计算机科学
COMPUTER SCIENCE

一种基于多粒度特征的软件多样性评估方法

迟宇宁, 郭云飞, 王亚文, 扈红超

引用本文

迟宇宁, 郭云飞, 王亚文, 扈红超.一种基于多粒度特征的软件多样性评估方法[J]. 计算机科学, 2022, 49(12): 118-124.

CHI Yu-ning, GUO Yun-fei, WANG Ya-wen, HU Hong-chao. Software Diversity Evaluation Method Based on Multi-granularity Features [J]. Computer Science, 2022, 49(12): 118-124.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211200029

一种基于多粒度特征的软件多样性评估方法

迟宇宁　郭云飞　王亚文　扈红超
解放军战略支援部队信息工程大学信息技术研究所　郑州 450001
(cyn_091981@163.com)

摘　要　针对现有软件多样性评估方法普遍采用单一特征,无法准确表征软件特性进而导致评估准确度较低的问题,提出了一种基于多粒度特征的软件多样性评估方法。该方法从程序的指令、函数、基本块、二进制文件4个粒度进行分析,首先通过小素数乘积法、动态权重分配等算法获取不同粒度的差异度特征,然后根据差异度分析该粒度的多样性,进而探讨多样化技术的有效性。实验部分采用GNU核心程序集,对指令替换、控制流平坦、伪控制流、NOP插入等7种软件多样化方法进行了综合评估,分析了不同软件多样化方法对不同粒度的特征带来的差异程度和多样性,验证了评估算法的适用性。实验结果表明,该评估方法能够从纵向和横向两个方向对软件多样化方法的有效性进行准确评估,对后续多样化技术的研究具有参考价值。

关键词:软件多样化;多粒度特征;多样性分析;小素数乘积法;量化评估
中图法分类号　TP393

Software Diversity Evaluation Method Based on Multi-granularity Features

CHI Yu-ning,GUO Yun-fei,WANG Ya-wen and HU Hong-chao
Institution of Scientific and Technical Information,People's Liberation Army Strategic Support Force Information Engineering University,Zhengzhou 450001,China

Abstract　Aiming at the problem that existing software diversity evaluation methods generally adopt single feature,a software diversity evaluation method based on multi-granularity feature is proposed. This method analyzes four granularity of program:instruction,function,basic block and binary file. First,different granularity are obtained by small prime product method and dynamic weight distribution algorithm. Then,the granularity is analyzed according to the effectiveness of diversification technology. In the experimental part,GNU coreutils is used to comprehensively evaluate 7 software diversification methods. The result is analyzed to verify the applicability of the evaluation algorithm. Experimental results show that this evaluation method can accurately evaluate the effectiveness of software diversification methods from both vertical and horizontal directions,which has reference value for the research direction of subsequent diversification technology.

Keywords　Software diversity,Multi-granularity feature,Diversity analysis,Prime product method,Quantitative evaluation

1　引言

随着信息技术及互联网的飞速发展,众多的应用软件在让人们的生活变得更加便利的同时,也引入了更大的安全风险。究其原因,大多数软件在其设计阶段采用了相似或相同的架构、协议,使得软件漏洞易被黑客在同质化环境中利用。

为了抵御此类威胁,各类安全机构研究了多种防御方法,例如限制内存的可执行页来抵抗大多数代码注入攻击[1],通过动态污点追踪信息流的方法来识别并破坏内存攻击[2]。但这些方法大多只能针对某种漏洞进行防御,具有较大的局限性,且目前已有不同的攻破方式[3]。

软件多样化是新兴的软件保护技术,指更改软件的内部接口和结构以生成其独特的多样化版本。多样化打破了"单一文化主义",并在软件部署过程中引入了"多元文化主义"。目前的软件多样化着眼于程序布局的不同粒度的随机化,可以在不同地方用自动化的方式实现,如通过编译器实现编译时软件的指令、函数或基本块级多样化[4],在加载时对程序逆向后替换loader加载器[5]实现多样化等。现代操作系统对程序中对象的基址进行随机化,实现了地址空间布局的随机化(ASLR)。

大量理论和实践证明了软件多样化技术能够有效提高攻击成本以及增强软件安全性,但如何对软件多样化技术的多样性效果进行通用评估是一个难点。Hernandez-castro[6]等从香农-维纳信息熵[7]的角度,利用物种多样性来分析操作系统的多样性。也有依靠逻辑认证[8]证明多样化方法能改变攻击依赖的程序属性,导致原有的攻击失效,说明防御成功。现有

到稿日期:2021-12-02　返修日期:2022-05-16
基金项目:国家重点研发计划(2021YFB1006200,2021YFB1006201);国家自然科学基金(62072467)
This work was supported by the National Key Research and Development Program of China(2021YFB1006200,2021YFB1006201) and National Natural Science Foundation of China(62072467).
通信作者:王亚文(wyw@ndsc.com.cn)

软件多样化评估方法缺少对多样化技术引入的程序多样性分析。因此，本文提出了一种基于二进制文件多粒度量化的软件多样化技术评估方法，对比程序中指令、函数、基本块以及二进制文件，从多样性的角度进行定量分析，以实际讨论软件多样化技术对程序的异构影响。

