高健壮性二进制应用程序裁剪

丁铎, 孙聪, 郑涛

引用本文

丁铎, 孙聪, 郑涛. 高健壮性二进制应用程序裁剪[J]. 计算机科学, 2024, 51(10): 208-217.

DING Duo, SUN Cong, ZHENG Tao. Robust Binary Program Debloating[J]. Computer Science, 2024, 51(10): 208-217.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230700008

高健壮性二进制应用程序裁剪

丁 铎[1,2] 孙 聪[1] 郑 涛[3]
1 西安电子科技大学网络与信息安全学院 西安 710071
2 中国电子科技集团公司第五十四研究所 石家庄 050050
3 中国航空工业集团公司西安航空计算技术研究所 西安 710068
(dingduo54@qq.com)

摘 要 应用程序的常用功能仅占其所有功能的小部分。冗余功能代码造成应用程序攻击面扩大,从而增大代码重用攻击风险。二进制程序裁剪能够在分析应用程序二进制的基础上,识别并删除程序冗余代码,减小程序攻击面。现有二进制裁剪方法依赖人工构造的输入产生初始控制流,并依赖启发式方法扩展控制流图,导致方法健壮性和可扩展性受限。文中提出并实现了一种高健壮性二进制应用程序裁剪方法(RBdeb),使用黑盒模糊测试技术获取具有更高健壮性的合法执行轨迹集合,基于图同构算法自动分类相似库函数,提出的路径发现算法从初始执行轨迹构成的二进制控制流子图出发,扩展二进制控制流路径和同类库函数调用,生成高健壮性的裁剪结果二进制文件。实验结果表明,相比现有方案,RBdeb具有更高的路径覆盖率和裁剪后二进制健壮性,路径发现算法和库分类方法具有更强的可扩展性,所提方法能够裁剪大规模实际应用程序。

关键词:程序裁剪;二进制分析;模糊测试;二进制重写;程序分析
中图分类号 TP314

Robust Binary Program Debloating

DING Duo[1,2],SUN Cong[1] and ZHENG Tao[3]
1 School of Cyber Engineering,Xidian University,Xi'an 710071,China
2 The 54th Research Institute of China Electronics Technology Group Corporation Electronic Equipment,Shijiazhuang 050050,China
3 AVIC XI'AN Aeronautics Computing Technique Research Institute,Xi'an 710068,China

Abstract The frequently used functionalities usually constitute a small portion of applications' functionalities. The redundant code for rarely used functionalities raises the attack surface of the applications,thus causing the potential risk of code reuse attacks. Binary program debloating can identify and remove the redundant code based on the binary analysis of the application,so as to reduce the attack surface. The state-of-the-art binary program debloating approach relies on artificially crafted inputs to derive the initial control flows. It uses heuristics to extend the binary control-flow graph for debloating. Such an approach has limited robustness and scalability. This paper proposes and implements a robust binary program debloating approach(RBdeb). It uses black-box fuzzing to derive highly-robust valid execution traces of the binary,and categorizes similar library functions automatically based on the graph isomorphism algorithm. The proposed path discovery algorithm extends the binary control flows with the classified library function calls from the control-flow sub-graph of the initial execution traces and generates the robust binary file as the debloating result. Experimental results demonstrate that RBdeb has higher path coverage and debloated binary robustness than the state-of-the-art approaches. The path discovery algorithm and library function categorization are more scalable. RBdeb can effectively debloat large real-world applications.

Keywords Program debloating,Binary analysis,Fuzzing,Binary rewriting,Program analysis

1 引言

应用程序功能愈渐复杂,但用户常用功能仅占小部分。研究表明[1],firefox的典型使用仅执行其所包含代码的不到30%,而JVM的典型运行过程仅有30%的函数和32%的代码被执行。应用程序的可执行程序中的冗余代码不仅会带来不必要的运行时开销,还具有多方面危害。一方面,大量软件漏洞源自应用程序的不常用代码,如OpenSSL心脏滴血漏洞

到稿日期:2023-07-03 返修日期:2023-11-14
基金项目:国家自然科学基金(62272366);陕西省重点研发计划(2023-YBGY-371)
This work was supported by the National Natural Science Foundation of China(62272366) and Key Research and Development Program of Shaanxi Province(2023-YBGY-371).
通信作者:孙聪(suncong@xidian.edu.cn)

