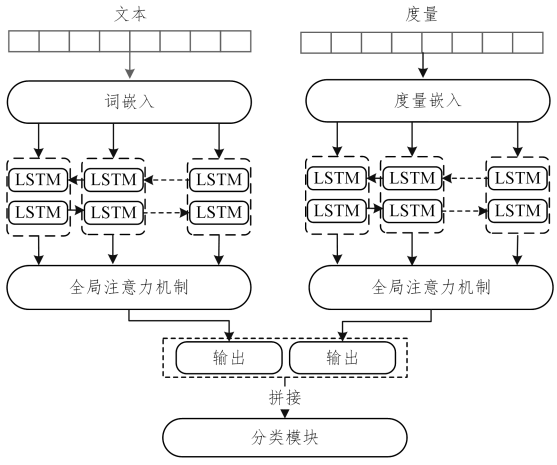

图 7 Bi-LSTM 代码检测双流模型架构

Fig. 7 Bi-LSTM code detection dual stream model

RF，DNN 和 Bi-LSTM 3 种模型为代码异味检测常用模型，本文已将其复现并应用于 Dockerfile 安全误配置检测，其参数设置如表 4 所列。CNN-FPN 模型为检测 Dockerfile 风险而提出，与本文模型检测目标最为相似，因其未开源代码，故直接采用原文实验结果进行对比。

表 4 基线模型参数设置

Table 4 Baseline model parameter settings

模型	超参数	值
RF	决策树个数	128
	最大深度	16
DNN Bi-LSTM	令牌嵌入维度	128
	学习率	0.001
	训练轮次	100
	批大小	64

4.4 评估指标

采用查准率(Precision，P)、查全率(Recall，R)、F1 值、ROC 曲线面积(Area Under the ROC Curve，AUC)等指标来评估方法性能。P 反映了预测结果的精确程度；R 代表分类器找到所有正例的能力；F1 值是查准率和查全率的调和平均值，用于综合评价分类器的性能；AUC 反映了分类器区分正负样本的能力，其值越接近于 1，表示分类器性能越好。AUC 的计算式较复杂，使用内置函数来计算；P，R 和 F1 的计算式如下：

$$P=\frac{TP}{TP+FP} \tag{6}$$

$$R=\frac{TP}{TP+FN} \tag{7}$$

$$F1=2\times\frac{P\times R}{P+R} \tag{8}$$

其中，TP 表示将存在安全误配置的 Dockerfile 文件预测为存在的数量；FP 表示将不存在安全误配置的 Dockerfile 文件预测为存在的数量；FN 表示将存在安全误配置的 Dockerfile 文件预测为不存在的数量。

4.5 度量特征优化实验

构建 AST 统计每个初选特征出现的频次，使用 Mann-Whitney U 检验比较正负样本。设置显著水平 $\alpha=0.05$，若 $p<\alpha$，则表明正负样本在该特征上有较大差异。选取 $p<$

0.05 的属性，使用 Cliff 's Delta 方法对特征差异进行量化，量化结果分为大、中、小、可忽略 4 个等级。选取隶属于前 3 个等级的特征训练 RF 获得特征权重，将权重大于 0.02 的特征作为关键度量特征。如图 8 所示，最终获得 15 个关键度量特征。

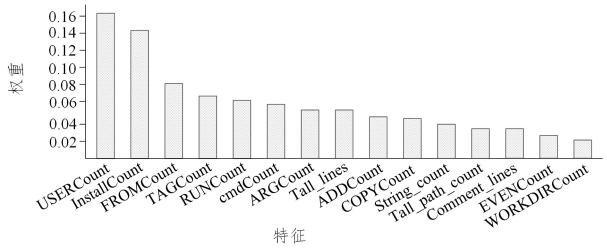

图 8　关键度量特征权重

Fig. 8　Key metric features weights

4.6　双流模型性能评估实验

从 D2 数据集中随机抽取 2000 个样本，分别采用本文方法、基线模型和传统方法进行安全误配置检测。本文方法与基线模型的实验结果对比如图 9 所示，本文方法与传统方法的实验结果对比如图 10 所示。