2 相关工作

Sebastians 等[9-10]用符号执行来评估各种代码混淆方法的强度，结果表明代码混淆减小了程序覆盖率。但符号执行主要关注对程序逆向的干扰程度，由于程序大小的限制，容易受到路径状态空间爆炸的影响。

Coffman 等[11]采用了 3 种模糊散列算法，对二进制文件的符号表进行了相似性分析，但该方法适用于描述静态链接 libc 的代码重用情况，未考虑程序的函数、基本块等逻辑关系，低估了相似性。

Liu 等[12]通过软件复杂度的通用指标，从程序长度、循环复杂度、扇入/扇出复杂度和数据结构复杂度 5 个方面评估了不同软件多样化技术的复杂性。

Gearhart 等[13]提出了非结构化的方法，利用非结构化文本分析的技术来获取多样化软件的特征向量，以区分多样性策略。

通常来说，评估一项技术引入的多样性程度是评估该技术泛化程度的一种方法。直观地说，与引入较少多样性且相似性更高的技术相比，造成差异度更高且引入多样性更多的技术更能有效地延缓同质攻击。因此，本文的目标是通过直接量化由各种技术引入的差异度来评估多样性。

目前比较不同二进制文件的方法较多，但大多聚焦于代码剽窃鉴定、补丁对比和恶意代码变种同源检测，且只关注文件的整体相似度。但由于多样化方法存在不同粒度分类，因此同一粒度下的不同技术的实现效果也有差异。为了全面地衡量差异度，需要对程序文件的多粒度特征也进行评估，即从纵向多维地反映不同粒度的多样化方法的有效性，并在同一粒度之间进行多样性效果的横向对比。

3 多粒度的多样性评估算法

3.1 算法预处理

本文采用补丁对比工具 BinDiff 作为分析的基础工具。BinDiff 具有多种特征，在匹配基本块和函数方面总体上比其他工具更有效，并且提供了两个二进制文件的详细对比情况。但在具体的相似度计算上，函数相似度只考虑了匹配的函数数量占总数量的比例，而未考虑函数的具体匹配程度。在整体文件的相似度计算中，采用统一的权重分配方法不能满足动态分析不同粒度的多样化技术的要求。本文在 BinDiff 原有匹配数据的基础上进行针对性的算法改进，计算两个程序的不同粒度特征的差异度，用差异度来反映多样性。

3.2 指令多样性度量

BinDiff 原有的指令相似性计算采用的是通过遍历两个二进制文件基本块的指令级图，以指令序列为基本单位匹配相同的字符串引用、子函数调用及助记符序列[14]，但该方法强调以基本块调用关系为主的指令序列整体匹配度，未考虑指令本身内部的变化及差异，且不同程序在不同编译器生成时可能伴随着汇编指令序列不一致，使得指令级图发生差异。在多样化技术中，堆栈类多样化技术未直接对指令进行多样化操作，但可能因为堆栈布局变化影响了指令排序，从而低估指令的相似性。为了尽可能从细粒度反映指令之间的差异，减少客观指令重排的影响，考虑到指令和函数的逻辑 关系，本文采用小素数乘积法[15]来代替原有的指令流匹配算法以完成多样性计算。

小素数乘积法是基于正整数的唯一分解定理而提出的，即任何一个大于 1 的正整数，要么本身就是素数，要么可以写为两个或两个以上的素数的积，而且若不考虑排列的顺序，正整数分解为素数乘积的方式是唯一的。

$$Q=P_1^{a_1}P_2^{a_2}\cdots P_n^{a_n} \tag{1}$$

为汇编指令的每个助记符分配一个唯一小素数，将指令流转化为小素数的乘积，比较两个素数乘积的关系。根据整数分解的唯一性和整数乘法的顺序可交换性，若两个程序的指令助记符集合的素数乘积相等，则说明这两个程序的指令集合相同。

对于本文来说，关键在于如何对不相等的两个助记符素数乘积进行进一步的差异性计算，从而得到两个指令集的整体多样性。考虑程序的函数逻辑关系，设计指令差异度计算的相关定义和算法如下。

定义 1 二进制文件 A 的函数集合表示为 $F=\{f_1,f_2,\cdots,f_n\}$，二进制文件的函数指令集合表示为 $I=\{I_1,I_2,\cdots,I_n\}$，其中 I_i 为对应的单个函数 f_i 的指令集。其中，每一个函数 f_i 的第一条指令是它的入口，最后一条指令是它的出口，单个函数 f_i 的指令集合表示为 $I_i=\{i_1,i_2,\cdots,i_m\}$。即二进制文件 A 有 n 个函数，对应 n 个指令集，其中每个函数的指令集各自有不同数量的 m 条指令。

定义 2 设有助记符列表 $M=\{m_1,m_2,\cdots,m_k\}$，对每个助记符分配一个唯一素数 t，得到助记符和素数的映射表 L：$\{M,T\mid m_1\rightarrow t_1,m_2\rightarrow t_2,\cdots,m_k\rightarrow t_k\}$，如表 1 所列。根据映射表 L，集合 I_i 中每个指令 i_j 的助记符将映射为一个素数 t_j，最后得到单个函数指令集 I_i 和素数的映射集合 $P_i=\{p_1,p_2,\cdots,p_j,\cdots,p_m\mid p_j\in S\}$，以及整个二进制文件的素数集合 $P=\{P_1,P_2,\cdots,P_n\}$。