的 memcpy()前边界检查缺失就出自用户不常用的库功能[2]，因此冗余代码蕴含不必要的漏洞利用；另一方面，冗余代码给代码重用攻击提供了更大的代码基和攻击面[3]。

程序裁剪技术通过程序分析识别应用程序中的冗余代码，通过编译技术、程序重写技术或定制装载技术删除内存中的应用程序冗余代码，减少潜在程序漏洞并缩小攻击面。健壮性和安全性是评估程序裁剪方案的关键指标，但二者在被裁剪程序上往往存在此消彼长的关系[4]。基于特定输入的程序裁剪必然使得裁剪结果程序倾向于与这些输入相关的程序功能，从而无法保证裁剪后所有程序功能完备，但实现健壮性与安全性之间的相对平衡仍具有重要实用价值。

现有针对程序源代码的裁剪方案[5-11]无法裁剪闭源软件的冗余代码，其裁剪过程还需处理与编译器的冲突或需要定制编译过程，且领域特定的编译器还会在编译过程中引入额外的攻击面，对二进制程序进行裁剪可避免以上问题。典型的二进制裁剪方案已被应用于领域特定的软件[12-14]、第三方库[15-17]和通用二进制软件[18-20]中。相比针对领域特定软件和第三方库的裁剪，通用二进制的裁剪在避免漏洞利用方面更具灵活性优势。相比固件裁剪方法[19-20]，Razor[18]适用于通用系统程序而不受硬件约束。Razor 首先使用构造的输入样本集运行待裁剪程序，跟踪输入样本集的执行轨迹生成部分控制流图，然后使用启发式方法识别相关路径加入控制流图，最后根据扩展的控制流图生成裁剪后的二进制代码，替换原二进制代码节。Razor 的裁剪方法减少了程序二进制代码总量和潜在漏洞数量，缩小了应用程序攻击面，但存在以下问题：

1)人工构造输入样本集，导致输入并非路径覆盖导向，收集到的执行轨迹数量少，覆盖率低，会影响最终生成的可执行文件的健壮性。

2)启发式的控制流图扩展，限制了相关路径识别和控制流图的扩展能力，且人工划分相似库函数类的方式仅划分了 libc，libm 等系统库，无法分析第三方库，影响了裁剪方法的可扩展性和结果的健壮性。

为了提高裁剪方法可扩展性和裁剪结果的健壮性，本文提出了一种高健壮性二进制应用程序裁剪方法(Robust Binary-level Debloating，RBdeb)。本文采用模糊测试技术构建具有高路径覆盖率的输入样本集合，并收集对应的样本执行轨迹集合；使用 VF2 图同构算法自动分析库函数二进制控制流图之间的相似性，将控制流图同构的库函数划分为同类函数，用于路径发现过程；提出了一种新的路径发现算法，从样本执行轨迹集合出发，结合库函数分类结果，识别并加入相关路径，生成符合裁剪要求的控制流子图；最后基于现有二进制代码重写技术生成裁剪后二进制程序。本文的主要贡献包括：

1)基于模糊测试的样本执行轨迹收集方法提高了输入样本集合的路径覆盖率，使得 RBdeb 相比现有方法，裁剪结果二进制程序的健壮性更高；

2)基于图同构算法的自动库函数分类及相应的路径发现算法具有更强的扩展性，提升了裁剪过程对二进制控制流图的扩展能力；

3)对公开样本集的裁剪结果分析表明，RBdeb 在提升裁剪后二进制健壮性的同时，能达到与现有方法相当的安全性。对 SQLite，Nginx，FFmpeg 和 httpd 等实际应用程序的有效裁剪说明了方案的适用性。

