CIDefuse：融合数据流分析与语义嵌入的命令注入漏洞检测系统

陈霄[1,2]，沙乐天[1,2]，潘家晔[1]，孙瑞[3]，董建阔[1,2]，肖甫[1,2]
（1.南京邮电大学计算机学院，江苏 南京 210023；2.江苏省物联网智能感知与计算重点实验室，江苏 南京 210023；
3.南京大学医学院附属鼓楼医院，江苏 南京 210008）

摘　要：针对物联网设备中命令注入漏洞危害严重，而现有静态分析误报率高、动态分析覆盖率低及代码相似性检测难以处理跨函数漏洞的问题，提出一种融合数据流分析与语义嵌入的漏洞检测系统CIDefuse。首先，利用轻量级的反向可达定义分析，从固件二进制代码中快速剪枝并精确提取跨函数的漏洞候选路径。随后，通过层次化图嵌入网络捕捉代码的深层结构与语义信息，实现漏洞精准识别。实验结果表明，CIDefuse取得了0.93的曲线下面积（AUC）值、93.75%的精确率和90.91%的F1值，性能优于主流方法。此外，CIDefuse成功挖掘出3个未知漏洞，并均已获得国家信息安全漏洞共享平台（CNVD）的官方认证，证明了其有效性和实际应用价值。

关键词：物联网安全；命令注入；数据流分析；语义嵌入；二进制分析
中图分类号：TP393.08
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2026018

CIDefuse: a command injection vulnerability detection system via data-flow analysis and semantic embedding

Chen Xiao[1,2], Sha Letian[1,2], Pan Jiaye[1], Sun Rui[3], Dong Jiankuo[1,2], Xiao Fu[1,2]
1. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
2. Jiangsu Key Laboratory of Intelligent Sensing and Computing for Internet of Things, Nanjing 210023, China
3. Nanjing Drum Tower Hospital, The Affiliated Hospital of Nanjing University Medical School, Nanjing 210008, China

Abstract: To address the critical security threats of command injection vulnerabilities in IoT devices, where high false-positive rates were exhibited by traditional static analysis, false negatives were caused by insufficient path coverage in dynamic analysis, and cross-function vulnerabilities were not handled effectively by code similarity-based approaches, CIDefuse was proposed as a vulnerability detection system fusing data-flow analysis with semantic embedding. Firstly, a lightweight backward reaching definition analysis was employed to rapidly prune and precisely extract cross-function candidate vulnerability paths from firmware binaries. Subsequently, a hierarchical graph embedding network was utilized to capture the deep structural and semantic information of the code, enabling accurate vulnerability identification. It is demonstrated by experimental results that an area under the curve (AUC) of 0.93, a precision of 93.75%, and an F1-score of 90.91% are achieved by CIDefuse, and mainstream methods are significantly outperformed. Moreover, three unknown vulnerabilities are successfully discovered by CIDefuse, and all of them are officially acknowledged by the China National Vulnerability Database (CNVD). This achievement underscores the system's effectiveness and practical value in real-world scenarios.

Keywords: IoT security, command injection, data-flow analysis, semantic embedding, binary analysis

收稿日期：2025–11–27；修回日期：2026–01–10
通信作者：沙乐天，ltsha@njupt.edu.cn
基金项目：国家自然科学基金资助项目（No.62572255, No.62302238）；江苏省2024前沿技术研发计划基金资助项目（No.BF2024071）
Foundation Items：The National Natural Science Foundation of China (No.62572255, No.62302238), The 2024 Frontier Technology Research and Development Program of Jiangsu (No.BF2024071)

0 引言

近年来，物联网（Internet of things, IoT）设备的数量呈现爆炸式增长，据统计，2025年全球活跃的IoT设备超过310亿台[1]。然而，与这种快速扩张不相匹配的是其普遍存在的安全脆弱性。研究报告指出，约有25%的IoT设备存在可被远程利用的高危漏洞[1]。在众多安全威胁中，命令注入漏洞因普遍性高、危害性大，已成为IoT安全领域最严峻的挑战之一，并持续受到攻击者青睐[2]。一旦命令注入漏洞被成功利用，通常会导致远程代码执行，使攻击者能够完全夺取设备的控制权，甚至将受控设备整合进僵尸网络，作为发起大规模分布式拒绝服务攻击的跳板。

传统的Mirai僵尸网络[3]正是利用了大量IoT设备中存在的弱口令和命令注入漏洞感染了数百万台设备。2025年4月，相关安全团队监测到，停产的GeoVision物联网设备中的两个命令注入漏洞正被黑客利用，用于构建新型的Mirai僵尸网络变种。2025年6月，研究人员再次发现一种新型的僵尸网络正利用数字视频录像机设备中的命令注入漏洞进行攻击，据估计约有5万台设备暴露于该风险之下[4]。这些数据表明，命令注入漏洞仍是当前IoT安全防御体系中亟须解决的核心问题之一。

针对IoT设备中日益严峻的命令注入漏洞威胁，学术界与工业界已提出多种漏洞自动化检测方法。静态分析方法[5-7]覆盖率高，但其主要依赖于预定义的规则和模式匹配，对用户输入如何影响命令字符串缺乏深入的追踪能力，因而容易产生大量的误报。动态分析技术[8-10]通过在模拟或真实环境中执行固件并提供非预期输入来触发漏洞。该方法准确率高，但其测试效果严重依赖于仿真环境的构建复杂度和测试用例的质量，对于难以触发的深层路径和特定状态下的漏洞，存在覆盖率不足的问题，导致漏报。近年来，二进制代码相似性检测（binary code similarity detection, BCSD）被用于快速识别已知漏洞的变种。当前流行的BCSD方法[11-14]通常依赖于控制流图（control flow graph, CFG）及其变体。然而，由于CFG本身结构的复杂性和冗余性，这些方法在处理大量二进制文件时，检测效率和准确性往往受到限制。此外，这些方法在处理跨函数的数据流分析时面临巨大的挑战，这可能导致关键漏洞的遗漏。

为此，本文设计并实现了一个融合数据流分析与语义嵌入的物联网设备命令注入漏洞检测系统CIDefuse。CIDefuse通过利用数据流分析中轻量且高效的反向可达定义分析技术，对固件二进制代码进行快速剪枝，从而精确定位并提取出漏洞候选路径。然后，针对这些与漏洞高度相关的路径，采用一个层次化的图嵌入网络捕捉代码的深层结构与语义信息，最终实现对命令注入漏洞的精准识别。

本文的主要贡献如下。

1) 提出了一种基于反向可达定义分析的跨函数漏洞路径提取方法，从复杂的固件二进制文件中精确识别出外部输入到危险函数的候选路径。与传统正向的数据流分析方法相比，该方法有效去除了大量与漏洞无关的路径，克服了路径爆炸与输入冗余的难题，为后续的深度语义分析提供了高质量的精确输入，同时实现了跨函数的漏洞检测。

2) 设计了一个层次化的图嵌入网络，能够获取多层次代码语义。通过在指令、基本块和函数3个粒度上逐层进行信息聚合与嵌入，生成了能够全面表征路径结构与上下文语义的嵌入向量，捕获了传统方法容易遗漏的大量语义信息，同时提高了漏洞检测效率。

3) 设计并实现了CIDefuse的系统原型，实验表明，在可疑漏洞代码相似性检测任务上，CIDefuse取得了0.93的受试者工作特征曲线下面积（area under the curve, AUC）值。在端到端的漏洞检测场景中，CIDefuse取得了最高的前k个结果的精确率。此外，CIDefuse还成功挖掘出3个未知的跨函数漏洞，均得到了国家信息安全漏洞共享平台（China national vulnerability database, CNVD）的官方认可。

1 相关研究

针对物联网设备中广泛存在的高危漏洞，例如命令注入漏洞，安全研究人员提出了多种漏洞检测和挖掘方法，主要分为传统的动静态漏洞检测方法和目前主流的二进制代码相似性检测方法。

1.1 传统的动静态漏洞检测方法

传统的漏洞检测技术主要分为静态分析和动态分析。这些方法通过分析固件文件本身或监控其运行时行为来寻找安全隐患，为自动化漏洞挖掘提供了基础框架，但也存在固有的局限性。

文献[5]通过分析指针别名和过程间数据流构建数据流图，实现了对命令注入等污点型漏洞的识别。针对固件中多二进制文件交互的复杂场景，文献[6]结合污点分析与符号执行，追踪跨越文件边界的数据流传播。然而，符号执行的高昂时间开销和约束求解复杂性，严重制约了其可扩展性。为突破这一瓶颈，文献[7]利用 Web 前端代码与后端二进制文件共享关键词的特性，以此指导污点分析路径的探索，显著降低了误报。文献[15]引入按需别名分析技术优化了间接调用的解析，避免了对无关变量的冗余计算。文献[16]采用从危险函数到输入源的反向追踪策略，进一步缩减了无效路径的探索空间。文献[17]采用模糊匹配策略识别更多潜在污点源，并结合轻量级可达性分析以平衡检测深度与速度。为了进一步抑制误报，文献[18]提出采用一种双标签策略来优化污点关键词的识别，显著提升了静态检测的精确度。