表 1 常用助记符和小素数映射

Table 1 Mnemonics and prime numbers

常用指令助记符	对应的小素数
MOV	3
PUSH	5
POP	7
XOR	11
JMP	13
LEA	17
SUB	19
ADD	23
—	—

算法 1 指令差异度算法

输入：正常二进制文件 A，多样化二进制文件 B

输出：A 和 B 的指令差异度 H_I

Step1 由 BinDiff 获取 A 和 B 的函数集合 F_a 和 F_b，以及 A 和 B 的函数匹配情况表，建设匹配的函数关系集合 $\alpha=\{(f_{a1},f_{b1}),(f_{a2},f_{b2}),\cdots,(f_{ai},f_{bi}),\cdots,\mid f_{ai}\in F_a,\ f_{bi}\in F_b\}$ 和不匹配的函数集合

$\beta=\{(f_{aj}\in F_a$，且 $f_{aj}\not\subset\alpha)$，$(f_{bj}\in F_b$，且 $f_{bj}\not\subset\alpha)\}$。

Step2 依次对 α 和 β 中的函数指令进行扫描，得到指令集合表 $I_\alpha=\{(I_{a1},I_{b1}),(I_{a2},I_{b2}),\cdots,(I_{ak},I_{bi}),\cdots\}$ 和 $I_\beta=\{(I_{aj},\cdots,I_{an}),(I_{bj},\cdots,I_{bn})\}$。通过助记符和素数的映射表 L，得到素数集 P_α 以及 P_β。对 P_α 的每个素数对分别求素数乘积，如式(2)所示，最后得到乘积对集合 $Q_\alpha=\{(Q_{a1},Q_{b1}),(Q_{a2},Q_{b2}),\cdots,(Q_{ai},Q_{bi}),\cdots\}$。

$$Q_i=\prod_{k=1}^{n}p_k,p_k\in P_i \tag{2}$$

Step3 扫描 Q_α，取当前的乘积对 Q_{ai} 和 Q_{bi} 进行对比，如果 $Q_{ai}=Q_{bi}$，转 Step4；如果 $Q_{ai}\neq Q_{bi}$，转 Step5；如果 Q_α 已扫描结束，说明匹配函数对的指令对比完毕，转 Step6。

Step4 对于相等的乘积对 Q_{ai} 和 Q_{bi}，计算其中任意一个乘积 Q_i(选用 Q_{bi})含有的素数数量 N_i；计算完成后转 Step3。

Step5 对于不相等的乘积对 Q_{ai} 和 Q_{bi}，计算 Q_{ai} 和 Q_{bi} 总共的素数数量 O_i。不考虑排序因素，两个素数乘积存在两种差异：1)某个素数 p 只在某个素数积中出现；2)某个素数 p 在 P_{ai} 和 P_{bi} 中出现的次数不同。从素数集中取出一个素数 p，计算 p 在 P_{ai} 和 P_{bi} 中出现的次数差，以及同时出现的次数。遍历素数集的素数后，累加得到该乘积对的总体次数差 D_i，累加得到乘积对的相同素数数量 G_i。Q_{ai} 和 Q_{bi} 含有的不重叠素数数量 $N_i=O_i-G_i$。计算完成后转 Step3。

Step6 扫描 P_β，计算 P_β 总共素数数量 O_j。从素数集中取出一个素数 p，统计 p 在子集 $\{P_{aj},\cdots,P_{an}\}$ 和子集 $\{P_{bj},\cdots,P_{bn}\}$ 中出现的次数差，以及同时出现的次数。遍历素数集后，累加得到这两个子集的总体次数差 D_j 和相同的素数数量 G_j。A 和 B 的不匹配函数中含有的不重叠素数数量 $N_j=O_j-G_j$。

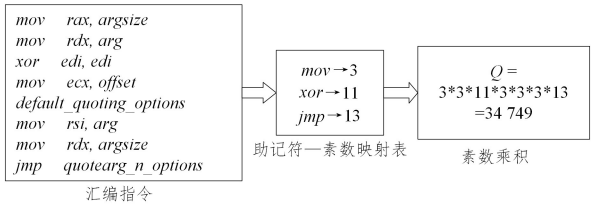
Step7 对 Step4 和 Step5 得到的总体次数差进行累加，得到匹配函数的指令差异数 D，对不重叠素数总数量进行累加，得到匹配函数的并集指令数量 N，如式(3)所示：

$$D=\sum D_i,N=\sum N_i \tag{3}$$

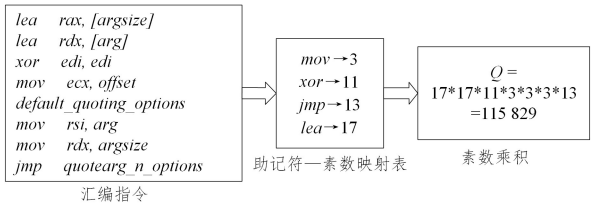
Step8 计算 A 和 B 的指令整体差异度 H_I(Instruction Heterogeneity)，如式(4)所示，即指令差异数和并集总指令数量的比值。

