移动 APP 多核架构安全漏洞并行检测

严文昊[1*]，王宏岩[2]，董　蓓[1]，曲　艺[1]，孙德艳[2]
（1. 国家电网有限公司客户服务中心，天津 300300；
2. 北京中电普华信息技术有限公司，北京 100031）

摘　要：移动 APP 多核架构的安全漏洞具有多样化、大规模特征，现有检测方法受限于参数选择适应性和特征提取的准确性，难以准确并行检测移动 APP 的多种漏洞。为了解决移动 APP 多核架构下大规模安全漏洞数据的准确检测问题，提出了一种基于 SimHash 算法和 LightGBM 的移动 APP 多核架构安全漏洞并行检测方法。该方法利用 SimHash 算法对漏洞数据进行特征提取和编码。利用 TF-IDF 算法融合 Jaccard 指数改进 SimHash 算法，优化特征词的权重分配计算，生成唯一特征哈希值。结合 LightGBM 算法构建二分类器，判断漏洞是否存在可利用的 EXP（漏洞利用代码）。利用贝叶斯超参数优化 LightGBM 算法，通过多核架构的并行处理，实现对大量漏洞数据的准确检测。通过实验表明，该方法针对多种常见漏洞的 MAE 值、RMSE 值、MAPE 值、R^2 值分别为 0.032、1.017、0.124%、0.976，有效提升了漏洞检测的适应性、精度、稳定性和拟合能力，为移动 APP 的安全管理提供了有力支持。

关键词：SimHash 算法；LightGBM；移动 APP；多核架构；安全漏洞；并行检测
中图分类号：TP391.4　　**文献标识码**：A　　**文章编号**：1673-629X(2025)02-0079-07
doi：10.20165/j.cnki.ISSN1673-629X.2024.0314

Parallel Detection of Security Vulnerabilities in Multi-core Architecture of Mobile APP

YAN Wen-hao[1*]，WANG Hong-yan[2]，DONG Bei[1]，QU Yi[1]，SUN De-yan[2]
（1. State Grid Customer Service Centre，Tianjin 300300，China；
2. Beijing China Power Information Technology Co.，Ltd.，Beijing 100031，China）

Abstract：The security vulnerabilities of multi-core architecture in mobile APP have diverse and large-scale characteristics. Existing detection methods are limited by the adaptability of parameter selection and the accuracy of feature extraction，making it difficult to accurately detect multiple vulnerabilities in mobile APP in parallel. In order to solve the problem of accurate detection of large-scale security vulnerability data in mobile APP multi-core architecture，a parallel detection method for security vulnerabilities in mobile APP multi-core architecture based on SimHash algorithm and LightGBM is proposed. This method utilizes SimHash algorithm for feature extraction and encoding of vulnerability data. Using TF-IDF algorithm and Jaccard index to improve SimHash algorithm，the weight allocation calculation of feature words is optimized，and the unique feature hash values are generated. A binary classifier is built by the LightGBM algorithm to determine if there are exploitable EXP（exploit code）vulnerabilities. By utilizing Bayesian hyperparameters to optimize the LightGBM algorithm and parallel processing with a multi-core architecture，accurate detection of large amounts of vulnerability data can be achieved. Through experiments，it has been shown that the proposed method for various common vulnerabilities have MAE values，RMSE values，MAPE values，and R^2 values of 0.032，1.017，0.124%，and 0.976，respectively，which effectively improves the adaptability，accuracy，stability，and fitting ability of vulnerability detection，providing strong support for the security management of mobile APP.

Key words：SimHash algorithm；LightGBM；mobile APP；multi-core architecture；security vulnerabilities；parallel detection

0　引　言

随着移动 APP 功能的日益复杂和多样化，其安全漏洞问题日益凸显。尤其是在多核架构下，移动 APP 的安全漏洞数据呈现出爆炸性增长，给安全检测工作带来了极大的挑战[1]。因此，如何高效地处理和准确识别移动 APP 多核架构下的安全漏洞数据，成为当前亟待解决的问题[2-3]。传统的安全漏洞检测方法往往依赖于人工审计和规则匹配，这些方法在处理大规模

收稿日期：2024-07-23　　**修回日期**：2024-11-27
作者简介：严文昊（1979-），男，通讯作者，硕士，高级工程师，研究方向为电子计算机、信息化技术等。

数据时效率低下,且难以适应复杂多变的漏洞模式。现今大数据和机器学习技术蓬勃发展,基于机器学习的安全漏洞检测方法逐渐崭露头角。这些方法能够自动提取漏洞数据的特征,构建分类器模型,实现对安全漏洞的快速检测和识别。在现有研究中,一些学者提出了基于哈希算法和机器学习算法的安全漏洞检测方法。哈希算法,如 MD5 和 SHA-1,被广泛用于数据压缩和唯一性验证,但在处理相似漏洞时效果不佳。