2　相关工作

现有程序裁剪技术可针对程序源代码或程序二进制进行。

对于通用 C/C++程序，基于源代码的程序裁剪技术在减小代码规模的同时，并不能保证程序运行时行为的保持：如基于 Delta-debugging 算法[21]的 C-Reduce[5]和 Chisel[6]能够分析目标程序的缺陷和安全漏洞并据此减小代码规模，Chisel 还进一步使用强化学习模型加速源代码的裁剪过程，但此类方法不保证裁剪后源代码可编译。DomGad[7]在裁剪源代码的同时能够保证程序输入子域上的行为正确性，但对子域外输入拒绝执行。TRIMMER[8]根据用户配置确定软件的使用上下文，通过过程间常量传播分析，识别并删除冗余代码。Piece-Wise[9]定制的装载模块根据其编译过程生成的存在于 ELF 文件中的程序依赖图，动态按需装载程序运行所需的二进制代码，减小库攻击面。与 Piece-Wise 定制装载模块不同，BlankIt[10]使用动态二进制插桩，依据编译阶段为每个库函数调用点生成的决策树预测器的预测结果，按需装载库代码。Ancile[11]在源码级利用覆盖导向的模糊测试获得动态控制流图和待保留功能，并利用定制的编译过程结合动态控制流图实现对源码的裁剪。

典型的二进制裁剪方案已被应用于领域特定的软件(如浏览器[12]、蓝牙软件栈[13]、地址清洗器[14]等)、第三方库[15-17]和通用二进制软件[18-20]。对第三方库的裁剪需求源于动态库中大量代码的使用率远低于用户代码。Nibbler[15]反汇编待裁剪的库二进制，得到库函数调用图，然后删除调用图中未执行函数的二进制代码，再用二进制重写生成裁剪后的库文件。BinTrimmer[16]使用抽象解释技术，通过定义符号无关的跨距间隔(SASI)域实现准确的值集分析，得出用于指导二进制裁剪的准确控制流图，实现输入无关的库裁剪。μTrimmer[17]使用输入无关的过程间控制流图，保守估计可能的库依赖，通过陷入指令覆盖无关基本块，实现二进制裁剪。在通用二进制裁剪方面，Razor[18]以构造的输入样本集运行待裁剪程序，跟踪输入样本得到的执行轨迹，生成部分控制流图，使用启发式方法扩展控制流图，根据扩展控制流图生成裁剪后二进制替换原二进制。DECAF[19]使用动态迭代的爬山优化方法，以输入相关的方式逐步裁剪 UEFI 固件的冗余模块；IRQDebloat[20]利用目标固件中与具体外围设备对应的中断处理代码，重写目标固件以删除特定的硬件特性。二进制裁剪与二进制翻译过程中的冗余指令优化[22]的应用场景和目标完全不同，前者缩小二进制文件的整个代码节以减小攻击面，后者则通过优化指令序列中的冗余指令来提高二进制执行的效率。

现有方法常利用模糊测试手段检验裁剪后的应用程序的可用性，而裁剪方案本身使用了模糊测试技术的工作目前仅有 Ancile[11]和 IRQDebloat[20]。Ancile 的模糊测试基于源码，而 IRQDebloat 通过对内存映射 I/O(MMIO)寄存器的模糊

测试找到中断与执行轨迹之间的对应关系。本文利用二进制模糊测试构建输入样本集合的方法与上述用法均不同。

3 高健壮性二进制应用程序裁剪方案

RBdeb 二进制应用程序裁剪方案的流程如图 1 所示，其中虚线表示逻辑等价。具体地：1)基于一个初始种子集，使用二进制黑盒模糊测试生成并精简得出用于执行轨迹收集的输入样本集合；2)使用输入样本集合运行待裁剪程序，基于动态二进制插桩收集输入样本集合对应的执行轨迹集合；3)对二进制使用的库函数根据相似性进行分类，划分出相似库函数集合；4)使用路径发现算法，结合相似库函数分类，扩展执行轨迹集合对应的控制流子图，得到最终控制流子图；5)基于 BinCFI 的二进制重写机制，生成与最终控制流子图对应的裁剪后应用程序可执行文件。