动态分析技术通过在真实或模拟环境中运行固件，并借助模糊测试等手段来触发和捕获漏洞。文献[8]通过分析配套移动应用的用户界面交互来识别协议字段，从而指导变异测试。文献[9]将 Web 接口作为模糊测试的切入点，并注入运行时监视器以实现上下文感知的细粒度监控。文献[10]利用键值对等数据模型指导测试用例生成，有效解决了请求序列依赖问题。针对传统黑盒测试缺乏反馈机制的难题，文献[19]结合全系统模拟与用户模式模拟，实现了高吞吐量的灰盒测试。在此基础上，文献[20]将模糊测试与混合符号执行相结合，利用符号执行提取关键约束词来辅助模糊测试器绕过复杂的输入校验。文献[21]提出基于响应的执行踪迹推断机制，从黑盒设备中有效推断代码覆盖率。文献[22]构建了一种基于 App 辅助的远程模糊测试框架，通过解析 App 中的非 Java 组件构建变异数据包，实现了对远程 IoT 接口的深层漏洞挖掘。

尽管上述传统方法取得了显著进展，但在应对命令注入漏洞的检测场景时，这些传统方法的局限性愈发凸显。许多嵌入式设备的固件并不开源，这使依赖源码的静态分析面临巨大困难。命令注入漏洞的触发往往需要构造精巧的输入来绕过过滤机制，并可能涉及深藏在复杂业务逻辑中的调用链，这给动态模糊测试的代码覆盖率带来了巨大挑战。

1.2　二进制代码相似性检测方法

BCSD 作为一种新兴的漏洞挖掘范式应运而生。其核心思想在于，许多相似的漏洞往往体现在相似的代码片段中。因此，通过比较目标固件与已知漏洞库中二进制代码的相似性，可以高效地发现潜在风险。目前，主流的方法可分为基于控制流图的方法和基于指令序列语义分析的方法。

基于控制流图的方法将函数的结构信息抽象为图，再进行相似性比较。文献[11]首次引入了属性控制流图（attributed control flow graph, ACFG），利用统计特征对基本块进行编码。然而，传统图匹配算法的高昂计算开销和精度瓶颈限制了其大规模应用。为了解决此问题，文献[12]将 ACFG 映射为低维嵌入向量，并通过孪生网络高效度量函数间的余弦相似度。文献[13-14]提出将数据流图与 CFG 相结合，构建了标记语义流图（labeled semantic flow graph, LSFG），从而更全面地捕获函数的语义。针对早期方法依赖手工定义特征的问题，文献[23]引入了三级属性控制流图（3-level-feature attributed control flow graph, 3LACFG），显著提升了精度。文献[24]创新性地使用程序依赖图作为分析基础，利用其对数据依赖关系的敏感性，为漏洞成因的可解释性分析提供了新的视角。针对复杂的函数内联场景，文献[25]给出了一种基于跨内联模式匹配的检测方法，结合属性控制流图与图神经网络，有效缓解了因编译器优化导致的函数边界模糊问题。

基于指令序列语义分析的方法借鉴自然语言处理（natural language processing, NLP）技术领域的经验，将汇编代码视为一种“语言”来处理。文献[26]采用无监督学习模型，通过捕获指令间的关系来生成嵌入向量，对编译器优化和代码混淆具有较好的鲁棒性。随着深度学习技术的飞速发展，基于 Transformer 和注意力机制的方法逐渐成为该领域的研究热点。这类方法利用预训练模型强大的上下文捕捉能力，旨在学习代码的深层语义表征。例如，文献[27]采用基于 Transformer 的双向编码器表示（BERT）的架构为指令生成上下文感知的嵌入。针对序列模型容易丢失结构信息的缺陷，文献[28]创新性地提出了跳转感知机制，将控制流跳转信息显式地嵌入 Transformer 的注意力层中，显著提升了模型对程序控制流的理解能力。文献[29]进一步结合

动态分析，采用Transformer架构对微执行踪迹进行建模，通过捕捉指令间的长期依赖关系来增强语义理解。文献[30]引入了多头注意力机制来融合指令级特征，实现了特征权重的可视化与可解释性。

然而，将上述技术直接应用于命令注入漏洞的检测场景时，仍面临着诸多挑战。以Genius[11]、Gemini[12]为代表的早期方法依赖手动定义的块级特征，缺乏对危险字符串等关键信息的感知。FIT[23]等自动化方法处理的输入为函数级的CFG，引入了大量无关代码，在处理大型函数时开销高昂。以jTrans[28]和Trex[29]为代表的先进模型虽然在通用语义理解上表现优异，但其语言模型的视角可能使其忽视了对命令注入漏洞至关重要的特定数据流模式和危险函数调用关系。此外，命令漏洞的利用路径往往跨越多个函数，而当前BCSD方法普遍缺乏跨函数分析能力，难以发现大量高危的跨函数漏洞。

2　系统设计

2.1　系统总览

针对现有物联网设备命令注入漏洞检测方法在处理复杂数据流和深层语义理解上的不足，本文提出了一个融合数据流分析与语义嵌入的漏洞检测系统CIDefuse，其整体架构如图1所示。首先经由反向可达定义分析对二进制代码进行剪枝与漏洞候选路径提取，随后通过层次化图嵌入网络逐级聚合语义特征，最后利用孪生网络完成相似性度量。系统由3个核心模块构成：漏洞候选路径提取、路径图语义嵌入以及嵌入特征相似性检测。

1) 漏洞候选路径提取。该模块以固件二进制文件为输入，通过从危险函数（Sink）出发进行反向数据流分析，追踪关键变量的定义能否到达一个数据源（Source）。若能到达，则系统识别出一条从Source到Sink的数据依赖路径，并将其作为漏洞候选路径。每条候选路径被转化为其对应的CFG形式，称为漏洞特征控制流图（Vul-CFG）。

2) 路径图语义嵌入。为实现对Vul-CFG的深度语义表征，该模块设计了一个层次化图嵌入网络，在3个粒度上逐级学习特征。首先，在指令级捕捉基础的词汇语义；其次，在基本块级通过序列建模来理解局部上下文信息；最后，在图级聚合全局的结构化信息，生成能够全面表征路径结构与语义的图嵌入向量。

3) 嵌入特征相似性检测。该模块通过计算待测固件二进制文件中的Vul-CFG与已知漏洞样本Vul-CFG的图嵌入向量之间的余弦相似度，来量化其语义相似性。若该相似度得分超过预设阈值，则判定该路径存在与已知漏洞模式高度相似的命令注入漏洞。

2.2　漏洞候选路径提取

命令注入漏洞主要源自用户可控的输入对后续处理函数的影响，特别是恶意输入对系统级函数的影响。因此，如何有效提取出漏洞候选路径，即用户输入点到危险函数的完整可达路径，是前期工作中最为关键的一步，也是后续漏洞检测的基础。现有研究工作，尤其是主流的静态分析方法[5-6]，通常采用数据流分析技术中的污点分析技术来定位可疑漏洞链，即漏洞候选路径。

污点分析技术通常通过追踪用户输入等外部数据的流向判断其是否到达了敏感位置，如系统命令拼接或执行点。虽然污点分析能够有效找出数据流的潜在安全风险，但其存在一些固有问题。首先，

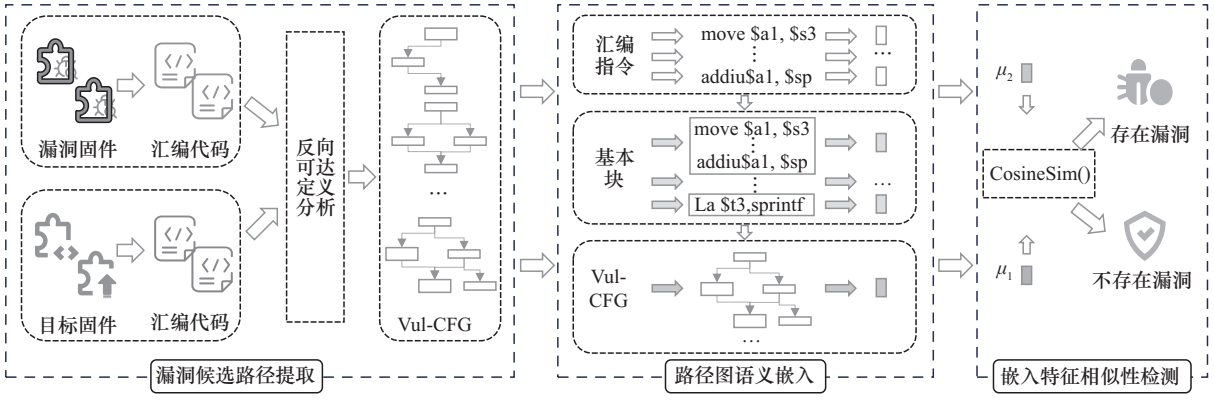


图1　CIDefuse整体架构

污点分析对数据传播路径较为敏感，容易受到信号丢失问题或复杂控制流的影响；其次，污点分析倾向于描述“数据从源到目标的传播过程”，对于某些漏洞触发条件，例如复杂逻辑条件或特定上下文，可能缺乏足够的精准性。基于此，一些研究开始采用可达定义分析进行漏洞候选路径的提取[17]。

