计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
2026,35(7):316−327 [doi: 10.15888/j.cnki.csa.010176] [CSTR: 32024.14.csa.010176]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

面向信创产品的固件解析与漏洞扫描框架①

江　楠, 赵创业, 田　叶

(南京南自数安技术有限公司, 南京 211106)
通信作者: 田叶, E-mail: tianye@chdnz.com.cn

摘　要: 信创产业作为国家信息技术自主可控战略的关键组成部分, 其固件安全对保障关键信息基础设施的安全具有重要意义. 面对国产处理器架构多样性、国密算法集成以及复杂供应链环境所带来的安全挑战, 本文提出了一种面向信创产品的多平台固件解析与漏洞扫描框架. 该框架通过扩展反汇编工具以支持非主流指令集, 结合动态构建的指纹库与自动化特征提取技术, 实现了对固件中已知漏洞及潜在安全风险的有效检测. 在龙芯、鲲鹏、飞腾这 3 大主流国产平台的固件样本上开展实验, 结果表明, 该框架的总体固件解析成功率达到 75.61%, 优于现有通用工具. 在漏洞检测方面, 平均精确率为 72.70%, 召回率为 79.00%, *F*1 分数达到 75.72%, 在保持较低误报率的同时实现了较高的检测覆盖率, 展现出良好的实用性与可扩展性. 本文为信创产品固件的安全评估提供了可行的技术方案, 具有重要的现实意义与应用价值.

关键词: 信创产品; 固件解析; 漏洞扫描; 信息安全; 国产平台

引用格式: 江楠,赵创业,田叶.面向信创产品的固件解析与漏洞扫描框架.计算机系统应用,2026,35(7):316–327. http://www.c-s-a.org.cn/1003-3254/10176.html

Firmware Analysis and Vulnerability Scanning Framework for Information Technology Application Innovation Products

JIANG Nan, ZHAO Chuang-Ye, TIAN Ye

(Nanjing NanZi Digital Security Technology Co. Ltd., Nanjing 211106, China)

Abstract: As a key component of the national strategy for independent and controllable information technology, firmware security in the information technology application innovation industry is of great significance for ensuring the security of critical information infrastructure. To address security challenges arising from diverse domestic processor architectures, the adoption of national cryptographic algorithms, and complex supply chain environments, this study proposes a multi-platform firmware analysis and vulnerability scanning framework for information technology application innovation products. By extending disassembly tools to support non-mainstream instruction sets and integrating a dynamically constructed fingerprint library with automated feature extraction techniques, the proposed framework effectively detects known vulnerabilities and potential security risks in firmware. Experiments are conducted on firmware samples from three major domestic platforms, including Loongson, Kunpeng, and Phytium. The results show that the overall firmware analysis success rate of the proposed framework reaches 75.61%, which is superior to existing general-purpose tools. In terms of vulnerability detection, the average precision is 72.70%, the recall is 79.00%, and the *F*1-*score* reaches 75.72%. The framework achieves high detection coverage while maintaining a relatively low false positive rate, demonstrating strong practicality and scalability. This study provides a feasible technical solution for the security assessment of firmware in information technology application innovation products and has important practical significance and application value.

Key words: information technology application innovation product; firmware analysis; vulnerability scanning; information security; domestic platform

① 收稿时间: 2025-11-13; 修改时间: 2025-12-02, 2025-12-19; 采用时间: 2025-12-30; csa 在线出版时间: 2026-05-22
CNKI 网络首发时间: 2026-05-25

国家信息安全战略下, 自主可控日益关键. 信创产业即信息技术应用创新产业, 旨在利用中国自主研发的软硬件产品实现对 x86 框架下的软硬件平台的替代或改进, 以解决关键领域的技术瓶颈问题[1]. 构成信创产业链上游的核心业务主要为硬件层的芯片、固件, 而复杂网络攻击的入口点通常是通过固件[2], 固件作为连接硬件与操作系统的最底层核心程序, 直接决定设备启动、运行及功能. 其安全性一旦受损, 攻击者可绕过上层防护, 直接控制硬件, 引发数据泄露、系统瘫痪, 甚至植入持久化威胁, 因此分析其安全性至关重要.

传统的固件分析工具主要围绕 x86/ARM 等通用架构设计[3], 难以有效应对国产处理器的非标准指令集、定制化的安全启动机制、国密算法应用以及异构文件系统格式等信创特有技术栈. 这种适配性缺陷导致在信创固件安全评估中, 出现关键组件识别困难、逆向分析效率低下、漏洞检测覆盖不足等问题[4]. 同时, 信创产品供应链中的闭源组件引入与国产化改造过程, 也可能带来难以识别的潜在安全风险[5].

在固件解析与漏洞检测领域, 国内外研究已从格式解析、动态仿真到跨架构语义分析等方向展开. 朱晓东等人[6]提出了基于结构化特征库的递进式固件解析方法, 实现了固件关键组成的自动识别, 但其特征库面向通用嵌入式体系构建, 对国产处理器架构差异与国密算法集成的适配能力仍有限. 面向动态分析的研究开始关注外设依赖问题, DICE 通过自动建模与仿真 DMA 输入路径, 解决固件动态分析中外设依赖导致的执行中断问题, 从而提高嵌入式固件的可执行覆盖率和动态分析有效性[7]. AIM 通过自动化构建精确的固件中断模型, 使动态分析环境能够逼真复现设备级中断行为, 从而显著提升动态固件分析在覆盖率与可执行性方面的有效性[8]. 在大规模语义分析方向, de Freitas 等人[9]提出了一种基于 BinclustRE 的大规模固件语义聚类方法, 通过自动提取与归纳跨架构函数特征, 实现了在无源代码条件下的大规模漏洞模式识别, 为异构固件的批量安全分析提供了可扩展的技术路径. 此外, FITS 框架通过构建结合静态特征与运行时特征的函数行为表示, 自动推断可作为中间污点源的函数, 并借由行为聚类与相似度排序显著提升固件污点分析在大规模闭源 IoT 固件上的覆盖率与漏洞发现能力[10].

现有固件分析体系虽然在通用架构上已较成熟, 但其方法论基础依赖于 x86/ARM 的统一语义和标准化生态, 因此难以直接迁移至信创环境. 面对国产处理器的异构性、国密算法的深度整合以及供应链组件的多源化, 传统工具无法构建稳定的解析路径, 也难以支撑信创场景所需的跨组件、跨语义的系统级安全分析. 进一步来看, 国产处理器在指令语义、ABI 约定和编译优化策略上的差异, 使控制流重建、函数边界识别与二进制语义恢复更易出现偏差; 国密算法多以封闭或高度优化的二进制形式存在, 使传统静态与动态分析难以识别其内部逻辑; 供应链异构性则限制了跨模块依赖与漏洞传播链的有效建模. 基于上述因素, 现有技术体系整体上尚不足以支撑面向国产平台的固件解析与漏洞扫描需求, 需要构建具备更强适配性与更高分析粒度的专用框架.