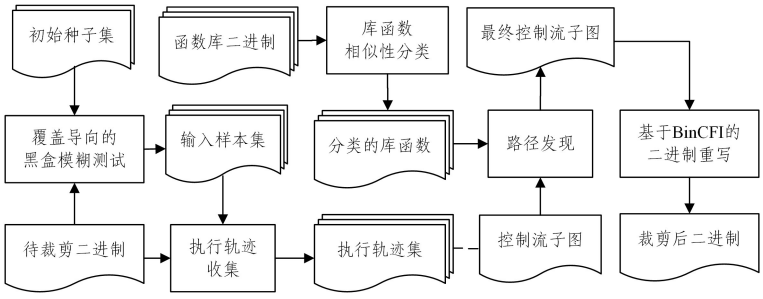


图 1 RBdeb 裁剪方案流程图

Fig. 1 Workflow of RBdeb debloating approach

3.1 构建输入样本集合

模糊测试变异算法构造的畸形输入可以发现常规输入难以触发的执行路径。利用模糊测试构造输入样本集合，能够使后续轨迹收集过程收集到更多执行路径，间接提高裁剪得到的应用程序的健壮性。构建输入样本集合的具体流程为：

1)初始种子集合获取。首先确定待裁剪程序是否存在公开可用的测试用例集合。若存在，则直接将该集合作为模糊测试的初始种子集合；若不存在，则采用测试用例生成框架(boofuzz[23]和 FormatFuzzer[24])，构建框架接受的特定文件或消息格式(包括 MP3、AVI、ZIP、HTTP 请求等)的生成规则，生成初始种子集合。

2)对于待裁剪程序，在初始种子集合上使用 AFL-QEMU[25]进行模糊测试，模糊测试过程记录触发新路径覆盖的合法变异输入，结合初始种子输入，产生输入样本集合。对于依赖程序参数的二进制(如 GNU coreutils)，通过二进制重写方法自动向二进制中引入参数解析函数及解析处理；对输入为 socket 请求的程序，预装载 preeny 库[26]，构造对服务器软件或网络程序的输入，将 socket 请求转化为标准输入输出，以加快收集触发新执行路径的合法输入。模糊测试过程对于目标程序异常中断的处理依赖于 AFL-QEMU 的内建机制。

3)为确保轨迹收集的效率，使用 afl-cmin 对模糊测试得到的触发新执行路径的输入集合进行精简，精简后的输入集合作为输入样本集合，用于执行轨迹收集过程。

3.2 执行轨迹收集

本文采用基于 DynamoRIO[27]的动态二进制插桩收集输入样本集合对应的执行轨迹集合。具体地，使用输入样本集合的每个输入样本依次运行插桩的待裁剪程序；跟踪记录每个输入样本在执行过程中的控制流信息，包括覆盖到的二进制基本块的起始与结束地址，直接、间接跳转边和调用边，以及跳转、调用的次数；将收集到的每个输入样本的控制流信息合并，合并结果即输入样本集合对应的执行轨迹集合，代表一个控制流子图。

3.3 基于图同构的库函数分类

本文程序裁剪方法需要对收集到的执行轨迹对应的控制流子图进行扩展，以保证裁剪结果的健壮性。而扩展的重要原则之一，是针对同类库函数使用的扩展，即如果当前控制流子图已使用某库函数，则应将不在当前控制流子图中的对同类库函数的使用扩展到控制流子图中。因此，需要预先对程序使用的库函数进行分类，得到一系列相似库函数集。

分类方法不依赖库函数源代码，从库的二进制文件出发，遍历库中的函数，判断函数相似性。为避免遍历顺序不同造成的分类结果不同，函数相似关系需满足，对于来自同一个库的函数 A，B，C：

