

基于候选函数组的固件间函数对应关系构建方法

肖睿卿, 祝跃飞, 刘胜利, 芦斌

(数学工程与先进计算国家重点实验室, 河南 郑州 450001)

摘要: 由于固件的特点, 传统二进制比对方法在匹配函数节点传播匹配的过程中易产生误匹配。针对匹配函数传播效果不理想的问题, 设计了基于候选函数组的函数对应关系构建方法, 并引入了函数 n 层局部网络匹配的概念。然后, 结合 3 种候选函数组构造策略形成候选函数组构造方法及候选函数组匹配方法, 并分析了时间开销。最后, 基于所提方法实现了原型系统, 并与 Bindiff 进行比较。通过随机抽样和人工核对, 所提方法匹配结果的 86.04% 与 Bindiff 匹配结果一致, 所提方法匹配结果的 11.3% 可以修正 Bindiff 的匹配错误, 缓解了传播带来的误匹配问题。

关键词: 固件; 二进制比对; 函数匹配; 候选函数组

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2021018

Method for constructing function correspondence between firmware based on candidate function group

XIAO Ruiqing, ZHU Yuefei, LIU Shengli, LU Bin

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

Abstract: Due to the characteristics of firmware, traditional binary comparison methods are prone to mismatches during the propagation of the matching function. Aiming at the problem that the matching function propagation algorithm is not ideal, a method for constructing function correspondence based on candidate function groups was designed, and the concept of function matching in n layer local network is supplemented. Then, three candidate function group construction strategies and candidate function group matching methods are proposed, and the time overhead were analyzed. Finally, a prototype system was implemented based on the method and compared with Bindiff. Through random sampling and manual check, 86.04% of the matching results of the proposed method are consistent with Bindiff matching results, while 11.3% can correct Bindiff matching errors.

Keywords: firmware, binary comparison, function matching, candidate function group

收稿日期: 2020-02-20; 修回日期: 2020-06-21

通信作者: 刘胜利, dr_liushengli@163.com

基金项目: 科技委基础加强项目 (2019-JCJQ-ZD-113); 国家重点研发计划 (2019QY1300, 2016YFB0801505)

Foundation Items: Science and Technology Commission Foundation Enhancement Project (2019-JCJQ-ZD-113), The National Key R&D Program of China (2019QY1300, 2016YFB0801505)

论文引用格式: 肖睿卿, 祝跃飞, 刘胜利. 等. 基于候选函数组的固件间函数对应关系构建方法[J]. 网络与信息安全学报, 2021, 7(2): 110-125.

XIAO R Q, ZHU Y F, LIU S L, et al. Method for constructing function correspondence between firmware based on candidate function group[J]. Chinese Journal of Network and Information Security, 2021, 7(2): 110-125.

1 引言

二进制代码比对旨在确定两段代码(基本块、函数、程序)是否相似^[1], 也称作二进制比对。文献[2-5]的工作表明, 二进制比对技术具有广泛的应用场景, 可用于固件补丁/漏洞比对^[4]、软件抄袭检测^[6]、恶意软件检测^[7]等领域。

本文研究邻近版本固件间构建函数对应关系的方法, 对固件安全性进行分析, 此问题属于二进制比对范畴。文献[1]将二进制比对的粒度分为基本块、函数、程序三级; 输入形式包括一对一、一对多、多对多; 比对形式包括同构、等价、相似3种。依据此条件, 本文研究内容可以描述为, 对于输入两个固件程序, 以一定方法判别每个固件中的每一个函数在另一个固件中是否有函数与之对应(也称为匹配)。对应可以理解为, 即使是本身发生了一定幅度修改的补丁函数, 通过固件比对方法也可以在邻近版本的固件之间(补丁前后)判定补丁前后的两个函数最为相似。

从现有研究工作看, 二进制比对大体分为两个步骤。

1) 函数对相似性计算。通过各类特征判断一对函数的相似性。

2) 匹配节点传播。在各文献(如文献[4-11]等)中称为选择器、过滤器等, 主要目的是提供待计算相似性的函数对。通过不同的方法(如基于固定点传播算法)选择一组可能匹配的函数对, 而后计算相似性。目的是避免全局范围内寻找待匹配函数对, 借此减少计算量。

多数研究者对于二进制比对的研究聚焦于函数对相似性计算方法, 特别是聚焦于函数本身的信息提取以表达函数自身; 而对匹配节点的传播方法关注较少。

近年来二进制比对的研究聚焦于跨指令集比对和深度学习, 并未聚焦于匹配节点传播问题。文献[3]采用部分统计特征和结构特征表示控制流图(CFG, control flow graph)中的基本块。通过谱聚类形成质心和码本。利用质心计算编辑距离, 以代表该函数CFG。文献[4]沿用上述特征但不再使用码本表示CFG。在使用特定维度向量的基础上, 将CFG周边具有调用关系的图的向量以

一定权重整合入自身, 通过神经网络, 训练出合适的权重。其特征整合了函数所在局部子图的特征, 可以更好地用以区分函数。文献[5]利用中间语言模拟执行函数, 提取函数中分支指令的值及系统调用相关信息, 以代表函数语义。然后, 利用最大公共子序列进行匹配。文献[8]利用中间语言对基本块的输入输出对进行捕获, 判别函数语义是否一致。文献[9]使用函数调用数、逻辑指令等数字特征以及结构特征, 结合机器学习的KNN算法, 降低匹配时间代价, 完成匹配。

以结构化比对为代表的早期研究提出传播算法尝试解决函数对应关系构建问题, 但其测试用例程序规模较小, 相关实验无法暴露其在固件比对上的缺陷。文献[10]首次将CFG用于二进制比对, 之后基于该方法出现了二进制比对工具Bindiff。文献[11]在此基础上, 引入了属性的概念, 并且提出了基于固定点和传播的方法进行匹配。文献[12]对指令归一化处理后进行指令排序, 以此解决匹配过程中发现指令顺序变化的问题。文献[13]提取了汇编指令的操作码以及数据常量, 该文献仅提取CFG中大小为 k 的图或 k 图的子图, 减少了提取量。文献[14]使用Bindiff完成初步匹配, 对未匹配函数节点使用图编辑距离(GED)和Hungarian算法进行匹配。

由于固件相较于普通应用程序函数数量较大, 特别是网络设备固件, 以Bindiff为代表的传统二进制比对方法直接应用于固件易出现预期外的误匹配。其原因主要是传播算法对匹配准确性有较高的依赖, 函数匹配算法带来的错误被传播算法放大。另外, 固件函数数量大, 往往存在一个函数与多个函数相似接近, 如果基于匹配节点传播的过程选择待比较函数对较少, 则可能导致遗失真正匹配的函数节点; 如果选择过多的待比较函数对, 则会导致时间开销较大和误匹配。

为解决上述问题, 本文设计了基于候选函数组的函数间对应关系构建方法以提高函数匹配准确率, 本文工作如下。

1) 设计了基于候选函数组的函数间对应关系构建方法, 补充了适用于固件比对场景的函数匹配的定义。

2) 提出候选函数组构造方法。该方法结合了

基于统计特征、调用关系和函数地址分布的候选函数组构造策略。此外,分析了候选函数组构造带来的时间开销。

3) 提出基于候选函数组的节点匹配方法。使用低时间开销的指令统计特征提高函数对匹配的效率,利用局部网络特征提高节点对匹配准确率,并提供候选函数组节点匹配结果筛选策略。

4) 使用本文方法形成原型系统,对样本固件进行二进制比对。分析了各个候选函数组构造策略对匹配工作产生的影响,并与 Bindiff5^[23]的匹配结果进行比对。

2 模型建立

2.1 相关概念

现有研究多使用元组表示函数,认为函数匹配(matched)是元组值相等的,通过选择器选择唯一且相等的元组完成对目标函数的匹配(此处匹配采用文献[14]概念)。因此,通过量化计算函数相似性的办法确定函数匹配。即对于某个函数 f_1 ,如果存在函数 f_2 ,两个函数的相似度 $\text{Sim}(f_1, f_2)$ 大于阈值 δ ,且不存在其他函数与 f_1 相似度大于 $\text{Sim}(f_1, f_2)$,则认为 f_1, f_2 匹配。不同研究者的区别在于特征提取和相似度衡量的方法。

在固件升级的场景下,由于固件函数数量过大,基于上述概念构建函数对应关系会产生问题。例如,图1(a)指令序列与图1(b)完全一致(Bindiff认为这对函数匹配),但实际图1(a)应与结构不同的图1(c)匹配。

此例反映了两个问题:一是基于函数本身进行匹配分析有局限性;二是函数指令序列相同(基于指令提取的特征值必相同),并不代表函数匹配。推广而之,即使两个函数本身相似性最高甚至语义等价,也不能表示这对函数有对应关系,即匹配。