本文的主要目标是提出一种面向信创产品的固件解析与漏洞扫描框架, 旨在解决信创固件安全分析中的适配性与准确性难题. 该框架通过深度定制化与扩展, 使固件分析工具能够解析国产处理器架构的二进制代码, 构建包含信创特有安全特征的动态指纹库, 并引入自动化二进制特征提取与模式识别技术. 这有助于实现对固件中已知漏洞及潜在风险的高效检测, 弥补了传统通用工具在信创适配性上的不足.

实验结果表明, 该方法在多款信创产品固件的测试中, 展现出良好的漏洞检测能力, 总体指标平均值达到了 72.70% 的精确率、79.00% 的召回率和 75.72% 的 $F1$ 分数, 在控制误报与降低漏报之间保持了较好的平衡, 整体检测性能较为稳定, 能够满足真实信创场景的应用需求. 这些研究成果将推动固件安全分析技术进步, 并助力提升我国信息基础设施的整体安全水平.

1 相关技术

1.1 固件解析与逆向分析

现有固件解析与逆向分析工具能够有效恢复固件的内部结构与代码语义, 为后续安全分析提供基础性的结构化信息, 是当前固件安全研究中最成熟、最广泛使用的技术体系.

固件通常以二进制镜像文件的形式存在, 其内部功能与结构不直观可读, 因此, 对固件进行解析是开展安全分析的首要步骤. 固件解析旨在从原始固件镜像中识别并提取其内部组件, 包括引导加载程序、操作系统内核、文件系统、驱动程序及应用程序等[11]. 专用工具能够识别固件镜像中的嵌入文件、加密密钥以

及不安全的配置[12], 进而对整体二进制映像进行结构化拆解, 将其转换为可辨识的逻辑组件集合, 为后续的静态分析与漏洞挖掘提供基础支持.

逆向工程指的是将已编译代码反汇编为可读格式, 分析其结构, 并识别诸如加密流程、认证机制以及数据存储方式等关键组件[13]. IDA Pro[14]、Ghidra[15]等通用逆向工程工具提供强大的代码结构分析能力[16], 仿真技术如 Firmadyne 实现了对于固件二进制文件所需的可扩展性的自动化分析[17]. 然而, 此类方法存在自动化有限的挑战, 尤其面对复杂或混淆的固件. 二进制代码的高级语义信息丢失, 加剧了理解难度. 同时, 分析高度依赖目标处理器架构和平台知识, 使得此过程耗时且技术门槛较高.

在信创固件环境下, 上述方法还存在额外局限. 国产处理器架构在指令语义、ABI 约定和编译特性上与主流架构差异显著, 通用逆向工具在反汇编精度、控制流恢复及语义识别方面容易出现偏差, 导致结构提取不稳定. 固件中广泛集成的国密算法往往以封闭实现或高度优化的二进制存在, 缺乏可识别的语义标记, 使得现有工具难以准确分析其调用关系与安全属性. 此外, 固件组件来源多样且缺乏统一描述信息, 增加了跨模块溯源与依赖识别的难度.

1.2 二进制漏洞扫描方法

二进制漏洞扫描方法在无需源代码的条件下即可识别潜在安全隐患, 静态分析具备覆盖范围广的优势, 动态分析能够捕捉运行时行为, 两者在传统软件安全领域已形成较完整的分析框架.

二进制安全漏洞扫描主要通过静态分析和动态分析两类方法进行[18]. 静态分析在不执行程序的情况下, 通过代码结构和逻辑发现漏洞[19]. 典型的静态分析过程首先根据分析目的将应用程序代码转化为抽象模型 (如控制流图 (control flow graph, CFG)、调用图 (call graph, CG) 或 UML 类/序列图), 并在此基础上应用多种分析技术, 包括控制流分析[20]、数据流分析[21]、指向分析[22]、调用图算法 (如类层次分析[23]、快速类型分析[24]、变量类型分析[25]等).

动态分析在应用程序运行时检测漏洞[26]. 常见的动态分析测试包括模糊测试、Concolic 测试和基于搜索的测试[27]. 模糊测试被定义为一种自动化软件测试技术, 涉及提供无效、意外或随机数据作为计算机程序的输入, 监控程序是否存在异常, 例如崩溃、内置代码断言失败或发现潜在的内存泄漏[28,29]. Concolic 测试是一种混合软件验证技术, 融合了符号执行和具体执行. 符号执行将程序变量视为符号变量, 而具体执行则在特定的输入路径上执行测试[30]. 基于搜索的测试使用元启发式优化搜索技术来自动化测试任务, 例如模拟退火和遗传算法[31]. 然而, 传统漏洞扫描方法面临显著挑战. 静态分析常存在较高的误报与漏报率[32], 尤其在处理复杂指针、加密或动态生成代码时准确性下降. 其难以捕捉运行时依赖或特定逻辑漏洞, 且大规模二进制分析效率低下. 动态分析则受限于代码覆盖率[33], 且固件与特定硬件的紧密耦合使得测试环境搭建极为复杂.

在信创生态中, 传统静态与动态漏洞扫描方法也面临适配性不足的问题. 静态分析依赖于架构一致的指令语义与准确的控制流构建, 而国产处理器的多样性会导致路径恢复、指针分析和数据流推断出现偏差, 从而增加误报与漏报. 国密算法实现通常以黑盒形式出现, 缺乏可利用的中间语义, 使扫描工具难以识别加密逻辑相关的弱点. 动态分析则受限于硬件依赖度高、仿真支持不足等因素, 难以在非主流架构上实现足够的路径覆盖. 供应链复杂性进一步降低了跨模块行为分析的可见性, 使得传统方法对信创固件的漏洞识别能力受到限制.

2 框架设计

2.1 框架总览

为了实现对信创多平台固件的深入安全分析与漏洞识别, 本文提出一种面向信创产品的多平台固件解析与漏洞扫描框架. 整体流程如图 1 所示, 框架由信创固件解析模块、信创安全知识库、特征匹配模块以及深度学习分析模块这 4 大核心功能模块组成, 各模块紧密协作, 协同完成从原始信创固件二进制镜像到统一漏洞检测报告的全过程.

本文提出的框架首先接收原始的信创固件二进制镜像作为输入. 固件解析模块作为框架的起始环节, 负责执行固件的底层解包、文件系统重构和多架构二进制反汇编, 并将其结果转化为统一的中间表示, 为后续漏洞检测提供数据支撑. 为确保检测过程的全面性与准确性, 框架内置信创安全知识库, 涵盖漏洞指纹、安全规则与行为模式, 为后续各检测环节提供依据. 在核心检测阶段, 特征匹配模块承担主要功能.

该模块结合知识库中的漏洞指纹与预定义规则, 对固件进行系统化分析: 一方面识别已知或公开的开源组件、第三方库及其相关漏洞; 另一方面基于静态分析技术检测配置缺陷、不安全编程模式及协议层异常行为, 从而发现潜在的逻辑漏洞. 在此基础上, 深度学习分析模块引入神经网络模型, 从二进制代码中提取复杂特征[34], 识别具有变种特征的漏洞模式, 以进一步提升检测的覆盖范围与准确性. 所有检测结果经统一整合与关联后, 最终生成结构化漏洞审计报告, 为安全修复提供依据.

