引用格式:沈毅,于璐,赵军,等.一种基于随机游走的固件代码补丁修复判定方法[J]. 信息对抗技术,2024, 3(4):50-62[SHEN Yi, YU Lu, ZHAO Jun, et al. A method for determining firmware code patch repair based on random walk[J]. Information Countermeasure Technology, 2024, 3(4):50-62. (in Chinese)]

一种基于随机游走的固件代码补丁修复判定方法

沈 毅[1,2],于 璐[1,2*],赵 军[1,2],张童童[3]

(1. 国防科技大学电子对抗学院,安徽合肥 230037; 2. 网络空间安全态势感知与评估安徽省重点实验室,安徽合肥 230037; 3. 31455 部队,辽宁沈阳 110000)

摘 要 确定目标程序中的漏洞是否被修复,是软件安全性检测的途径之一,能够提高程序安全性。提出了一种基于随机游走的固件补丁存在性判定方法,利用程序分析技术对二进制固件函数进行代码特征提取和分析,判断固件中的函数是否进行了补丁修复,实现对固件漏洞代码的检测。该方法分别对固件中的目标函数和对应的漏洞函数、固件中的目标函数和补丁函数构建表征代码相似性程度的伴随图,并使用随机游走的方法筛选伴随图中的重要节点。基于重要节点信息,可以判断目标函数与漏洞函数、补丁函数的相似程度,实现对目标函数补丁修复情况的自动化判断。实验证明,提出的方法可以实现对固件补丁修复情况的高效判断,为提高二进制固件安全性提供支持。

关键词 漏洞分析;随机游走;补丁修复;固件漏洞

中图分类号 TP 393 **文章编号** 2097-163X(2024)04-0050-13

文献标志码 A **DOI** 10.12399/j.issn.2097-163x.2024.04.004

A method for determining firmware code patch repair based on random walk

SHEN Yi[1,2], YU Lu[1,2*], ZHAO Jun[1,2], ZHANG Tongtong[3]

(1. College of Electronic Engineering, National University of Defense Technology, Hefei 230007, China; 2. Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230007, China; 3. Unit 31455 of PLA, Shenyang 110000, China)

Abstract Determining whether vulnerabilities in the target program have been fixed is one of the approaches to software security detection, which can enhance the safety of the program. A method for determining the existence of firmware patches based on random walks was proposed. This method utilizes program analysis techniques to extract and analyze code features from binary firmware functions, judging whether functions in the firmware have undergone patch repairs, and achieving the detection of firmware vulnerability codes. The method constructs accompanying graphs representing the degree of code similarity between the target function in the firmware and its corresponding vulnerability function, as well as between the target function and the patch function. Important nodes in the accompanying graph are then selected using a random walk approach. Based on the information from these important nodes, it is possible to determine the similarity between the target function and both the vulnerability function and the patch function, enabling automated judgment on the patch repair

收稿日期:2023-08-30 **修回日期**:2023-11-10

通信作者:于璐,E-mail:yulu@nudt.edu.cn

基金项目:国家自然科学基金资助项目(62202484)

status of the target function. Experimental results show that the proposed method can efficiently judge the patch repair status of firmware, providing support for enhancing the security of binary firmware.

Keywords　vulnerability analysis; random walk; patch repair; firmware vulnerability

0　引言

云计算、大数据、人工智能、物联网(Internet of Things, IoT)、移动通信等信息技术的快速发展，一方面给人们的生活带来极大的便捷，另一方面，信息窃取、勒索攻击、钓鱼攻击等安全事件也日益增多，不仅导致信息泄漏和经济损失，而且对涉及国计民生的关键基础设施和新技术应用造成极大的安全隐患，隐性损失难以估量。随着 IoT 设备的广泛使用，传统架构的程序也逐渐适配到基于 MIPS 或者 ARM 架构开发的 IoT 设备中，IoT 固件中的安全漏洞也引发了安全人员的广泛关注。美国加州大学和密歇根大学的研究人员对 IoT 设备进行研究发现，IoT 设备(如路由器、防火墙，打印机等)在漏洞公布后，往往未及时进行漏洞修补[1]。即使发现了固件中的漏洞，对其进行修复一般需要手工更新补丁程序，比传统 PC 机上的漏洞修复更为困难，且出于对系统稳定性的考虑，厂商一般不会及时对固件进行补丁修复。导致漏洞公开和补丁发布后很长一段时间，大部分固件仍然没有进行补丁修复，依然存在安全问题。据 Shodan 统计，在发现著名的“心脏滴血”漏洞 3 年之后，依然有超过 20 万个 IoT 设备受该漏洞的影响。究其原因，主要在于对漏洞代码进行修复时，往往不会过多修改代码的内容，而是添加更多验证代码，在控制流图(control flow graph, CFG)添加一个验证分支，或者修改变量的长度与类型，并未对代码的整体结构做过多的更改。与源码已知的程序分析[2]相比，对二进制固件进行分析难度更大，因此对程序中可能存在漏洞的函数，特别是对固件文件进行补丁存在性判断是固件设备安全分析重要的研究内容之一[3]。

代码相似性比较[4]的方法可以应用于补丁比较中，可以对 2 组代码进行分析和比较，判断其相似性。在工业界和学术界，代码相似性比较都引发了研究人员的兴趣。代码比较工具，如 Diaphora[5]、BinDiff[6]和 Turbodiff[7]可以对二进制代码进行函数级别和基本块级别的比较，然而这些代码比对工具主要对二进制程序中的函数和基本块对应关系进行判断，并未专门针对补丁存在性判断进行设计；同时，代码映射关系判断是基于简单特征的结构化签名，特征提取较为简单，可能存在映射关系判断错误的情况。同时，结构化比较的方法扩展性差，基于简单代码特征的结构化签名的方法在代码差异较大情况下，对映射关系的分析存在精确率不高的问题。在学术领域，当前代码相似性比较多借助于机器学习的方法，以函数粒度进行分析，并取得了较好的效果[8-17]，但这些方法无法直接进行粒度更细的补丁存在性判断。

针对代码补丁修复情况判断的研究主要包括基于程序分析和基于机器学习 2 种主流方法。其中，针对基于程序分析的代码补丁修复性判断的研究较为分散，BinXray[18]利用补丁签名判断代码中的漏洞是否被修复，通过比较漏洞函数和补丁后的函数，得到发生变化的基本块(changed basic block, CBB)，并对 CBB 的前后边缘基本块进行提取，生成基本块序列作为补丁签名，在此基础上确定目标函数是否被修复，这种方法对 CBB 及其前后边缘基本块提取的粒度较细且存在一定难度，分析结果对 CBB 的依赖较强，一旦出现 CBB 判断错误，就会大大影响后续的分析结果；1dVul[19]以生成到达补丁代码(程序崩溃点)的输入为目的，对补丁代码和漏洞代码的比较结果进行分析，找到关键分支点，之后交替执行导向式模糊测试和导向式符号执行，生成到达关键分支点的输入；XU 等[20]认为，安全补丁引入的语义变化相对较小，若补丁后语义影响小，则认为相应代码为补丁，基于识别的特定语句，结合污点分析的方法找到漏洞模式，得到的模式用于漏洞搜索和补丁查找中。SUN 等[21]通过分析基本块之间的数据流切片，识别补丁代码对原二进制程序的修改。通过解析并分析这些修改，提取包含代表性运算符和操作数来源的补丁签名，然后通过词法比较匹配补丁签名来识