为了克服传统模型和现有模型的局限性,学者们针对移动 APP 多核架构下的安全漏洞检测进行了深入的研究,并提出了多种解决方案。文献[4]等针对 C 语言程序中因指针的直接内存操作可能导致内存泄露、缓冲区溢出等内存错误,而引起系统崩溃或内部数据破坏的漏洞问题,利用动态分析工具对程序进行插桩,并在运行时监测内存操作,使用模糊测试技术提供多样化的输入数据,以触发更多可能的内存错误情况,丰富其测试场景。然而 C 语言中的特定结构可能非常复杂和多样化,C 语言中的某些特定结构因其复杂性和多样性,使得实时分析工具在处理和插桩这些结构时面临挑战,有时可能无法完全正确地进行插桩,进而导致插桩后的代码无法被正确编译。文献[5]等针对神经网络模型因库外词过多导致的嵌入词向量的语义模糊的问题,提出一种增强可解释性的源代码漏洞检测方案。该方案基于卷积神经网络和全局平均池化,并结合类激活映射技术,实现对模型预测结果的可视化解释,虽然归一化可以减少库外词的数量,但也可能导致一些重要的标识符信息丢失,同时 CAM 的解释结果可能受到模型训练数据和参数设置的影响。文献[6]等针对智能合约漏洞检测过程中可能会引入过多的无效信息问题,修改源码特征为智能合约操作码序列,以提高智能合约漏洞检测的效率和准确率。然而胶囊网络和注意力机制的混合网络具有较高的复杂性,这可能导致模型在训练过程中需要更多的计算资源和时间,同时模型的复杂性也可能使得其难以解释和调试。文献[7]等针对静态分析往往需要在代码完全开发完成后再进行,无法实时地反映代码变更可能引入的新漏洞的问题,利用程序切片技术得到程序源码的特征矩阵,形成一个全局的代码图表示,使用多头图注意力网络(MHGAT)对全局代码图进行训练和学习。虽然模型能够准确地检测出漏洞,但很难直接给出漏洞产生的具体原因和修复建议。文献[8]等利用 LightGBM 和贝叶斯优化算法进行字词特征融合与漏洞自动评估。该方法融合的字词特征权重不够平衡,影响了漏洞评估效果。文献[9]等基于 GAGLight GBM 进行 PRFGRFECV 特征优选,完成网络安全态势评估。该方法未对 GAGLight GBM 模型进行超参数优化,对不同类型漏洞的检测适应性不强。文献[10]等利用遗传算法和 LightGBM 算法进行网络安全态势感知。该方法未准确提取网络特征,获得的漏洞感知效果不够精准。

SimHash 算法因为其部分敏感的特性,可以生成相似漏洞的相似哈希值,从而加速相似漏洞的匹配。LightGBM 作为一种高效的梯度提升框架,以其快速、准确和可解释性强的特点,在安全漏洞检测领域显示出巨大的潜力。在现有研究的基础上,该文提出了一种基于 SimHash 和 LightGBM 的移动 APP 多核架构安全漏洞并行检测方案,旨在自动化地提取特征并构建分类器,以实现海量漏洞数据的高效、准确检测。

1 基于改进 SimHash 算法的源代码特征提取

SimHash 是一种局部敏感哈希算法,其显著优势在于高效性和准确性。SimHash 算法流程如图 1 所示。

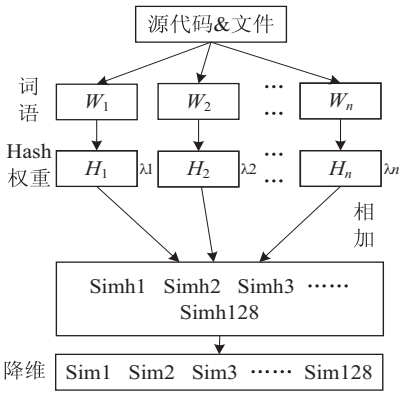

图 1 SimHash 算法流程

根据图 1 所示流程,利用 SimHash 算法进行源代码特征提取的具体步骤如下:

步骤 1:对源代码 & 文件进行分词,输入语句 S 的词语集合 $W=\{W_1,W_2,\cdots,W_n\}$。

步骤 2:设置哈希算法为 MurmurHash,使用 MurmurHash 映射转化词语集合 W 得到哈希签名值 $\{H_1,H_2,\cdots,H_n\}$,设置参数最大哈希位数 H_n 为 128 位的 hash 数字串。