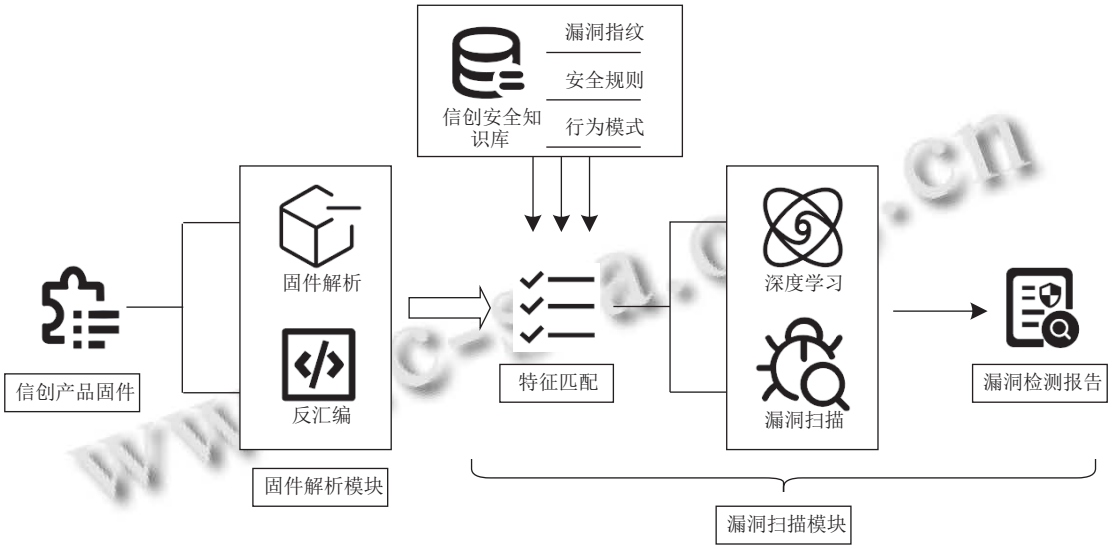

图 1 框架流程图

框架通过分层处理路径组织固件解析、特征匹配与深度学习分析, 并以统一知识库约束各模块的判定标准, 借助规则特征与深度特征的协同作用, 能够在结构、语义与行为多个层面形成互补的漏洞识别能力.

2.2 固件解析模块

固件解析模块旨在对原始信创产品固件镜像进行系统化处理, 提取关键结构与特征信息, 为后续漏洞挖掘与安全分析奠定基础, 整体流程如图 2 所示.

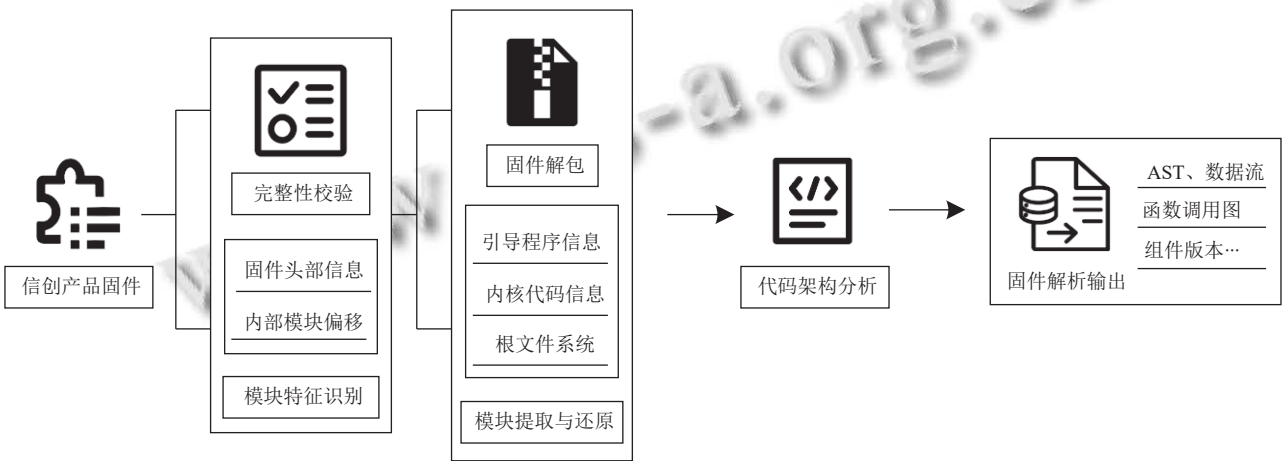

图 2 固件解析模块流程图

首先, 模块以未经处理的固件二进制镜像作为输入. 由于此类镜像通常包含压缩、加密及多种定制化结构, 因此需在解析之前进行完整性校验和初步识别, 提取固件头部信息, 如整体长度、版本、校验码以及内部模块偏移量, 以保证输入数据的有效性和可解析性.

在此基础上, 模块执行固件解包与模块化还原. 通过特征库与文件系统标识, 对镜像进行逐层解压缩与分区重构, 识别并恢复内部的引导程序、内核、根文

件系统及附加数据段. 针对固件中存在的多种文件系统类型, 模块能够实现虚拟挂载和目录树还原, 从而完整获取固件中的文件及配置信息.

文件系统解析完成后, 模块进入代码与架构分析阶段. 针对国产处理器架构, 模块执行多架构反汇编与指令级解析, 识别函数边界和基本块, 并进一步构建程序控制流图. 同时, 结合符号表与配置文件, 提取操作系统特有的启动加载程序、内核模块、关键系统服务及参数配置; 对于涉及国密算法实现的二进制片段, 还需进行特征定位与符号分析, 以标注潜在的密码学相关组件.

解析结果包括两类输出: 一类是标准化的中间表示, 包括抽象语法树、静态单赋值形式、符号表、函数调用关系、数据流依赖及关键代码属性, 为后续的语义分析和漏洞检测提供统一支撑. 另一类是固件元数据信息, 涵盖各组件的路径、哈希值、类型、处理器架构和操作系统版本等内容, 用于固件特征比对、安全溯源与版本追踪.

2.3 信创安全知识库

信创安全知识库在框架中发挥核心支撑作用, 其构建目标在于为各检测模块提供统一、系统化的漏洞信息与规则基础. 该模块的内容体系如表 1 所示, 主要包括 3 方面: 其一为漏洞指纹与规则数据, 涵盖信创产品及其依赖组件的已知漏洞信息, 内容包括漏洞编号、影响范围、典型特征码及相应检测规则; 其二为安全规则与配置规范, 面向系统配置、编码实践及协议实现等方面形成约束性条目, 为识别配置缺陷及不安全实现提供依据; 其三为行为模式与扩展特征, 总结来源于历史漏洞与攻击样本的抽象化模式, 为深度学习分析提供训练与识别的参考.

表 1 信创安全知识库内容体系

类别	主要内容	作用
漏洞指纹与规则数据	漏洞编号、特征码等	已知漏洞识别
安全规则与配置规范	配置项、编码规范等	缺陷检测依据
行为模式与扩展特征	攻击模式、特征向量等	深度分析参考

通过上述内容的集成, 知识库为解析模块提供规范化的特征数据支撑, 为特征匹配模块提供检测规则与判定依据, 并为深度分析模块提供必要的特征模式. 在这一基础上, 整体框架能够实现跨模块的一致性运行, 确保漏洞检测过程具备统一的知识来源与判定标准.