别补丁的存在；XIAO 等[22]也是通过使用切片的方法分别对漏洞和补丁进行签名，但是数据流切片方法对资源消耗较大，效率相对较低。在补丁修复和机器学习方法结合的研究中，文献[23-24]结合机器学习方法获取代码的提交信息与漏洞之间的关联关系，以此判断安全相关漏洞或是安全无关漏洞。文献[25-27]针对源码已知的程序进行漏洞和补丁指纹的生成，对程序是否进行补丁修复进行判断。PatchDB[28]则是重点关注补丁数据库的构建，通过收集漏洞数据库已有漏洞中的补丁数据、开源代码网站 Github 的代码以及利用补丁合成方法生成代码等 3 种途径构建机器学习的训练集。可以看出，针对补丁代码修复情况的研究目的不一，有的是为了复现程序的崩溃现场辅助漏洞分析，有的则重点关注于基于机器学习的漏洞信息抽取，且部分方法使用数据流切片、符号执行等方法，对于系统资源的需求较大，存在效率问题；而专门进行代码相似性比较的工具（Diaphora 、BinDiff 和 Turbodiff 等）是为了辅助程序分析，虽然通用性较强，但是未能实现对补丁代码的针对性分析和比较。

针对以上问题，本文聚焦固件文件的补丁修复判断方法，在利用相似性方法获取函数级别的比较代码后，以源码未知的二进制固件文件作为研究对象，针对漏洞修复过程中存在的代码变化不大的现实情况，对是否修复目标固件中的漏洞进行精细化判断。本文提出一种基于随机游走的固件补丁修复情况的判断方法，研究函数内部细粒度的补丁存在性判定方法，构建表征代码相似程度的伴随图，并基于随机游走方法判断目标文件中的函数是否进行了漏洞修复。

1 方法概述

基于随机游走的固件代码补丁存在性判断方法主要是对漏洞进行修复时，在代码变化不大的情况下，对固件程序是否已经进行了补丁修复给予判断；将给定的补丁二进制程序、固件程序和漏洞二进制程序作为分析对象，在已经确定固件程序与补丁对应的函数（也是漏洞对应的函数）的前提下，将补丁函数、漏洞函数分别与要进行补丁存在性判断的目标函数构建待匹配的函数对；对 2 个待匹配的函数代码进行针对性丰富特征提取，并受图匹配方法的启发，构建表征 2 个待匹配函数相似程度的伴随图；使用随机游走的方法得到伴随图中的对应节点，实现更为高效且精细的补丁存在性判断。系统的整体工作流程如图 1 所示。

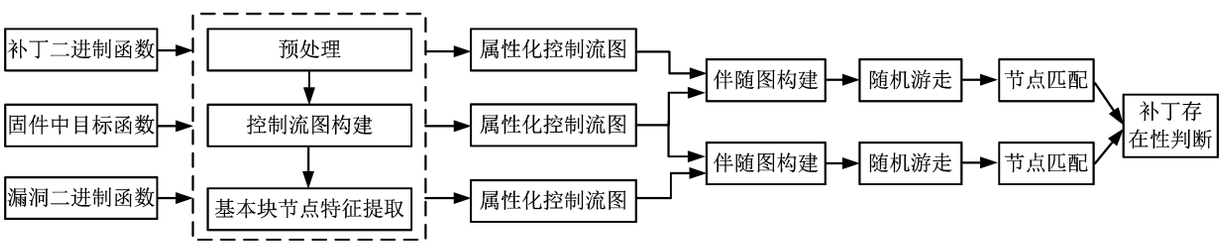


图 1 系统整体工作流程图

Fig. 1 Overall working flowchart of the system

本文提出的方法主要分为 2 个过程：一是构建伴随图，分别提取 2 个待比较函数的基本块节点特征，基于它们之间的基本块节点和边的相似程度构建一个伴随图，图中节点表示待比较函数中基本块节点之间的相似程度。按照该过程，分别构建固件中待分析的目标函数与漏洞函数、补丁函数的伴随图；二是基于伴随图，对其进行随机游走，获取重要节点并利用相应信息，对固件目标函数中是否进行补丁修复进行判断。在第 1 步的伴随图构建中，首先对函数进行预处理。由于目标函数为二进制形式，需要对其进行反汇编，之后基于反汇编分析结果得到 CFG，同时将 CFG 中基本块节点的特征进行提取，进而得到带有节点属性的 CFG，即属性化控制流图（attributed control flow graph，ACFG）；在此基础上，构建表征待比较图中节点和边相似程度的伴随图。将补丁函数和目标函数的 ACFG 以及目标函数和漏洞函数的 ACFG 分别作为输入，构建伴随图。在第 2 步的伴随图上随机游走过程中，使用匈牙利算法对随机游走得到的节点重要度向量进行节点匹配，根据匹配的节点判断目标函数与补丁函数、漏洞函数的相似程度，从而对固件函

数的补丁修复情况进行判断。

2　基于代码特征的伴随图构建

2.1　ACFG 构建

在二进制程序分析中，将二进制程序反汇编为汇编指令后，对指令序列进行基本块划分，得到以基本块为基本单位的 CFG。AHO 等[29]将二进制文件中的基本块看作程序中顺序执行的最大连续执行的指令序列，满足以下 2 个条件：一是控制流只能从基本块的第一个指令进入该块，即单入口；二是除了基本块的最后一个指令，控制流在离开基本块之前不会跳转，即单出口。

在将二进制程序按照基本块进行划分后，基于控制依赖关系构造 CFG。根据基本块的特征描述，如果 2 个基本块 B 和 C 之间存在控制依赖关系，则需要满足以上 2 个条件之一：即有一个从 B 最后一条指令跳转到 C 的起始指令的条件或无条件跳转语句；或者是在指令序列中，基本块 C 的第 1 条指令紧跟在 B 的最后一条指令之后，且在 B 的结尾不存在无条件跳转语句[29]。基于基本块 B 和 C 的控制依赖关系，将 B 称为 C 的前驱节点，将 C 称为 B 的后继节点。

基于以上讨论，CFG 可表示为 $G_{\mathrm{CFG}}=(V,E)$，其中，$v\in V$ 表示 *CFG* 中的基本块节点，$e\in E$ 表示 CFG 中存在控制依赖关系的边。

得到 CFG 后重点对 CFG 中的基本块节点特征进行提取，基本块的代码特征包括代码语法统计特征和代码结构特征。

代码语法统计特征包含基本块级别中的代码的数字化统计特征，考虑到特征提取的效率，本文选择了基本块中的汇编指令条数、调用函数的个数、算术指令的个数、堆栈操作指令的个数、跳转指令的个数 5 类统计特征（用 F_{sta} 表示），这些特征可以通过程序静态分析的方法得到，且对于整个补丁存在性判断的效率影响不大。

代码结构特征是基于函数内部的 CFG，使用 CFG 中基本块节点的结构特征进行表征。本文选择的代码结构特征包含了基本块节点在 CFG 中的介数、入度、出度 3 类结构特征（使用 F_{con} 表示），其中，介数指 CFG 中的节点介数，代表图中所有最短路径中经过该节点的数量比例，是反映节点在图中影响力的全局特征量。

基于基本块的代码语法统计特征和代码结构特征，对 ACFG 进行定义。

定义 1　ACFG 可以使用 $G_{\mathrm{ACFG}}=(V,E,\sigma)$ 表示。其中，V 表示基本块节点集合，E 表示基本块之间的控制依赖边的集合。若基本块 v_i 和 v_j 之间存在控制依赖关系，则 $\langle v_i,v_j\rangle\in E$；同时，$\sigma:V\rightarrow\Sigma$ 表示基本块节点及其属性的映射关系，$\Sigma=\{F_{\mathrm{sta}},F_{\mathrm{con}}\}$。