$$H_I=\frac{D+D_j}{N+N_j},H\in[0,1] \tag{4}$$

以图 1 中正常编译的 base64 程序 A 和采用指令替换多样化编译的 base64 程序 B 的函数对 quotearg_mem 为例进行分析。



(a)正常 base64 程序 A 的函数对 quotearg_mem



(b)指令替换多样化 base64 程序 B 的函数对 quotearg_mem

图 1 base64 的函数 quotearg_mem 指令及素数乘积

Fig. 1 Quotearg_mem instruction and prime number product

该函数对中实现的功能相同但指令有区别，助记符的素数乘积不相等。该乘积对的总体素数次数差 $D=4$(mov 次数差 2+lea 次数差 2)，总共素数量 $O=14$，相同的素数数量 $G=5$，不重叠的并集素数数量 $N=14-5=9$。

3.3 函数多样性度量

BinDiff 带有多种函数匹配算法，有较好的匹配效果。在函数相似性上，BinDiff 原有的方法是不区分每个相似函数对的相似值，直接计算相似函数对的总数量和两个程序的函数并集总数量的比值，为函数的相似性。实际上每个函数对的相似程度不同，有的函数对可达 100%的相似性，而有的函数对只有 10%甚至更低。低于 100%相似的函数对，其区别在于控制流、指令、函数调用或跳转关系、函数入口点基本块和循环次数等发生分歧。因此，原算法会高估函数相似性。

为了避免单纯计数而夸大函数的相似性，设计文件的函数差异度计算的算法如算法 2 所示。

算法 2 函数差异度算法

输入：正常二进制文件 A，多样化二进制文件 B

输出：A 和 B 的函数差异度 H_I

Step1 由 BinDiff 获取 A 和 B 的函数集合 F_a 和 F_b，以及 A 和 B 的函数匹配情况表，建设匹配的函数关系集合 $\alpha=\{(f_{a1},f_{b1}),(f_{a2},f_{b2}),\cdots,(f_{ai},f_{bi}),\cdots|f_{ai}\in F_a,f_{bi}\in F_b\}$ 和不匹配的函数集合 β，计算集合 β 中函数的个数 D。

Step2 根据集合 α 中的匹配关系，获取每一对函数的相似值 s_i，建立相似值集合 $s=\{s_1,s_2,\cdots,s_i,\cdots,|s_i\in(0,1]\}$。

Step3 遍历集合 s，将当前成员 s_i 分别选出并放入集合 $S_1=\{s_i\in(0,0.1)\}$，$S_2=\{s_i\in[0.1,0.2)\}$，$S_3=\{s_i\in[0.3,0.4)\}$，$\cdots$，$S_9=\{s_i\in[0.9,1)\}$，$S_{10}=\{s_i=1\}$。

Step4 遍历 S_1 到 S_{10}，对当前 S_i 计算成员的数量 n_i，并选出 S_i 成员的中位数 Mid_i，计算 $x_i=Mid_i*n_i$。

Step5 计算 A 和 B 的函数整体差异度 H_f(Function Heterogeneity)，如式(5)所示：

$$H_F=1-\frac{\sum_{i=1}^{10}x_i}{D+\sum_{i=1}^{10}n_i},H\in[0,1] \tag{5}$$

3.4 二进制文件多样性度量

原来的算法中，采用唯一一套统一的权重分配方法，权重偏向于文件控制流。但是，对于多样化技术，不同层次的技术针对的粒度各有侧重。对指令级的多样化方法采用原算法，无法突出其在指令级的优势，从而低估了多样性和有效性。为了更加准确地反映各层级的特点，针对不同粒度的多样化技术，设计多样化软件的文件差异度的算法如算法 3 所示。

算法 3 文件差异度算法

输入：由算法 1 和算法 2 得到的指令整体差异度 H_I 和函数整体差异度 H_F；由 BinDiff 得到的基本块整体差异度 H_B，控制流图边整体差异度 H_G 和调用图 md 索引值差异 H_{CJ}

输出：A 和 B 的文件差异度 H

Step1 判断文件 B 的多样化方法粒度 Type，有如表 2 所列的权重分配方法。

Step2 根据文件 B 的 Type 和相应权重分配方法，计算文件异质度 H，如式(6)所示：

$$H=\sum H_i*W_i,H\in[0,1] \tag{6}$$

表 2 文件差异度算法的权重分配

Table 2 Weight allocation of file heterogeneity algorithm

多粒度差异度 H_i	权重 W_i
控制流图边 H_G	0.3
基本块 H_B	If type: 基本块级/程序级 0.3 Other 0.1
函数 H_F	If type: 函数级 0.3 Other 0.1
指令 H_I	If type: 指令级 0.3 Other 0.1
调用图 md 索引值差异 H_{CJ}	0.2

4 实验结果与分析

4.1 实验数据集

现阶段常用的软件多样化技术通过编译器实现，本文结合 Multicompiler[16] 和 LLVM Obfuscator[17] 构建了具有多种软件多样化功能的编译器，根据程序的粒度[18]，主要采用以下多样化策略。