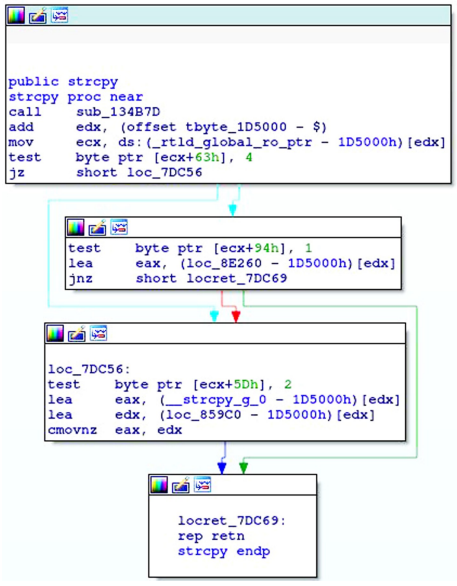
1)可交换性：若函数 A 与函数 B 相似，则函数 B 与函数 A 相似；

2)可传递性：若函数 A 与函数 B 相似，函数 B 与函数 C 相似，则函数 A 与函数 C 相似。

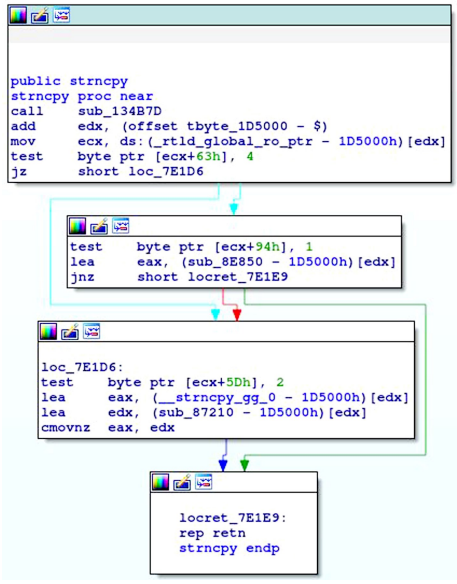
为了满足相似关系的可交换性与可传递性，并使库函数分类更准确，本文采用图同构关系判断库函数控制流图所对应的无向图之间的相似性。图同构关系满足以上交换性和传递性。如果一个库中两个函数的控制流图对应的无向图具有同构关系，则认为这两个函数的控制流图相似。由于库函数实现风格相对一致，这种无向图近似在提高了识别性能的同时，不会引入显著的误匹配，且相比 Razor 的人工库函数匹配可扩展性更强。以 Glibc 标准库函数 *strcpy*()和 *strncpy*()为例，IDA PRO 反汇编得到的控制流图分别如图 2(a)和图 2(b)所示，两个库函数控制流图的无向图同构，可认为两个函数控制流图相似，将它们归入同一库函数集。

本文使用 VF2 算法[28]判断给定的两个无向图是否同构，进一步使用算法 1 自动分类库函数。假定库 p 的库函数集 $F_p=\{f_1,f_2,\cdots,f_n\}$，对任一库函数 f_i 静态反汇编得到入口地址为 s_i 的无向控制流图结构 g_i，以由(s_i，g_i)键值对组成的未分类函数列表 U_p 作为输入，该算法可对 Glibc 库和任意第三方库进行函数分类，得到分类结果$\mathbb{C}_p=\{C_1,\cdots,C_m\}$。算法 1 的时间复杂度为 $O(M\cdot N^2)$，其中 N 为单个库函数平均

基本块数量，M 为库函数数量。基于库函数的强模块化特点，仅少量库函数基本块较多，因而实际复杂度取决于少量复杂库函数的基本块数量及库函数数量。



(a) CFG of *strcpy*()



(b) CFG of *strncpy*()

图 2　*strcpy*()函数和 *strncpy*()函数的控制流图

Fig. 2　Control-flow graphs of function *strcpy*() and *strncpy*()

算法 1　基于无向图同构的库函数分类算法

输入：未分类库函数列表 $U_p=\{(s_1,g_1),\cdots,(s_n,g_n)\}$

输出：库函数分类结果 $\mathbb{C}_p=\{C_1,\cdots,C_m\}$

1. $\mathbb{C}_p\leftarrow\varnothing$;
2. while $U_p\neq\varnothing$ do
3.　从 U_p 中移除(s,g);
4.　$C_s\leftarrow\{s\}$;
5.　foreach (s',g') in U_p do
6.　　if VF2(g,g') then
7.　　从 U_p 中移除(s',g');
8.　　$C_s\leftarrow C_s\cup\{s'\}$;
9.　　end if
10.　end for
11.　$\mathbb{C}_p\leftarrow\mathbb{C}_p\cup\{C_s\}$
12. end while

3.4　路径发现算法

由于模糊测试变异算法具有随机性，因此，构建出的输入样本集合的执行仅覆盖待裁剪程序的部分执行路径。通常还存在一些未被样本执行轨迹覆盖到的执行路径，其二进制代码在程序的典型运行中不可缺少。待裁剪程序中包含的这类执行路径被称为相关路径。需要使用路径发现算法，向执行轨迹收集过程得到的控制流子图中加入相关路径。