步骤 3:将步骤 2 中签名值加权后累计,转化为序列串,获取到 128 位的未降维 SimHash 值 $\{\mathrm{Simh}_1,\mathrm{Simh}_2,\cdots,\mathrm{Simh}_{128}\}$,表示为:

$$\mathrm{Simh}_j=\sum_{i=1}^{n}\omega_i H_{ij} \tag{1}$$

其中,H_{ij} 表示第 i 个词 hash 值的第 j 位,ω_i 表示每个词的加权值。

步骤 4:将步骤 3 中结果降维成二进制,得到算法最终签名 $\{\mathrm{Sim}_1,\mathrm{Sim}_2,\cdots,\mathrm{Sim}_{128}\}$。

$$\mathrm{Sim}_j = \mathrm{redu}(\mathrm{Simh}_j) = \begin{cases} 0 & \mathrm{Simh}_j \leqslant 0 \\ 1 & \mathrm{Simh}_j > 0 \end{cases} \tag{2}$$

其中, Sim_j 提取移动 App 中某些关键部分(如代码片段、配置文件、资源文件等)的特征,并生成对应的哈希值。

在移动 App 安全漏洞检测中,随着文本数据的增长,分词后特征词数量庞大,尽管降维为二进制,但 SimHash 在处理高维稀疏数据时仍可能面临共现词权重过高在检测源代码和文件差异时产生的干扰[11],降低检测效率。因此,该文首先利用 TF-IDF 算法改进 SimHash 算法,对分词结果进行筛选降维,在保持关键安全信息的同时,减少特征词数量,降低计算复杂度。然后,在 TF-IDF 模型的基础上,融合 Jaccard 指数进行特征词的权重分配计算,改进 SimHash 算法中特征词重要性比重的计算方法。

步骤 5:利用 TF-IDF 模型对分词筛选降维。

$$\omega_{dt} = \mathrm{tf}_{dt} \times \mathrm{idf}(N_{nt}) \tag{3}$$

式中, d 为文本, t 为特征词语, ω_{dt} 为 t 在 d 中的权重, f_{dt} 为 t 在 d 中出现的频次, N 为文本总量, $\mathrm{idf}(N_{nt})$ 为逆文本频次,即对 N 和 t 出现的文本量 n 的比值取对数,作用是表明 t 的重要程度。

步骤 6:对 t 的权重进行归一化操作。

$$\omega_{dt} = \frac{(m_{dt}/M_d) \times \lg(N/n_t + 0.01)}{\sqrt{\sum_{t \in d} [(m_{dt}/M_d) \times \lg(N/n_t + 0.01)]^2}} \tag{4}$$

其中, d 为文本, t 为特征词语, m_{dt} 为 t 在 d 中出现的次数, M_d 是 d 中 t 的总量, n_t 为文本集合中出现了 t 的文本数量。

步骤 7:利用 Jaccard 指数修正样本相似度阈值。Jaccard 指数是专门用于量化样本与样本之间近似情况的统计度量,如式 5:

$$J(x,y) = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \tag{5}$$

式中,对象 x,y 的相似度是 $J(x,y)$, f_{11} 为 $x=1, y=1$ 时的样本量, f_{01} 为 $x=0, y=1$ 时的样本量, f_{10} 为 $x=1, y=0$ 时的样本量。x,y 等于 1 代表其样本中存在对应的特征单词,反之不存在。在现实场景里, x,y 不论何时,其出现频率都是随机的,于是为了去除量纲的干扰,进行样本相似度阈值计算。

$$J(x,y) = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \times \frac{1}{1 + \lg\sqrt{1 + \dfrac{\sum_{k=1}^{n} (x_k - y_k)^2}{n}}} \tag{6}$$

式中,第 k 个样本中特征词 x 或 y 出现的数量分别用 x_k 、y_k 表示, n 为同时含 x,y 的总数。

当两个高度类似的特征词同时出现时,可以减小其中一个的重要程度占比,尤其是完全一致时,进而可以专注识别导致文本区别的特征词[12]。

步骤 8:结合 TF-IDF(词频-逆文档频率)和 Jaccard 相似度进行特征词权重分配优化。

$$\omega_{dy}^{`} = \begin{cases} \omega_{dy} - \omega_{dx} \times J(x,y), \omega_{dy} > \omega_{dx} \times J(x,y) \\ 0, \omega_{dy} \leqslant \omega_{dx} \times J(x,y) \end{cases} \tag{7}$$