(1)指令级

1)NOP 插入(NOP Insertion)。NOP 插入是随机在当前指令之前插入 1 个 NOP 指令[16]，NOP 指令不修改寄存器的状态和内存。在本文实验中，NOP 插入的方式为原程序每个指令前有 40%的随机概率被插入 NOP 指令，整个程序被插入的 NOP 指令占指令总数的 30%。

2)指令替换(Instruction Replace)。相同功能有不同的指令实现方式，如 $a/5=a*(0.2)$。该方法在保证原功能的前提下，进行等价的指令序列替换。

(2)函数级

1)栈填充(Fill Stack)。往栈空间里填充无用数据，增加实际栈的空间。实验的填充率为 50%，将预留的栈空间填充至 50%的利用率。

2)堆栈交换(Exchange Stack&heap)。该方法将 Linux 可执行文件布局中的堆栈内存布局进行分配区域交换，使栈处于低地址区域，堆处于高地址区域。

(3)基本块级

1)控制流平坦(Control Flow Flat)。利用分发器控制[19]基本块之间的直接跳转，模糊整个程序的跳转逻辑。分发器的控制让基本块都出现在同一层，隐藏原始跳转结构。

2)伪控制流(Fake Control Flow)。通过插入不影响程序功能的无效跳转基本块[20]，构造一些跳转条件，从而破坏原来的控制流图。

(4)程序级：函数随机化(Function Randomization)

这种方法在整个可执行文件中进行全局性的函数重排列，改变了编译器生成的目标代码中函数的顺序。

本文采用了以上 7 种多样化技术(控制流平坦 CFF、虚假控制流 FCF、NOP 插入 NOP、指令替换 IR、函数随机化 FR、栈填充 FS 和堆栈交换 ESH)以及一种正常的编译方法 Normal。对于每种多样化技术，选择开源大数据集 GNU 核心实用程序，用 ES2 集中编译链接。表 3 列出了关于数据集的基本统计信息，使用 David A. Wheeler 的“SLOC Count”(2.26)计算了数千行代码的数量(KLOC)。

表 3 GNU 核心程序集

Table 3 GNU coreutils

Database	Version	File	KLOC	Instruction
GNU coreutils	8.3	100	60.1	700000

利用本文构建的软件多样化编译器，采用以上提到的 7 种多样化方法以及未经多样化的方法对 GNU 核心程序集进行编译链接。每种方法生成一个包含 100 个二进制程序的数据集，一共 8 个数据集，共计 800 个程序。

用二进制分析工具 IDA7.5 和二进制文件对比工具 BinDiff 对每一个多样化程序和对应的未经多样化的程序进行逆向和初步相似性分析，并对 BinDiff 原有算法做本文所述 3 种算法的改进，初步得出 7 个多样化程序集的每一个程序数据，最后取程序集的平均值并将其作为实验的最终数据。

4.2 各粒度多样性实验结果

4.2.1 指令整体差异度评估

理想情况下，两个多样化的二进制文件应该尽可能地扩大指令多样性。本实验计算了正常执行体和其他所有多样化变体的指令整体差异度。

图 2 给出了每种多样化程序集相对于正常程序集的指令整体平均差异度。在采用的多样化技术中，NOP 插入最有效，异构度高达 92%，这说明指令整体相似数量最少。由于 NOP 指令的插入使整个程序的指令集变大，因此与原程序相比差异度增大。其次，控制流平坦技术在全局性的层次上打乱了原有的程序布局，因此指令多样性较大。

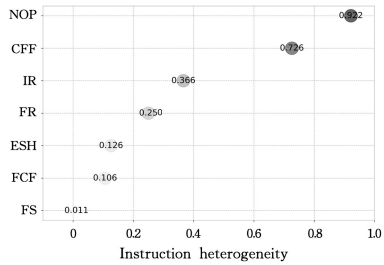


图 2 多样化程序集的指令整体差异度

Fig. 2 Instruction heterogeneity of diversified database

指令替换技术由于能产生逻辑相等但语义不同的替换，可导致采用的助记符发生分歧，例如”ADD”变为”SUB”，因此指令多样性表现不弱。函数随机化改变了函数的原有顺序，在本文的实现方法中还引入了额外的功能函数，总体的指令差异度源于引入的额外函数。而栈填充和栈反转主要影响堆栈布局，对指令的影响较小。

即便采用了多样化技术，指令中始终有共同存在的部分。不同技术的指令差异度取决于该多样化技术引入新指令或替换原有指令的程度，以及对整个程序的逻辑影响。通过本文的指令多样性分析，也能得知具体某种多样化方法的效率。指令级的指令替换技术和 NOP 插入技术引入了不同程度的差异度，这说明本文采用的这两种方法的多样化效率不同。

4.2.2 函数整体差异度评估

图 3 给出了函数整体的异构化计算结果。函数随机化引起的差异度高达 30%，由于本文实验采用的函数随机化方法

引入了额外封装函数，导致整体函数数量增加，并且排序顺序被打乱，引起的调用和跳转关系等发生了变化，因此匹配的函数对呈现的相似性也较小；控制流平坦同理。NOP 插入的函数异构性是源于匹配的函数对的指令高差异而引起的低相似性。指令替换的影响在于细微的函数内部指令差异。其他 3 种多样化方法的函数多样性差距不大。