2.2　伴随图构建

为了判断目标程序是否进行了补丁修复，需要在函数级别对目标固件程序代码（简称“目标代码”）和漏洞代码以及补丁代码分别进行比较，判断目标代码与后两者中的哪一个更为相似。本文提出的基于伴随图的代码相似性判定方法，将待比较代码对应的 ACFG 转换为伴随图，并在伴随图中使用随机游走的方法进行代码相似性分析。

基于伴随图的方法可以将 2 个待比较的图转换为单个图来完成它们的匹配过程。需要基于目标代码和漏洞代码、基于目标代码和补丁代码分别构建伴随图，即基于目标代码和漏洞代码的 ACFG 构建第 1 个伴随图，以及基于目标代码和补丁代码的 ACFG 构建第 2 个伴随图。

伴随图的构建基于节点和边之间的相似程度，即伴随图中的节点和边的权值与 2 个待比较的 ACFG 中的节点相似度和边的相似度有关。具体来说，伴随图中节点的权值反映分别属于 2 个待匹配图中节点之间的相似程度；伴随图中的边的权值则反映待匹配图中边与边之间的相似程度，得到的伴随图为一个无向权值图。

伴随图的构建示例如图 2 所示。在图 2 的左侧分别显示了包含节点(1,2,3)和(a,b,c,d)的 2 个待匹配图(G1 和 G2)，基于此得到包含节点相似度和边相似度的伴随图，如图 2 右侧所示。其中，伴随图节点 1a 表示待匹配图中节点 1 和节点 a 之间的相似度，节点 2b 表示待匹配图中节点 2 和节点 b 之间的相似度；伴随图中 1a 和 2b 之间连接边的权值表示 2 个待匹配图中节点 1 和 2 的边与节点 a 和 b 的边之间的相似度。

伴随图中的节点中包含了 2 个待匹配图中的节点的相似程度，因此使用相似度矩阵（affinity matrix）$\boldsymbol{K}$ 记录节点和边的相似度信息。使用伴随图可以将待匹配图之间的节点匹配过程转换为伴随图中的重要节点过程，即基于 $\boldsymbol{K}$，对伴随图

中的重要节点进行确定，就能够得到对应的 2 个待匹配图中存在对应关系的节点。

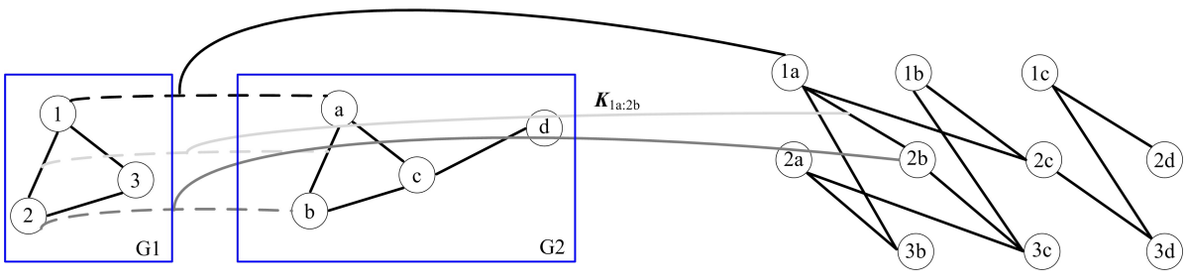


图 2 伴随图构建示例

Fig. 2 Example of association graph construction

计算 $\boldsymbol{K}$ 时，$\boldsymbol{K}$ 中的元素可以使用 $k_{mn:st}$ 表示，对于分别包含节点 m，s 和 n，t 的 2 个待匹配图 G1 和 G2，矩阵中对应的值为：

$$k_{mn:st}=\begin{cases}S_{\text{sim}}(v_m,v_n), & m=s \text{ 且 } n=t\\ S_{\text{sim}}(e_{ms},e_{nt}), & m\neq s \text{ 且 } n\neq t\\ 0, & \text{其他}\end{cases} \tag{1}$$

矩阵 $\boldsymbol{K}$ 中的对角线记录节点权值，非对角线记录边权值，节点权值和边权值分别使用 $S_{\text{sim}}(v_m,v_n)$ 及 $S_{\text{sim}}(e_{ms},e_{nt})$ 表示。其中，$S_{\text{sim}}(v_m,v_n)$ 表示分别属于待匹配图 G1 和 G2 的 2 个节点 m 和 n 的相似度，$S_{\text{sim}}(e_{ms},e_{nt})$ 表示分别属于 G1 和 G2 的边〈m，s〉和〈n，t〉之间的相似度。对于节点 m 和节点 n，其对应的语法统计特征向量 $\boldsymbol{M}=(m_1,m_2,\cdots,m_u)$ 和 $\boldsymbol{N}=(n_1,n_2,\cdots,n_u)$，其中，$m_1,m_2,\cdots,m_u$ 表示节点 m 的 u 个语法统计特征（即 F_{sta} 包含的元素数目），$n_1,n_2,\cdots,n_u$ 为节点 n 的 u 个语法统计特征。

节点的相似程度表示为：

$$S_{\text{sim}}(v_m,v_n)=D(\boldsymbol{M},\boldsymbol{N})=\frac{\min(\|\boldsymbol{M}\|,\|\boldsymbol{N}\|)+0.001}{\sqrt{\sum_{i=1}^{u}(m_i-n_i)^2}+0.001} \tag{2}$$

式中，$\min(\|\boldsymbol{M}\|,\|\boldsymbol{N}\|)$ 表示 2 个向量的长度中更小的值。

而表征边〈m，s〉和〈n，t〉之间相似度的边权值 $S_{\text{sim}}(e_{ms},e_{nt})$ 不仅与对应的节点相似度有关（即与节点的语法统计特征相关），还与节点的结构特征相关。因此使用式(3)将节点的语法统计特征和结构特征的相似度值分别赋不同权值，即权值 α 和权值 β 进行计算。

$$S_{\text{sim}}(e_{ms},e_{nt})=\alpha[S_{\text{sim}}(v_m,v_n)+S_{\text{sim}}(v_s,v_t)]+\beta[S_{\text{sim,con}}(v_m,v_n)+S_{\text{sim,con}}(v_s,v_t)] \tag{3}$$

式中，$S_{\text{sim,con}}(v_m,v_n)$ 表示基于基本块节点 m 和 n 的结构化特征得到的结构相似度值。节点 m 和 n 对应的结构特征向量分别记为 $\boldsymbol{M}_{\text{con}}$ 和 $\boldsymbol{N}_{\text{con}}$，节点之间的结构相似度值 $S_{\text{sim,con}}(v_m,v_n)$ 为：

$$S_{\text{sim,con}}(v_m,v_n)=1+\frac{\boldsymbol{M}_{\text{con}}\boldsymbol{N}_{\text{con}}}{\|\boldsymbol{M}_{\text{con}}\|\|\boldsymbol{N}_{\text{con}}\|} \tag{4}$$

基于式(2)～(4)得到的伴随图中节点权值和边权值，与匹配图中对应节点的相似程度有关，其中节点权值越大说明对应节点的相似程度越高；同样边权值反映了边之间的相似程度，边权值越小，则说明 2 个待匹配图中对应的边相似程度越低。

在得到伴随图的节点权值和边权值后，可以基于式(1)构建 $\boldsymbol{K}$。

3 基于随机游走的伴随图重要节点筛选和节点匹配

3.1 基于随机游走的节点重要度向量构建

由上述讨论可以看出，伴随图中重要节点反映了对应的待比较图中节点和边的关系。在得到 $\boldsymbol{K}$ 之后，需要根据 $\boldsymbol{K}$ 的值选择伴随图中重要节点，基于随机游走过程在伴随图中进行节点选择，最终将图匹配问题转换为找到伴随图中具有最大权重的多个节点的问题。因此本节首先基于 $\boldsymbol{K}$ 得到随机游走的状态转移矩阵的构建和优化，之后对表征节点重要程度的重要度向量进行迭代更新。