因此,需要补充限定条件,现引入相关概念如下。

定义 1 n 层局部网络。对于二进制文件 A 中的函数 f_1 , $f_1 \in G_A$, G_A 为文件 A 的函数调用图。若存在由函数构成的有限顶点集合 $V_{f_1_net_n}$, 集合内每个顶点 f_i (即 $\forall f_i \in V_{f_1_net_n}$) 满足以下条件

$$f_i \in G_A \quad (1)$$

$$d(f_1, f_i) \leq n \quad (2)$$

$$\forall (f_j \in G_A) \wedge d(f_1, f_j) \leq n \Rightarrow f_j \in V_{f_1_net_n} \quad (3)$$

则称 $\text{Local_Net}(n, f_1, G_A) = (V_{f_1_net_n}, E_{f_1_net_n})$ 构成的

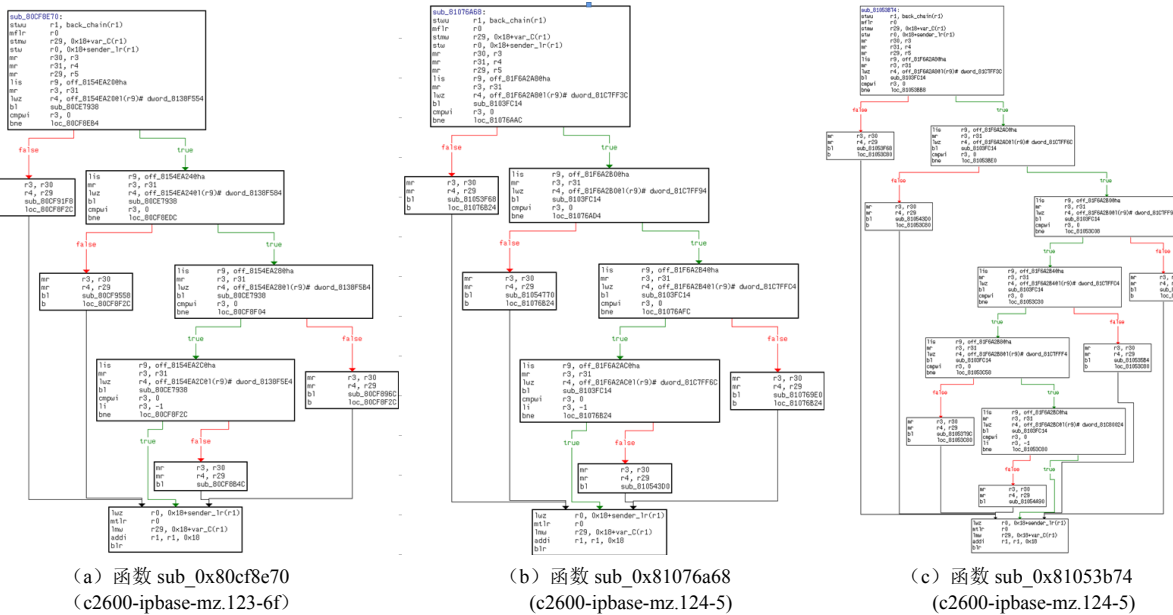


图1 两版本固件的3个函数控制流程图
Figure 1 Three CFGs of two firmwares with different version

有向图为函数 f_1 的 n 层局部网络。其中， $E_{f_1_net_n} \subseteq V_{f_1_net_n} \times V_{f_1_net_n}$ 是有向边集合，每个有向边代表一对函数节点间的调用关系；即由所有与顶点 f_1 距离小于 n 的函数节点及这些顶点间的调用关系，构成顶点 f_1 的 n 层局部网络。

定义 2 候选函数集合。对于待比较二进制文件 A, B ， G_A, G_B 为文件的函数调用图。基于 A, B 中已匹配节点集 $S_{Match} = Match_set(G_A, G_B)$ ，挑选有限顶点集合 $Ca_A = Ca_set(G_A, S_{Match}, Flag_{Strategy})$ 用以进行下一次函数相似性分析，集合内每个顶点都是未匹配函数节点，且属于文件 A 。该集合称为文件 A 当前的候选函数集合，其中， $Flag_{Strategy}$ 表示策略选择标志。

定义 3 候选函数组。对于待比较二进制文件 A, B ， G_A, G_B 为文件的函数调用图。基于 A, B 中已匹配节点集 $Match_set(G_A, G_B)$ ，挑选出的二元组 Ca_A, Ca_B 称为候选函数组。其中， Ca_A, Ca_B 分别是 A, B 挑选的候选函数集合。

定义 4 函数 n 层局部网络匹配。对于一个函数节点 f_1 ，有 $f_1 \in G_A$ ， G_A, G_B 为二进制文件 A, B 的函数调用图。若存在函数节点 $f_2, f_2 \in G_B$ ，两者在 n 层局部网络内相似度大于阈值 δ ，以 $Sim(n, f_1, f_2) > \delta$ 表示；且对于 G_B 内的任何函数节点 f_i ， $f_i \neq f_2$ ， $f_i \in V_{f_2_net_n}$ ，满足不等式 $Sim(n, f_1, f_2) - Sim(n, f_1, f_i) > \varepsilon$ ；那么对于节点 f_2, f_1 是其 n 层局部网络内的匹配点。若 f_1, f_2 互为对方的 n 层局部网络匹配点，称函数 f_1, f_2 匹配。这种匹配是 n 层局部网络匹配。由于函数调用图中函数即节点，下文也使用节点和节点匹配的说法。

其中， $Sim(n, f_1, f_2)$ 表示两节点的函数 n 层局部网络的相似度，下文使用此概念作匹配的基础，

层次 n 不作为变量讨论，因此使用 $Sim(f_1, f_2)$ 表示相似度。 ε 为正，表示相似度差异量下限。

函数匹配并不仅仅有 n 层局部网络匹配。但 n 层局部网络匹配，是现行条件下可以给出严格定义的一种情况。

2.2 匹配流程

本文采用二进制文件匹配流程如图 2 所示，算法表示如算法 1。

首先，从待比较的二进制文件中挑选函数节点构造候选函数集合，如果两个候选函数集合具有相似性，则生成候选函数组；然后，进行候选函数组匹配操作，主要针对候选函数组内的函数进行相似性分析，并从中筛选出函数匹配结果。重复上述过程，直到没有新的匹配结果产生。

算法 1 二进制文件比对算法

输入 二进制文件 A, B 的函数调用图 G_A, G_B ；
已匹配节点集合 $old_matchset$

输出 匹配节点集合 $matchset$

过程 1 二进制文件比对流程

- 1) $matchset = old_matchset$
- 2) DO{
- 3) $(Ca_A, Ca_B) = get_ca(matchset, G_A, G_B)$
- 4) //获取候选函数组
- 5) $new_matchset = math_ca(Ca_A, Ca_B)$
- 6) //获取新匹配结果集合
- 7) FOREACH $node_pair$ in $new_matchset$ DO
- 8) $matchset$ append($node_pair$)
- 9) END
- 10) } WHILE $new_matchset$ is empty
- 11) END//无新增匹配节点出现则结束比对
- 12) RETURN $matchset$

候选函数组构造：目的是从大量函数中选择部分具有关联性的函数用于后续分析，避免全局

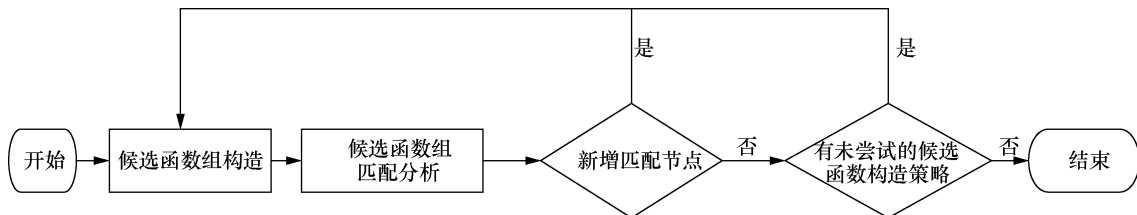


图 2 二进制文件匹配流程
Figure 2 Binary file matching process

性的比对, 借此减少工作量。此阶段工作侧重于研究匹配节点传播方法, 即利用已有的匹配结果, 挑选候选函数集合。

候选函数组匹配: 通过对候选函数组所有的函数对进行相似性分析, 计算相似度, 而后, 根据相似度筛选出函数匹配的结果。其中, 使用 n 层局部网络匹配的概念辅助函数匹配工作, 并研究匹配结果筛选策略, 用以挑选最优匹配结果。

3 候选函数组构造方法

候选函数组构造的目的是利用已有匹配结果, 确定下一步匹配工作范围。由于候选函数组是由候选函数集合构成的二元组, 对两个待比较文件采用相同的候选函数集合构造方法即可。如果两个候选函数集合具备一定的相似性 (如集合内函数数量、函数平均长度近似), 即可形成候选函数组。本节重点是从候选函数集合构建的角度进行说明, 对于集合相似性不再赘述。