可达定义分析通过分析变量定义的可达性，追溯敏感动作的根源，也就是“在执行关键命令时，有哪些数据赋值可能影响此处的变量”。这种方法能够更高效地聚焦于漏洞的触发点，避免因中间流程复杂而导致信号丢失。此外，可达定义分析还能帮助发现未初始化变量、死代码等潜在问题，为命令注入的检测提供更多语义层面的信息。

考虑到正向可达定义分析存在状态空间膨胀以及无效路径过多导致的低效问题，本文系统采用反向可达定义分析。与常规的正向可达定义分析方向不同，反向可达定义分析以危险函数为起点，从漏洞的潜在触发点向前溯源，分析敏感命令依赖的变量定义及其传播路径，最终找到潜在的漏洞入口和相关的危险路径。

具体来说，本模块在反向可达定义分析中对Sink和Source的角色定义如下所示。

1) Sink：代码中可能导致命令注入的危险操作位置，通常包括执行系统命令的函数调用，例如exec ()、system ()、popen () 等。在反向可达定义分析中，Sink不仅是分析的起点，也是初始关键定义的生成点。例如在分析system ($cmd) 时，本文会在此处为变量$cmd生成一个关键定义。

2) Source：导致命令注入漏洞的潜在输入点，通常包括用户可控的输入或外部传入的数据，例如用户输入的字段$_GET、$_POST、$_REQUEST、$_COOKIE等。本文的目标是判断在Sink生成的关键定义是否可以在不被覆盖（重新定义）的情况下成功到达某一个Source。

基于上述定义的Sink和Source，本模块从各个Sink开始，应用反向可达定义分析技术，追踪其关键参数的定义是否可达任意Source。若可达，则证明存在一条从Source到Sink的完整数据依赖路径（即可疑漏洞链），并将其转化为CFG形式，称为Vul-CFG。本文提出的采用反向可达定义分析提取Vul-CFG的伪代码如算法1所示。

算法1　采用反向可达定义分析提取Vul-CFG

输入　二进制文件的汇编代码

输出　Vul-CFG

1) 初始化 IN[·] = ∅, OUT[·]= ∅, Worklist = ∅, Vul-CFG = ∅
2) Sink, ArgsMap ← InitAnalysis(Binary) //初始化Sink及关键参数映射
3) for *s* in Sink do //初始关键变量加载
4)　　OUT[*s*] ← ArgsMap[*s*]
5)　　Worklist.push(*s*)
6) end for
7) while Worklist ≠ ∅ do
8)　　node ← Worklist.pop()
9)　　GEN ← GetDataDeps(node, OUT[node]) //基于关键变量提取依赖，更新GEN
10)　　KILL ← GetResolvedDefs(node, OUT[node]) //识别关键变量定义，更新KILL
11)　　NewIN ← (OUT[node] \ KILL) ∪ GEN //应用反向传递方程
12)　　if NewIN ≠ IN[node] then //若状态更新，则向前驱节点传播
13)　　IN[node] ← NewIN
14)　　for pred ∈ Predecessors(node) do
15)　　OUT[pred]←OUT[pred]∪IN[node]
16)　　Worklist.push(pred)
17)　　end for
18)　　end if
19)　　if IsSource(node) and (OUT[node] ∩ KILL ≠ ∅) then //若到达Source且关键变量存活
20)　　Vul-CFG ← BacktrackPath(node, OUT[node])
21)　　Vul-CFG.add(Vul-CFG)
22)　　end if
23) end while
24) return Vul-CFG

算法1定义了数据流方程中的核心集合：GEN[node]表示节点node内部产生的新变量定义；KILL[node]表示节点node内部被覆盖或无效化的变量定义；IN[node]和OUT[node]分别表示在节点入口和出口处有效的定义集合。为实现剪枝，关键变量判定规则设定为：以Sink参数为初始集进行反向遍历时，若指令定义了当前关键变量，则将其源操作

数作为新关键变量加入跟踪。

初始化阶段，算法首先识别二进制程序中的危险函数（Sink）及其对应的参数映射（Args-Map），并将关键参数加载至 Sink 节点的出口集合中以确定初始的关键变量，随后将 Sink 节点加入工作队列（第 1~6 行）。在迭代分析过程中，算法基于当前活跃的关键变量依次计算节点的新依赖（GEN）和被覆盖（KILL）的变量定义，并应用反向传递方程 IN[node] = (OUT[node] \ KILL[node]) ∪ GEN[node]计算节点的入口状态（第 7~11 行）。若状态发生更新，算法将新的入口状态通过前驱节点的出口集合向前传播，并将前驱节点加入队列以触发后续分析（第 12~18 行）。在此过程中，一旦检测到当前节点为用户可控的数据接收点（Source）且追踪的关键变量在此处被定义，算法将立即回溯并提取从 Source 到 Sink 的完整数据依赖路径，构建出用于后续检测的漏洞候选路径 Vul-CFG（第 19~22 行）。通过上述操作，本模块提取出固件二进制文件中所有的漏洞候选路径。这些候选路径的起点为用户可控输入点，终点为危险函数，能够有效覆盖函数内和跨函数的命令注入漏洞情况。

面对真实物联网固件中普遍存在的去符号化、编译器优化以及复杂的间接跳转等挑战，为了确保反向可达定义分析的稳健性与实效性，本模块在具体实施中采取了以下优化策略：基于调用约定的跨过程分析，通过内置 ARM、MIPS 等架构的寄存器传参规则，有效恢复跨函数的数据流依赖，解决了因符号缺失导致的调用链断裂问题；应用函数摘要技术对 sprintf、strcpy 等常见标准库函数预定义“输入–输出”依赖模型，在避免深入分析库函数内部逻辑的同时，有效防止了路径爆炸；控制流恢复，利用 IDA Pro 引擎预处理间接跳转，从而在保证分析精度的前提下，实现了对大规模固件的轻量级快速扫描。

2.3 路径图语义嵌入

获取漏洞候选路径之后，传统方法通常会直接对这些路径进行逐一验证。然而，由于命令注入的漏洞路径往往跨越多个函数，单纯通过静态方法进行漏洞分析会带来较大的时间消耗，效率较低。借鉴目前主流的代码相似性检测方法[13-14,23]，本系统将漏洞候选路径转换为向量的形式，使用向量来唯一表征 Vul-CFG。为了更好捕捉 Vul-CFG 内部指令元素之间、指令之间和代码块之间的深层次依赖关系，路径图语义嵌入模块包含了 3 个部分：指令嵌入、基本块嵌入和图嵌入。

2.3.1 指令嵌入

Vul-CFG 由代码块组成，而代码块由多条汇编指令构成，指令则包含若干个指令元素。为了解决汇编指令中操作数取值空间无限大导致的词汇表爆炸问题，并提高系统对地址偏移和常量变化的鲁棒性，本模块在进行嵌入学习前，先对原始指令序列执行严格的规范化与分词处理，具体规则如下：①保留指令的操作码助记符，如 mov、add、ldr，因为它们承载了核心的操作语义；②保留通用寄存器名称，以捕获寄存器分配模式和数据流向；③将所有的十六进制内存地址和指针统一替换为特殊标记<MEM>，消除因重定位或基址随机化带来的差异；④将超过一定阈值的立即数常量统一替换为标记<IMM>，仅保留小的控制流相关常量；⑤将代码中引用的字符串常量替换为<STR>，将用户自定义的函数调用目标替换为<FUNC>，从而聚焦于代码的结构逻辑而非特定的命名符号。经过上述处理后，每条指令被转化为一个由规范化词元（Token）组成的序列。

指令嵌入用于捕获指令中单词元素的上下文语义信息，并为每条指令生成包含高级语义信息的嵌入向量。在自然语言处理领域，词嵌入技术已发展成熟，其中 Word2vec[31]作为一种经典且高效的方法被广泛应用。Word2vec 的核心思想在于：如果两个词具有相似的语义，它们通常会在相似的上下文中出现。通过训练浅层神经网络，Word2vec 将文本中的词转化为高维向量表示，能够有效捕获目标词的语义和上下文关系。

Word2vec 包括两种模型结构：连续词袋（continuous bag-of-word, CBOW）模型和跳字模型（Skip-Gram）。CBOW 模型利用上下文来预测目标词，Skip-Gram 模型通过给定目标词来预测上下文。在实际的代码分析和漏洞检测中，二进制汇编指令通常具有语义稀疏性，即某些指令或操作符出现频率较低。相较于 CBOW 模型，Skip-Gram 模型更能有效地学习这些低频词语的语义表示。此外，汇编指令通常具有明确的上下文依赖关系。例如某条指令（中心词）可能决定下一步操作的语义特性。由

于Skip-Gram模型是从一个词推测其周围信息，并逐步形成语义嵌入，因此Skip-Gram模型在捕获功能相关上下文信息方面比CBOW模型更出色。基于上述分析，本模块采用Skip-Gram模型进行指令嵌入生成。