3.1.1 状态转移矩阵构建

对伴随图进行随机游走是从任意节点出发，根据图的马尔可夫转移过程随机选择下一步游走的边，马尔可夫链需要转移矩阵，因此基于得到的 $\boldsymbol{K}$，对其操作构建转移矩阵 $\boldsymbol{P}$：

$$\boldsymbol{P}=\boldsymbol{D}^{-1}\boldsymbol{K} \tag{5}$$

式中，$\boldsymbol{D}$ 是对角线矩阵，满足 $D_{ii}=d_i=\sum_j K_{ij}$。

保证转移矩阵中每行的和为 1，且每一项不小于 0。

3.1.2　状态转移矩阵优化

基于得到的 $\boldsymbol{P}$，对伴随图进行随机游走，得到稳态的马尔可夫链，对其进行状态变换后，使用一定的迭代次数得到节点重要度向量 $\boldsymbol{R}$ 的近似稳态，即不断对 $\boldsymbol{R}$ 迭代更新至其收敛，收敛后 $\boldsymbol{R}$ 中每一个元素为对应节点的重要程度。节点重要度向量生成依赖于优化后的转移矩阵。

具体来说，得到 $\boldsymbol{P}$ 后，设置系数 γ 定义节点的随机游走策略：在任意节点以概率 γ 进行下一个节点的跳转，按照概率 $1-\gamma$ 进行任意节点的跳转。本文将系数 γ 设置为 0.85，使基于转移矩阵进行随机游走的影响更大。将随机游走策略添加到转移矩阵中，得到优化后的转移矩阵为：

$$\boldsymbol{M}=\gamma\cdot\boldsymbol{P}+\frac{(1-\gamma)}{e\cdot f}\boldsymbol{E} \tag{6}$$

式中，e 和 f 分别为待匹配图 G1 和 G2 中节点的个数，它们的乘积表示构造图中的行和列的数目，$\boldsymbol{E}$ 为 $h\times h(h=e\cdot f)$ 的单位矩阵。

3.1.3　$\boldsymbol{R}$ 迭代更新

$\boldsymbol{R}$ 的维度为 h，初始值为 $\boldsymbol{R}^0=\left[\frac{1}{h},\frac{1}{h},\cdots,\frac{1}{h}\right]^{\mathrm{T}}$，基于 $\boldsymbol{M}$，对 $\boldsymbol{R}$ 进行迭代更新，并判断 $\boldsymbol{R}$ 是否收敛：

$$\boldsymbol{R}^{(t+1)}=\boldsymbol{M}\cdot\boldsymbol{R}^{(t)} \tag{7}$$

式中，$t+1$ 和 t 分别表示迭代次数，若当前迭代前后的差值小于给定的最小值 ε，则说明 $\boldsymbol{R}$ 收敛，结束该过程，即：

$$\sum_{i=1}^{n}\left|\boldsymbol{R}_i^{(t+1)}-\boldsymbol{R}_i^{(t)}\right|<\varepsilon \tag{8}$$

收敛后得到的 $\boldsymbol{R}$ 中的数据值反映了对应节点的重要度，值越大说明节点越重要。

3.2　基于匈牙利算法的节点匹配和补丁修复判断

本节主要是对 $\boldsymbol{R}$ 中的重要元素进行选择，基于选择的重要元素判断目标函数是否进行补丁修复。利用匈牙利算法对 $\boldsymbol{R}$ 进行分析，得到最大指派（即节点之间的相似程度），通过对目标函数与漏洞函数、目标函数与补丁函数得到不同伴随图的最大指派元素值进行比较，基于比较的结果判断目标函数与漏洞函数、补丁函数之间的相似程度，判断目标函数是否进行了补丁修复。

3.2.1　基于匈牙利算法的矩阵最大指派

由于 $\boldsymbol{R}$ 的维度为 h，且 $h=e\cdot f$，可以将该 $\boldsymbol{R}$ 转为一个 e 行 f 列的重要度矩阵 $\boldsymbol{R}'$，基于重要度矩阵 $\boldsymbol{R}'$ 得到对应的节点对应关系，本文使用匈牙利算法得到该重要度矩阵的最优分配值，即每一行和每一列中不重复且总和最大对应的分配关系，从而得到节点的对应关系，具体步骤为：

1）$\boldsymbol{R}'$ 每一行的最大值减去该行的所有对应值。

2）每一列的最大值减去该列的所有对应值。

3）使用横线或者竖线穿过矩阵中的所有 0，并记录达成该目的所需的最小线路总数。

4）若线路总数等于矩阵的行数 $\min(e,f)$，则一种最优的分配是可能的，完成；若总数小于 $\min(e,f)$，则执行下一步。

5）找到线路未覆盖地方的最小项，存在未覆盖项的行减去该项，覆盖项的列加上该项。

6）对得到的元素进行独立 0 元素指派，直到得到独立 0 元素的个数为 $\min(e,f)$。

得到最大指派后，其中的行列之间的对应关系就是 2 个图中节点的对应关系（$\min(e,f)$ 个对应关系），记录节点之间的对应关系，并记录对应的矩阵重要度值。

3.2.2　基于最大指派的函数补丁存在性判断

根据匈牙利算法得到的矩阵 $\boldsymbol{R}'$ 的最大指派后，可以根据节点之间的相似度关系对目标函数是否进行补丁修补进行判断。对得到的值及其表示的存在对应关系的节点进行记录，最大指派中包含 $\min(e,f)$ 个元素，对这些元素进行排序。对于目标函数、漏洞函数和补丁函数而言，将目标函数与漏洞函数得到的伴随图进行随机游走得到的最大指派的元素集合记为 X_1，按照由高到低进行排序后得到的最大指派的元素集合记为 $\mathrm{middle}(x_1)$；同样，将目标函数与补丁函数得到的最大指派的元素集合记为 X_2，对应的中位数记为 $\mathrm{middle}(x_2)$，若 $\mathrm{middle}(x_1)>\mathrm{middle}(x_2)$，则认为目标函数与漏洞函数的伴随图中的较为重要的节点数更多，即目标函数与漏洞函数的相似程度更高，固件中的函数未进行修补的可能性更大；反之，若 $\mathrm{middle}(x_1)<\mathrm{middle}(x_2)$，则认为目标函数与补丁函数的相似程度更高，即固件进行补丁修复的可能性更大。

4　实验分析

为了验证方法的有效性，本文收集了大量真实

的固件程序，涉及的厂商包括 AirGrim、AGM、Ubiquiti Networks（airrouter/airGrid）、华硕 Asus、TP-Link、AMG、领势/LINKSYS、D-link，等等。

由于固件中包含的程序不同，本文选择了固件中广泛采用的 Busybox/OpenSSL 组件，对其包含的漏洞进行补丁存在性分析。本文构建了测试固件数据集，数据集中包含了 5 925 个固件文件，通过分析发现，固件中多数包含了 libssl、busybox 以及 tcpdump 等二进制组件，因此，将 libssl、busybox 以及 tcpdump 作为实验分析的目标，对其中的漏洞函数是否进行了修复进行测试分析。