本节提出了一种路径发现算法用于对控制流子图进行拓展，如算法 2 所示。算法 2 的输入执行轨迹集合 $subG_p$ 是第 2.2 节收集的轨迹集合的后向展开集，对于执行轨迹集合中的每一条执行轨迹，以轨迹 $tr=b_1\rightarrow b_2\rightarrow b_3\rightarrow b_4$ 为例，后向展开集包含 3 条轨迹：$\{tr,b_2\rightarrow b_3\rightarrow b_4,b_3\rightarrow b_4\}$，其中 b_1-b_4 为基本块。

在算法 2 中，pathGen 和 subGen 过程交互递归，最终由 pathGen 返回路径扩展后的控制流子图。初始情况下，控制流子图扩展过程 pathGen 以二进制的入口基本块 e 和执行轨迹集合后向展开集 $subG_p$ 作为输入。在交互递归过程中，pathGen 从当前作为入口的基本块调用 subGen 沿控制流向前查找，subGen 用于向前找到那些尚未保存在当前执行轨迹展开集中的条件跳转目标，对这些条件跳转目标递归应用 pathGen 算法。如果当前入口基本块不在当前执行轨迹集合 $subG_p$ 中，则直接对 $subG_p$ 做一步后向扩展。

算法 2　二进制 p 的路径发现算法 pathGen

输入：二进制的入口基本块 e，执行轨迹集合后向展开集 $subG_p=\{tr_1,\cdots,tr_n\}$

输出：路径发现后的控制流子图 $subG_p^{final}$

1. procedure pathGen(e,$subG_p$)
2. for each $tr_i\in subG_p$ do
3.　subGen(entry(tr_i),$subG_p$,tr_i);
4.　if e 不是任一 $tr\in subG_p$ 的起始 then
5.　　获得 e 的控制流直接后继集合 T_e;
6.　　foreach $t\in T_e$ do
7.　　if (e,t)为跳转 $\wedge$ $t\in tr_i$ then
8.　　$subG_p\leftarrow subG_p\cup\{(e,t)\}$;
9.　　else if e 以 call(f_i)结尾 $\wedge$ 库函数 f_i 的同类库函数已在 $subG_p$ 中使用 then
10.　　$subG_p\leftarrow subG_p\cup\{(e,r_e)\}$;
11.　　else if e 以用户函数调用 call(q)结尾 then
12.　　$subG_p\leftarrow subG_p\cup CFG_q\cup$pathGen($r_e$,$subG_p$);
13.　　end if
14.　end for
15.　end if
16. end for
17. return $subG_p$;
18. end procedure
19. procedure subGen(e,$subG_p$,tr_i)
20. s←e;
21. for each $s\in tr_i$ do

```
22.   s′←next(tr_i,s);
23.   if (s,s′)为无条件跳转或顺序执行 then
24.     s←s′;
25.     continue;
26.   else   //条件跳转、间接跳转、间接调用
27.     T_s←{t|t 为 s 的直接后继∧t≠s′};
28.     if (s,s′)为条件跳转 then
29.       for each t′∈T_s do
30.         if t′∉subG_p then
31.           subG_p←subG_p∪pathGen(t′,subG_p);
32.         end if
33.         subG_p←subG_p∪{(s,t′)};
34.       end for
35.     end if
36.   end if
37. end for
38. end procedure
```

$entry(tr)$返回轨迹 tr 的入口基本块。$next(tr,s)$返回基本块 s 在轨迹 tr 上的直接后继基本块，若 s 为 tr 的末尾节点，则返回 s 本身。r_e 为由基本块 e 末尾的 *call* 指令调用的函数返回 e 所在函数的目标基本块。当路径发现过程中出现对用户函数的间接调用时，预先使用 BinPointer[29-30] 进行调用目标发现，将潜在被调用函数对应的控制流子图都加入 $subG_p$，间接调用目标发现过程独立于二进制裁剪过程，此处不再赘述。

由于算法 2 本质上是将新的控制流边加入 $subG_p$，当 $subG_p$ 稳定后即终止交互递归。$subG_p$ 的上限是整个二进制控制流图的路径后向展开集，因而算法 2 的时间复杂度为 $O(E^2)$，其中 E 为二进制控制流图的边数量。算法使用的控制流子图操作均基于二进制反汇编框架 Capstone 的反汇编结果，利用通用的图操作方式实现。