具体来说，Skip-Gram模型需要通过学习得到一个映射函数 $V \to R^d$，其中 V 是词汇表，即所有可能的词，d 是嵌入向量空间的维度。在Skip-Gram模型中，给定一个特定中心词 w_c，模型的输出预测目标是其上下文词 $\{w_{c-k},\cdots,w_{c-1},w_{c+1},\cdots,w_{c+k}\}$ 的出现概率 P，其中 k 是上下文滑动窗口的尺寸。对于特定中心词 w_c，上下文中每个相邻词 w_o 的条件概率为

$$P(w_o|w_c)=\frac{\exp(\boldsymbol{v}_{w_o}\cdot\boldsymbol{u}_{w_c})}{\sum_{w\in\mathcal{V}}\exp(\boldsymbol{v}_{w_o}\cdot\boldsymbol{u}_{w_c})} \tag{1}$$

其中，$\boldsymbol{u}_{w_c}$ 和 $\boldsymbol{v}_{w_o}$ 分别是中心词 w_c 和上下文词 w_o 的嵌入向量。

对于序列长度为 T 的汇编指令，Skip-Gram模型的目标是最大化对数似然函数 J

$$J=\frac{1}{T}\sum_{t=1}^{T}\sum_{-k\leqslant j\leqslant k,j\neq 0}\ln P(w_{t+j}|w_t) \tag{2}$$

考虑到Skip-Gram模型中对整个词汇表进行求和会带来较大的时间开销，因此本文系统采用负采样模型来优化Skip-Gram模型。负采样通过采样少量负样本来提高效率，在改善词向量质量的同时实现更鲁棒的语义捕捉。最终，Vul-CFG中每条指令都被转换为嵌入向量的形式。

2.3.2　基本块嵌入

代码块由若干条指令构成，指令之间通常存在顺序性和上下文依赖关系。为了更好地捕捉指令序列的长距离上下文依赖关系，本文系统采用长短期记忆网络（LSTM）来聚合指令序列信息，生成代码块的块嵌入。LSTM作为一种改进的循环神经网络，专门用于解决序列数据中的长距离依赖问题[32]，通过引入记忆单元和多维度的门控机制来控制信息的流动，能够有效地捕获复杂的序列关系。

采用LSTM进行基本块嵌入生成，具体流程如下。首先，对LSTM中隐藏状态 h_0 和记忆单元状态 c_0 进行初始化，将它们都初始化为零向量。接着，将代码块 B 输入LSTM中。代码块 B 包含该块中的指令嵌入序列 $\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_T\}$，作为LSTM的原始输入特征，其中，$\boldsymbol{x}_t\in\boldsymbol{R}^d$ 是由2.3.1节生成的 d 维指令嵌入向量，T 表示基本块中的指令数量。为了适应LSTM的批处理训练，对于长度不足的序列进行零填充，对于过长的序列进行截断处理。在每个时间步 t，LSTM单元根据当前指令嵌入 $\boldsymbol{x}_t$、上一时刻的隐藏状态 $\boldsymbol{h}_{t-1}$ 以及记忆单元状态 $\boldsymbol{c}_{t-1}$ 进行更新。完整的计算过程如式(3)~式(8)所示。

$$\boldsymbol{i}_t=\sigma\left(\boldsymbol{W}_{xi}\boldsymbol{x}_t+\boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}+\boldsymbol{b}_{\mathrm{i}}\right) \tag{3}$$

$$\boldsymbol{f}_t=\sigma\left(\boldsymbol{W}_{xf}\boldsymbol{x}_t+\boldsymbol{W}_{hf}\boldsymbol{h}_{t-1}+\boldsymbol{b}_{\mathrm{f}}\right) \tag{4}$$

$$\boldsymbol{o}_t=\sigma\left(\boldsymbol{W}_{xo}\boldsymbol{x}_t+\boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}+\boldsymbol{b}_{\mathrm{o}}\right) \tag{5}$$

$$\tilde{\boldsymbol{c}}_t=\tanh\left(\boldsymbol{W}_{xc}\boldsymbol{x}_t+\boldsymbol{W}_{hc}\boldsymbol{h}_{t-1}+\boldsymbol{b}_c\right) \tag{6}$$

$$\boldsymbol{c}_t=\boldsymbol{f}_t\odot\boldsymbol{c}_{t-1}+\boldsymbol{i}_t\odot\tilde{\boldsymbol{c}}_t \tag{7}$$

$$\boldsymbol{h}_t=\boldsymbol{o}_t\odot\tanh\left(\boldsymbol{c}_t\right) \tag{8}$$

其中，$\boldsymbol{i}_t$、$\boldsymbol{f}_t$ 和 $\boldsymbol{o}_t$ 分别表示输入门、遗忘门和输出门，用于控制信息的流动；$\tilde{\boldsymbol{c}}_t$ 表示当前时刻的候选细胞状态，$\boldsymbol{c}_t$ 和 $\boldsymbol{h}_t$ 分别表示更新后的细胞状态和隐藏状态；$\sigma(\cdot)$ 为Sigmoid激活函数，$\tanh(\cdot)$ 为双曲正切激活函数，$\odot$ 表示矩阵的哈达玛积；$\boldsymbol{W}_{xi}$、$\boldsymbol{W}_{xf}$、$\boldsymbol{W}_{xo}$ 和 $\boldsymbol{W}_{xc}$ 为输入权重矩阵，而 $\boldsymbol{W}_{hi}$、$\boldsymbol{W}_{hf}$、$\boldsymbol{W}_{ho}$ 和 $\boldsymbol{W}_{hc}$ 为循环权重矩阵，$\boldsymbol{b}_i$、$\boldsymbol{b}_f$、$\boldsymbol{b}_o$ 和 $\boldsymbol{b}_c$ 为偏置向量。

最终，代码块 B 在最后一个时间步 T 输出隐藏状态 $\boldsymbol{h}_T$。由于LSTM的循环结构和门控机制，$\boldsymbol{h}_T$ 已经编码并且聚合了从指令 $\boldsymbol{x}_1$ 到 $\boldsymbol{x}_T$ 的全部序列信息，因此 $\boldsymbol{h}_T$ 被视为整个基本块的语义摘要，即该代码块 B 的块嵌入 $\boldsymbol{h}_B$。

2.3.3　图嵌入

经过指令嵌入和基本块嵌入，Vul-CFG中各个代码块都被转换为块嵌入向量的形式。然而，仅有单个块的语义是不够的。考虑到命令注入漏洞的利用路径通常会跨越多个基本块，而这些基本块之间的控制流关系对最终的漏洞判决起着重要作用。因此，需要进一步对整个Vul-CFG的结构信息进行建模。参考相关研究[11-12]，本系统引入Structure2vec来生成Vul-CFG的图嵌入。Structure2vec是一种能够捕获节点特征及图结构关系的图嵌入网络，可以很好地处理Vul-CFG这种图结构数据。本系统直接使用由基本块嵌入部分生成的深层语义块嵌入向量作为图节点的初始特征，从而构建一个端到端的特

征学习系统，避免了相关研究中对人工特征工程的依赖[11-12,14]，具体流程如下。

1) 首先，构建层次化网络的顶层输入。每个Vul-CFG被形式化为图结构$g=(V,E)$。其中，V是图中顶点的集合，表示代码块，E是边集合，代表控制流依赖关系。输入特征包含两个部分：一是拓扑结构特征，由边集合E构成的邻接矩阵表示；二是节点属性特征，对于图中每个顶点v，采用2.3.2节输出的深层语义块嵌入向量$\boldsymbol{h}_B$作为其初始特征向量$\boldsymbol{x}_v$，从而实现了局部语义向全局结构的无损传递。

2) 根据边集合E中的控制流关系，初始化图中每个顶点v的邻居集合$N(v)$。

3) 对于图中每个顶点v，初始化其嵌入$\boldsymbol{\mu}_v^{(0)}=0$，表示初始时刻尚未融合邻居信息。

4) 通过消息传递和嵌入更新机制，将每个顶点的信息与邻居顶点的结构化信息逐步融合，捕捉顶点间的依赖关系。嵌入更新使用的非线性聚合函数为$\boldsymbol{\mu}_v^{(t+1)}=\tanh(\boldsymbol{W}_1\boldsymbol{x}_v+\sigma(\sum_{u\in N(v)}\boldsymbol{\mu}_u^{(t)}))$，其中，$t$表示$t$轮更新迭代；$\boldsymbol{W}_1$为权重矩阵，用于对顶点特征$\boldsymbol{x}_v$进行线性变换，捕捉顶点的本地信息；$\sigma$为非线性激活函数，用于对邻居顶点聚合的消息进行非线性变换。

5) 经过T轮迭代更新后，每个顶点的嵌入充分包含其自身特征、本地结构信息以及多跳邻居顶点的远程依赖信息。为了获取能够表征整个图的嵌入向量，将所有顶点的嵌入$\boldsymbol{\mu}_v^{(T)}$聚合生成最终全局的图嵌入$\boldsymbol{\mu}_g$。聚合公式为$\boldsymbol{\mu}_g=\boldsymbol{W}_2(\sum_{v\in V}\boldsymbol{\mu}_v^{(T)})$，其中，$\boldsymbol{W}_2$是线性权重向量。

经过上述操作，每个Vul-CFG g都生成了对应的图嵌入向量$\boldsymbol{\mu}_g$，该嵌入向量充分融合了图结构和顶点属性的特性，能够用于后续的漏洞检测任务。