式中,使用 TFC 算法后, ω_{dx} 为 x 在文本 d 中的重要程度占比, ω_{dy} 为 y 在文本 d 中的重要程度占比。

由此获得源代码特征 SimHash 值和权重。作为漏洞检测依据。

2　基于 LightGBM 的安全漏洞检测模型

在得到源代码的 SimHash 值后,使用 LightGBM 算法构建安全漏洞检测模型。将已知的带有安全漏洞的 APP 源代码作为正样本,将没有安全漏洞的 APP 源代码作为负样本,通过 LightGBM 算法对这些样本进行训练,得到安全漏洞检测模型。

2.1　基于 LightGBM 算法的输入数据集设置

模型输入数据集为 $D = \{(x_{ij}, \mathrm{Sim}_i) \mid_i^n\}, i = 1,2,\cdots,n, j = 0,1,2,3$,其中 n 为样本数量, Sim_i 表示移动 App 中某些关键部分(如代码片段、配置文件、资源文件等)的标签, x_{ij} 为 4 个极化参数。算法具体执行为:

(1)初始化极化参数 x_{ij} 在第 k 类的预测状态 $F_{k,0} = (x_{ij})$,气象目标分类标签为 $k = 1,2,\cdots,K$ 。

$$F_{k,0}(x_{ij}) = 0 \tag{8}$$

(2)迭代计算,迭代次数 $m = 1,2,\cdots,M$ 。

①计算极化参数 x_{ij} 在各类别上的概率。

$$P_{k,m-1}(x_{ij}) = \frac{1}{\sum_{k=1}^{K} \mathrm{e}^{F_{k,m-1}(X_{ij})}} \tag{9}$$

②计算极化参数 x_{ij} 的负梯度 $\tilde{y}_{i,k}$,极化参数 x_{ij} 的真实概率为 $y_{i,k}$ 。

$$\tilde{y}_{i,k} = y_{i,k} - P_{k,m-1}(x_{ij}) \tag{10}$$

③ $h = 1,2,\cdots,H$ 为叶子节点的数量, $\gamma_{h,k,m}$ 表示叶子节点在分裂后的值, $R_{h,k,m}$ 表示叶子节点上的样本集。

$$\gamma_{h,k,m} = \frac{k-1}{K} \frac{\sum_{x_{ij} \in R_{h,k,m}} \tilde{y}_{i,k}}{\sum_{x_i \in R_{h,k,m}} |\tilde{y}_{i,k}| (1 - |\tilde{y}_{i,k}|)} \tag{11}$$

④模型优化刷新,学习率设置为 η 。

$$F_{k,m}(x_{ij}) = F_{k,m-1}(x_{ij}) + \eta \sum_{h=1}^{H} \gamma_{h,k,m} I \tag{12}$$

(3)获取最终模型。

$$F_k(X) = \eta \sum_{m=1}^{M} \sum_{h=1}^{H} \gamma_{h,k,m} I \tag{13}$$

2.2 基于贝叶斯的超参优化

机器学习算法的训练效果是由许多超参数决定的[13],针对 LightGBM 算法超参数较多且不易调优的问题,采用贝叶斯优化算法进行超参数搜索和优化。LightGBM 模型超参数优化的模版函数可表示为:

$$x^{`} = \arg\max_{x \in \nu} f(x) \tag{14}$$

式中,$f(x)$ 是反映超参数与 LightGBM 模型性能之间关系的目标函数。x 为待优化模型的超参数。其中 ν 代表超参数搜索空间[14]。

贝叶斯优化算法以贝叶斯定理为基础,优化的目标是找到超参数 x 的合适组合,使 LightGBM 模型性能 $f(x)$ 达到最大,即找到公式 14 的最优解。

$$p(f \mid D_{1:t}) = \frac{p(D_{1:t} \mid f)p(f)}{p(D_{1:t})} \tag{15}$$

式中,$p(f)$ 为先验概率,$p(D_{1:t} \mid f)$ 为观测点,$p(D_{1:t})$ 为归一化常数,$p(f \mid D_{1:t})$ 为给定观测点 $D_{1:t}$ 的 f 的后验概率。

$D_{1:t} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_t, y_t)\}$ 为输入的 t 个数据组成的数据集,每个数组中 x 为一组超参数,y 为对应结果。使用贝叶斯优化模型,获取预测最优点 x_t 与计算函数目标值 y_t,(x_t, y_t) 组合后加入到预测点集里,迭代执行后获取最佳 LightGBM 结果。