2.4 特征匹配模块

特征匹配模块负责识别已知漏洞与特定行为异常, 用于弥补传统静态分析在大规模二进制处理和运行时依赖捕捉上的不足, 该模块流程如图 3 所示. 其核心由漏洞匹配、静态污点分析与协议逻辑验证这 3 个子模块组成, 从语法、数据依赖和协议行为层面识别潜在安全风险. 模块最终输出可疑代码片段及上下文信息, 为深度学习分析模块提供候选输入.

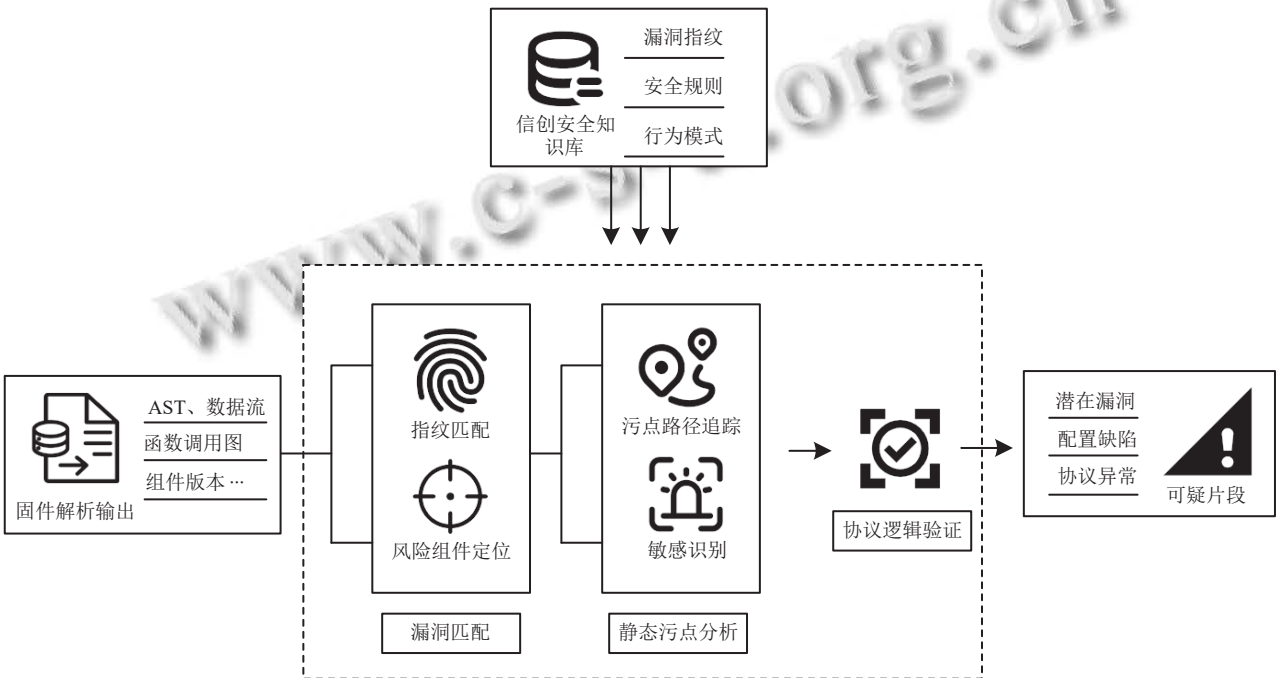

图 3 特征匹配模块流程图

2.4.1 漏洞匹配

漏洞匹配模块旨在利用知识库中的漏洞签名与固件解析结果建立映射关系, 实现已知漏洞的快速定位. 模块结合组件版本信息、语法结构与局部语义特征, 以分层方式进行比对. 在版本识别阶段, 模块根据文件元数据构建组件指纹, 并与知识库存储的版本特征集合进行匹配, 用于判断固件中是否包含具有已知安全风险的组件. 在语法匹配阶段, 系统基于抽象语法树 (abstract syntax tree, AST) 展开模式匹配, 依据语法层面的风险特征, 将数组索引越界风险、字符串处理不安全等结构模式与漏洞模板进行比对. 针对存在编译器优化造成结构变形的情况, 模块进一步基于基本块和指令序列构建指令级语义片段, 利用操作类型集合与数据依赖特征进行语义一致性检验. 具体实现如算法 1 所示.

算法 1. 漏洞匹配算法

输入: *IR*, *META*, *vuln_signatures*, *comps*.
输出: *sig_hits*.

```
1. procedure VulnerabilitySignatureMatch(IR, META, vuln_signatures, comps)
2.   sig_hits=∅
3.   for each c in comps do
4.     ast_c, ir_c=ExtractIR(IR, c)
5.     patterns=ExtractPatterns(ast_c, ir_c)
6.     for each sig in vuln_signatures[c] do
7.       if MatchPattern(patterns, sig) then
8.         hit=BuildSigHit(c, sig, location=Locate(ast_c, sig))
9.         sig_hits=sig_hits∪{hit}
10.      end if
11.    end for
12.  end for
13.  return sig_hits
14. end procedure
```

2.4.2 静态污点分析

静态污点分析用于判断外部输入是否能够传播至敏感操作点, 从而建立可利用的攻击路径. 模块在固件解析生成的控制流图 (control flow graph, CFG) 和数据流图 (data flow graph, DFG) 基础上, 对标识为 Source 的输入点进行初始化, 并依据赋值、指针操作、数组访问及跨函数调用等传播规则执行固定点迭代分析. 传播过程中结合过滤函数和边界检查语句对污点状态进行截断, 以降低误报率. 完成全局传播后, 系统依据知识库定义的 Sink 集合分析污点状态是否与漏洞匹配阶段发现的位置相关联. 如果某个漏洞匹配条目位于污点路径上, 则说明该风险具备输入可控性, 有更高的可信度. 具体实现如算法 2 所示.

算法 2. 静态污点分析算法

输入: *IR*, *taint_rules*, *sig_hits*.
输出: *taint_hits*.

```
1. procedure StaticTaintConfirm(IR, taint_rules, sig_hits)
2.   CFG, DFG=BuildCFGAndDFG(IR)
3.   taint_state=InitTaintState(taint_rules.sources)
4.   repeat
5.     changed=false
6.     for each node n in CFG do
7.       in_state=taint_state.in[n]
8.       out_state=TransferTaint(n, in_state, DFG, taint_rules)
9.       if HasSafeCheck(n, taint_rules.filters) then
10.        out_state=RemoveTaint(out_state)
11.      end if
12.      if UpdateState(taint_state.out[n], out_state) then
13.        changed=true
14.      end if
15.    end for
16.  until changed=false
17.  taint_hits=∅
18.  for each sig in sig_hits do
19.    if ExistsTaintPath(sig.location, taint_state, taint_rules.sinks) then
20.      taint_hits=taint_hits∪{sig}
21.    end if
22.  end for
23.  return taint_hits
24. end procedure
```

2.4.3 协议逻辑验证

协议逻辑验证模块面向固件中的通信协议与状态机驱动逻辑, 通过比较实际实现的状态迁移关系与知识库中规范模型的一致性来识别逻辑漏洞. 模块从协议处理函数的控制流中抽取状态更新语句、字段解析步骤及错误处理路径, 构建实际状态迁移图, 并检测是否存在异常跳转、缺失验证或未处理异常字段等问题. 对比过程基于有限状态机 (finite state machine, FSM) 匹配框架, 通过映射函数将实际迁移边与规范模型中的状态转换对应. 若某一迁移不满足协议规范或未覆盖协议要求的边界条件, 即视为协议实现缺陷, 并输出不一致项及其位置. 具体实现如算法 3 所示.