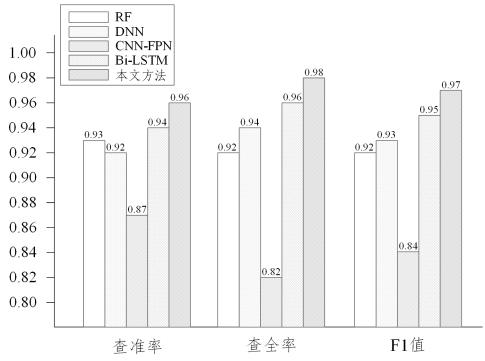

图 9　双流模型与基线模型的检测性能比较

Fig. 9　Performance comparison of dual stream model and baseline model

由图 9 可知，RF 模型使用度量特征进行检测，取得了较好的效果，其查准率、查全率和 F1 值分别达到了 0.93，0.92 和 0.92，表明了所筛选的 15 个关键安全度量特征的有效性。

Bi-LSTM 模型的整体性能明显优于 DNN 模型和 CNN-FPN 模型。其主要原因在于，Bi-LSTM 模型融合了文本信息与度量信息进行双流信息检测，而 DNN 模型和 CNN-FPN 模型都是单流模型，在进行检测时仅关注度量信息或文本信息中的一种。在代码检测领域，双流模型的性能高于单流模型，实验结果表明，此结论在 Dockerfile 检测领域同样适用。

对比 Bi-LSTM 双流模型，本文方法检测查准率、查全率和 F1 值均提升了 0.02，证实了该模型在 Dockerfile 检测任务上的优越性。这是由于 Bi-LSTM 模型在处理文本信息时，所使用的 LSTM 模型是单向的，不能同时获取 Token 序列的前后依赖关系；在处理度量信息时，所使用的 LSTM 模型更关注序列的顺序关系，无法有效捕捉度量特征间的相互依赖关系。本文采用 Bi-LSTM 模型，同时处理文本的双向信息，获取 Token 序列间的前后依赖；采用 Transformer 模型，利用其自注意力机制捕捉特征之间的全局依赖关系，动态调整特征权重；采用 CNN 子层和 ReLU 激活函数，统一特征维度，融合双流信息，提高模型的整体检测性能。

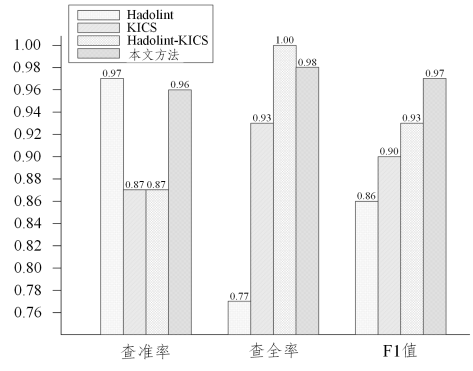

图 10　双流模型与传统方法的检测性能比较

Fig. 10　Performance comparison of dual stream model and traditional methods

由图 10 可知，对比传统基于规则的检测方法，本文方法的检测性能具有较大提升。由于 Hadolint 和 KICS 工具仅能检测部分安全误配置的规则，因此单独使用任一工具检测时，查全率较低；Hadolint 工具相比 KICS 工具，规则覆盖率更小，因而查全率也更低，仅为 0.77；KICS 在进行敏感信息检测时容易产生误报，导致查准率较低，仅为 0.87；同时使用双工具进行检测时，实验将出现歧义的样本归为异常样本，造成了较高的查全率和较低的查准率。

4.7　消融实验

为了评估本文方法中双流模型关键组件对识别 Dockerfile 安全误配置的贡献，本节进行了 6 组消融实验。实验沿用 4.6 节实验的数据集，实验结果如图 11 所示。

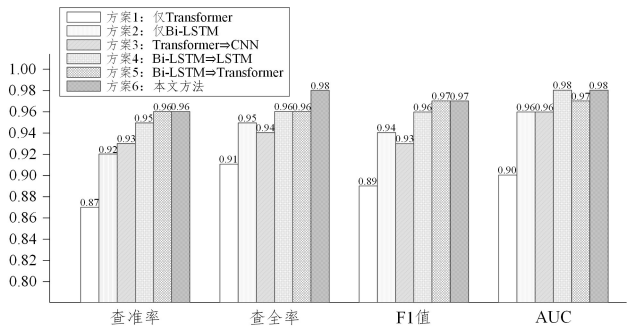

图 11　消融实验结果比较

Fig. 11　Comparison of ablation study results

通过比较方案 1、方案 2 和方案 6 的实验结果，可以观察到每个流在双流模型中的贡献。当移除任何一流时，模型的整体性能均显著下降，这进一步证实了融合度量特征和语义特征可有效提升模型性能。