采集函数期望增量(Expected Improvement,EI)的函数表示如下:

$$\mathrm{EI}(x) = \begin{cases} [\mu(x) - f(x^+)]\theta(Z) + \\ \qquad \sigma(x)\varphi(Z)\sigma(x) > 0 \\ \max(0, \mu(x) - f(x^+))\sigma(x) = 0 \end{cases} \tag{16}$$

$$Z = \frac{\mu(x) - f(x^+)}{\sigma(x)} \tag{17}$$

式中,θ 与 φ 分别为高斯分布的累积分布函数、概率密度函数。

2.3 安全漏洞检测

CVE(Common Vulnerabilities and Exposures)是一个全球公认的漏洞编号系统,用于为广泛认可的信息安全漏洞和暴露提供公共名称和描述[15]。EXP(exploit)则是指利用这些已知 CVE 漏洞的工具或代码,通常用于攻击目标系统或网络,获取未授权的访问权限或执行恶意操作。每年都会有大量的 CVE 漏洞被公开,而 EXP 的编写和使用往往紧随其后,因此及时了解和修复 CVE 漏洞对于保护系统和网络安全至关重要。

通过 SimHash 算法,将 APP 的源代码或二进制文件转化为简洁的哈希值,从而大幅度降低数据处理的复杂度和计算成本。同时,利用 LightGBM 模型进行漏洞分类,更加准确地识别出不同类型的安全漏洞,为后续的漏洞修复和风险防范提供了有力的支持,具体流程如图 2 所示。

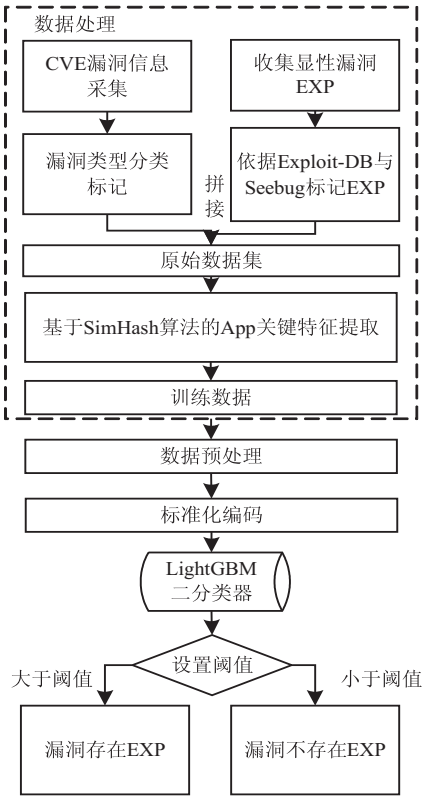

图 2 基于 LightGBM 算法漏洞分类模型

Step1:特征提取与 SimHash 编码。

收集 CVE 漏洞信息和显性漏洞 EXP;对数据进行预处理,包括依据 Exploit-DB 和 Seebug 标记 EXP;进行特征选择,提取最具代表性的特征,这些特征可能包括代码片段、漏洞类型、影响范围等;应用 SimHash 算法对提取的特征进行编码,生成每个漏洞的唯一哈希值。这些哈希值将用于后续的快速比较和分类。

Step2:数据标准化与拼接。

将经过 SimHash 编码的数据进行标准化编码,确保数据格式的统一性;将标准化后的数据进行拼接操作,构建数据集。

Step3:LightGBM 二分类器构建。

利用 LightGBM 算法,基于拼接后的数据集构建二分类器;设定分类器的阈值,用于判断漏洞是否存在 EXP。

Step4:模型训练、评估与结果输出。

使用训练数据对二分类器进行训练;使用评估数据对分类器的性能进行评估;根据分类器的判断结果,输出“漏洞存在 EXP”或“漏洞不存在 EXP”的结论。

采用了多核架构进行并行检测。在检测过程中,将待检测的 APP 源代码划分为多个部分,每个部分分配给一个核心进行并行处理。每个核心使用训练好的安全漏洞检测模型对分配的源代码部分进行检测,并将检测结果汇总得到最终的检测结果,如图 3 所示。

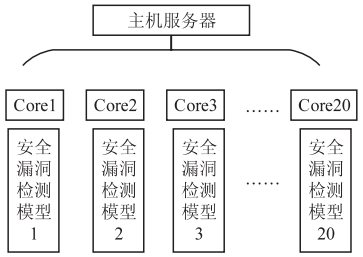

图 3　多核并行检测架构

3　实　验

3.1　实验环境

3.1.1　软硬件环境

深度学习模型需要处理大量数据,高性能计算机提供强大的计算能力和数据处理速度,确保算法模型训练过程的高效进行,尤其是处理器(CPU)的运算能力、内存(RAM)的存储能力。具体实验环境参数如表 1 所示。