2.4 嵌入特征相似性检测

经过图嵌入后，每个Vul-CFG都被表示为一个浓缩了其语义与结构信息的向量。本模块的目标是设计一个能够精确度量不同CFG间语义相似性的模型，从而实现对已知漏洞的检测。传统的分类模型难以处理开放世界中不断出现的新代码变体，特别是命令注入漏洞的变体，而度量学习可以有效地解决该问题。度量学习的核心思想是学习一个高效的嵌入空间，在该空间中，语义相似的样本在几何上相互靠近，而语义不同的样本则相互远离。

为此，本模块使用经典的孪生神经网络架构结合Structure2vec，来比较嵌入向量的相似性。两个共享相同权重和参数的Structure2vec被嵌入Siamese架构中，分别对两个Vul-CFG g_1和g_2进行处理，生成两个图嵌入向量$\boldsymbol{\mu}_{g_1}$和$\boldsymbol{\mu}_{g_2}$。Siamese架构的训练基于构建的正负样本对和对比损失函数，样本对的构建策略如下。正样本对由源自同一漏洞但属于不同固件版本或CPU架构的两个Vul-CFG构成，对应的标签值$y=1$。该样本对旨在训练模型学习识别具有相同漏洞逻辑的、跨平台的代码变体。负样本对对应的标签值$y=0$，包含两种类型：①来自同一固件的Vul-CFG与其在补丁版本中对应函数构成的Vul-CFG；②来自两个完全不同漏洞对应的Vul-CFG。该样本对旨在训练模型区分细微的语义差异以及本质不同的漏洞模式，从而平衡系统的泛化能力与敏感度。为了防止系统仅通过识别宏观结构差异来获取高分，本文在训练过程中严格控制了负样本对中两种类型样本的比例，从而迫使损失函数优化的重心始终聚焦于消除细微的语义差异。

对于输入的一对嵌入向量$(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})$及其标签$y$，对比损失函数$L$的定义为

$$L(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2},y)=y\cdot D(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})^2+(1-y)\cdot \max(0,m-D(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2}))^2 \tag{9}$$

其中，$D(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})$代表两个向量之间的欧氏距离，即$D(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})=\|\boldsymbol{\mu}_{g_1}-\boldsymbol{\mu}_{g_2}\|_2$；$m$是一个预设的超参数，定义为负样本对在嵌入空间中应保持的最小距离。

训练完成后，将已知漏洞的Vul-CFG和待测固件中提取出的Vul-CFG输入Siamese架构中，得到两个嵌入向量并继续输入余弦函数中。通过式(10)计算它们的相似性得分$\mathrm{CosineSim}(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})$。

$$\mathrm{CosineSim}(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})=\frac{\boldsymbol{\mu}_{g_1}\cdot\boldsymbol{\mu}_{g_2}}{\|\boldsymbol{\mu}_{g_1}\|\ \|\boldsymbol{\mu}_{g_2}\|} \tag{10}$$

其中，$\boldsymbol{\mu}_{g_1}\cdot\boldsymbol{\mu}_{g_2}$表示两个嵌入向量的点积，即$\boldsymbol{\mu}_{g_1}\cdot\boldsymbol{\mu}_{g_2}=\sum_{i=1}^{d}\boldsymbol{\mu}_{g_1}^{(i)}\cdot\boldsymbol{\mu}_{g_2}^{(i)}$，$d$是嵌入向量的维度，$\|\boldsymbol{\mu}_{g_1}\|$表示

向量$\boldsymbol{\mu}_{g_1}$的L2范数，即$\|\boldsymbol{\mu}_{g_1}\| = \sqrt{\sum_{i=1}^{d}(\boldsymbol{\mu}_{g_1}^{(i)})^2}$。相似性得分$\mathrm{CosineSim}(\boldsymbol{\mu}_{g_1},\boldsymbol{\mu}_{g_2})$的结果范围为[−1,1]，得分越接近1，表示两个向量越相似；得分越接近−1，表示两个向量越不相似。如果其中一个向量来源于目标固件，另一个向量来源于漏洞固件，则可以通过相似性得分来判断目标固件中命令注入漏洞的存在性。

3　实验评估

3.1　实验环境

本文所有实验均在一台高性能服务器上完成，硬件配置为一个 Intel Core i9-13900K CPU（主频 3.00 GHz），并配备了 64 GB 的系统内存，以及 24 GB 显存的NVIDIA GeForce RTX 4090 GPU。软件层面选用 Ubuntu 20.04.3 作为操作系统。使用 Binwalk（版本 v2.3.4）进行固件解包，并借助 IDA Pro 9.2 进行二进制文件的反汇编与控制流图提取。

CIDefuse 基于 PyTorch 框架（版本 2.1.0）和 Python（版本3.13）实现。在模型训练过程中，选用 Adam W 作为优化器，初始学习率设置为 1×10^{-4}，批处理大小为128。模型的内部嵌入向量维度 d 统一为256维，图神经网络的消息传递迭代轮数 T 为10。为了防止过拟合并提高训练效率，本文设定了最多100个训练周期，并启用了早停机制。对比损失函数的边界值 m 设置为1.0。

3.2　实验数据集

1) 数据集1。本文系统性分析CNVD和通用漏洞披露（common vulnerabilities and exposures, CVE）数据库，收集了2020年1月至2025年6月公开披露的物联网设备命令注入漏洞。通过人工甄别来确保其有效性，并尽量从多重渠道获取对应的固件。对于难以获取漏洞固件的漏洞，进行舍弃。最终，经过去重、筛选等操作，数据集1共包含185个不同的命令注入漏洞。围绕这些漏洞，收集了562个受影响的固件版本。同时，针对上述漏洞中的一部分，成功获取了421个对应的补丁固件版本。补丁固件数量与漏洞固件数量并非严格的一一对应关系，存在部分漏洞缺乏对应补丁固件的情况。这一数量差异客观反映了物联网固件分析在现实场景中面临的数据获取挑战。许多受影响的旧款设备已停止支持，或者官方未提供历史固件的公开存档，导致部分漏洞固件无法获取其对应的修复版本。

2) 数据集2。该数据集旨在训练和验证Siamese网络对漏洞代码语义相似性的判别能力，数据来源于数据集1中70%的固件。本文利用2.2节提出的方法，从固件中提取Vul-CFG，并根据2.4节的策略来构建正负样本对，其中补丁样本在负样本中占据70%。通过上述策略，能够确保CIDefuse的高检测性能并非源于对简单任务的依赖，而是在保持对异构代码宏观区分能力的同时，有效获得区分漏洞与补丁微观语义的能力。最终，共生成了117 560个函数样本对（正负样本比例控制为1:1），并将其以8:1:1的比例划分为训练集、验证集和测试集。

3) 数据集3。该数据集由数据集1中剩余的30%固件构成，其目的是在模拟真实场景下对本文系统与其他基线方法的检测性能进行端到端对比。为确保评估的公正性与客观性，数据集3中的所有固件均未在模型训练或验证阶段被使用，从而保证了对模型泛化能力的公正评估。

3.3　评估指标

为了客观、定量地评估本文系统的性能，本节采用了在二进制分类任务中被广泛认可的一系列标准评估指标，具体包括以下定义。

1) 真正例（true positive, TP）：实际为漏洞样本，且被模型正确预测为漏洞。

2) 假正例（false positive, FP）：实际为无漏洞样本，但被模型错误预测为漏洞。

3) 真反例（true negative, TN）：实际为无漏洞样本，且被模型正确预测为无漏洞。

4) 假反例（false negative, FN）：实际为漏洞样本，但被模型错误预测为无漏洞。

基于以上定义，本文采用5个核心指标进行评估，包括精确率（Precision）、召回率（Recall）、F1-Score（F1分数）、准确率（Accuracy）和假阳性率（false positive rate, FPR）。

3.4　性能评估

3.4.1　最优参数评估

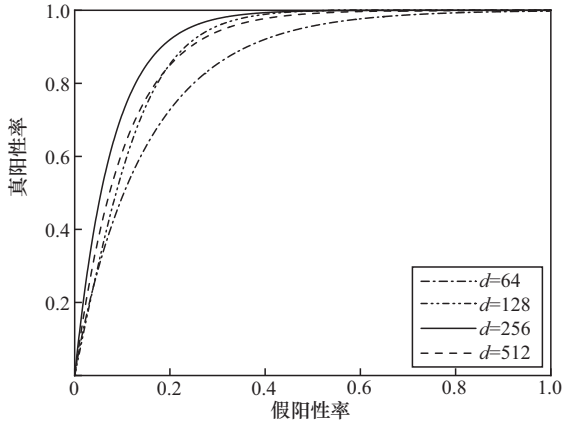
本节首先对层次化图嵌入网络中的关键参数进行评估与确定。实验采用控制变量法，利用数据集2的训练集训练模型，并在验证集上评估嵌入向量维

度d、消息传递迭代轮数T和超参数m对模型检测性能的影响。嵌入维度决定了语义向量的特征表达容量，实验测试了嵌入维度d为主流的64、128、256和512时的模型表现。本文采用受试者操作特征（receiver operating characteristic, ROC）曲线和AUC作为评估指标。ROC曲线的理想情况位于（0,1）点。AUC值代表坐标轴和ROC曲线围成的区域面积，AUC值越接近1，表明模型的性能越好。不同嵌入向量维度下的ROC曲线如图2(a)所示。