4.1 二进制程序固件的规模分析

固件中的补丁检测主要包含了伴随图构建和随机游走方法，这 2 个阶段的实现过程与伴随图中的节点数目相关，而节点的数目与待比较的 2 个函数的规模，即函数中包含的基本块数目有关，因此在讨论本文方法的效率和准确率前，首先对目标固件中函数包含的基本块数目进行分析。为了便于讨论，本文将程序规模定义为函数包含的基本块的数目。随机选择了 337 个固件文件进行规模分析，记录每一个固件文件中包含的函数对应的基本块数目，从而获取固件文件中函数的规模分布。随机选择的 337 个固件文件中共包含 140 866 个函数。其中，规模不大于 10 的函数数目为 94 250 个，占比达到 66.90%；而规模大于 100 的函数数目为 2 260 个，仅占整个函数的 1.60%，不同规模占比示意图如图 3 所示。

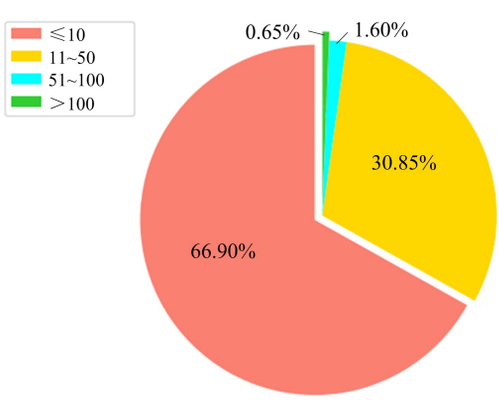


图 3 函数规模分布

Fig. 3 Distribution of function size

可以看出，固件中的函数规模分布并不均匀，规模较小的函数占的比例更高，在记录的 140 866 个函数中，规模不超过 50 的函数数目占固件中函数总数的 97.75%。分析出现该情况的原因主要有 2 个方面：一是函数规模分布本身就具有不均匀的特征，标准库函数的规模分布同样存在规模小的函数占比较高的情况。例如，本文分析了 Busybox 中的 31 692 个标准库函数，发现其中有 25 286 个函数的规模小于 20，占整个标准库函数集合的 79.8%，而规模超过 50 的函数占比则小于 6%，可以看出，标准库函数的规模分布规律与固件函数的规模分布一致；此外，固件中的函数是厂商对标准库函数进行了部分修改得到的，为了适应实际情况下固件内存有限的问题，厂商往往对标准库进行无用函数的删减，只保留固件设备顺利运行的相应功能，因此会将与固件功能无关的，特别是规模较大的函数进行相应的删减。

4.2 固件漏洞检测效果

为了验证方法的有效性，本文以构建的测试固件数据集为研究对象，判断其是否受到漏洞函数的影响。通过对数据集中固件的分析发现，其中有 4 017 个固件包含 libssl 库、474 个固件包含 tcpdump。因此本文选择了 libssl 和 tcpdump 中存在的典型漏洞函数作为分析对象，漏洞函数包括 ssl3_get_key_exchange、SSL_shutdown、ssl3_get_new_session_ticket、ssl3_get_cert_verify、icmp_print 以及 ldp_tlv_print。

在上节的讨论中，由于内存的限制，固件厂商在实际的固件开发实现过程中，往往会对标准库函数进行删减后使用，保证在内存有限的情况下能够完成固件的功能，这就导致固件中并未包含对应标准库中的所有函数。例如，在 4 017 个包含 libssl 库的固件中，包含漏洞函数 ssl3_get_key_exchange 的固件数目为 3 828 个，包含漏洞函数 ssl3_get_new_session_ticket 的固件数目为 3 614 个，而在 474 个包含 tcpdump 的固件中，包含漏洞函数 icmp_print 的数目为 31 个，上述漏洞对应包含的固件数目见表 1 所列。

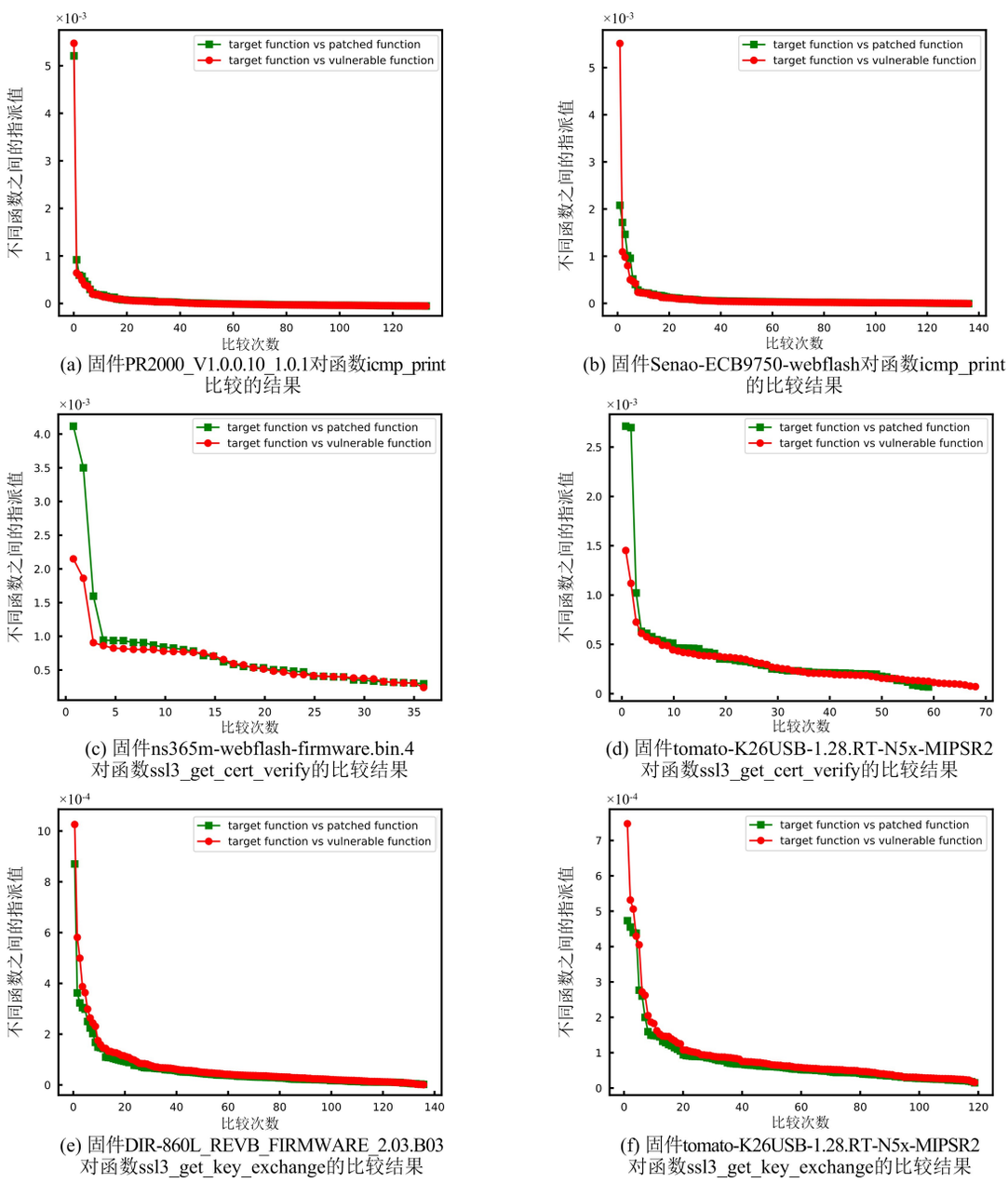
由于固件中补丁存在性判断结果与目标函数与漏洞函数、目标函数与补丁函数之间的相似程度相关，而函数之间的相似程度通过本文提出的伴随图上随机游走，并使用基于匈牙利算法的最大指派元素集合得到。若目标函数与漏洞函数得到的最大指派元素集合中的值大于目标函数与补丁函数得到的最大指派元素集合值，则说明目标函数与漏洞函数之间的相似程度更高，未进行修补的可能性更大。