算法 3. 协议逻辑验证算法

输入: *IR*, *proto_models*, *comps*.
输出: *proto_hits*.

```
1. procedure ProtocolFSMVerify(IR, proto_models, comps)
2.   proto_hits=∅
3.   for each component c in comps do
```

```
    handlers=ExtractProtocolHandlers(IR, c)
    G=BuildStateGraph(handlers)
    model=proto_models[c]
    mismatches=CompareFSM(G, model)
    for each m in mismatches do
      hit={component=c, mismatch=m}
      proto_hits=proto_hits ∪ {hit}
    end for
  end for
  return proto_hits
end procedure
```

综合 3 个子模块的分析结果进行整合, 依据条目的组件标识与代码位置信息, 将其映射至具体的函数或基本块范围, 并抽取相邻指令与语句形成局部上下文, 同时保留相应的分析依据, 如签名特征、污点传播链或协议状态不一致描述. 经过整理后的输出以可疑代码片段及其上下文信息的形式呈现, 为深度学习模块提供结构一致的候选输入, 并便于后续人工审查.

2.5 深度学习分析模块

深度学习分析模块建立于特征匹配模块的输出之上, 其目标在于对已有规则难以覆盖或表现为变种特征的漏洞进行进一步识别, 该模块流程如图 4 所示.

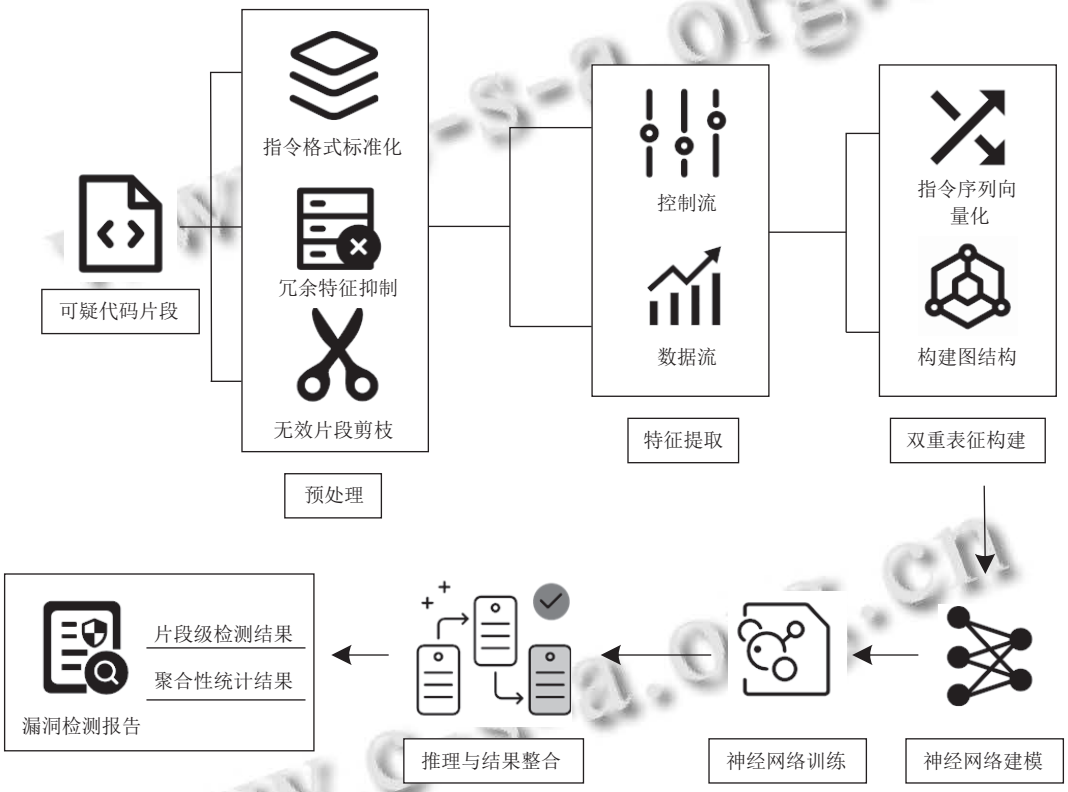

图 4 深度学习分析模块流程图

2.5.1 数据预处理

深度学习模型无法直接处理原始二进制代码, 因此需将来自特征匹配阶段的可疑代码片段转换为结构化输入表示. 本模块生成两类特征表示: 指令序列表示和图结构表示, 用于后续多模态特征学习.

指令序列表示通过对反汇编得到的指令进行 Token 化, 将操作码、寄存器和立即数等字段转换为离散符号, 并由嵌入层映射为稠密向量序列 $E=[e_1,e_2,\cdots,e_L]\in\mathbb{R}^{L\times d_e}$, 其中 L 为指令序列长度, d_e 为嵌入维度, 用于捕捉局部语义模式与指令间的基本依赖关系. 图结构表示基于固件解析模块恢复的控制流图 (CFG) 与数据流图 (DFG) 构建. 图的节点表示基本块或指令, 边描述控制跳转或数据依赖关系. 得到的图结构作为结构信息输入, 在后续模型训练阶段由图神经网络进一步编码, 以提取跨基本块与跨路径的结构特征. 经由上述步骤构建的两类表示分别提供语义与结构维度的信息, 为联合模型训练阶段的多模态特征融合奠定基础.

2.5.2 联合模型训练

联合模型训练阶段通过融合指令序列特征与图结构特征, 实现对漏洞相关模式的联合学习. 指令序列表

示首先输入 Transformer 编码器, 通过自注意力机制建模指令间的语义关联, 其核心计算如式 (1) 所示:

$$Attention(Q,K,V)=Softmax\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V \tag{1}$$

其中, Q、K、V 分别为由输入序列线性映射得到的查询、键和值矩阵, d_k 为键向量维度. Transformer 经过多层堆叠后, 输出能够表征指令片段语义特征的向量. $F_{\text{seq}}\in\mathbb{R}^{d_{\text{seq}}}$.

图结构表示则输入多层图卷积网络 (graph convolutional network, GCN), 通过邻域特征聚合学习控制流与数据流的结构关系. GCN 的更新方式如式 (2) 所示:

$$H^{(l+1)}=\sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2}$$

其中, $H^{(l)}$ 为第 l 层节点特征矩阵, $\tilde{A}=A+I$ 为加入自环后的邻接矩阵, $\tilde{D}$ 为对应的度矩阵, $W^{(l)}$ 为可学习权重, $\sigma(\cdot)$ 为激活函数. 最终得到图嵌入向量 $F_{\text{graph}}\in\mathbb{R}^{d_{\text{graph}}}$.