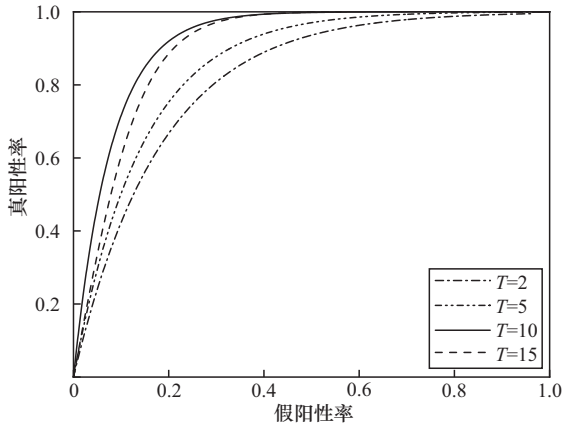
图2(a)结果表明，随着维度从64增加至256，ROC曲线显著向左上方最优点（0,1）移动，这说明模型捕捉代码复杂语义和结构信息的能力随着嵌入维度的增加而增强。然而，当维度进一步增加至512时，ROC曲线出现下降趋势。数据表明，当维度为256时，模型的AUC达到最高的0.920 5；当维度为512时，对应的AUC值降为0.894 5。这可能是由于过高的维度引入了噪声，同时模型在训练过程中出现了过拟合。综合考虑，本文最终确定d = 256为平衡检测精度与计算效率的最优选择。

迭代轮数决定了图神经网络聚合邻域信息的范围，实验测试了迭代轮数T为2、5、10和15时的模型表现。不同迭代轮数下的ROC曲线如图2(b)所示。当T较小（如2或5）时，ROC曲线收缩，说明模型此时难以追踪跨越多个基本块的长距离数据依赖。随着T增加至10，模型能够有效覆盖大多数函数的控制流深度，检测性能达到峰值。当T继续增加至15时，由于图神经网络中常见的过平滑现象，不同节点的特征表示趋于同质化，性能出现下降。因此，本文将迭代轮数设定为T = 10。

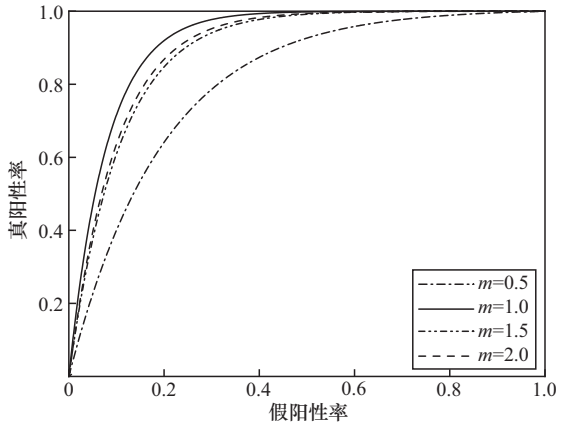
对比损失函数的边界值决定了负样本对在嵌入空间中应保持的最小距离，实验测试了边界值m为0.5、1.0、1.5和2.0时的模型表现。不同边界值下的ROC曲线如图2(c)所示。当m较小（如0.5）时，ROC曲线位置较低，表明由于边界约束过于宽松，模型泛化能力不足。随着m增加至1.0，模型在正负样本间建立了足够的区分度，检测性能达到峰值。然而，当m进一步增加至1.5和2.0时，性能出现下降。这是因为过大的边界约束增加了优化难度，导致损失函数在训练过程中振荡甚至难以收敛。因此，本文将边界值m设定为1.0。



(a)不同嵌入向量维度下的ROC曲线



(b)不同迭代轮数下的ROC曲线



(c)不同边界值下的ROC曲线

图2　不同模型参数对模型性能的影响

3.4.2　代码相似性检测性能

CIDefuse的核心思想是将漏洞检测转化为计算目标代码与已知漏洞代码的相似度，因此，代码相似性检测的准确性直接决定了漏洞检测能力的上限。基于已确定的最优参数配置，本节首先评估CIDefuse在二进制代码相似性检测任务上的有效性。然后将其与多款主流的二进制代码相似性检测工具进行性能对比，具体包括Gemini[12]、Vul-

Seeker[14]、FIT[23]、CI-Detector[25]、jTrans[28]、Trex[29]。实验基于数据集 2，并使用其独立的测试集进行性能评估。

由于上述基线工具的输入并非 Vul-CFG 格式，本文根据数据集中的每个 Vul-CFG，定位其在原始二进制文件中关联的单个或多个函数，并为它们生成各自所需的输入表示。具体而言，为 Gemini、CI-Detector 和 FIT 构建 ACFG 和 3LACFG；为 VulSeeker 构建 LSFG；对于 Trex 和 jTrans 这类方法，本文提取相应的汇编指令序列作为输入。此外，考虑到 Vul-CFG 常常涉及跨函数调用，本文对涉及多个函数的样本采用了拼接策略。将这些函数对应的 ACFG、3LACFG 或 LSFG 进行拼接以构建一个更大的图，并将它们的汇编指令序列连接后分别输入 jTrans 和 Trex，从而为这些基线模型提供尽可能完整的跨函数语义。

Gemini、FIT 等基线方法最初旨在解决函数级代码相似性问题，并未针对拼接后结构复杂的跨函数图进行特定优化。然而，鉴于命令注入漏洞普遍涉及跨函数的数据流依赖，有效的漏洞检测强烈依赖于完整的语义上下文信息。采用拼接策略能够最大程度地保留漏洞的上下文完整性，从而在统一的语义维度上考察各方法对跨函数漏洞的捕获能力。因此，本文采用拼接策略来提供完整的跨函数语义。

所有基线模型均使用其官方推荐的最优超参数进行训练，并应用提前停止策略。训练完成后，本文在测试集上评估各模型的性能。图 3 展示了 CIDefuse 与各基线工具在测试集上的 ROC 曲线。

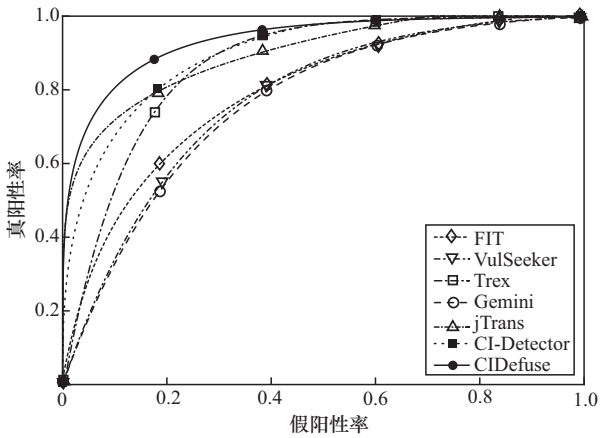


图 3　CIDefuse 与各基线工具在测试集上的 ROC 曲线

如图 3 所示，CIDefuse 的 ROC 曲线位于所有基线方法的上方，其 AUC 值也大于其他所有方法。基于 ACFG 和 LSFG 的 Gemini 和 VulSeeker 表现出相近的性能，AUC 值分别为 0.763 0 和 0.771 4。这两种方法主要依赖于对 CFG 的结构化特征进行匹配。然而，这两种方法对指令序列的深层语义信息挖掘不足，导致在面对语义相似但结构有变的代码变体时识别能力受限。

FIT 表现有所提升，AUC 值达到 0.790 7。通过在函数、基本块和指令 3 个层面提取更丰富的静态属性，FIT 增强了对代码的表征能力，但其本质仍未摆脱对静态图结构的依赖，同时输入的图结构较为冗余。Trex 取得了 0.868 4 的优异 AUC 值，其性能优势得益于引入了动态执行轨迹和汇编指令序列。然而，该方法侧重于线性序列信息，可能无法充分利用代码路径内的结构化信息，导致在区分某些逻辑复杂但执行路径相似的负样本对时存在一定的局限性。

jTrans 通过将控制流跳转信息显式嵌入 Transformer，成功在序列表示中恢复了程序的拓扑结构，将 AUC 值显著提升至 0.897 9。CI-Detector 则凭借对复杂函数内联结构的模式匹配能力，在处理拼接策略导致的边界模糊样本时展现出更强的鲁棒性，取得了 0.901 5 的次优表现。这充分验证了其在深度学习模型中融合结构特征与深层语义的必要性。

相比之下，本文提出的 CIDefuse 取得了最优的性能，AUC 值高达 0.934 2。首先，不同于 jTrans 等通用相似性检测方法容易受到无关代码噪声的干扰，CIDefuse 不分析孤立或冗余的结构，而是先通过反向可达定义分析精确提取从用户可控输入到危险函数的完整数据流路径，确保了分析的针对性和准确性。同时，这一过程也删减了大量与漏洞无关的路径以及不可达的路径，使相似性比较更为聚焦。其次，其层次化的图嵌入网络实现了优势互补。CIDefuse 既能像 Trex 一样捕捉指令级的执行语义，又能像 FIT 一样理解全局的图结构，更重要的是，它能在基本块级别学习局部上下文。这种设计使其能够深刻洞察代码的内在逻辑，从而在代码相似性检测任务中表现出最高的准确性和鲁棒性。