表 1　包含漏洞的固件数目统计

Tab. 1　The number of firmware containing vulnerabilities

漏洞函数名称	所属组件	固件总数	包含该函数的固件数目
ssl3_get_key_exchange	libssl	4 017	3 828
SSL_shutdown	libssl	4 017	3 935
ssl3_get_new_session_ticket	libssl	4 017	3 614
ssl3_get_cert_verify	libssl	4 017	3 691
icmp_print	tcpdump	474	31
ldp_tlv_print	tcpdump	474	30

因此本文选择了 3 组固件文件：PR2000_V1.0.0.10_1.0.1 和 Senao-ECB9750-webflash、ns365m-webflash-firmware.bin.4 和 tomato-K26USB-1.28.RT-N5x-MIPSR2 以及 DIR-860L_REVB_FIRMWARE_2.03.B03 和 tomato-K26USB-1.28.RT-N5x-MIPSR2，分别对其中的函数 icmp_print、ssl3_get_cert_verify 和 ssl3_get_key_exchange 进行补丁存在性判断，记录得到的最大指派元素集合，并将指派元素集合中的元素值使用图的形式展示，如图 4 所示，其中绿色的曲线表示将目标函数与补丁函数构造伴随图得到的最大指派值，而红色曲线则表示将目标函数与漏洞函数进行伴随图构建后得到的指派值。



(a) 固件PR2000_V1.0.0.10_1.0.1对函数icmp_print比较的结果

(b) 固件Senao-ECB9750-webflash对函数icmp_print的比较结果

(c) 固件ns365m-webflash-firmware.bin.4对函数ssl3_get_cert_verify的比较结果

(d) 固件tomato-K26USB-1.28.RT-N5x-MIPSR2对函数ssl3_get_cert_verify的比较结果

(e) 固件DIR-860L_REVB_FIRMWARE_2.03.B03对函数ssl3_get_key_exchange的比较结果

(f) 固件tomato-K26USB-1.28.RT-N5x-MIPSR2对函数ssl3_get_key_exchange的比较结果

图 4　不同固件中的函数与漏洞函数和补丁函数的比较结果

Fig. 4　Comparison results of the functions in different firmware with vulnerable functions and patched functions

由图 4 可以看出，对于固件文件 PR2000_V1.0.0.10_1.0.1 和 Senao-ECB9750-webflash 包含的 tcpdump 中的函数 icmp_print 而言(见图 4(a)～(b))，2 个目标函数与漏洞函数之间的指派元素集合中的元素值要高于目标函数与补丁函数之间的指派元素集合中的元素值，因此，判断目标函数未进行补丁修复，通过实际的分析发现，目标函数的确并未进行补丁修复，说明方法对该函数的补丁修复判断是正确的。而对于固件文件 ns365m-webflash-firmware.bin.4 和 tomato-K26USB-1.28.RT-N5x-MIPSR2 而言，发现目标函数与补丁函数之间的指派元素结合的元素值高于其与漏洞函数之间的指派元素值，判断目标函数已经进行了补丁修复，通过实际分析发现，目标函数已经进行了补丁修复，与分析结果一致。

除了上面实验中选择的 3 组固件对，实验也重点关注表 1 中的 6 个漏洞函数，并对分别包含这 6 个函数的固件进行分析，对其补丁修复情况进行判断，固件使用的是上文分析得到的包含对应函数的所有固件，包含函数的固件文件数目见表 2 所列。同时，为了验证本文方法的有效性，选择使用被广泛使用的二进制比对工具 BinDiff 进行比较，BinDiff 使用了最小素数法来完成函数和基本块级别的比较，使用打分值来表示代码之间的相似程度，能够被应用于补丁存在性判断。本文将 BinDiff 分别比较目标函数和漏洞函数、目标函数和补丁函数，判断其相似性分值，如果目标函数与漏洞函数的相似性分值高于目标函数与补丁函数的相似性分值，则认为目标函数未进行漏洞修复的可能性更大。

实验使用准确率指标对补丁存在性判断结果进行比较，准确率计算的是所有预测正确的数目占总数的比重，能够反映对补丁存在性判断结果的有效性，具体正确检测的固件数目和检测的准确率见表 2 所列。

可以看出，与工具 BinDiff 相比，本文方法的补丁检测准确率更高，对于固件中的漏洞函数 ssl3_get_key_exchange、SSL_shutdown 和 ssl3_get_new_session_ticket 而言，能够实现对漏洞中补丁存在性 100％的正确判断，表现好于 BinDiff；而在其他的漏洞函数检测中，本文的方法也能够保证较高的补丁存在性检测准确率。

表 2 补丁存在性检测结果

Tab.2 Patch existence detection result

漏洞函数	包含该函数的固件数目	本文方法		BinDiff	
		正确检测的固件数目	检测准确率(％)	正确检测的固件数目	检测准确率(％)
ssl3_get_key_exchange	3 828	3 828	100	3 625	94.70
SSL_shutdown	3 935	3 935	100	3 854	97.94
ssl3_get_new_session_ticket	3 614	3 614	100	3 549	98.20
ssl3_get_cert_verify	3 691	3 688	99.92	3 612	97.86
icmp_print	31	30	96.77	25	80.65
ldp_tlv_print	30	28	93.33	27	90.00

4.3 效率分析

基于函数规模的讨论，本文选择了在规模上具有代表性的 3 个函数：ssl3_get_new_session_ticket 、ssl3_get_cert_verify 以及 SSL_shutdown 进行效率分析。函数 ssl3_get_new_session_ticket 在对应漏洞已经修复的版本中的规模为 22，未修复版本中的规模为 15；函数 ssl3_get_cert_verify 对应漏洞已经修复版本中的规模为 64，未修复版本中的规模为 65；函数 SSL_shutdown 对应漏洞已经修复的版本规模为 6，未修复版本中的规模为 4。对其进行时间消耗分析可以客观反映不同规模的函数在伴随图构建、随机游走以及基于匈牙利算法的重要节点筛选过程的效率。

本节主要针对本文提出的方法进行效率分析，讨论固件文件比较中影响效率的 3 个主要过程：伴随图构建、随机游走和基于匈牙利算法的重要节点筛选的时间消耗。这 3 个过程的时间消耗与待比较函数的规模，即函数包含的基本块节点数量相关，因此本节在进行时间消耗的讨论之前，首先讨论了实际固件文件中包含的函数规模

以及其数量分布情况。基于此，结合在规模上具有代表性的函数 SSL_shutdown、ssl3_get_new_session_ticket 以及 ssl3_get_cert_verify 进行效率分析，来判断本文方法能否在保证较高效率的同时得到较好的比较结果。

4.3.1　伴随图效率分析

伴随图的构建过程主要完成 2 个待比较图中的节点和边相似性计算和构建，而后续的重要节点选择与伴随图中节点的相似性相关，因此，本节重点关注伴随图构建中节点构建的时间消耗。在进行伴随图的节点构建时，需要计算待比较图中的每一个节点与另一个图中的每一个节点之间的相似程度。如果 2 个待比较的图中分别包含 m 和 n 个节点，则进行伴随图中节点构建的时间复杂度为 $O(m \cdot n)$。