表 1　实验环境参数

项目	版本
处理器	Inter(R)Core(TM)i9-14900KF CPU@4.2 GHz
内存	128 GB DDR5 2166 MHz
显卡	NVIDIA GeForce 4090Ti 16G
操作系统	Ubuntu/CentOS 双系统
编程语言	Python 3.12.3 64 位
数据挖掘框架	scikit-learn 1.4
分词工具	NLPIR
仿真平台	Matlab 2024a

3.1.2　参数设置

为了确定改进 SimHash 算法的特征词权重参数的最佳设定,基于本模型进行训练。设置相似度阈值为 0.4,则去重率阈值大于 0.6。从数据集中随机选择了 1 000 个文本作为实验数据,其中每个类别的文本数量尽量保持均衡。分别选择了 5 个类别进行实验,每个类别包含 200 个文本,综合五轮计算,取其平均值进行记录,实验结果如图 4 所示。

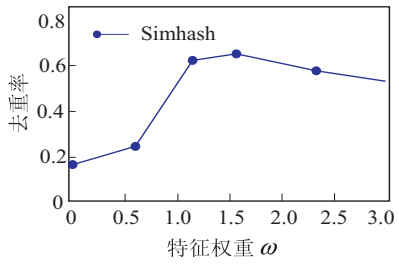

图 4　特征权重参数取值

如图 4 所示,ω 在模型训练初期,当 ω 值较低时,模型因可能无法充分学习到源码中的差异特征,导致去重率较低;随着 ω 值增大,去重率呈现上升趋势,但当 ω 值超过一定范围后,模型可能过于敏感,将一些相似但不完全相同的源码部分错误地判定为重复代码,去重率反而会下降。实验结果表明,只有当 $1.4 \leqslant \omega \leqslant 1.6$ 时,模型的去重率指标最佳,因此设置参数 ω 值为 1.5。

基于贝叶斯优化算法进行 LightGBM 算法超参数的迭代选择和优化,过程如表 2 所示。

表 2　LightGBM 算法超参数选择优化过程

算法:贝叶斯优化 LightGBM 参数
输入:超参数类型;调优范围;算法模型 LightGBM
输出:最优参数组合[params]
1. 设置需要优选的超参数与搜索空间: v = {列采样比例:(0.5,1), 子叶最小量:(20,200), 叶子节点数:(1,200), 树模型深度:(2,30), 学习器的数量:(1,300), *L*1 正则化权重系数:(0.1,10), *L*2 正则化权重系数:(0.1,10), 学习率:(0.01,1)}
2. 初始化获取数据集 $D_{1:t} = \{(x_1,y_1),(x_2,y_2),\cdots,(x_t,y_t)\}$
3. 根据采集函数公式,超参数组合 $x_{t+1} = \text{argmax}EI(x,D_{1:t})$ 获得下一个最有潜力的评估点 x_{t+1}
4. 计算采样点 x_{t+1} 对应的算法模型的性能 f_{t+1}
5. 更新 x_{t+1} 到观察点 $D_{1:t+1} = \{D_{1:t},(x_{t+1},f_{t+1})\}$,更新 f 的后验分布
6. 循环迭代 t 次,输出最优参数组合 params = $x'_{\max}$
7. 整理证书类型参数组合,并更新最优参数组合,输入 LightGBM 进行运行测试
8. return[params]

经过上述流程优选获得的关键超参数如表 3 所示。

表 3　超参数

超参数名称	默认值
树的数量	100
树最大深度	10
叶子节点个数	31
叶子节点最小样本数	20
学习率	0.1

3.2　数据集

数据集来源于美国官方漏洞数据库,时间范围覆盖 1999 年至 2023 年,包含如漏洞标识符(CVE ID)、漏洞描述、影响范围、解决方案、评分(如 CVSS 评分)等漏洞详细信息,同时排除掉“REJECT”标签相关的数据和包含 null 或空字符串的数据。

从数据集中随机选择了 10 000 个文本作为实验样本数据,其中每个类别的文本数量尽量保持均衡。分别选择死锁漏洞、缓冲区溢出漏洞、空指针解引用漏洞、代码注入、侧信道攻击 5 种常见的移动 APP 多核架构安全漏洞进行实验,每个类别包含 2 000 个文本。

将所有样本随机划分为训练集 7 000 个、测试集 3 000个。

3.3 评价指标

模型准确率评价指标包括平均绝对误差、均方根误差、平均绝对百分比误差、决定系数等,对方法进行验证。指标具体含义及计算公式如下:

平均绝对误差是所有实际取值减预测取值的绝对值平均数。平均绝对误差越小,表示模型越精准,计算公式如下:

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i-y_i\right| \tag{18}$$

式中,y_i 为真实值,n 为样本 i 的组数,$\hat{y}_i$ 为预测值。

均方根误差是误差平方和的平均值的平方根,取值范围为 0 ~ ∞。均方根误差值越低说明预测值越接近于真实值,模型越精确,计算公式如下:

$$\mathrm{RMSE}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i-y_i\right|^2} \tag{19}$$

平均绝对百分比误差是预测值与实际值之间误差的平均百分比,范围为 0 ~ ∞。平均绝对百分比误差值越低,预测值越接近于真实值,模型越精确,公式如下:

$$\mathrm{MAPE}=\frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i-y_i}{y_i}\times 100\%\right| \tag{20}$$

决定系数 R^2 是模型解释的变异量占总体变异量的比例,计算公式为:

$$R^2=1-\frac{\sum_{i=1}^{n}(\hat{y}_i-y_i)^2}{\sum_{i=1}^{n}(\bar{y}_i-y_i)^2} \tag{21}$$

式中,$\bar{y}_i$ 为平均值。

3.4 结果分析

3.4.1 消融实验

针对移动 APP 多核架构的复杂性、多样性以及安全漏洞的隐蔽性,设计了针对性的消融实验来验证联合算法的有效性以及各优化改进措施的实际效果,其数据集中随机选取 2 000 个样本展开实验,以决定系数 R^2 作为指标,测试每个模块对模型性能的影响,实验结果如表 4 所示。

表 4 消融实验

序号	SimHash 算法特征提取	TF-IDF 算法改进 SimHash 算法	LightGBM 分类	贝叶斯的超参数优化	数据处理编码	漏洞检测	R^2
1	/	/	/	/	/	√	0.653
2	√	/	/	/	/	√	0.712
3	√	√	/	/	/	√	0.813
4	√	/	√	/	/	√	0.866
5	√	√	√	/	/	√	0.915
6	√	√	√	√	/	√	0.961
7	√	√	√	√	√	√	0.975

如表 4 可知,传统漏洞检测模型的 R^2 值仅为 0.653,在模型中引入了 SimHash 算法特征提取后,模型的性能显著提升,R^2 值提升到了 0.712;经过 TF-IDF 算法改进 SimHash 算法后,R^2 值提升到了 0.813,证明 SimHash 算法改进后的性能更优。增加 LightGBM 分类步骤后,R^2 值提升到了 0.915。而当增加贝叶斯超参数优化后,模型的 R^2 值达到了 0.961,最后通过数据处理编码,模型的性能达到最佳,R^2 值为 0.975。可见 SimHash 算法特征提取、SimHash 算法改进、LightGBM 分类、贝叶斯超参数优化、数据处理编码都能够帮助模型更准确地识别关键信息,进一步提高模型的泛化能力和检测准确性。

3.4.2 对比实验

为了对比不同算法在特定数据集上的性能表现,通过量化指标评估模型的预测精度和泛化能力。实验比较了包括文献[4-7]在内的已有算法与文献[8]使用的 LightGBM + 贝叶斯优化算法、文献[9]使用的 PRFGRFECV+GA-LightGBM 算法、文献[10]提出的 GA+LightGBM 算法。

分别选择死锁漏洞、缓冲区溢出漏洞、空指针解引用漏洞、代码注入、侧信道攻击 5 种常见移动 APP 多核架构安全漏洞类别 3 000 个测试文本进行测试。测试 8 个方法针对 5 种常见漏洞的检测性能。由于文中方法研究的是多漏洞并行检测,因此为了直观体现性能差异,统计每种漏洞检测指标的均值,结果如表 5 所示。

如表 5 可知,针对死锁漏洞、缓冲区溢出漏洞、空指针解引用漏洞、代码注入、侧信道攻击 5 种常见移动

APP 多核架构安全漏洞,文中模型在多个性能指标上均展现出显著的优势。其中,MAE 均值低至 0.032,显示出模型在检测结果与实际值之间的平均差异最小,检测精度最高;RMSE 均值也远低于其他算法,表明模型在检测过程中的波动性较小,检测结果更为稳定可靠;尽管 MAPE 均值与其他算法相差不大,但文中模型仍取得了最低值,进一步印证了其在检测精度上的优势;尤为重要的是,R^2 均值接近 1,表明模型检测结果与实际值之间的拟合程度极佳,证明了模型具有很高的解释性和预测能力。可见文中算法在适应性、检测精度、稳定性和拟合能力等方面均表现出色。

表 5　不同算法性能对比

模型	MAE	RMSE	MAPE/%	R^2
文献[4]	0.073	1.653	0.169	0.938
文献[5]	0.065	1.526	0.165	0.911
文献[6]	0.089	2.015	0.154	0.925
文献[7]	0.076	1.632	0.149	0.934
文献[8]	0.072	1.802	0.156	0.909
文献[9]	0.074	1.753	0.162	0.911
文献[10]	0.071	1.685	0.158	0.927
文中模型	0.032	1.017	0.124	0.976