3.4.3　漏洞检测性能

基于 3.4.2 节的代码相似性检测结果，可以明

确CIDefuse在检测可疑漏洞代码上的优异性能。本节将场景迁移至真实环境中，进一步评估CIDefuse在端到端的漏洞检测应用场景中的实际表现。本文模拟了安全研究员利用已知漏洞信息在大量固件中搜寻同源漏洞的真实工作流程，旨在衡量CIDefuse及各基线工具在真实固件库中进行漏洞挖掘的准确性和效率，对比基线选取主流的代码相似性检测工具。实验基于为漏洞检测任务构建的数据集3，该数据集共包含286个固件（169个漏洞固件和117个补丁固件），构成了本文的搜索目标库。

经过统计，286个固件中共包含49个已知不同的命令注入漏洞。对于每一个漏洞，本文从其已知实例中提取一个代表性样本作为查询。最终，构建了一个包含49个独立漏洞的查询集。使用CIDefuse及所有基线工具，在数据集3中进行相似性搜索，采用信息检索领域中广泛使用的前k个结果的精确率Precision@k（P@k）和平均精度均值（mean average precision, MAP）作为主要评估指标。P@k衡量在返回的前k个搜索结果中真正属于漏洞的比例，本文计算了所有49个查询的平均P@k，其中k的取值为1、5、10、30和50。MAP综合考虑了所有相关结果的排名情况，能更全面地反映模型的整体排序性能，对排名靠前的正确结果给予更高的权重，MAP值越高，表明模型的综合性能越好。图4展示了CIDefuse与各基线工具在漏洞搜索任务上的总体性能。

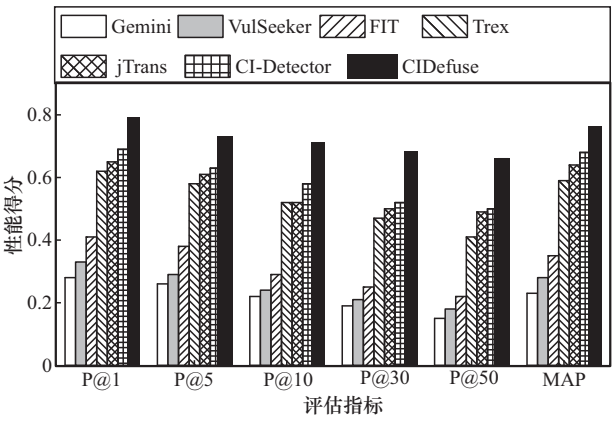


图4 CIDefuse与各基线工具在漏洞搜索任务上的总体性能

随着k值的增加，各基线的性能都呈现一定程度的下降趋势，但是CIDefuse依然在所有P@k指标及MAP指标上处于最高位，特别是在衡量首选命中率的P@1指标上大幅领先于次优的CI-Detector。CIDefuse的P@1指标达到了0.79，这意味着当安全研究员使用CIDefuse进行搜索时，有极高的概率在第一个结果中就直接找到目标漏洞，这极大地提升了漏洞挖掘的效率和确定性。同时，其MAP得分（0.76）也以明显优势领先，证明了其排序结果的整体质量非常高。

相比之下，Gemini和VulSeeker依赖传统人工定义特征和冗余的CFG结构，无法有效捕捉决定漏洞本质的数据流路径，导致其在语义理解上存在较大缺陷。FIT虽然表现稍好，但其性能从P@5迅速下滑，说明其对代码变体的鲁棒性不足。Trex通过学习执行踪迹来捕捉语义，表现相对较好，但它依赖于预先收集的踪迹，可能无法覆盖所有执行路径，从而限制了其在静态分析场景下的精度上限，其检索性能明显弱于融合了结构感知的jTrans和CI-Detector。然而，即便是jTrans和CI-Detector这类先进方法，其分析对象仍包含大量冗余的非漏洞逻辑，导致在搜索精度上存在难以突破的瓶颈。

考虑到CIDefuse融合了数据流分析技术，因此本节还选取了几款典型的传统静态分析工具进行横向对比，包括Karonte[6]、SaTC[7]和Mango[16]。本文从数据集3中随机选取了17个漏洞固件和对应的11个补丁固件共28个固件构成对比数据集，实验设定漏洞只存在于原始固件的二进制文件中。

对于BCSD方法，本文从漏洞固件中提取出所有的漏洞函数链作为查询集，固件中所有的函数作为目标集。检测过程中，BCSD方法计算目标集中各函数与查询集中的相似度，从而判断是否存在漏洞。由于补丁代码仅包含细微的逻辑修复，其向量表征与漏洞代码极度接近。为了有效区分补丁并控制误报，本文经过预实验初步验证后实施了严格的判定标准：当函数间的语义相似度高于0.9时，才判定为“检测到漏洞”，否则视为安全。表1展示了CIDefuse与上述基线工具在该数据集上的详细性能表现，其中平均分析时间指标记录了各工具分析单个二进制文件（含预处理、特征提取与向量生成及相似度比对）所需的平均端到端时间。

由表1可知，CIDefuse的精确率达到了最高的93.75%，F1值达到了90.91%，相较于表现最好的BCSD基线CI-Detector（81.25%）和静态工具

Mango（77.42%），均取得了显著提升。这表明，CIDefuse有效克服了现有方法的局限性：一方面，它通过精确的数据流分析有效过滤了补丁固件中大量无关的伪相似路径，极大地降低了BCSD方法难以避免的误报；另一方面，其层次化语义嵌入网络能够有效捕捉到复杂的，甚至跨函数的漏洞模式，弥补了静态分析因路径解析失败而导致的低召回率缺陷。因此，CIDefuse最终实现了高达88.24%的召回率，且假阳性率仅为9.09%。

表1　　各种方法在对比数据集上的性能

方法	Precision	Recall	F1值	Accuracy	FPR	平均分析时间/min
Karonte[6]	66.67%	39.29%	46.15%	50.00%	27.27%	10.28
SaTC[7]	72.73%	47.06%	57.14%	57.14%	27.27%	12.67
Mango[16]	85.71%	70.59%	77.42%	75.00%	18.18%	8.15
Gemini[12]	44.44%	23.53%	30.77%	35.71%	45.45%	2.51
Vul-Seeker[14]	45.45%	29.41%	35.71%	35.71%	54.55%	2.12
FIT[23]	66.67%	47.06%	55.17%	53.57%	36.36%	3.06
jTrans[28]	76.92%	58.82%	66.67%	64.29%	27.27%	4.33
Trex[29]	71.43%	58.82%	64.52%	60.71%	36.36%	3.47
CI-Detector[25]	86.67%	76.47%	81.25%	78.57%	18.18%	4.75
CIDefuse	93.75%	88.24%	90.91%	89.29%	9.09%	6.04

在检测效率方面，CIDefuse采用轻量级的反向可达定义分析快速完成了对无关代码的剪枝，使计算开销较大的深度学习模型仅需在少量高风险的候选路径上运行。因此，单个二进制文件平均分析时间为6.04 min。尽管其耗时略高于CI-Detector，但考虑到BCSD方法需进行全量函数的特征提取且难以精准区分补丁，CIDefuse仅以微小的时间成本增量，就换取了假阳性率的大幅降低。

本文还使用CIDefuse对数据集3进行未知漏洞的挖掘。通过对输出的漏洞候选路径进行人工核对，CIDefuse不仅成功识别了数据集中全部已知的命令注入漏洞，还发现了3个未被公开的漏洞，如表2所示。作者已对这些漏洞进行申报，并获得CNVD的官方认证，其中两款的评级为中危，一款的评级为高危。3个漏洞的利用链都跨多个函数，进一步说明了命令注入漏洞的高隐蔽性和挖掘难度。

表2　CIDefuse系统挖掘到的未知命令注入漏洞（已申报CNVD）

设备厂商	设备类型	设备型号	固件版本	跨函数	危害等级	CNVD ID
D-Link	路由器	DI-7003G	xxxx	√	高危	CNVD-2025-17832
D-Link	路由器	DI-7003G	xxxx	√	中危	CNVD-2025-20039
TOTO LINK	路由器	LR350	xxxx	√	中危	CNVD-2025-21916

本文使用主流静态分析工具Mango和BCSD方法CI-Detector对上述3个漏洞所在的固件进行分析，设定超过30 min无结果则为超时。实验数据显示，静态分析工具主要出现超时情况，BCSD方法则因缺失跨函数上下文导致漏报，CIDefuse是唯一实现全部成功检测的工具。具体来说，在分析CNVD-2025-17832和CNVD-2025-20039时，由于漏洞涉及复杂的循环结构与多层函数调用，Mango陷入了严重的路径爆炸，分析耗时超过30 min仍无法输出结果。即便在未超时的案例中，耗时也长达20.2 min，且最终因间接跳转导致的约束求解失败而产生漏报。CI-Detector则表现出语义捕捉能力的不足，尽管其平均耗时较短，但均未能检测出这3个漏洞。CIDefuse成功检测出全部3个漏洞，其平均耗时仅为4.7 min，证明了通过反向可达定义分析进行剪枝的策略，不仅有效避免了静态分析的路径爆炸问题，还弥补了BCSD方法在处理跨函数语义时的缺陷，实现了对复杂漏洞链的高效精准定位。