3.1 候选函数组构造流程

以待比较二进制文件 A, B 为例, G_A, G_B 为文件的函数调用图。在已有匹配节点集合 $S_M = \text{Match_set}(G_A, G_B)$ 的条件下, 已知上一轮匹配集合为 S_{LM} , 策略选择标志为 Fg_i 。候选函数集合 Ca_A 构造如式(4)所示。

$$Ca_A = \begin{cases} \text{Ge}_{\text{Sta}}(G_A, G_B), S_{LM} = \emptyset, Fg_i = 0 \\ \text{Ge}_{\text{Pro}}(G_A, S_M), S_{LM} \neq \emptyset \\ \text{Ge}_{\text{Add}}(G_A, S_M, Fg_i), S_{LM} = \emptyset, Fg_i > 0 \end{cases} \quad (4)$$

其中, $\text{Ge}_{\text{Sta}}(G_A, G_B)$ 是基于统计特征构建的策略, $\text{Ge}_{\text{Pro}}(G_A, S_M)$ 表示基于调用关系构建函数集合的策略, $\text{Ge}_{\text{Add}}(G_A, S_M, Fg_i)$ 是基于地址分布的策略。

Fg_i 的值根据每次生成候选函数组的策略进行更新, 如式(5)所示, i 表示第 i 轮进行候选函数组构造:

$$Fg_{i+1} = \begin{cases} 1, Fg_i = 0 \\ 1, S_{LM} \neq \emptyset, Fg_i > 0 \\ Fg_i + 1, S_{LM} = \emptyset, Fg_i > 0 \end{cases} \quad (5)$$

其中, Fg_i 永不为负。初值为 0, 表示首次进行候选函数组构造采用基于统计特征的策略, 避

免已匹配节点集合为空。若已匹配节点集合不为空, 采用基于调用关系的策略。在基于调用关系的策略无法提供新的匹配点时, 使用基于地址分布的策略。此外, 候选函数组构造方法并非仅仅生成一个候选函数组, 而是生成由若干候选函数组构成的候选函数组集合。

单轮次的候选函数组构建流程如图 3 所示, 使用符号如前文所述。其中, G_A, G_B 表示二进制文件的 FCG 图, S_M 表示匹配节点集合, S_{LM} 表示上一轮匹配的匹配节点集合, Fg 表示策略选择标识 (Fg 不为负), Ca_{gp} 表示构造的候选函数组集合。

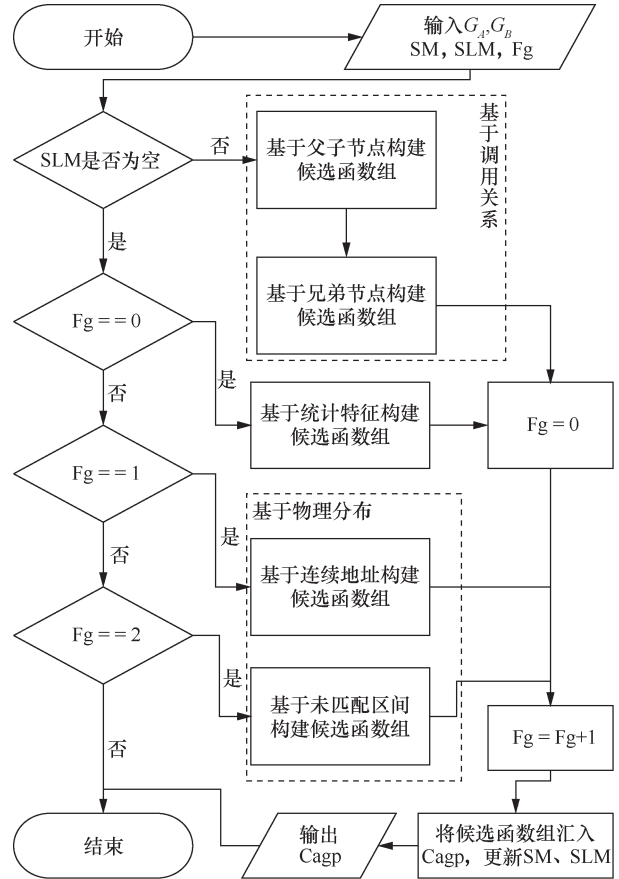


图 3 单轮次的候选函数组构建流程

Figure 3 Single round of candidate function group construction process

3.2 基于统计特征构建候选函数组

候选函数组的构造基于已函数匹配节点。若没有函数匹配节点, 应当使用基于统计特征构建候选函数组的策略, 筛选特征特征值相同或相似的节点以构建候选函数组。

本文选择的特征为函数长度、基本块数、子

节点数、被调用次数, 并依照上述次序对函数进行排序, 选取特征值相同的函数构建候选函数组。以上统计特征, 特征值越大, 同一文件内出现函数具有相似特征的可能性越小。

统计特征本身不像其他两种方法具备传播能力, 即从已匹配节点关联其他节点能力, 因此不依赖已匹配节点也可工作。统计特征是对函数部分语义信息的提取, 相较于全函数匹配, 运算量更小、筛选速度快。在筛选过程中, 应当尽量避免同文件内特征相同的函数加入候选组, 这会对后续分析带来干扰。

当基于调用关系和地址分布不产生新的匹配节点时, 可考虑再次使用基于统计特征来构建候选函数组。但匹配后期的剩余函数以短小函数为主, 将存在大量同文件内函数相似的情况, 实际效果可能不够理想。

本文基于统计特征构建候选函数组主要是为了替代人工提供初期的匹配节点, 给后续的传播类算法提供基础, 因此并不追求利用统计特征解决全部匹配问题。使用统计特征筛选初始匹配节点的过程中, 候选函数集合包含函数节点的数量以及筛选函数节点的长度下限是需要讨论的。

对基于特征筛选的候选函数集合, 其集合内节点数量最大值(后文以 K 表示)的选择, 将影响初始匹配节点的生成量, 进而对后续分析的耗时产生影响。实验表明, 基于统计特征策略构建候选函数组, 其数量限定 K 为 100 时实验效果最好。

对于函数长度下限的设定, 则将影响函数匹配的准确率。在代码中修改同样数量的寄存器、操作码, 对于长函数而言, 修改量所占百分比低于短函数。这意味着同等数量的修改, 对长函数而言影响更小。因此本文基于统计特征的匹配过程中, 特征值越大越可靠。

此外, 实验使用候选函数长度下限为 50 条指令。固件函数数量较大, 即使构建候选函数组的过程中达到候选函数数量限制, 所选函数的长度远大于长度限制, 不影响后续分析, 因此本文对候选函数长度不做实验进行分析。

3.3 基于调用关系构建候选函数组

3.3.1 基于父子节点的构建候选函数组

以待比较二进制文件 A, B 为例, G_A, G_B 为文

件的 FCG。基于父节点已匹配构建候选函数组的步骤如下。

1) 提取上轮匹配函数集合 S_{LM} 的所有匹配结果。

2) 对于一组函数匹配结果 (f_1, f_2) , 查找两个函数的所有子节点。

3) 将 f_1 的子节点汇总形成候选函数集合 Ca_A 。

4) 将 f_2 的子节点汇总形成候选函数集合 Ca_B 。

5) 如果两个集合具有一定相似度, 形成一个候选函数组 $\langle Ca_A, Ca_B \rangle$ 。

6) 重复步骤 2)~步骤 5), 将生成的候选函数组汇总, 并输出。

基于子节点已匹配的候选函数组构建策略与上述流程类似, 区别仅在于, 候选函数集合是上轮匹配节点的子节点构成的集合。

文献[11]提出针对已匹配函数节点的子节点进行相似性分析的方法。实际上, 这是较为可靠的分析方法, 这种传播方法将节点周边局部网络的信息间接融入分析流程; 若子函数语义一致, 计算子节点相似性的过程中可以不再进行局部网络的相似性分析。

并没有文献提出利用已经匹配函数节点分析其父节点的思路。此法同样可行, 但分析难度较前者略大。因为匹配函数的子节点易做到一一对应, 便于候选函数组构造分析。但函数父节点常难以找到对应关系, 在构造候选函数组时易包含较多不匹配节点, 后续的候选函数组匹配工作负担相对大。但匹配函数节点的父函数集合, 仍是一种降低匹配量的思路, 可以用作构造候选函数组。