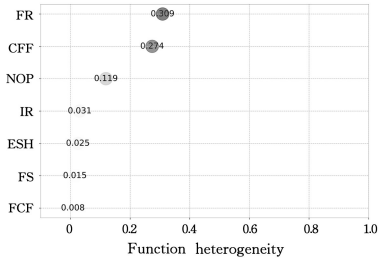


图 3 多样化程序集的函数整体差异度

Fig. 3 Function heterogeneity of diversified database

通过衡量函数差异度，可以反映不同多样化技术对函数的改变程度不同，差异度越高，多样性越大，对函数的直接作用带来的多样性最大，如函数随机化。因为该技术不仅影响了函数布局，还通过全局性的作用改变了函数的入口点基本块。而栈填充和堆栈交换的多样化方法，虽然被归类为函数级的多样化技术，但主要作用于堆栈内存布局上，对函数本质的影响整体较弱。未来的新兴函数类多样化技术，可以采用该方法度量比较具体技术的有效性。

4.2.3 基本块整体差异度评估

图 4 给出了基本块的整体差异度。控制流平坦属于基本块级的多样化技术，因为使整个程序的基本块跳转和调用关系趋于扁平，造成了基本块差异较大，所以差异度最高，多样性最大。同是基本块级的伪控制流技术未能像控制流平坦技术一样带来较大的差异，原因可能是采用的具体多样化方法效率不高，或未能真正实现对应的多样化效果。

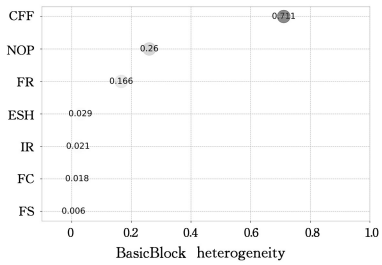


图 4 多样化程序集的基本块整体差异度

Fig. 4 Basicblock heterogeneity of diversified database

4.2.4 文件整体差异度评估

图 5 给出了文件差异度的计算结果。综合指令、函数等细粒度指标后，控制流平坦的差异度高达 62%，是总体效果最理想的多样化技术。这个结果与文献[12]采用程序复杂度方法得到的结果相同，证明了本文算法的有效性。NOP 插入和指令替换相比，指令替换的效果低于 NOP 插入，因为本文采用的 NOP 插入扩充了 30%的指令，对整个文件影响大，该结果符合采用的技术特点。函数重排序方法因为打乱了原有的函数顺序，导致整个二进制文件全局发生改变，因此差异度排第三。

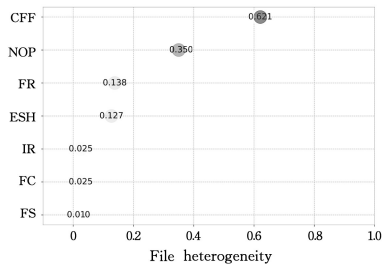


图 5 多样化程序集的文件整体差异度

Fig. 5 File heterogeneity of diversified database

根据实验结果可知，该算法能够有效地反映各种技术对二进制文件造成的多样性差异。对程序整体影响越大的多样化技术，如控制流平坦直接影响了整个程序的控制流布局，最后综合能力就越好。NOP 插入则是因为插入的 NOP 指令较多，引起了程序膨胀，从而激发了多样性。虽然函数重排序影响了全局的函数顺序，但因为实际的调用关系未变，所以效果不如同样影响了全局的控制流平坦技术。堆栈交换和栈填充的差异度均未超过 5%，因为这两种方法着重于内存中堆栈的改变。总的来说，对程序的控制流、函数、基本块引起的全局变化越大，整体文件就越大。

4.3 实验算法对比分析

因为本文采用的主要算法是对 BinDiff 原有相似度计算方法的改进，所以下面从各粒度差异度的整体准确性和个体准确性两个方面对本文算法与 BinDiff 算法进行分析比较，结果如表 4 所列。

表 4 各粒度差异度比较分析

Table 4 Comparison and analysis of different granularity heterogeneity

（单位：%）

多样化技术	指令差异度		函数差异度		文件差异度	
	BinDiff 算法	本文算法	BinDiff 算法	本文算法	BinDiff 算法	本文算法
控制流平坦 CFF	84.9	72.6	2.4	27.4	51.0	62.1
虚假控制流 FCF	35.2	10.6	0.7	0.8	0.2	2.5
函数随机化 FR	36.8	25.0	23.3	30.9	10.0	13.8
指令替换 FI	73.2	36.6	0.7	3.1	3.5	12.7
NOP 插入 NOP	96.1	92.2	0.7	11.9	27.0	35.0
栈填充 FS	32.3	1.1	0.7	1.5	1.0	1.5
堆栈交换 ESH	34.1	12.6	0.7	2.5	2.0	2.5