4　结束语

随着移动互联网技术的迅猛发展,移动 APP 的安全漏洞问题日益成为公众关注的焦点。特别是在多核架构下,APP 的复杂性和多样性使得安全漏洞数据的处理与识别变得尤为复杂和困难。基于当前背景,提出了一种基于 SimHash 算法和 LightGBM 的移动 APP 多核架构安全漏洞并行检测方法。通过利用 SimHash 算法对漏洞数据进行特征提取和编码,结合 LightGBM 算法构建的二分类器,准确判断漏洞是否存在可利用的 EXP,通过多核架构的并行处理,充分利用计算资源,实现对大量漏洞数据的快速检测,在保障数据安全的前提下,显著提高了漏洞检测的效率和准确性,为移动 APP 的安全管理提供了有力的支持。

然而,该方法仍存在一定的不足之处。例如,在特征提取和分类器构建过程中,可能需要进一步优化算法参数以提高检测性能。此外,随着新技术和新漏洞的不断涌现,如何及时更新和优化该方法以适应新的安全挑战,也是未来研究的重要方向。

展望未来,将继续深入研究基于 SimHash 算法和 LightGBM 的移动 APP 多核架构安全漏洞并行检测方法,探索更多优化和改进的可能性。同时,也将关注新技术和新应用对安全漏洞检测带来的新挑战和新机遇,为移动 APP 的安全管理提供更加全面、高效、可靠的解决方案。

参考文献:

[1] KSIBI S,JAIDI F,BOUHOULA A. A comprehensive study of security and cyber-security risk management within e-health systems:synthesis, analysis and a novel quantified approach [J]. Mobile Networks & Applications,2023,28(1):107-127.

[2] CHEN Z,KOMMRUSCH S,MONPERRUS M. Neural transfer learning for repairing security vulnerabilities in C code[J]. IEEE Transactions on Software Engineering,2023,49(1):147-165.

[3] ALFADEL M,COSTA D E,SHIHAB E. Empirical analysis of security vulnerabilities in Python packages[J]. Empirical Software Engineering,2023,28(3):1.1-1.37.

[4] 马莺姿,陈　哲,殷家乐,等. 结合模糊测试和动态分析的内存安全漏洞检测[J]. 计算机科学,2024,51(2):352-358.

[5] 王　剑,匡洪宇,李瑞林,等. 基于 CNN-GAP 可解释性模型的软件源码漏洞检测方法[J]. 电子与信息学报,2022,44(7):2568-2575.

[6] 陆　璐,赖锦雄. 基于胶囊网络和注意力机制的智能合约漏洞检测方法[J]. 华南理工大学学报:自然科学版,2023,51(5):36-44.

[7] 程靖云,王布宏,罗　鹏. 基于图表示和 MHGAT 的代码漏洞静态检测方法[J]. 系统工程与电子技术,2023,45(5):1535-1543.

[8] 张　哲,王　勇. 基于字词特征融合与 BO-LightGBM 的自动漏洞评估方法[J]. 微电子学与计算机,2023,40(7):27-35.

[9] 任高科,莫秀良. 基于 PRFGRFECV 特征优选的 GAGLightGBM 的网络安全态势评估[J]. 计算机科学,2023,50(S1):769-774.

[10] 胡　锐,徐　芳,熊郁峰,等. 基于遗传算法和 LightGBM 的网络安全态势感知模型[J]. 网络安全与数据治理,2024,43(3):14-20.

[11] 徐晓君,常会丽. 多线程交互学习软件系统安全漏洞自动化检测[J]. 计算机仿真,2022,39(4):335-340.

[12] 陈春雷,王省欣,谭　静,等. 基于 Yosys 的硬件信息流安全验证与漏洞检测[J]. 计算机应用研究,2021,38(6):1865-1869.

[13] 白英民,师智斌,信文阁,等. 基于词嵌入与 Shapelet 时序特征的智能合约漏洞检测方法研究[J]. 中北大学学报:自然科学版,2023,44(4):381-387.

[14] AHMAD H,DHARMADASA I,ULLAH F B M A. A review on C3I systems′ security: vulnerabilities, attacks, and countermeasures[J]. ACM Computing Surveys,2023,55(9):1.1-1.38.

[15] BOJANOVA I,GALHARDO C E C. Bug,fault,error,or weakness:demystifying software security vulnerabilities[J]. IT Professional,2023,25(1):7-12.