实验结果表明, 基于已匹配节点的父节点构造候选函数组, 最终贡献的匹配节点量远大于利用已匹配节点子节点的策略贡献的匹配节点量。

3.3.2 基于兄弟节点构建候选函数组

若两个节点具有相同的父节点或者相同的子节点, 则称为兄弟节点。本策略利用上轮匹配节点集合 S_{LM} , 首先查找 S_{LM} 中所有节点的父子节点哪些已经匹配成功; 然后针对这些匹配的父子节点使用基于父子节点的候选函数组构建策略, 再次构建候选函数组; 最终将构建的候选函数组形成候选函数组集合并输出该集合, 如算法 2 所示。

算法 2 基于兄弟节点构建候选函数的算法

输入 二进制文件 A, B 的 FCG 图 G_A, G_B ; 上轮匹配节点集合 S_{LM}

输出 候选函数组集合 new_ca

```

1)  FOREACH ( $f_1, f_2$ ) in  $S_{LM}$  DO
2)      // 提取所有已匹配节点对
3)   $Son\_set = get\_son\_matched(f_1, f_2, G_A, G_B)$ 
4)      // 获取匹配节点对的已匹配子节点
5)  FOREACH ( $fs1, fs2$ ) in  $Son\_set$  DO
6)       $(Ca_A, Ca_B) = get\_pa(fs1, fs2, G_A, G_B)$ 
7)  // 获取匹配子节点的父节点集合
8)   $new\_ca.append(<Ca_A, Ca_B>)$ 
9)      // 录入候选函数组集合
10) END
11)  $Pa\_set = get\_pa\_matched(f_1, f_2)$ 
12) // 获取匹配节点对的父节点
13) FOREACH ( $fp1, fp2$ ) in  $Pa\_set$  DO
14)      $(Ca_A, Ca_B) = get\_son(fp1, fp2, G_A, G_B)$ 
15) // 获取匹配父节点的子节点集合
16)  $new\_ca.append(<Ca_A, Ca_B>)$ 
17) // 录入候选函数组集合
18) END
19) END
20) RETURN  $new\_ca$ 

```

此策略的设计基于如下考虑。理论上, 只要反复判别已匹配节点的父子节点集合是否会出现新的匹配点, 就可以完成基于调用关系的匹配工作。但实际上, 由于匹配工作是基于候选函数集合进行的, 如果候选函数集合中存在无法区分的相似节点(同文件内的相似函数), 则策略上会放弃匹配这种节点, 以避免产生误匹配或由于相似度过于接近以至无法选择匹配结果的问题。此时, 若后期成功匹配了一个早期被放弃的相似节点, 那么同文件内相似节点应当重新尝试寻找匹配点, 因为早期放弃匹配的节点已经成功匹配, 不再影响其分析。反复对早期节点进行重复的调用关系分析, 就是在处理这类问题。

已匹配节点数量是随着每一轮分析而递增的, 如果每次产生新增匹配节点时都对前几轮的匹配节点进行再分析, 会带来大量的时耗。实际上, 后期匹配节点对早期不匹配节点的影响, 主

要发生在兄弟节点上。例如, 某个节点的若干父节点相似性较高, 被放弃分析; 如果后续分析过程中, 其中一个被放弃的父节点已经完成匹配, 则可以尝试重新分析这些被放弃的父节点。这样, 不再需要每次对全部的早期匹配节点进行重复分析, 只需要根据新增匹配节点查找兄弟节点即可, 减少了运算量。

实验表明, 在所有基于调用关系匹配贡献的匹配节点中, 基于兄弟节点的策略贡献了 50% 以上的匹配量, 充分说明对早期匹配节点进行重匹配的必要性, 也说明此策略降低时耗的必要性。

3.4 基于地址分布构建候选函数组

本节主要介绍基于地址分布的构建候选函数组策略, 包括基于连续地址和基于未匹配区间两个角度。目的是突破基于调用关系构建候选函数组的局限。

3.4.1 基于连续地址构建候选函数组

基于连续地址构建候选函数组的策略, 主要筛选与已匹配函数节点在地址上邻接的函数节点, 检查其是否已经匹配, 如未匹配, 则加入候选函数组。连续地址可以突破调用关系的传播壁垒, 此策略对于以短小函数为代表的低特征函数具有较好的匹配效果。

基于调用关系构建候选函数组的方法本身属于传播算法。实际上, 基于已有匹配结果进行扩展分析的方法, 应当视为传播算法, 本质上是为了利用已匹配节点筛选候选函数组, 缩小后续匹配工作的匹配范围。理想状态下的二进制文件, 如果其 FCG 图以一棵树的形态存在, 使用前文策略即可完成匹配。即任意两个函数节点之间存在一条路径可以从一个节点经过若干节点后到达另一个节点。然而, 现实中更为可能的情况是, 二进制文件的 FCG 是以森林状态存在的, 此种情况下会出现传播壁垒, 以图 4 为例。

假设存在文件 A, B 是需要进行匹配的具有相似性的二进制文件, 已知函数 f_2 和 f_2' 匹配。此时如果使用基于调用关系的策略, 无法自动化的完成函数 f_4 和 f_4' 匹配, 此为传播壁垒。基于连续地址的策略可缓解此类问题。

这种方法的合理性在于, 如果没有特殊要求, 编译器在编译过程中会将同一文件内的函数连续

编译,这些函数在物理上连续存在是大概率事件。逻辑上讲,程序中某段特定功能相关的代码一般存在于一段邻近空间,这也符合局部性原理和开发规律。此外,这种基于连续地址进行分析的策略,也可以看作对最大公共序列问题的延伸。连续匹配的函数越多,连续匹配的指令就越多,匹配结果就更为可靠。

实验表明,使用本策略后,新增匹配节点占总匹配函数数量的38%。

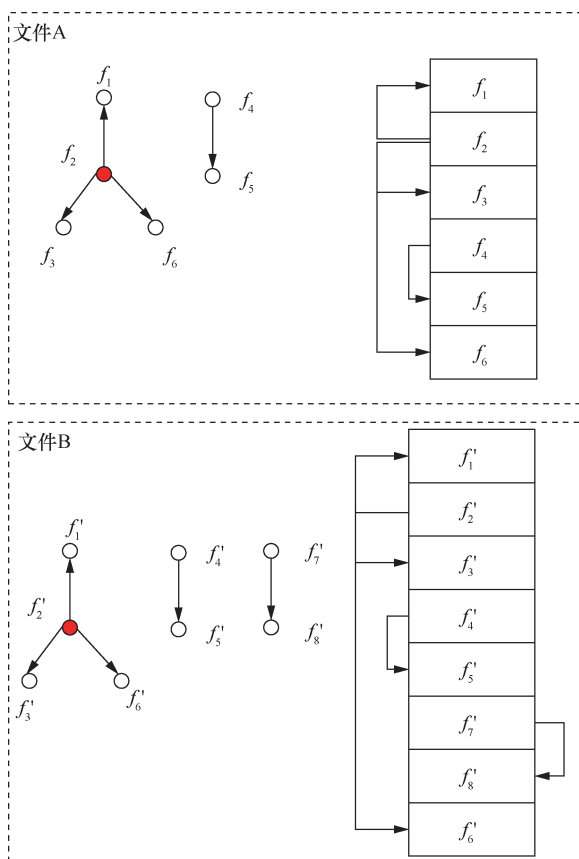


图4 调用关系隔离的匹配示例
Figure 4 Matching example of call relation isolation

3.4.2 基于未匹配区间构建候选函数组

在匹配分析的后期,整个二进制文件的未匹配函数节点被已匹配节点分割成若干段。此时,从局部性原理考虑,对于两段连续未匹配函数区间,如果其边界的匹配节点是互相匹配的,则直接将两段函数构造成候选函数组,用以后续分析。

此策略是为了缓解基于连续地址的传播壁垒,示例如图5所示。对于文件A、B,已经完成函数 f_1, f_5, f_7 的匹配工作,人工分析条件下已知

f_3, f_3' 是完全一致的函数;假设此时 f_2, f_4 函数存在一定差异,无法自动化地完成匹配工作,则无法使用基于连续地址构建候选函数组的方法,利用已匹配节点完成 f_3 的自动化匹配。

此情况产生的原因可能是多样的。如编译器在编译过程中,不同文件在链接的过程中可能插入未知数据;文件编译时,文件内的小函数往往分布在编译结果的边缘;将编辑结果链接成可执行文件的过程中,排序可能有变化。虽然有局限性,此策略依然可以用作突破调用关系的传播壁垒。由于传播过程没有融入语义信息,一旦产生新的匹配节点,仍优先采用更为可靠的基于调用关系的策略。

对于本节的策略,仍以图5为例,在已知函数 f_1, f_5 完成匹配的情况下,即使 f_2, f_3, f_4 这3个连续函数构成的连续区间没有函数匹配,其与 f_2', f_3', f_4' 这3个函数也存在匹配的可能。将 f_2, f_3, f_4 与 f_2', f_3', f_4' 构造候选函数组,用于后续分析。