以CNVD-2025-17832为例，CIDefuse的挖掘流程如下。首先，利用Binwalk对目标路由器固件进行解包，提取出内部的55个二进制文件，并借助IDA Pro恢复其控制流图。随后，基于预定义的Source-Sink集合，系统在核心二进制文件jhttpd中锁定危险点，并应用反向可达定义分析算法，精确抽取出5条Vul-CFG。在图嵌入与相似性检测阶段，通过计算发现，其中一条Vul-CFG与已知漏洞CNVD-2025-10952的嵌入向量表现出高度的语义一致性，相似度得分高达0.92，显著超过判定阈值。该路径被定位为一条跨越5个函数的复杂调用链。经过人工审计与漏洞复现，攻击者能够通过在Source注入恶意指令，经由该跨函数链路触发Sink的命令执行，从而验证了检测结果的准确性。

本文遵循负责任的漏洞披露原则，已将这3个漏洞的详细技术报告提交给相关设备厂商。为防止潜在的恶意利用，受影响的具体固件版本将暂时不予公开。

4 结束语

本文针对物联网设备中命令注入漏洞难以兼顾检测精度与效率的难题，提出了一种融合数据流分析与语义嵌入的物联网设备命令注入漏洞检测系统CIDefuse，其核心特点在于数据流剪枝与语义嵌入的有机结合。系统利用反向可达定义分析剔除大量非漏洞路径，有效解决了代码相似性检测方法在区分补丁固件时的高误报问题。同时通过基于神经网络的层次化语义建模，克服了传统静态分析在面对复杂跨函数调用链时的漏报缺陷。

尽管效果显著，CIDefuse仍存在对预处理质量依赖较高、在大规模实时扫描任务中效率有待提升等局限。未来研究将围绕上述瓶颈展开：一方面，计划引入大语言模型作为辅助语义提取器，对汇编指令进行语义摘要，弥补传统NLP模型在处理混淆指令时的语义缺失，同时结合轻量级动态分析，恢复间接跳转目标，解决静态分析中的控制流断裂问题；另一方面，将探索利用局部敏感哈希技术快速过滤非相似函数，仅对高可疑函数进行深度图嵌入计算，同时，利用并行计算技术对图神经网络的信息传递过程进行加速。此外，针对缓冲区溢出等其他污点型漏洞，计划引入数值范围约束机制。通过污点分析同步追踪缓冲区大小与输入数据长度的约束关系，实现对内存破坏类漏洞的有效识别。

参考文献：

[1] Martin L. 57 IoT statistics for 2025[R]. 2025.

[2] Ye J, Fei X, Carnavalet X C D, et al. Detecting command injection vulnerabilities in Linux-based embedded firmware with LLM-based taint analysis of library functions[J]. Computers & Security, 2024, 144: 103971.

[3] 邹福泰, 谭越, 王林, 等. 基于生成对抗网络的僵尸网络检测[J]. 通信学报, 2021, 42(7): 95-106.
Zou F T, Tan Y, Wang L, et al. Botnet detection based on generative adversarial network[J]. Journal on Communications, 2021, 42(7): 95-106.

[4] Bill T. New Mirai botnet infect TBK DVR devices via command injection flaw[R]. 2025.

[5] Cheng K, Li Q, Wang L, et al. DTaint: detecting the taint-style vulnerability in embedded device firmware[C]//Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Piscataway: IEEE Press, 2018: 430-441.

[6] Redini N, Machiry A, Wang R Y, et al. Karonte: detecting insecure multi-binary interactions in embedded firmware[C]//Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2020: 1544-1561.

[7] Chen L B, Wang Y H, Linghu J, et al. SaTC: shared-keyword aware taint checking for detecting bugs in embedded systems[J]. IEEE Transactions on Dependable and Secure Computing, 2024, 21(4): 2421-2433.

[8] Chen J Y, Diao W R, Zhao Q C, et al. IoTFuzzer: discovering memory corruptions in IoT through app-based fuzzing[C]//Proceedings 2018 Network and Distributed System Security Symposium. Internet Society, 2018: 1-15.

[9] Srivastava P, Peng H, Li J H, et al. FirmFuzz: automated IoT firmware introspection and analysis[C]//Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things. New York: ACM Press, 2019: 15-21.

[10] Zhang Y, Huo W, Jian K P, et al. ESRFuzzer: an enhanced fuzzing framework for physical SOHO router devices to discover multi-Type vulnerabilities[J]. Cybersecurity, 2021, 4: 24.

[11] Feng Q, Zhou R D, Xu C C, et al. Scalable graph-based bug search for firmware images[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2016: 480-491.

[12] Xu X J, Liu C, Feng Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2017: 363-376.

[13] Gao J, Yang X, Jiang Y, et al. Semantic learning based cross-platform binary vulnerability search for IoT devices[J]. IEEE Transactions on Industrial Informatics, 2021, 17(2): 971-979.

[14] Gao J, Yang X, Fu Y, et al. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary[C]//Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2020: 896-899.

[15] Cheng K, Zheng Y W, Liu T, et al. Detecting vulnerabilities in linux-based embedded firmware with SSE-based on-demand alias analysis[C]//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2023: 360-372.

[16] Gibbs W, Raj A S, Vadayath J M, et al. Operation mango: scalable discovery of taint-style vulnerabilities in binary firmware services[C]//Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24). Berkeley: USENIX Association, 2024: 7123-7139.

[17] Gao Z C, Zhang C, Liu H T, et al. Faster and better: detecting vulnerabilities in linux-based IoT firmware with optimized reaching definition analysis[C]//Proceedings of the 2024 Network and Distributed System Security Symposium. Virginia: the Internet Society, 2024: 26.

[18] Yin X K, Cai R J, Zhu X Y, et al. Precise discovery of more taint-style vulnerabilities in embedded firmware[J]. IEEE Transactions on Dependable and Secure Computing, 2025, 22(2): 1365-1382.

[19] Zheng Y, Davanian A, Yin H, et al. FIRM-AFL: high-throughput grey-box fuzzing of IoT firmware via augmented process emulation[C]//Pro-

ceedings of the 28th USENIX Security Symposium (USENIX Security 19).Berkeley: USENIX Association, 2019: 1099-1114.

[20] Yu J, Kim J, Yun Y, et al. Poster: combining fuzzing with concolic execution for IoT firmware testing[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2023: 3564-3566.

[21] Liu H T, Gan S T, Zhang C, et al. Labrador: response guided directed fuzzing for black-box IoT devices[C]//Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2024: 1920-1938.

[22] Liu K Z, Yang M, Ling Z, et al. RIoTFuzzer: companion app assisted remote fuzzing for detecting vulnerabilities in IoT devices[C]//Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2024: 2341-2354.

[23] Liang H L, Xie Z S, Chen Y X, et al. FIT: Inspect vulnerabilities in cross-architecture firmware by deep learning and bipartite matching[J]. Computers & Security, 2020, 99: 102032.

[24] Zhang Y T, Fang B X, Xiong Z H, et al. A semantics-based approach on binary function similarity detection[J]. IEEE Internet of Things Journal, 2024, 11(15): 25910-25924.

[25] JIA A, FAN M, XU X, et al. Cross-inlining binary function similarity detection[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. Piscataway: IEEE Press, 2024: 1-13.

[26] Ding S H H, Fung B C M, Charland P. Asm2Vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]//Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2019: 472-489.

[27] Li X, Qu Y, Yin H. PalmTree: learning an assembly language model for instruction embedding[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2021: 3236-3251.

[28] Wang H, Qu W J, Katz G, et al. jTrans: jump-aware transformer for binary code similarity detection[C]//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2022: 1-13.

[29] Pei K X, Xuan Z, Yang J F, et al. Learning approximate execution semantics from traces for binary function similarity[J]. IEEE Transactions on Software Engineering, 2023, 49(4): 2776-2790.

[30] Jiang S, Fu C, Qian Y K, et al. IFAttn: Binary code similarity analysis based on interpretable features with attention[J]. Computers & Security, 2022, 120: 102804.

[31] Mallik A, Kumar S. Word2Vec and LSTM based deep learning technique for context-free fake news detection[J]. Multimedia Tools and Applications, 2024, 83(1): 919-940.

[32] Bamber S S, Katkuri A V R, Sharma S, et al. A hybrid CNN-LSTM approach for intelligent cyber intrusion detection system[J]. Computers & Security, 2025, 148: 104146.

[作者简介]

陈霄（1995– ），男，江苏宿迁人，博士，南京邮电大学讲师，主要研究方向为物联网安全、软件安全、机器学习等。

沙乐天（1985– ），男，江苏徐州人，博士，南京邮电大学教授、硕士生导师，主要研究方向为漏洞挖掘、恶意软件分析、软件安全等。

潘家晔（1985– ），男，江苏扬州人，博士，南京邮电大学副教授，主要研究方向为系统安全、软件安全、网络安全等。

孙瑞（1985– ），男，江苏宿迁人，南京大学医学院附属鼓楼医院工程师，主要研究方向为医院智能化管理、信息安全等。

董建阔（1992– ），男，河北邢台人，博士，南京邮电大学副教授、硕士生导师，主要研究方向为公钥密码、后量子密码、并行计算等。

肖甫（1980– ），男，湖南邵阳人，博士，南京邮电大学教授、博士生导师，主要研究方向为无线传感网、物联网安全、信息安全等。