序列特征与图结构特征在融合层拼接形成统一表示, 如式 (3) 所示:

$$F_{\text{code}}=\left[F_{\text{seq}};F_{\text{graph}}\right] \tag{3}$$

并输入分类器进行漏洞类别预测. 训练过程采用交叉熵损失, 如式 (4) 所示:

$$L=-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}y_{ic}\log(\hat{y}_{ic}) \tag{4}$$

其中, N 为样本数量, C 为类别数, y_{ic} 为真实标签, $\hat{y}_{ic}$ 为预测概率. 通过参数优化, 模型学习序列语义特征与结构依赖特征的联合表征能力.

2.5.3 神经网络推理

推理阶段对新的固件代码片段执行与训练一致的预处理策略, 将构建好的序列表示与图结构表示输入联合模型以获得漏洞预测结果. 分类器通过 *Softmax* 计算每个类别的后验概率得到预测类别及其置信度, 如式 (5) 所示:

$$P(y=c\mid F_{\text{code}})=\frac{\exp\left(w_c^{\mathrm{T}}F_{\text{code}}+b_c\right)}{\sum_{j=1}^{C}\exp\left(w_j^{\mathrm{T}}F_{\text{code}}+b_j\right)} \tag{5}$$

其中, F_{code} 为深度学习模型经过特征融合得到的代码片段表示向量; w_c 和 b_c 分别表示分类器中对应类别 c 的权重向量与偏置项; C 为漏洞类别总数; $P(y=c\mid F_{\text{code}})$ 表示模型在给定特征向量 F_{code} 时预测该片段属于类别 c 的后验概率. 分母中的求和项对所有类别进行归一化, 保证输出概率满足概率分布要求.

此外, 模型通过基于梯度的重要性分析方法对关键指令或基本块进行突出标识, 为模型判定提供辅助解释. 推理过程能够识别结构或语义上具有偏移的潜在异常行为, 为规则匹配未覆盖的风险提供补充证据.

2.5.4 漏洞结果整合

检测结果在函数、文件与组件等层次上实现聚合, 从而形成稳健的多层级漏洞分布图谱. 模块输出由两部分构成: 其一为片段级检测结果, 涵盖文件位置、函数或基本块标识、预测漏洞类别、置信度分值及其与规则检测结果的一致性说明; 其二为聚合性统计结果, 呈现不同漏洞类型在组件与固件范围内的分布特征, 并揭示与特征匹配结果之间的交集与差异. 该输出经报告生成机制进一步组织与可视化, 最终形成标准化的漏洞检测报告, 为后续人工复核、修复策略制定与知识库迭代更新提供依据.

3 实验与评估

3.1 实验设置

为验证本文提出的信创产品固件解析与漏洞扫描框架的有效性与性能, 本实验将针对来自不同类型和架构的真实信创固件样本进行测试, 通过量化评估该框架在固件解析能力、漏洞检测性能和扫描效率方面的表现.

3.1.1 数据集

本实验使用的数据集涵盖龙芯、鲲鹏和飞腾这 3 大国产处理器平台的多型号固件, 固件样本的收集来源于龙芯、飞腾和鲲鹏的开源固件仓库.

龙芯部分主要涵盖 5000 系列与 6000 系列的 18 个固件样本, 涉及笔记本、台式机、服务器及虚拟机等多种应用平台. 飞腾部分包含 D2000、E2000、FT2004 及 X100 GPU 平台的 10 个固件样本, 其中 GPU 平台固件又细分为 Gmem 与 NoGmem 两种版本. 鲲鹏部分共计 54 个固件样本, 覆盖 C8、C2000、C3000、C5800、C4800 等核心系列, 固件发布时间跨度为 2022–2025 年, 覆盖了多种硬件架构与版本迭代. 实验数据集如表 2 所示.

3.1.2 评估指标

为全面评估模型性能, 实验从以下 3 个维度设计评估指标.

(1) 从框架在固件内部结构解析与关键组件提取的效率上, 本实验采用解包成功率来进行评估.

解包成功率: 固件解析模块能够成功识别并提取固件中的文件系统层级结构或至少一个主要的可执行二进制组件的固件样本数量占总测试固件样本数量的百分比. 该指标主要评估框架在固件宏观结构解包和关键文件定位能力.

表 2 实验数据集

处理器平台	型号	固件数量	总计
龙芯	5000Series	15	18
	6000Series	3	
鲲鹏	C6	5	54
	C8	6	
	C9	2	
	C2000	7	
	C3000	3	
	C5800	4	
	C4800	4	
	N6	4	
	N2700	3	
	N6300	2	
	N6500	2	
	其他	12	
飞腾	D2000	3	10
	E2000	3	
	FT2004	3	
	X100	1	

(2) 通过精确率、召回率和 $F1$ 分数综合评估分类性能.

精确率 (*Precision*): 衡量框架预测为攻击的样本中真实攻击的比例, 如式 (6) 所示:

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

召回率 (*Recall*): 衡量真实存在的漏洞实例中被正确识别的比例, 如式 (7) 所示:

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

$F1$ 分数: 综合精确率与召回率的调和平均值, 用于平衡两者在类别不平衡场景下的表现, 如式 (8) 所示:

$$F1\text{-}score = 2\times\frac{Precision\times Recall}{Precision+Recall} \tag{8}$$

其中, TP 表示真实漏洞被正确识别为漏洞的数量, FP 表示实际无漏洞区域被误报为漏洞的数量, FN 表示真实漏洞未被识别的数量, TN 表示实际无漏洞且被正确识别为无漏洞的数量. 精确率高表明模型的误报率低, 召回率高表明模型的漏报率低.

(3) 从框架在固件解析和漏洞扫描任务中的处理效率上, 本实验采用固件解析时间和漏洞扫描时间来进行评估.

固件解析时间: 单个固件样本从固件输入到解析完成的平均耗时 (单位: min/样本), 用于评估固件解析模块的处理效率.

漏洞扫描时间: 单个固件样本从固件输入到扫描完成的平均耗时 (单位: min/样本), 用于评估漏洞扫描模块的计算效率.

3.2 结果分析

3.2.1 固件解析性能评估

本实验以解包成功率作为核心评估指标, 用于验证本框架在识别并提取信创产品固件内部结构方面的适配性与稳定性, 并与常用的两款通用固件分析工具 Binwalk[35]与 Firmware Mod Kit (Fmk)[36]进行了对比测试. 结果如表 3 所示.