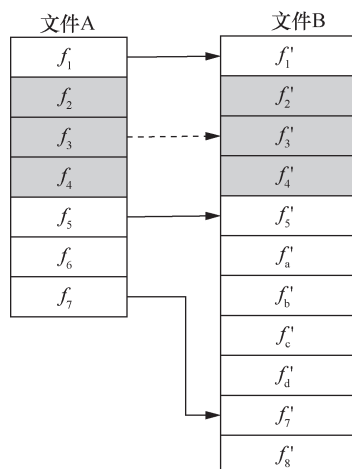


图5 匹配结果示例
Figure 5 Examples of matching results

在实际操作过程中,可以考虑利用函数序列关系、剔除函数特征相似节点、优先分析邻近已匹配点函数等策略加以优化。

实验表明,使用基于未匹配区间构建候选函数组策略后,新增匹配节点量占总匹配量的5%。

3.5 时间开销分析

候选函数组构造过程本身不涉及函数节点的比较,主要是基于已有匹配结果查找具有

关联性的未匹配节点,并汇入候选函数组。在已经构建 FCG 的条件下,这种查找过程的时间开销是常量级的。因此,构造过程本身的时耗可以忽略。

但函数节点相似性分析是具有一定时间开销的,候选函数组的构建结果直接影响每个候选函数组需要进行的节点相似性分析计算次数。因此,本节主要分析候选函数组构造方法对节点匹配过程带来的影响,进而分析对整个固件匹配分析流程带来的时耗影响。

假设 1 有两个函数数量一致的二进制文件 A,B,其函数数量都是 n ,以表示其规模相同。

对于假设 1,理想状态下,如果整个分析流程构建了 n 个候选函数组,每个候选函数组内候选只有一对函数节点,且组内节点一次分析即可成功匹配,那么对于整个固件分析流程,共需进行 n 次分析,时间复杂度 $O(n)$ 。由此可知,整个分析流程的节点分析次数在区间 $[n,n^2]$ 波动,时间复杂度为 $O(n) \sim O(n^2)$ 。

3.5.1 等尺寸候选函数集合的时间开销分析

在假设 1 的条件下,补充假设 2。

假设 2 假设 $n=ab$, $a>b$, $a \in \mathbb{N}$, $b \in \mathbb{N}$, $b>0$ 。

为便于分析,先不考虑每次构建候选函数集合大小不一的情况,假设每次构建的函数集合包含相同数量的函数节点,数量是 κ 个, κ 是远远小于 n 的常量,直到未匹配节点数量少于 κ 时,构建候选函数集合的节点数量为当前未匹配节点量。

理想条件下,每个函数仅一次用于构建候选函数组,构建即保证匹配。这种条件下,如果候选集合内节点数量满足 $\kappa=a$,那么会构建 b 个候选函数组,则整个流程需要进行 aab 次节点比较,即 an 。因此,候选函数组越大,比较次数越多,时耗越大。

除了初始阶段,本文候选函数组的构建是需要依赖匹配节点的。因此,每次候选函数组至少应当有一个新匹配节点产生,才能保证生成用于分析的新的候选函数组。

由此引入关于匹配成功率的假设,对于具有 κ 个节点的候选函数集合,每次节点相似性分析

集合内的每个函数可以有 p 的概率成功匹配,且 $0<p \leq 1$ 。则每个候选函数组可以匹配 $\lfloor \kappa p \rfloor$ 个节点。

如果整个文件所有函数节点都可以匹配(此时开销最大),要完成匹配,至少需要构建 $\frac{n}{\lfloor \kappa p \rfloor}$ 个

候选函数组(不考虑剩余未匹配节点数量小于 κ 的情况)。整个分析流程需要进行的节点匹配次数为 $\frac{\kappa^2 n}{\lfloor \kappa p \rfloor}$,如果所有相似性计算结果保存,其计算

上限仍为 $O(n^2)$ 。由此可知,匹配的时间开销受候选函数集合大小的影响,仍然为 $O(n) \sim O(n^2)$,但只要候选函数组构建的足够小,时耗会趋于 $O(n)$ 。

3.5.2 不等尺寸候选函数集合节点的时间开销分析

在假设 1 的条件下,补充假设 3。

假设 3 对于所有候选函数组内的节点,都有 p 的概率成功完成匹配, $0<p \leq 1$, 假设

$n = \sum_{i=1}^t a_i p$, 其中, a_i 是每个候选函数组的尺寸。

那么需要进行的匹配次数为 $\sum_{i=1}^t a_i^2$, 如果有

$a_i = \frac{n}{tp} + b_i$, 且 $\sum_{i=1}^t b_i = 0$, b_i 表示每个候选函数组

与平均尺寸的差值,则 $\sum_{i=1}^t a_i^2 = t \left(\frac{n}{tp} \right)^2 + \sum_{i=1}^t b_i^2$ 。由

此可知,不等尺寸的候选函数集合时耗大于等尺寸的候选函数集合。同理可知,差值均匀的情况比差值不均的情况时耗小。

3.5.3 节点相似性重分析对时间开销的影响

除上述因素外,需要对节点重分析带来的时耗进行讨论。部分早期不匹配节点,随着匹配过程的推进,可以完成匹配。因此,重新构建候选函数组进行分析,以图 6 为例。

现有函数节点 $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8$, 其调用关系如图 6 所示。由于 f_1 与 f_2, f_3, f_4 具有较高相似性,早期基于 f_6 节点构建候选函数组时, f_1, f_2, f_3, f_4 没有成功匹配,但后期基于 f_5 节点完成了 f_1 节点的匹配,此时,可以基于 f_6 节点重新构建候选函数组进行匹配分析。

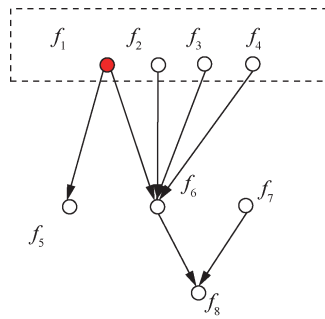


图6 重匹配示例
Figure 6 Rematch example

但实际情况中, f_6 节点可能由于 f_7 、 f_8 节点的匹配被关联到, 引发重匹配。而只要 f_1 节点的匹配没有完成, 重新构建候选函数组是无法解决 f_1 节点带来的匹配失败情况的。此时, 整个重匹配工作不会产生新增匹配节点。因此, 应当避免每个匹配节点的产生引发候选函数组重建和重分析的情况。

对每一轮匹配产生的匹配节点关联到所有节点汇总后(如基于 f_5 、 f_7 确定 f_6), 再分析可以有效降低此类时耗。汇总公共节点需要进行排序和查找以确定是否有重复节点, 引入了新的时耗。

公共节点汇总工作主要受两个因素影响: 首先是单轮匹配节点数量带来的影响, 匹配节点数量越多, 关联到的节点越多, 排序时耗越大; 其次, 匹配轮次的影响, 匹配轮次越多, 排序查找次数越多, 时耗越大。

而这些因素都受初始匹配节点量的影响, 即对基于特征筛选候选函数数量的最大值 K 的选择。由于匹配是基于传播规律的, 对于固定大小的文件, K 越大, 整个匹配轮次降低。但初始匹配节点的数量, 会影响到单轮匹配节点数量的大小, K 越大, 单轮匹配节点量越大, 查找排序时耗越大。因此, K 值的选择具有重要价值。

4 候选函数组匹配分析方法

本节对候选函数组内的节点匹配方法进行阐述, 具体流程如图7所示。其中, 输入是候选函数组 Ca_A, Ca_B , Ca_A, Ca_B 分别是二进制文件 A, B 某个轮次构建的候选函数集合, 图中为 Ca_A, Ca_B 。输出为匹配节点集合 S_{Match} , 图中为 SM 。

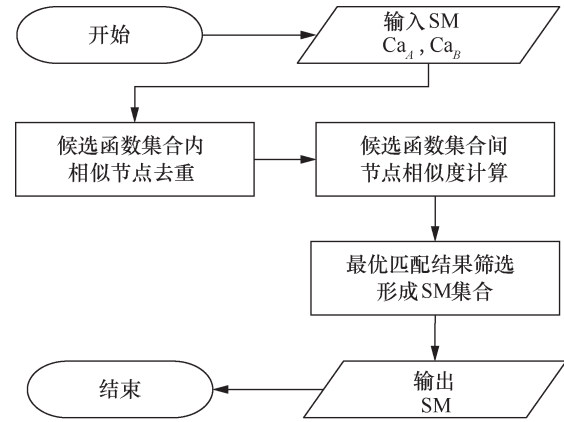


图7 节点匹配流程
Figure 7 Node matching process

其主要思路如下。

1) 候选函数集合内相似节点去重。分别对两个候选函数集合 Ca_A, Ca_B 各自内部相似度较高的函数节点进行去重工作; 以 Ca_A 为例, $\forall f_i \in Ca_A$, $\forall f_j \in Ca_A$, 若 $\text{Sim}(f_i, f_j) > \varepsilon$, 则 Ca_A 中去掉 f_i 和 f_j 。 Ca_B 照此法处理。