在进行伴随图效率分析时，将函数 SSL_shutdown、ssl3_get_new_session_ticket 以及 ssl3_get_cert_verify 作为分析对象，分别构建伴随图并记录时间消耗。具体来说，对于每一个函数，获取其已经进行补丁修复的函数 P 和存在漏洞的函数 V，同时将每一个固件文件中的对应函数 cand_i 作为待测函数，对函数 P 和函数 cand_i、函数 V 和函数 cand_i 分别构建伴随图，记录伴随图构建过程耗费的时间，由于时间长度与伴随图中的节点个数相关，因此，实验同步记录了单个节点的构建时间，将单个节点构建时间计算为总时间消耗与节点个数乘积（$m \cdot n$）的比值。对于每一个分析函数而言，可以记录多个伴随图的构建时间。3 个函数的多个伴随图的构建时间、构建的伴随图数目以及单节点构建时间的平均值和中位值见表 3 所列。为了便于直观展示，使用图 5 展示 SSL_shutdown、ssl3_get_new_session_ticket 以及 ssl3_get_cert_verify 这 3 个函数的单节点构建时间的箱体图。

表 3　伴随图构建时间消耗

Tab. 3　Time cost of association graph construction

函数名称	构建的伴随图数目	伴随图构建时间平均值/s	单节点构建时间平均值/s	单节点构建时间中位值/s	伴随图中节点的平均数目
SSL_shutdown	7 870	1.84	0.056	0.057	32.1
ssl3_get_new_session_ticket	7 226	26.36	0.084	0.083	312.3
ssl3_get_cert_verify	7 382	793.54	0.200	0.220	3 797.1

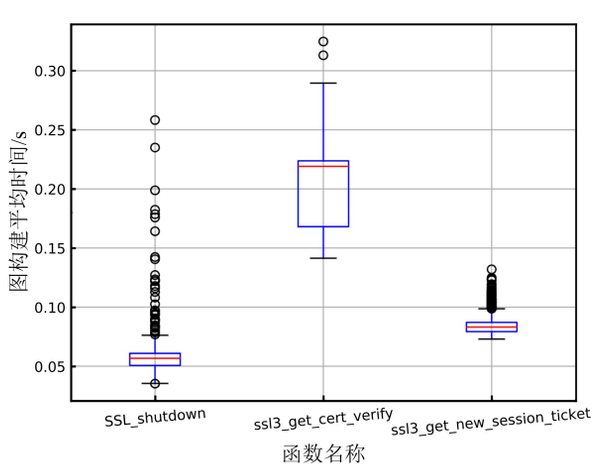


图 5　函数伴随图构建过程中单节点耗费时间

Fig. 5　Time cost of a single node in constructing association graph

通过实验结果可以看出伴随图的构建时间较长，这主要是因为伴随图的构建需要首先计算分别属于 2 个待匹配图中的节点之间，以及边之间的相似度值，需要对节点和边进行遍历计算，所以它的构建过程与节点和边的数目相关：SSL_shutdown 对应函数已经修复的版本规模为 6，未修复版本中的规模为 4，对应的伴随图构建时间相对较短；而函数 ssl3_get_new_session_ticket 和 ssl3_get_cert_verify，特别是函数 ssl3_get_cert_verify 的构建时间相对较长。通过上文的讨论可知，函数 ssl3_get_new_session_ticket 在对应漏洞已经修复的版本中的规模为 22，未修复的版本中的规模为 15；函数 ssl3_get_cert_verify 对应漏洞已经修复的版本中的规模为 64，未修复的版本中的规模 65；因此，函数 ssl3_get_cert_verify 构建伴随图时需要计算相似性的节点和边的数目要远大于 ssl3_get_new_session_ticket（规模为对应的 2 个函数中节点的数目乘积），导致函数 ssl3_get_cert_verify 的时间消耗要更大。而在上文的讨论中，函数规模大于 50 的比例较低，例如，在统计的固件函数中规模大于 50 的占比为 3.1%，因此虽然规模大于 50 的函数在构建伴随图时的时间消耗较大，但其占比很小，对整个补丁检测效

率的影响相对有限。

4.3.2 基于随机游走的伴随图重要节点筛选和节点匹配效率分析

基于随机游走的伴随图重要节点筛选和节点匹配主要包括随机游走和基于匈牙利算法的重要节点筛选 2 个主要过程。

一是随机游走时间消耗。在伴随图中的随机游走过程主要是构建节点重要度矩阵，这是基于转移矩阵完成的，其中，转移矩阵的秩为($m \cdot n$)(假定 2 个待比较的函数中分别包含 m 和 n 个基本块)，可以看出，随机游走过程的效率与待比较函数中的基本块节点个数同样相关，但是由于在随机游走过程中，不存在大量的相似度计算，因此随机游走的时间消耗要远小于伴随图构建过程。同样，使用与伴随图构建过程效率分析过程相同的目标函数，记录多个伴随图的随机游走时间，同时计算伴随图中单个节点的平均耗费时间，单个节点的耗费时间箱体图如图 6 所示，而时间消耗的平均值记录见表 4 第 4 列。

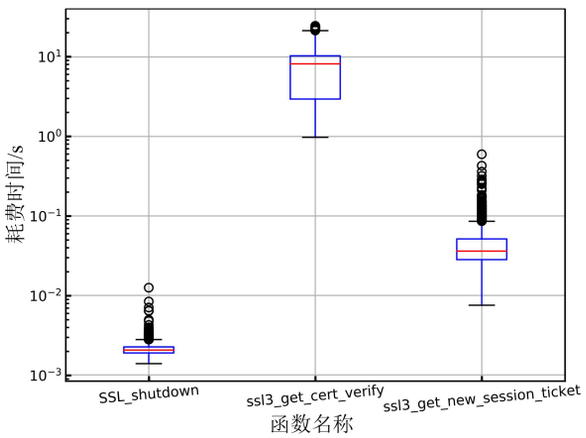


图 6 不同函数随机游走的耗费时间

Fig. 6 Random walk time cost of different functions

二是基于匈牙利算法的重要节点筛选时间消耗。基于匈牙利算法的重要节点筛选过程主要完成最大指派集的计算，记录节点筛选过程使用的时间，并同时计算单个节点的筛选过程耗费时间为节点筛选时间与节点数目的比值，单个节点的筛选时间如图 7 所示，筛选过程的平均时间记录见表 4 第 5 列。

表 4 随机游走和重要节点选择过程的时间消耗

Tab. 4 Time cost of random walk and important node selection

函数名称	构建的伴随图数目	伴随图中节点的平均数目	随机游走时间平均值/s	节点筛选平均耗费时间/s
SSL_shutdown	7 870	32.1	0.002 1	0.000 77
ssl3_get_new_session_ticket	7 226	312.3	0.042	0.006 9
ssl3_get_cert_verify	7 382	3 797.1	6.89	0.095

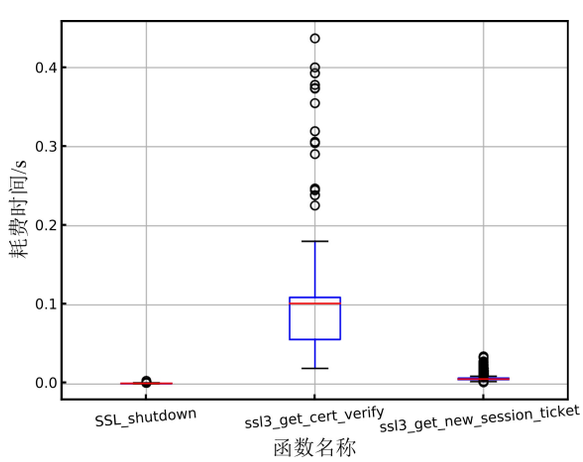


图 7 基于匈牙利算法节点筛选的耗费时间

Fig. 7 Time cost of node selection based on Hungarian algorithm

可以看出，随机游走的时间与伴随图中的节点数目成正比，且随机游走的时间消耗与伴随图构建的时间消耗相比相对要少，而节点筛选过程时间消耗比随机游走的时间消耗更少，因此，随机游走的时间消耗和基于匈牙利算法的重要节点筛选过程的时间消耗，对于整个补丁存在性判断过程的效率影响要小。