表 3 不同处理器平台固件解析性能统计表

处理器平台	固件样本总数	固件解包成功数			解包成功率 (%)			本框架固件解析时间 (min/样本)
		本框架	Binwalk	Fmk	本框架	Binwalk	Fmk	
龙芯	18	13	12	14	72.22	66.67	77.78	1.53
飞腾	10	7	6	6	70.00	60.00	60.00	0.92
鲲鹏	54	42	35	40	77.78	64.81	74.07	1.22
总计	82	62	53	60	75.61	64.63	73.17	1.25

实验结果表明, 本框架在总体解包成功率上取得 75.61%, 较 Binwalk 的 64.63% 和 Fmk 的 73.17% 均有所提升. 这主要得益于本框架在固件解析模块中针对国产处理器架构和信创固件的文件系统结构、压缩与打包方式进行了深度定制与优化. 相较之下, Binwalk 作为一款通用固件分析工具, 其签名库和解包逻辑主要面向 x86/ARM 等主流架构和常见文件系统, 对信创固件中定制化的引导程序、非标准文件系统和特定压缩算法的识别能力较弱, 导致其成功率偏低. Fmk 虽然在某些方面具备较好的通用性, 但在面对信创特有的

异构文件系统类型和多样化的固件加密/混淆机制时, 其解包策略的灵活性和针对性不如本框架, 因此整体表现也低于本框架.

本框架在不同处理器平台上的固件解析时间保持在较为接近的水平, 平均单个样本处理时间为 1.25 min, 未出现明显的性能波动, 说明其解析流程能够较好地适应不同平台的固件特性并保持稳定效率. 总体来看, 本框架在多平台测试中均能实现稳定的固件结构解析与关键组件提取, 尤其在鲲鹏平台上的表现验证了针对特定固件打包逻辑优化的成效.

3.2.2 漏洞扫描性能评估

本实验以精确率、召回率和 $F1$ 分数作为核心评估指标, 用于验证本框架在识别信创产品固件中安全漏洞方面的准确性与完整性, 并补充考量了漏洞扫描时间以评估其运行效率. 结果如表 4 所示.

表 4 固件漏洞扫描性能与效率评估

处理器平台	固件样本总数	精确率 (%)	召回率 (%)	$F1$分数 (%)	总漏洞数	漏洞扫描时间 (min/样本)
龙芯	18	76.47	80.00	78.20	65	1.82
鲲鹏	54	70.14	77.89	73.81	190	2.12
飞腾	10	78.72	82.22	80.43	45	1.92
总体平均	82	72.70	79.00	75.72	300	2.03

实验结果表明, 本框架在涵盖龙芯、鲲鹏、飞腾这 3 大平台的 82 个信创固件样本上表现出较好的综合检测性能. 平均精确率为 72.70%, 说明在被识别为漏洞的结果中, 大多数属于真实漏洞, 误报控制在合理水平; 平均召回率为 79.00%, 表明能够发现大部分实际存在的漏洞, 漏报风险相对较低. $F1$ 分数为 75.72%, 反映了框架在查准与查全之间保持了较好的平衡. 在不同平台的表现中, 飞腾平台精确率为 78.72%, 召回率为 82.22%, $F1$ 分数为 80.43%, 检测结果稳定且整体性能领先. 鲲鹏平台精确率为 70.14%, 召回率为 77.89%, $F1$ 分数为 73.81%, 检测精度相对较低. 由于鲲鹏平台的测试样本数量最多, 其性能水平对总体结果影响显著, 这一结果也提示在该平台上模型可能面临更复杂的固件结构或更分散的漏洞特征, 从而增加了精确识别的难度.

在效率方面, 漏洞扫描的总体平均处理时间为每样本 2.03 min, 3 大平台的处理时间差异较小, 均能在分钟级完成扫描. 这样的速度使得本框架在批量化检测任务中具备较高的可扩展性, 能够在保持较低误报与漏报的同时完成高效检测, 为信创固件的安全评估提供了可靠的技术支撑.

3.2.3 与现有工具对比

为了验证本文框架的优势, 我们将其与两种现有的固件安全检测工具 EMBA[37]与 Cybellum[38]进行了对比. 实验在第 3.1.1 节所述的信创固件数据集上进行, 包含龙芯、鲲鹏和飞腾这 3 大平台共 82 款固件样本. 图 5 是本框架与现有方法的对比结果. 实验结果表明, Cybellum 在漏洞检测中能够保持一定的稳定性, 但在精确率和召回率上均低于本文方法; EMBA 在精确率上表现较高, 但召回率显著偏低, 导致整体检测覆盖不足.

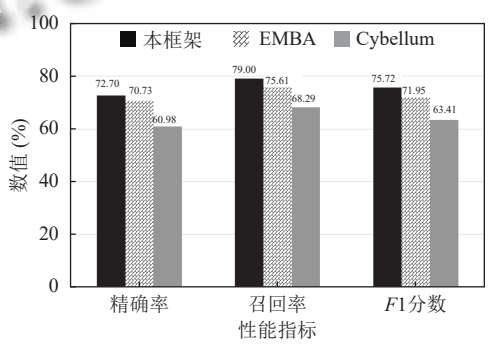

图 5 本框架与现有方法对比

本框架在召回率方面相对较高, 对实际漏洞的覆盖范围较广, 精确率处于中等水平, 整体表现较为均衡. Cybellum 的精确率相对偏低, 在多平台固件环境下适配性有限. EMBA 在部分测试中精确率表现尚可, 但在整体平衡性上略低于本框架. 总体来看, 本框架在多平台信创固件扫描中能够在覆盖范围与检测准确性之间维持一定的平衡.

此外, 我们对 ARM Cortex-A 与 Intel x86 这两款国外主流架构的固件进行了参考性测试, 实验结果如表 5 所示. 由于这些平台的固件结构更趋标准化、工具链亦更加成熟, 本框架在此类固件上的表现并未体现出明显优势. 相比之下, 在本研究重点覆盖的龙芯、飞腾和鲲鹏平台上, 解析与检测效果提升更为显著, 进一步说明框架设计在信创生态中的适配性和针对性价值.

表 5 国外处理器平台实验评估 (%)

处理器平台	固件样本总数	固件漏洞扫描性能			解包成功率		
		精确率	召回率	$F1$分数	本框架	Binwalk	Fmk
ARM Cortex-A	30	63.83	68.18	65.93	80.00	90.00	86.67
Intel x86	30	66.15	70.49	68.25	83.33	93.33	90.00

4 总结与展望

针对信创固件安全分析中适配性与准确性不足的问题, 本文提出了一种多平台固件解析与漏洞扫描框架, 通过多维度指纹匹配与自动化特征提取技术, 结合国产处理器架构与操作系统特性, 实现了对高度定制化固件的漏洞检测. 实验结果表明, 该方法在已知漏洞检测方面具有一定实用性, 有效缓解了通用工具在信创适配性上的不足. 未来将引入更精细的行为建模与深度学习算法, 扩大固件样本库规模, 并优化核心算法与计算机制, 以提升复杂逻辑漏洞检测能力、扫描效率及结果可靠性.

参考文献

1 梁炜, 李雅箐. 信创环境与 x86 架构下系统网络安全等级测评的异同分析. 网络安全技术与应用, 2023(4): 11–12.