2) 候选函数集合间节点相似度计算。即 $\forall f_a \in Ca_A$, $\forall f_b \in Ca_B$, 计算节点对 (f_a, f_b) 的相似度 $\text{Sim}(f_a, f_b)$ 。

3) 最优匹配结果筛选。根据所有节点对的相似度计算结果, 首先基于函数 m 层局部网络匹配的概念完成部分函数节点匹配工作。然后, 对于多函数对相似度接近等不同情况, 从候选函数集合的角度, 挑选出整个集合的最优匹配结果的组合。

4.1 节点相似度计算方法

对于一对函数节点相似度的计算应当包含两部分, 函数自身特征的相似性和函数所在局部网络的相似性。因此, 节点相似性计算如式(6)所示。

$$\text{Sim}(f_1, f_2) = W_{\text{self}} P_{\text{fsim}} + W_{\text{net}} P_{\text{nsim}} \quad (6)$$

W_{self} 表示自身特征的权重, W_{net} 表示局部网络特征的权重, 满足 $W_{\text{self}} + W_{\text{net}} = 1$, 且权重值都是非负数。 P_{fsim} 和 P_{nsim} 分别表示基于自身特征的相似度和基于局部网络特征的相似度, 两者皆是小于等于1的非负数。权重的取值可以自行调整, 在匹配过程的前期和后期采用不同的权重分配可能具有更好的效果。

4.1.1 基于函数自身特征的相似度计算方法

基于函数自身特征的相似性分析, 本文直接使用函数中汇编指令序列进行比对。如果函数指令序列相同, 则视为函数自身特征相同。理论上, 依靠 LCS 算法, 可以直接计算两个函数之间指令序列的最大公共子序列, 以此来完成相似度计算。在实际处理上略有差异。

在早期匹配过程中, 为了减少对候选函数组构造过程中传播算法的影响, 本文在计算最大公共子序列时, 逐个基本块利用 LCS 计算公共指令序列长度, 累加公共长度求和后, 除以较大的函数长度, 得到函数相似度 P_{sim} , 具体如式(7)所示。

$$P_{\text{sim}} = \frac{\sum_{i=0}^{b_count(f_1)} \text{lcslen}(\text{bk}(f_1, i), \text{bk}(f_2, i))}{\text{Max}(\text{len}(f_1), \text{len}(f_2))} \quad (7)$$

其中, f_1, f_2 为待比较节点, $b_count(f_1)$ 表示函数的基本块数量; $\text{len}(f_1)$ 表示函数指令数量, 即函数长度; $\text{lcslen}(\text{bk}(f_1, i), \text{bk}(f_2, i))$ 表示两个函数第 i 个基本块的指令序列最大公共子序列长度。

4.1.2 基于局部网络特征的相似度计算方法

局部网络特征的相似度 P_{nsim} 的计算方法, 如式(8)所示。

$$P_{\text{nsim}} = 1 - \frac{d(\text{net_fea}(f_1), \text{net_fea}(f_2))}{\text{Max}(|\text{net_fea}(f_1)|, |\text{net_fea}(f_2)|)} \quad (8)$$

其中, f_1, f_2 为待比较节点; $\text{net_fea}(f_1)$ 将计算出一个 m 维的特征向量, 用以表示函数所在的局部网络; $d(\text{net_fea}(f_1), \text{net_fea}(f_2))$ 表示两个向量的欧式距离, $|\text{net_fea}(f_1)|$ 是特征向量的模, Max 表示取两者中最大的值。在实际分析过程中, 特征向量 net_fea 的构建, 可以选择块数、子节点数等简单、易于统计的特征来降低计算开销。

4.2 匹配节点筛选策略

匹配节点筛选工作主要是利用前文计算出各个节点对相似度, 在候选函数组内挑选出匹配节点, 形成匹配节点集合。其中, 要解决单个函数与多个函数相似度较高等问题。

4.2.1 无序列关系的候选函数组匹配方法

构建候选函数组时, 候选函数集合内的函数节点并没有直接顺序关系。在完成各个节点相似

度计算后, 匹配筛选流程如下。

1) 确定满足 n 层局部网络匹配的节点对。相似度阈值 δ 的大小影响到候选函数组成功匹配节点数量。实验表明, δ 取 0.8 时效果最好。

2) 孤立不匹配节点对剔除。对于节点对 (f_1, f_2) , 其相似度小于 δ , 若不存在节点对 (f_1, f_j) , 两者节点对相似度接近 (参考 ε , 定义 4), 则将 (f_1, f_2) 相似度归 0。若 (f_1, f_2) 与 (f_1, f_j) 相似度相同, 则此轮剔除节点 f_1 , 不对其进行分析。

3) 多相似节点匹配结果筛选。在完成前两步工作后, 剩余的相似度大于 0 的节点多数含有多个匹配节点。参考解决皇后问题的思想, 解决剩余子节点匹配问题。

4.2.2 有序候选函数组匹配方法

候选组生成过程中, 候选组内函数会存在具有序列关系的情况, 如连续地址区间、连续调用序列等。有序候选组相较于无序候选组实际上仅多了一个可以用于辅助分析的有序条件。在此, 不再叙述候选结果筛选流程, 仅讨论如何在无序筛选的基础上加以改进。

一个较为简单的方法是将 LCS 算法融入筛选过程。由于 LCS 中各个节点仅存在匹配与不匹配情况, 而函数比对的相似性是一段连续的区间, 相似度有高低区分。需要对算法进行调整, 其流程如下。

1) 不考虑区分度 ε , 将相似度大于 δ 的节点对相似度置 1, 其余节点对相似度置 0。

2) 利用 LCS 算法计算匹配结果。

3) 将步骤结果与 4.2.1 节结果进行比对, 修正步骤 2) 的误匹配, 以此避免由于 LCS 算法导致的连续误匹配问题。

5 实验与分析

本章对实验进行说明, 实验主要包括: 初始匹配节点量 K (即基于特征筛选函数的数量限制) 与相似度阈值 δ 对匹配流程的影响; 最优参数选择; 最优参数条件下, 各种构建函数策略对匹配流程的贡献; 与 Bindiff5 匹配结果比对, 本文各策略分析结果的差异量; 各策略分析差异结果抽样验证; 跨固件系列条件下的各策略分析差异结果抽样验证。

5.1 实验环境

本文实验环境为 Win10 x64 操作系统, IntelXeonGold6150CPU@2.70 GHz 处理器, 128 GB 内存, SamsungT5SSD2TB 硬盘。使用软件 IDApro^[14] 进行辅助分析。实现语言是 python2.7x64。实验基于收集的 Cisco^[22]C2600 系列(约 660 个)和 C800 系列固件进行, 收集方法如文献[24]。从 C2600 系列固件中挑选两组固件, 每组随机抽取 10 个, 标号分别是 A1-A10 和 B1-B10, 两组之间的固件两两比对以进行相似性分析, 共计 100 对比对样例; 从 C800 系列固件中选择 4 个版本号、编译次数、功能号相近的固件, 标号是 C1-C4。所挑选的固件列表如表 1 所示。

C 组固件的函数数量波动小于 10%, 主要用于参考和跨系列比对验证; 而 A、B 两组固件的函数数量波动达到了 90%, 且选择的固件源自不

同的功能号, 足以验证本文方法在多种场景下的分析效果。

5.2 预处理

固件比对分析前, 需要对固件进行解析。主要依靠 IDApro 进行反汇编, 并提取函数比对需要的基本信息。本文对收集的 600 余个固件进行了预处理, 包括为 Bindiff 比对提前生成 Binexport 文件, 共耗时 23 h。

5.3 参数选择

针对初始匹配节点量 K (即基于特征筛选函数的数量限制) 与相似度阈值 δ 的选择, 使用 A、B 两组样本进行比对实验。本文对 K 值为 1、10、50、100、500 共计 5 个选择。 δ 为 70%、80%、90%、100% 共 4 个选择, 进行实验。由 3.5 节中分析可知, 两个参数都会对实验结果产生影响。