5 结束语

本文提出一种针对源码未知的二进制固件程序的补丁存在性判断方法，通过对固件中的目标函数与对应的漏洞和补丁函数进行对比分析，对目标函数是否进行补丁修复进行细粒度判断。基于属性化控制流图构建伴随图，从基本块级别的代码进行相似程度判断，并利用对图的随机游走方法实现对重要节点的筛选，判断目标函数与

漏洞函数、补丁函数之间的相似程度。最后，通过实验证明了该方法的有效性。本文提出的方法能够帮助安全人员实现对 IoT 固件文件的安全性检测，也可以为恶意软件分析提供基础支持。但是，该方法针对的是函数级别的代码分析和比较，下一步基于本文的方法，将分析范围进行扩展，实现对涉及函数间调用时存在问题的代码的分析和比较。

参 考 文 献

[1] MCMILLAN R. It's crazy what can be hacked thanks to Heartbleed[EB/OL]. (2014-04-28)[2023-10-25]. https://www.wired.com/2014/04/heartbleed-embedded/.

[2] 纪守领，王琴应，陈安莹，等. 开源软件供应链安全研究综述[J]. 软件学报，2023，34(3)：1330-1364.
JI Shouling，WANG Qinying，CHEN Anying，et al. Survey on open-source software supply chain security[J]. Journal of Software，2023，34(3)：1330-1364.(in Chinese)

[3] 于颖超，甘水滔，邱俊洋，等. 二进制代码相似度分析及在嵌入式设备固件漏洞搜索中的应用[J]. 软件学报，2022，33(11)：4137-4172.
YU Yingchao，GAN Shuitao，QIU Junyang，et al. Binary code similarity analysis and its applications on embedded device firmware vulnerability search[J]. Journal of Software，2022，33(11)：4137-4172.(in Chinese)

[4] 方磊，武泽慧，魏强. 二进制代码相似性检测技术综述[J]. 计算机科学，2021，48(5)：1-8.
FANG Lei，WU Zehui，WEI Qiang. Summary of binary code similarity detection techniques[J]. Computer Science，2021，48(5)：1-8.(in Chinese)

[5] Diaphora[EB/OL]. (2020-11-15)[2023-10-25]. https://github.com/joxeankoret/diaphora.

[6] Zynamics BinDiff[EB/OL]. (2016-03-20)[2023-10-09]. https://www.zynamics.com/software.html.

[7] Turbodiff[EB/OL]. (2020-08-01)[2023-09-15]. https://www.coresecurity.com/core-labs/open-source-tools/ turbodiff-cs.

[8] XU X J，LIU C，FENG Q，et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security.[S.l.]：ACM，2017：363-376.

[9] TUFANO M，WATSON C，BAVOTA G，et al. Deep learning similarities from different representations of source code[C]//Proceedings of the 15th International Conference on Mining Software Repositories.[S.l.]：ACM，2018：542-553.

[10] LIU B C，HUO W，ZHANG C，et al. αDiff：cross-version binary code similarity detection with DNN[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.[S.l.：s.n.]，2018：667-678.

[11] ZUO F，LI X P，YOUNG P，et al. Neural machine translation inspired binary code similarity comparison beyond function pairs[C]//Proceedings of 2019 Network and Distributed Systems Security Symposium.[S.l.：s.n.]，2019：1-15.

[12] 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 2019 IEEE Symposium on Security and Privacy.[S.l.]：IEEE，2019：472-489.

[13] TIAN D H，JIA X Q，MA R，et al. BinDeep：a deep learning approach to binary code similarity detection[J]. Expert Systems with Applications，2021，168：114348.

[14] YANG S G，CHENG L，ZENG Y C，et al. Asteria：deep learning-based AST-encoding for cross-platform binary code similarity detection[C]//Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks.[S.l.]：IEEE，2021：224-236.

[15] SUN H，CUI L，LI L，et al. VDSimilar：vulnerability detection based on code similarity of vulnerabilities and patches[J]. Computers & Security，2021，110：102417.

[16] WU Y M，ZOU D Q，DOU S H，et al. VulCNN：an image-inspired scalable vulnerability detection system[C]//Proceedings of the 44th International Conference on Software Engineering.[S.l.：s.n.]，2022：2365-2376.

[17] KIM G，HONG S，FRANZ M，et al. Improving cross-platform binary analysis using representation learning via graph alignment[C]//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis.[S.l.：s.n.]，2022：151-163.

[18] XU Y F，XU Z Z，CHEN B H，et al. Patch based vulnerability matching for binary programs[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis.[S.l.：s.n.]，2020：376-387.

[19] PENG J Q，LI F，LIU B C，et al. 1dVul：discovering 1-day vulnerabilities through binary patches[C]//Pro-

ceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]:IEEE, 2019: 605-616.

[20] XU Z Z, CHEN B H, CHANDRAMOHAN M, et al. SPAIN: security patch analysis for binaries towards understanding the pain and pills[C]//Proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering. [S. l.]:IEEE, 2017: 462-472.

[21] SUN P Y, YAN Q B, ZHOU H Y, et al. Osprey: a fast and accurate patch presence test framework for binaries[J]. Computer Communications, 2021, 173: 95-106.

[22] XIAO Y, XU Z Z, ZHANG W W, et al. VIVA: binary level vulnerability identification via partial signature[C]//Proceedings of 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. [S. l.]:IEEE, 2021: 213-224.

[23] YANG X, CHEN B H, YU C D, et al. MVP: detecting vulnerabilities using patch-enhanced vulnerability signatures[C]//Proceedings of the 29th USENIX Security Symposium. [S. l. : s. n.], 2020: 1165-1182.

[24] ZHANG H, QIAN Z Y. Precise and accurate patch presence test for binaries[C]//Proceedings of the 27th USENIX Security Symposium. [S. l. : s. n.], 2018: 887-902.

[25] WANG X D, SUN K, BATCHELLER A, et al. Detecting"0-day" vulnerability: an empirical study of secret security patch in OSS[C]//Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]: IEEE, 2019: 485-492.

[26] TAN X, ZHANG Y, MI C Y, et al. Locating the security patches for disclosed OSS vulnerabilities with vulnerability-commit correlation ranking [C]//Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l.]: ACM, 2021: 3282-3299.

[27] WANG X D, WANG S, FENG P B, et al. PatchRNN: a deep learning-based system for security patch identification[C]//Proceedings of 2021 IEEE Military Communications Conference. [S. l.]: IEEE, 2021: 595-600.

[28] WANG X D, WANG S, FENG P B, et al. PatchDB: a large-scale security patch dataset[C]//Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]: IEEE, 2021: 149-160.

[29] AHO A V, LAM M S, SETHI R, et al. Compilers: principles, techniques, and tools [M]. 2nd ed. Boston: Addison Wesley, 2006.

作 者 简 介

沈 毅

男,1985 年生,副教授,研究方向为 Web 应用安全

E-mail:shenyi@nudt.edu.cn

于 璐

女,1985 年生,博士,讲师,研究方向为软件与系统安全

E-mail:yulu@nudt.edu.cn

赵 军

男,1976 年生,高级工程师,研究方向为软件安全

E-mail:zhaojun17@nudt.edu.cn

张童童

女,1983 年生,工程师,研究方向为网络安全技术

E-mail:13309888031@189.cn

责任编辑 董 莉