2 Patil M, Patil SC. A blended approach of static binary mining and exploratory data analysis to obtain the security posture of embedded systems firmware. International Journal of Information and Computer Security, 2025, 26(1–2): 1–21. [doi: 10.1504/ijics.2024.10065236]

3 于颖超, 陈左宁, 甘水滔, 等. 嵌入式设备固件安全分析技术研究. 计算机学报, 2021, 44(5): 859–881.

4 蒋松. A 银行基础软件信创国产化项目质量管理研究 [硕士学位论文]. 成都: 电子科技大学, 2025.

5 刘家豪, 张佳发. 软件供应链安全在信创产业下的现状及思考. 2024 电力行业信息化年会论文集. 南昌: 中国电机工程学会电力信息化专业委员会, 2024. 334–340.

6 朱晓东, 尹青, 常瑞, 等. 基于结构化特征库的递进式固件格式解析. 武汉大学学报 (理学版), 2017, 63(2): 125–132. [doi: 10.14188/j.1671-8836.2017.02.005]

7 Mera A, Feng B, Lu L, *et al*. DICE: Automatic emulation of DMA input channels for dynamic firmware analysis. Proceedings of the 2021 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2021. 1938–1954. [doi: 10.1109/SP40001.2021.00018]

8 Feng B, Luo M, Liu CM, *et al*. AIM: Automatic interrupt modeling for dynamic firmware analysis. IEEE Transactions on Dependable and Secure Computing, 2024, 21(4): 3866–3882. [doi: 10.1109/TDSC.2023.3339569]

9 de Freitas OB, Júnior LAP. Unveiling firmware weaknesses: An approach for large-scale security analysis. Proceedings of the 24th Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. Porto Alegre: Sociedade Brasileira de Computação, 2024. 73–80.

10 Liu PZ, Zheng YW, Sun CN, *et al*. FITS: Inferring intermediate taint sources for effective vulnerability analysis of IoT device firmware. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Vancouver: ACM, 2024. 138–152.

11 Tang QH, Du F. Internet of Things Security: Principles and Practice. Singapore: Springer, 2021. 71–120.

12 Al-Zuraiqi A, Greer D. Evaluation of static analysis and Transformer-based LLMs for IoT firmware security. Proceedings of the 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things. Lucca: IEEE, 2025. 389–396. [doi: 10.1109/DCOSS-IoT65416.2025.00069]

13 Adeniran AA. Research survey on the use of reverse engineering embedded systems in security analysis of IoT device firmware and FCC's voluntary IoT labeling program. International Journal of Computer Applications Technology and Research, 2025, 14(7): 32–44. [doi: 10.7753/ijcatr1407.1004]

14 Lee A, Payne A, Atkison T. A review of popular reverse engineering tools from a novice perspective. Proceedings of the 2018 International Conference on Software Engineering Research and Practice. 2018. 68–74.

15 Rohleder R. Hands-on ghidra—A tutorial about the software reverse engineering framework. Proceedings of the 3rd ACM Workshop on Software Protection. London: ACM, 2019. 77–78.

16 冯健, 曹竞博. 跨平台移动应用中的安全漏洞分析与逆向工程破解防护策略探讨. 信息与电脑, 2025, 37(1): 92–94.

17 Chen DD, Woo M, Brumley D, *et al*. Towards automated dynamic analysis for Linux-based embedded firmware. Proceedings of the 23rd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2016. 1:1.1–1:8.1.

18 储宝龙. 基于信息技术的网络安全漏洞监测与防范策略研究. 信息记录材料, 2025, 26(2): 61–63. [doi: 10.16009/j.cnki.cn13-1295/tq.2025.02.043]

19 曾宇恒, 王娟, 朱倪宏, 等. 基于神经网络的代码漏洞检测研究进展与趋势. 智能安全, 2025, 4(1): 77–91. [doi: 10.12407/j.issn.2097-2075.2025.01.077]

20 Wang XL, Zhu SC, Zhou DH, *et al*. Droid-AntiRM: Taming control flow anti-analysis to support automated dynamic analysis of Android malware. Proceedings of the 33rd Annual Computer Security Applications Conference. Orlando: ACM, 2017. 350–361.

21 Wei FG, Roy S, Ou XM, *et al*. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android Apps. ACM Transactions on Privacy and Security (TOPS), 2018, 21(3): 14.

22 Li L, Bissyandé TF, Papadakis M, *et al*. Static analysis of android apps: A systematic literature review. Information and Software Technology, 2017, 88: 67–95. [doi: 10.1016/j.infsof.2017.04.001]

23 Dean J, Grove D, Chambers C. Optimization of object-oriented programs using static class hierarchy analysis. Proceedings of the 9th European Conference on Object-oriented Programming. Aarhus: Springer, 1995. 77–101.

24 Bacon DF, Sweeney PF. Fast static analysis of C++ virtual function calls. Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. San Jose: ACM, 1996. 324–341.

25 Sundaresan V, Hendren L, Razafimahefa C, *et al*. Practical virtual method call resolution for Java. ACM SIGPLAN Notices, 2000, 35(10): 264–280. [doi: 10.1145/354222.353189]

26 颜琪森, 康明, 杨艺垚, 等. 基于双阶段标签传播的半监督源码漏洞检测模型. 计算机科学, 2025: 1–17. https://link.cnki.net/urlid/50.1075.TP.20251205.1058.010. (2025-12-05)[2025-12-25].

27 Amin A, Eldessouki A, Magdy MT, *et al*. Androshield: Automated Android applications vulnerability detection, a hybrid static and dynamic analysis approach. Information, 2019, 10(10): 326. [doi: 10.3390/info10100326]

28 Mahmood R, Esfahani N, Kacem T, *et al*. A whitebox approach for automated security testing of Android applications on the cloud. Proceedings of the 27th International Workshop on Automation of Software Test (AST). Zurich: IEEE, 2012. 22–28.

29 Godefroid P, Kiezun A, Levin MY. Grammar-based whitebox fuzzing. Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. Tucson: ACM, 2008. 206–215.

30 Edalat E, Sadeghiyan B, Ghassemi F. ConsiDroid: A concolic-based tool for detecting SQL injection vulnerability in Android Apps. arXiv:1811.10448, 2018.

31 McMinn P. Search-based software testing: Past, present and future. Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops. Berlin: IEEE, 2011. 153–163.

32 刘云飞. 基于双向预训练模型的软件漏洞自动修复技术研究 [硕士学位论文]. 西安: 西安工业大学, 2025.

33 袁寰宇, 傅建明. 基于挂钩的 Android 应用密码函数识别. 计算机工程, 2025: 1–14. https://link.cnki.net/urlid/31.1289.TP.20250519.1105.001. (2025-05-19)[2025-08-22].

34 韩烨, 孙治, 赵童, 等. 基于机器学习的二进制代码相似性分析技术综述. 通信技术, 2022, 55(9): 1105–1111.

35 ReFirmLabs. https://github.com/ReFirmLabs/binwalk. (2024-10-31)[2025-11-13].

36 Firmware Mod Kit. https://github.com/rampageX/firmware-mod-kit. [2025-11-13].

37 EMBA. https://github.com/e-m-b-a/emba. [2025-11-13].

38 Cybellum. https://github.com/Cybellum. [2025-11-13].

(校对责编: 张重毅)