因此, 两个参数联合进行实验, 共计进行 20 组

表 1 待比较文件列表
Table 1 List of files to be compared

文件标号	文件原名	函数数量
A1	c2600-advipservicesk9-mz.124-15.t4	187 365
A2	c2600-advsecurityk9-mz.123-14.t7	85 864
A3	c2600-ds-mz.121-5.t7	57 947
A4	c2600-entbase-mz.123-11.t	79 355
A5	c2600-ik9s-mz.122-26	61 916
A6	c2600-io3-mz.121-7	26 619
A7	c2600-ipbase-mz.123-6f	53 877
A8	c2600-is-mz.122-2.xt	68 918
A9	c2600-ix-mz.122-11.t11	41 452
A10	c2600-jk9o3s-mz.122-28	73 856
B1	c2600-advipservicesk9-mz.123-11.t8	125 416
B2	c2600-advipservicesk9-mz.123-8.t	125 781
B3	c2600-advipservicesk9-mz.124-8	140 723
B4	c2600-advsecurityk9-mz.124-23	873 57
B5	c2600-i-mz.123-18	46 303
B6	c2600-ipbase-mz.124-5	66 091
B7	c2600-is56i-mz.121-14	46 830
B8	c2600-ix-mz.123-11.t	59 583
B9	c2600-j1s3-mz.123-24	91 589
B10	c2600-jsx-mz.123-11.t	120 031
C1	c800-universalk9-mz.SPA.154-3.M10	388 468
C2	c800-universalk9-mz.SPA.155-3.M5	420 071
C3	c800-universalk9-mz.SPA.156-3.M4	435 324
C4	c800-universalk9-mz.SPA.157-3.M	434 759

实验, 每组都使用文件 A1-A10 和 B1-B10 进行两两比对, 即每组 100 对对比样例。每组比对后, 累加所有匹配节点数量和时耗秒数, 形成表 2。

实验结果表明, 当 K 取 100、 δ 取 80% 时, 效果最佳。之后实验在此参数下进行。从单位时间函数匹配数量的角度而言, 两个参数不会对匹配效率带来单调增减的影响, 即 K 和 δ 两者之一的增减, 并不会使匹配效率单调增减。

5.4 候选函数组构造策略匹配效果

在 K 取 100、 δ 取 80% 的条件下, 使用 A、B 两组样本进行比对实验。统计各候选函数组构造策略 (下文简称策略) 贡献的匹配量和时耗如表 3 所示。

基于上述结果可以确定, 调用关系贡献了最多的节点匹配数量, 约占匹配总量的 60%; 基于连续地址的策略是单位时间内贡献匹配量最多的策略, 影响了匹配总量的 38%。相当于提升了前者效果的 50%; 基于未匹配区间的匹配策略同样发挥了自身的价值。

需要说明的是, 由于实际程序所限, 基于连续地址与基于调用关系策略是混用的。因此在统计过程中, 部分结果重复统计在表中, 与上文中总的匹配数量有出入。但通过差值计算不难得出, 纯粹依靠连续地址匹配的函数节点仍然有

782 709 个, 相当于 36% 的贡献量。

在时耗问题上, 受程序提供 API 影响, 程序时耗的精度是 1 s, 因此带来误差。结合连续地址和调用关系混用导致的时耗重计算, 其数据的总和是高于第 5.3 节的。但从单位时间匹配节点量的角度看, 即使是混合的数据, 也可以证明连续地址策略的效率是绝对高于调用关系策略的。

对于调用关系内的 3 种不同策略, 其统计结果如表 4 所示。

基于兄弟节点策略本质上是基于父子节点关系, 从匹配数量上可以看出, 基于父子关系匹配的节点总数是基于调用关系匹配的总数。但父子关系的总时耗上少于基于调用关系的时耗, 因为基于兄弟节点本身有筛选兄弟节点的流程, 以降低无效重匹配的时耗。这部分时耗差正是兄弟节点策略独有的时耗。兄弟节点策略贡献了半数以上的匹配量。

从单位时间匹配量而言, 基于已匹配父节点的策略效率更高, 但是基于已匹配子节点的策略可以匹配更多的节点。从失败消耗上讲, 后者更高。

5.5 与 Bindiff 比较实验

本节介绍本文方法匹配结果与 Bindiff5 匹配结果的差异, 并使用 A、B 两组样本进行比对实

表 2 单位时间节点匹配量 (匹配个数/s)
Table 2 Number of matching nodes per unit time (number of matches/s)

K	δ			
	70%	80%	90%	100%
1	2.2 (2249727/1022386)	2.05 (2142435/1046574)	2.24 (1869028/835100)	2.57 (1572394/610913)
10	2.25 (2268523/1006288)	2.28 (2159454/945639)	2.24 (1886530/841410)	2.58 (1578812/611188)
50	2.44 (2273280/932350)	2.23 (2162729/967990)	2.27 (1904032/840368)	1.88 (1584180/841044)
100	2.19 (2279676/1042411)	2.65 (2104008/794232)	2.28 (1909337/836784)	2.3 (1589379/691464)
500	2.56 (2299658/899247)	2.3 (2193307/954281)	2.09 (1930581/922702)	2.1 (1609010/765023)

注: 每一单元格表示, 在相应参数下, 比对 100 对样例后, 匹配量、时间开销的总和, 单位是个数和秒。

表 3 各策略匹配函数效果
Table 3 Matching function effect of each strategy

策略	时耗/s	匹配数量	匹配轮次	失败时耗/s	失败轮次
调用关系	759 176	1 325 891	1 651	4 438	525
连续地址	215 337	816 915	392	2	31
未匹配区间	25 920	52 016	564	12 123	271
统计特征	3 015	9 511	99	0	0

注: 轮次表示候选函数组匹配的次數, 失败表示一轮匹配未产生新的匹配节点。

验后,完成采样分析。Bindiff5 是 2019 年 3 月最新更新的工具,且其算法^[14]主要依赖已匹配父节点的调用关系,具有比较价值。实验表明,本文方法匹配的结果与 Bindiff 有差异,各个策略带来的差异量如表 5 所示。

实验结果表明以下结论。

1) 基于连续地址和基于未匹配区间两种非调用关系的策略提供了较高的差异量。差异函数的平均长度小于 30,而两种基于地址分布的策略提供的差异函数,其平均长度普遍小于其他方法,说明基于地址分布策略影响的匹配结果对小函数更有效。

2) 基于调用关系和统计特征的策略与 Bindiff5 的分析结果相比有 97%以上的相似度,说明两者实验效果接近。

3) 基于地址分布策略累积贡献的差异量占所有策略匹配总量 13.18%,是很可观的差异量。这些差异量是直接基于该方法产生的差异量,如果分析这些差异匹配节点影响到的匹配节点,可能会在调用关系的匹配结果中找到。由此说明,基于地址分布的策略对差异匹配的影响可能高于预期,而基于调用关系贡献的差异可能低于预期。

针对差异匹配结果,需要人工核实 Bindiff5 与本文方法差异的函数匹配结果哪个正确。由于

差异量较大,采用抽样分析的方法,将 A1-A10 和 B1-B10 产生的 100 对测试样例,针对不同的候选函数组构造策略抽取 500 个差异匹配结果,共 2 215 个(基于统计特征仅有 215 个差异结果,无法抽取更多),对结果进行人工核对,时耗 10 天,核对结果如表 6 所示。通过人工核对,差异结果中本文方法至少有 70%的正确率,本文方法匹配结果的 11.8%可以修正 Bindiff 的错误。需要说明的是,本文测试用例中确实存在人工无法判别是否匹配的情况,这主要发生在小函数中,对于以往的研究,研究者往往忽略了这些数据。

5.6 跨系列比较

本节使用 A、C 两组样本进行比对实验,通过分析本文方法比对结果和 Bindiff 比对结果的差异结果可知,本文方法各匹配策略的准确率。

本实验采用抽样分析的方法,将 A1-A10 和 C1-C4 产生的 40 对测试样例针对不同的候选函数组构造策略抽取 100 个差异匹配结果,共计 466 个(基于统计特征仅有 77 个差异结果,基于未匹配区间仅有 89 个结果),对结果进行人工核对,结果如表 7 所示。

实验结果表明,在跨系列固件比对条件下,本文方法依然可以对 Bindiff 比对结果的错误进行修正,且准确率在 60%以上。

表 4 调用关系策略匹配函数效果
Table 4 Matching function effect of calling relationship strategy

策略	时耗/s	匹配数量	匹配轮次	失败时耗	失败轮次
调用关系	759 176	1 325 891	1 651	4 438	525
兄弟节点	565 724	950 268	14 372	96 458	5 220
父节点已匹配	129 327	537 577	28 744	8 308	9 942
子节点已匹配	508 578	788 314	28 744	168 168	13 064

表 5 各策略匹配结果与 Bindiff5 的差异量
Table 5 The amount of difference between the matching results of each strategy and Bindiff 5

策略	差异匹配数量	占该法匹配总量百分比	函数平均指令数
父节点已匹配	10 094	1.87%	21
子节点已匹配	20 094	2.55%	26
连续地址	244 166	31.19%	14
未匹配区间	33 133	64.70%	10
统计特征	215	2.26%	121

注:差异匹配数量,本文匹配结果与 Bindiff5 不同的函数数量。

表 6 同系列固件比对各策略匹配差异结果抽测情况

Table 6 Comparing the same series of firmware to test the results of the matching differences of each strategy