通过比较方案 3、方案 4、方案 5 和方案 6 的实验结果，可以观察到替换模型中的关键组件对模型性能产生的影响。将 Transformer 替换成 CNN、将 Bi-LSTM 替换成 LSTM，或将 Bi-LSTM 替换成 Transformer 时，模型的整体性能均有所下降。其中，CNN 擅长提取局部特征，无法准确捕捉 Dockerfile 度量信息间的全局依赖关系，因此将 Transformer 替换成 CNN 时，模型性能下降最多，查准率、查全率、F1 值和 AUC

分别下降 0.03,0.04,0.04,0.02。相较于 Bi-LSTM,LSTM 只能进行正向推理,无法获取序列间的逆向依赖关系;Transformer 更擅长捕捉长距离依赖关系,导致部分局部微小顺序依赖丢失,因此将 Bi-LSTM 替换成 LSTM 或 Transformer 来处理 Token 序列时,模型整体性能下降。由此可见,Transformer 和 Bi-LSTM 是现有模型的最佳组合。

结束语 本文提出一种基于双流模型的 Dockerfile 安全误配置检测方法 DSDC。该方法使用静态检测工具标注样本,构建抽象语法树提取文本信息和度量信息,设计双流模型进行安全误配置检测。实验表明,与目前已有的检测方法相比,DSDC 具有更高的 F1 值。本文使用的数据集将 Dockerfile 文件分为存在安全误配置和不存在安全误配置两种标记,后续研究将进一步细分存在的误配置类型,进行多分类检测。此外,本文所使用的数据集是依据与安全相关的规则违反情况,使用静态检测工具提取的。该数据验证了所提方法的有效性,但同时也限制了所训双流模型检测已知规则之外的安全问题的能力。后续将通过多种技术手段(如检测容器安全)对现有公开数据集进行分类,构建更加权威、全面的数据集,以拓展模型的检测能力。

参 考 文 献

[1] CITO J, SCHERMANN G, WITTERN J E, et al. An Empirical Analysis of the Docker Container Ecosystem on GitHub[C]// 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017: 323-333.

[2] WU Y W, ZHANG Y, WANG T, et al. Development Exploration of Container Technology Through Docker Containers: A Systematic Literature Review Perspective[J]. Journal of Software, 2023, 34(12): 5527-5551.

[3] HENKEL J, BIRD C, LAHIRI S K, et al. Learning from, Understanding, and Supporting DevOps Artifacts for Docker[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ACM, 2020: 38-49.

[4] HENKEL J, SILVA D, TEIXEIRA L, et al. Shipwright: A Human-in-the-Loop System for Dockerfile Repair[C]// 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 1148-1160.

[5] WIST K, HELSEM M, GLIGOROSKI D. Vulnerability Analysis of 2500 Docker Hub Images[C]// Advances in Security, Networks, and Internet of Things: Proceedings from SAM'20, ICWN'20, ICOMP'20, and ESCS'20. Springer, 2021: 307-327.

[6] LI M, BAI X, MA M, et al. DockerMock: Pre-build detection of dockerfile faults through mocking instruction execution[J]. arXiv: 2104.05490, 2021.

[7] WU Y, ZHANG Y, CHANG J, et al. Using Configuration Semantic Features and Machine Learning Algorithms to Predict Build Result in Cloud-Based Container Environment[C]// 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2020: 248-255.

[8] BOROVITS N, KUMARA I, KRISHNAN P, et al. DeepIaC: deep learning-based linguistic anti-pattern detection in IaC[C]// Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation. 2020: 7-12.

[9] SHAO S S, LI K, RAO H C, et al. Research on a Docker risk prediction method based on deep learning[J]. Journal of Nanjing University of Posts and Telecommunications (Natural Science Edition), 2021, 41(2): 104-112.

[10] DE GIORGI L A. Security Misconfigurations Detection and Repair in Dockerfile[D]. Torino: Politecnico di Torino, 2022.