4.3.1 指令差异度对比分析

采用本文算法与 BinDiff 算法对多样化程序所得指令整体异构性对比数据的准确性进行对比分析，结果如图 6 所示。

(1)整体准确性。通过图 6 可以得出，本文算法得出的指令差异度整体比 BinDiff 算法低，这是由于本文算法尽可能地忽略了客观因素导致的指令重排问题，更加关注指令流的整体逻辑，而 BinDiff 采用的算法则只聚焦于单条指令，较大程度地受编译器优化或客观因素的影响。此处随机抽取 GUN 核心程序集中 5 个程序的源代码，采用 Linux 自带程序和本文编译平台正常编译（非多样化）得到二进制文件，并用本文算法和原算法进行指令差异度计算，最后求得平均值。原算法的指令整体差异度为 18.5%，而本文算法的整体差异度为 6.3%。因为未加多样化技术，自带程序和其他编译平台编译

的主要区别为不同编译方式的指令重排。可见,本文算法可以有效地减轻客观指令重排的影响。

(2)个体准确性。BinDiff 算法中,伪控制流、函数随机化、栈填充和堆栈交换 4 种多样化方法的异构度较为接近。但本文算法中,这 4 种方法的指令差异度呈现较大差异,如 4.2.1 节中的分析,这种差异更符合 4 种多样化方法的特点,准确性更高。

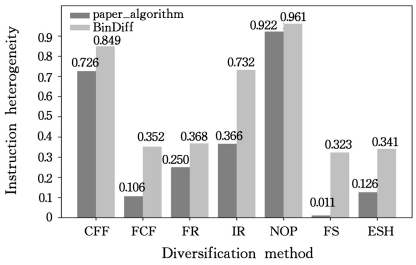


图 6 本文算法与 BinDiff 算法的指令整体差异度对比

Fig. 6 Comparison of instruction heterogeneity between the proposed algorithm and BinDiff

4.3.2 函数差异度对比分析

图 7 给出了本文算法与 BinDiff 算法所得函数整体差异度的对比数据。

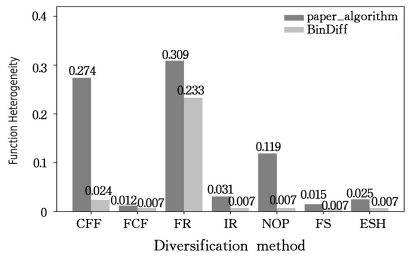


图 7 本文算法与 BinDiff 算法的函数整体差异度对比

Fig. 7 Comparison of function heterogeneity between the proposed algorithm and BinDiff

(1)整体准确性。由图 7 可知,本文算法得出的函数差异度整体高于 BinDiff 得出的差异度。单个函数对的相似度呈现高低不同,这种区别反映了函数对并非百分百相似。而 BinDiff 的算法只在乎匹配的函数对数量,未考虑单个函数对的相似情况,相似度只有 0.01 的函数对也被视为和相似度为 1 的函数对性质相同。若匹配的函数对中相似度低的占多数,则 BinDiff 算法显然会低估函数的差异性。本文算法考虑到了这种情况,对函数对做相似性区分处理,更能准确反映函数差异度及多样性。此处随机选取 5 个 Linux 自带程序,采用交叉对比,例如 cp 程序和 pwd 程序对比,取多次结果的平均值。原算法的函数整体差异度为 40.6%,而本文算法的整体差异度为 68.4%。因为对比的两个程序不同,除去小部分因静态 libc 引用和编译共同部分,其余匹配函数的分值较小。本文算法考虑了不同函数对的相似度分值差异较大的情况,提高了准确性。

(2)个体准确性。BinDiff 算法中,伪控制流、指令插入、NOP 插入、栈填充和堆栈交换 4 种多样化方法的差异度相同。但本文算法对这 4 种方法做出了更贴合其特点的区分。NOP 插入的多样性源于大量 NOP 指令引起的函数对相似性较小,因此差异度稍大,指令替换同理。

4.3.3 文件差异度对比分析

对比本文算法和 BinDiff 算法的文件差异度,结果如图 8 所示。

(1)整体准确性。本文算法所得的文件差异度整体高于 BinDiff 算法。由于 BinDiff 采用的加权计算法中权重是单一分配,而针对不同特性的文件利用了同样的权重分配,这必然会使文件相似性比较结果产生一定的误差。若对指令级多样化方法采用函数权重高的加权法,则会因为指令级多样化技术对函数整体改变有限,从而低估差异度,这很难客观反映该技术的真实效率。本文考虑到了这种情况,根据多样化粒度进行权重不同的分配处理,不同程度地提高了文件差异度,也更能反映真实的符合技术特点的差异度指标。

(2)个体准确性。控制流平坦、指令替换和 NOP 插入的文件差异度中,本文算法得出的文件差异度高出 BinDiff 算法 10%。通过后两种指令级技术可看出,本文算法更全面地考虑了其所属的粒度层次。栈填充和堆栈交换的文件差异度略有提高,但依旧符合其方法对整个文件的影响。而这两种方法之间的差异,也符合堆栈交换对布局的影响高于栈填充的特点。

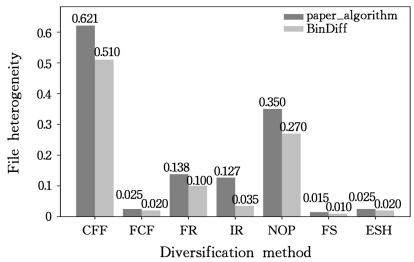


图 8 本文算法与 BinDiff 算法的文件差异度对比

Fig. 8 Comparison of file heterogeneity between the proposed algorithm and BinDiff