策略	差异匹配数量/个	本文方法正确数量/个	Bindiff 正确数量/个	两种方法都判别错误数量/个	人工无法鉴定两种方法哪种正确
父节点已匹配	500	433	5	40	22
子节点已匹配	500	402	13	60	24
连续地址	500	435	2	27	36
未匹配区间	500	351	2	64	82
统计特征	215	210	0	4	1

表 7 跨系列固件比对各策略匹配差异结果抽测情况

Table 7 Cross-series firmware comparison of each strategy matching difference result sampling test situation

策略	差异匹配数量	本文方法正确数量	Bindiff 正确数量	两种方法都判别错误数量	人工无法鉴定两种方法哪种正确
父节点已匹配	100	66	3	16	15
子节点已匹配	100	71	4	16	9
连续地址	100	77	0	10	13
未匹配区间	89	73	0	16	0
统计特征	77	68	0	8	1

实际上,跨系列的固件本身仅有部分功能模块相似,固件间的相似函数数量较少。使用 A、B 两组样本进行同系列不同功能和版本的固件间比对,足以证明本文方法在不同相似程度固件间的效果, A、C 组的跨系列比对实验,主要作用是更直观地证明本文方法在低相似度固件间比对的效果。

6 相关工作

除引言提到的研究外,二进制比对领域也有其他分支,但与本文目标关联较低。文献[15]提取了函数调用图(FCG, function call graph),引入图编辑距离的概念进行相似性比对。文献[16]针对 FCG 使用 SDMF 来计算图的相似性距离,该法主要通过测量匹配过程中,最小匹配成本的路径,形成最优匹配路径,以此计算相似性。文献[17]针对已有签名方法易被小变化欺骗、无法形成匹配的情况,使用调用图,规避这类问题,并使用模拟退火算法来近似调用图编辑距离。文献[18]基于子图匹配的层次分析方法,在剔除确定匹配节点后形成待检测的相似子图,利用函数调用序列,进行相似序列比对,以此形成新的相似节点匹配。文献[19]主要比对源码中的补丁。文献[20]通过提取 Tracelet(一段短而连续的执行路径)完成比对。文献[21]先将二进制文件基本

块进行符号简化,以此抵抗语法变化。而后,针对符号简化形成树形图,使用基于树编辑距离等的匹配方法,进行相似性计算。

7 结束语

本文设计了基于候选函数组的函数对应关系构建方法,补充完善了函数 n 层局部网络匹配的概念;利用统计特征、调用关系、函数地址分布 3 种策略缓解了传统传播算法的传播壁垒,并分析了时间开销。其中,基于地址分布的策略有效弥补其他方法的不足,具有最为优良的实验效果。本文提出了基于候选函数组的节点匹配方法,使用本文方法形成原型系统,进行了实验分析,确定了初始匹配节点量 K 与相似度阈值 δ 的最佳选值 100 和 80%,验证了各个候选函数组构造策略对匹配工作产生的影响(包括匹配量、时耗),与 Bindiff5 的匹配结果进行比对,本文方法匹配结果的 86.04%与 Bindiff 匹配结果一致,而 11.3%可以修正 Bindiff 的匹配错误。

参考文献:

- [1] A survey of binary code similarity[R]. 2019.
- [2] GAO J, YANG X, FU Y, et al. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary[C]//Conference on Automated Software Engineering(ASE 2018). 2018.

- [3] FENG Q, ZHOU R, XU C, et al. Scalable graph-based bug search for firmware images[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 480-491.
- [4] XU X, 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. 2017: 363-376.
- [5] HU Y, ZHANG Y, LI J, et al. Cross-architecture binary semantics understanding via similar code comparison[C]//2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). 2016: 57-67.
- [6] 刘春红, 郭涛, 崔宝江, 等. 二进制文件同源性检测的结构化相似度计算[J]. 北京邮电大学学报, 2012, 35(3): 56-60.
LIU C H, GUO T, CUI B J, et al. Similarity computation for executable objects homology detection based on structural signature [J]. Journal of Beijing University of Posts and Telecommunications, 2012, 35(3): 56-60.
- [7] CESARE S, XIANG Y. Classification of malware using structured control flow[C]//Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing-Volume 107. 2010: 61-70.
- [8] PEWNY J, GARMANY B, GAWLIK R, et al. Cross-architecture bug search in binary executables[C]//2015 IEEE Symposium on Security and Privacy. 2015: 709-724.
- [9] ESCHWEILER S, YAKDAN K, GERHARDS-PADILLA E. DiscoverRE: efficient cross-architecture identification of bugs in binary code[C]//NDSS. 2016.
- [10] Flake H. Structural comparison of executable objects[C]//Proc of the International GI Workshop on Detection of Intrusions and Malware & Vulnerability Assessment. 2004: 161-174.
- [11] DULLIEN T, ROLLES R. Graph-based comparison of executable objects[J]. SSTIC, 2005, 5(1): 3.
- [12] 王建新, 杨凡, 韩锋. 二进制文件比对中的指令归并优化算法[J]. 计算机应用与软件, 2013, 30(12): 40-42+126.
WANG J X, YANG F, HAN F. Instruction merging optimization algorithm in binary files comparison[J]. Computer Applications and Software, 2013, 30(12): 40-42+126.
- [13] KHOO W M, MYCROFT A, ANDERSON R. Rendezvous: a search engine for binary code[C]//Proceedings of the 10th Working Conference on Mining Software Repositories. 2013: 329-338.
- [14] BOURQUIN M, KING A, ROBBINS E. Binslayer: accurate comparison of binary executables[C]//Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop. 2013: 4.
- [15] HU X, CHIUEH T, SHIN K G. Large-scale malware indexing using function-call graphs[C]//Proceedings of the 16th ACM Conference on Computer and Communications Security. 2009: 611-620.
- [16] 刘星, 唐勇. 恶意代码的函数调用图相似性分析[J]. 计算机工程与科学, 2014, 36(3): 481-486.
LIU X, TANG Y. Similarity analysis of malware's function-call graphs[J]. Computer Engineering & Science, 2014, 36(3): 481-486.
- [17] KOSTAKIS O, KINABLE J, MAHMOUDI H, et al. Improved call graph comparison using simulated annealing[C]//Proceedings of the 2011 ACM Symposium on Applied Computing. 2011: 1516-1523.
- [18] 孙贺, 吴礼发, 洪征, 等. 基于函数调用图的二进制程序相似性分析[J]. 计算机工程与应用, 2016, 52(21): 126-133.
SUN H, WU L F, HONG Z, et al. Research on function call graph based method for similarity analysis of binary files[J]. Computer Engineering and Applications, 2016, 52(21): 126-133.
- [19] JANG J, AGRAWAL A, BRUMLEY D. ReDeBug: finding unpatched code clones in entire OS distributions[C]//2012 IEEE Symposium on Security and Privacy. 2012: 48-62.
- [20] DAVID Y, YAHAV E. Tracelet-based code search in executables[J]. ACM Sigplan Notices, 2014, 49(6): 349-360.
- [21] PEWNY J, SCHUSTER F, BERNHARD L, et al. Leveraging semantic signatures for bug search in binary programs[C]//Proceedings of the 30th Annual Computer Security Applications Conference. 2014: 406-415.
- [22] Cisco 2600 Series Multiservice Platforms - Retirement Notification[EB].
- [23] Bindiff5[EB]. 2019.
- [24] COSTIN A, ZADDACH J, FRANCILLON A, et al. A large-scale analysis of the security of embedded firmwares[C]//23rd {USENIX} Security Symposium ({USENIX} Security 14). 2014: 95-110.
- [25] 肖睿卿, 刘胜利, 颜猛, 等. 节点层次化的二进制文件比对技术[J]. 计算机工程与应用, 2017, 53(21): 144-150.
XIAO R Q, LIU S L, YAN M, et al. Comparison technology of binary files based on hierarchical nodes[J]. Computer Engineering and Applications, 2017, 53(21): 144-150.

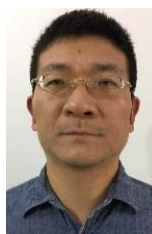
[作者简介]



肖睿卿(1992-), 男, 黑龙江牡丹江人, 数学工程与先进计算国家重点实验室博士生, 主要研究方向为二进制分析。



祝跃飞(1962-), 男, 浙江兰溪人, 数学工程与先进计算国家重点实验室教授、博士生导师, 主要研究方向为安全协议和密码学。



刘胜利(1973-), 男, 河南周口人, 数学工程与先进计算国家重点实验室教授、博士生导师, 主要研究方向为网络设备安全和网络攻击检测。



芦斌(1982-), 男, 山西灵石人, 数学工程与先进计算国家重点实验室副教授, 主要研究方向为信息安全、机器学习。