[11] HAO J, LU H, JIANG Y, et al. DFScan: Security Scanner of the Dockerfile Based on Instruction Coverage and Attack Perspective[J]. Human-centric Computing and Information Sciences, 2024, 14: article 10.

[12] WU Y, ZHANG Y, WANG T, et al. Dockerfile Changes in Practice: A Large-Scale Empirical Study of 4,110 Projects on GitHub[C]// 2020 27th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2020: 247-256.

[13] DURIEUX T. Empirical Study of the Docker Smells Impact on the Image Size[C]// Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024: 1-12.

[14] WU Y. Exploring the relationship between dockerfile quality and project characteristics[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. ACM, 2020: 128-130.

[15] ZHANG Y, VASILESCU B, WANG H, et al. One Size Does Not Fit All: An Empirical Study of Containerized Continuous Deployment Workflows[C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2018: 295-306.

[16] SCHERMANN G, ZUMBERI S, CITO J. Structured information on state and evolution of dockerfiles on github[C]// Proceedings of the 15th International Conference on Mining Software Repositories. ACM, 2018: 26-29.

[17] BUI Q C, LAUKOTTER M, SCANDARIATO R. DockerCleaner: Automatic Repair of Security Smells in Dockerfiles[C]// 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2023: 160-170.

[18] ZHOU Y, ZHAN W, LI Z, et al. DRIVE: Dockerfile Rule Mining and Violation Detection[J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 1-23.

[19] YU D J, YANG Q X, CHEN X, et al. Actionable code smell identification with fusion learning of metrics and semantics[J]. Science of Computer Programming, 2024, 236: 103110.

[20] ZHANG Y, DONG C H, LIU H, et al. Code Smell Detection Approach Based on Pre-training Model and Multi-level Information[J]. Journal of Software, 2022, 33(5): 1551-1568.

[21] WANG H, LIU J, KANG J, et al. Feature Envy Detection based on Bi-LSTM with Self-Attention Mechanism[C]// 2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE, 2020: 448-457.

[22] DALLA PALMA S, DI NUCCI D, PALOMBA F, et al. Within-

Project Defect Prediction of Infrastructure-as-Code Using Product and Process Metrics[J]. IEEE Transactions on Software Engineering,2022,48(6):2086-2104.

[23] RAHMAN A,WILLIAMS L. Characterizing Defective Configuration Scripts Used for Continuous Deployment[C]// 2018 IEEE 11th International Conference on Software Testing,Verification and Validation (ICST). IEEE,2018:34-45.

[24] RAHMAN A,WILLIAMS L. Source Code Properties of Defective Infrastructure as Code Scripts[J]. Information and Software Technology,2019,112:148-163.

[25] ZHANG Y,GE C,LIU H,et al. Code smell detection based on supervised learning models: A survey[J]. Neurocomputing, 2024,565:127014.

[26] AHAMED W S S,ZAVARSKY P,SWAR B. Security Audit of Docker Container Images in Cloud Architecture[C]// 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC). IEEE,2021:202-207.

[27] RAHMAN A,WILLIAMS L. Source Code Properties of Defective Infrastructure as Code Scripts[J]. Information and Software Technology,2019,112:148-163.

[28] DALLA PALMA S,DI NUCCI D,TAMBURRI D A. AnsibleMetrics:A Python library for measuring Infrastructure-as-Code blueprints in Ansible[J]. SoftwareX,2020,12:100633.

[29] VAN DER BENT E,HAGE J,VISSER J,et al. How good is your puppet? An empirically defined and validated quality model for puppet[C]// 2018 IEEE 25th International Conference on Software Analysis,Evolution and Reengineering (SANER). IEEE,2018:164-174.

[30] YILMAZ S,TOKLU S. A deep learning analysis on question classification task using Word2vec representations[J]. Neural Computing and Applications,2020,32(7):2909-2928.

[31] MUKHOTI J,KULHARIA V,SANYAL A,et al. Calibrating deep neural networks using focal loss[J]. Advances in Neural Information Processing Systems,2020,33:15288-15299.

[32] LIU H,JIN J,XU Z,et al. Deep learning based code smell detection[J]. IEEE Transactions on Software Engineering, 2019, 47(9):1811-1837.

ZHAO Ning, born in 1993, postgraduate. Her main research interest is system security.

WANG Jinshuang, born in 1978, Ph.D, associate professor. His main research interest is system security.

(责任编辑:柯颖)