4.4 实验评估总结

通过上述分析可清楚得知,不同粒度的多样化技术对整个文件的多样性影响不一致。对文件的全局性影响越大的技术,如控制流平坦,产生的差异度越大,引入的多样性就更高。同时,由于细粒度多样性度量的提出和算法的改进,使得本文算法的数据比 BinDiff 能更准确地反映多样化技术的有效性,如综合几个粒度的分析,可判断本文采用的伪控制流技术可能存在成功率低的问题。

本文算法既能纵向性地给出不同粒度的多样化效果的综合评估,还能横向性地深入挖掘同一粒度多样化技术的有效性差异,以此作为指导,日后可根据这些指标研究差异度更大的多样化技术。

结束语 针对软件多样化技术带来的程序多样性难以量化评估的问题,本文采用多粒度量化分析方法,基于多样化技术所属的粒度级别来计算整个文件的差异度,建立量化评价体系。本文利用 GNU 核心程序数据集开展实验,评估了 7 种不同的多样化技术,并与 BinDiff 方法进行对比,验证了该方法在软件多样化领域的适用性以及有效性。

参 考 文 献

[1] LITCHFIELD D. Buffer Underruns, DEP, ASLR and improving the Exploitation Prevention Mechanisms (XPMs) on the Windows platform[J]. Next Generation Security Software, 2005. https://www.nccgroup.com/globalassets/our-research/uk/whitepapers/xpms.pdf.

[2] LIVSHITS V B, LAM M S. Finding Security Vulnerabilities in Java Applications with Static Analysis[C]// USENIX Security Symposium. 2005, 14: 18-18.

[3] YAO D, ZHANG Z, ZHANG G F, et al. A Survey on Multi-Variant Execution Security Defense Technology[J]. Journal of Information Security, 2020, 5(5): 77-94.

[4] DULLIEN T, ROLLES R. Graph-based comparison of executable objects (english version)[J]. SSTIC, 2005, 5(1): 3.

[5] CRISTIANO G, ANTON K, ANDREW S T. Enhanced operating system security through efficient and fine-grained address space randomization[C]// Proceedings of the 21st USENIX Security Symposium. 2012: 475-490.

[6] HERNANDEZ-CASTRO J, ROSSMAN J. Measuring soft- ware diversity, with applications to security[EB/OL]. [2020-04-13]. https://arxiv.org /abs/1310.3307vl.

[7] SHANNON C E. A mathematical theory of communication[J]. Bell System Technical Journal, 1948, 27(3): 379-423.

[8] COHEN F B. Operating system protection through program evolution[J]. Computers & Security, 1993, 12(6): 565-584.

[9] SEBASTIAN B, CHRISTIAN C, VIJAY G, et al. Code Obfuscation Against Symbolic Execution Attacks[C]// Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC '16). 2016: 189-200.

[10] SEBASTIAN B, CHRISTIAN C, ALEXANDER P. Predicting the resilience of obfuscated code against symbolic execution attacks via machine learning[C]// Proceedings of the 26th USENIX Security Symposium. 2017: 661-678

[11] COFFMAN J, CHAKRAVARTY A, RUSSO J A, et al. Quantifying the Effectiveness of Software Diversity using Near-Duplicate Detection Algorithms[C]// Proceedings of the 5th ACM Workshop on Moving Target Defense. 2018: 1-10

[12] LIU Z W, SUI R, ZHANG Z, et al. Software Diversity Evaluation Based on Information Entropy and Software Complexity [J]. Journal of Information Engineering University, 2020, 21(2): 207-213.

[13] GEARHART A S, HAMILTON P A, COFFMAN J. An Analysis of Automated Software Diversity Using Unstructured Text Analytics[C]// 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2018.

[14] DULLIEN T, ROLLES R. Graph-based comparison of executable objects (english version)[J/OL]. SSTIC, 2005. https://www.docin.com/p-1472608287.html.

[15] DONG Q H, WANG Y G. Partition-based binary file similarity comparison method [J]. Journal of Computer Applications, 2015, 35(10): 2896-2900.

[16] HOMESCU A, NEISIUS S, LARSEN P, et al. Profile-guided automated software diversity [C]// Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2013: 1-11.

[17] JUNO D P, RINALDINI J, WEHRLI J, et al. Obfuscator-LLVM—Software Protection for the Masses[C]// 2015 IEEE/ACM 1st International Workshop on Software Protection (SPRO). ACM, 2015: 3-9.

[18] LARSEN P, HOMESCU A, BRUNTHALER S, et al. SoK: Automated Software Diversity[C]// 2014 IEEE Symposium on Security and Privacy. 2014: 276-291.

[19] LÁSZLÓ T, KISS Á. Obfuscating C++ programs via control flow flattening[J]. Annales Universitatis Scientarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica, 2009, 30(1): 3-19.

[20] COLLBERG C, THOMBORSON C, LOW D. Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs[C]// Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 1998: 184-196.

CHI Yu-ning, born in 1995, postgraduate. Her main research interests include software diversification and mimicry defense.

WANG Ya-wen, born in 1991, Ph.D. His main research interests include cloud computing security and scientific workflow security.

(责任编辑:喻藜)