以上路径发现算法的特点是，对于由条件跳转触发的路径扩展，其后续扩展或是用户函数对应的控制流子图，或是已使用过的库函数的同类函数调用。而对于在扩展上下文中的进一步跳转，要求收敛回到现有执行轨迹集合，从而限制路径扩展范围，达到兼顾裁剪健壮性和安全性的目的。该算法相比 Razor 的启发式导向的路径推断方法，能够达到更高的二进制代码覆盖率和健壮性(见 4.2 节)，更适用于本文生成的执行轨迹集合和库函数集合。

本文路径发现算法可将图 3 中未执行的相关路径加入控制流子图。图 3 中，实线表示第 2.2 节获得的执行轨迹，虚线为未执行过的路径，S 表示用户函数，a 和 b 表示同类库函数。根据算法 2 的第 26－35 行，无需进入 30－31 行的 if 分支即可将边 $L1\rightarrow L5$ 作为相关路径加入 $subG_p$。对于相关路径 $L2\rightarrow L3\rightarrow L4$，先通过 30－31 行的 if 分支递归进入 pathGen，然后通过第 11－12 行分支将 *call* 指令调用的函数 S 的控制流子图和后续返回路径加入 $subG_p$，而其中第 12 行的 pathGen 又利用第 7－8 行将 $L3\rightarrow L4$ 的无条件跳转加入 $subG_p$。对于相关路径 $L5\rightarrow L6\rightarrow L7$，通过第 31 行递归进入 pathGen，通过第 9－10 行将代表对库函数 b 调用的一条抽象边加入 $subG_p$，这样添加是因为与 b 同类的库函数 a 已经在现有执行轨迹的 $L7$ 中使用过。

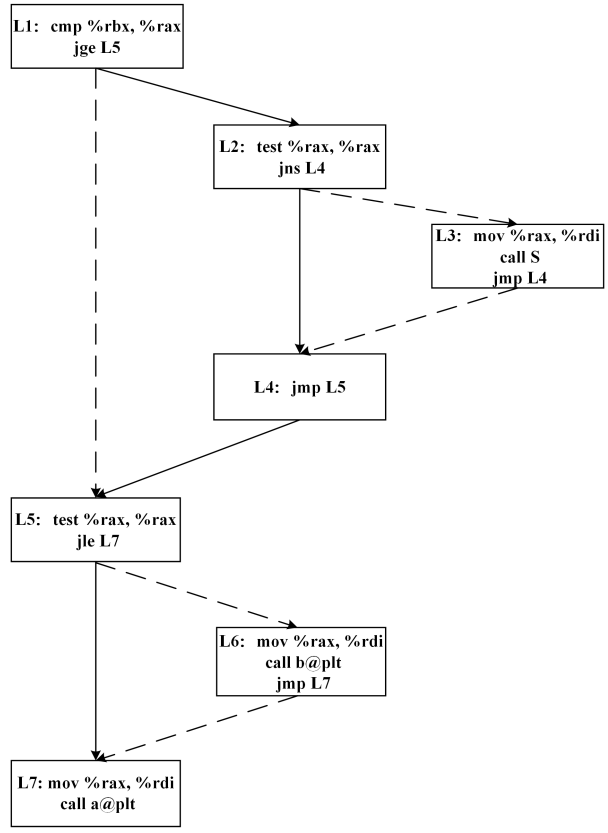


图 3 相关路径发现示例

Fig. 3 Example of related path discovery

3.5 生成裁剪后的程序

为使裁剪后二进制文件能够运行，本文沿用 binCFI[31] 的二进制代码重写方法，用经过路径发现算法扩展的最终控制流子图 $subG_p^{final}$ 构造裁剪后二进制。首先，根据扩展后的控制流子图，对待裁剪的二进制进行反汇编。根据 $subG_p^{final}$ 中每个基本块的起始地址和结束地址，找到待裁剪的二进制 text 节中相应的基本块，生成包含这些基本块汇编代码指令的伪汇编代码。其次，用 binCFI 将伪汇编代码修改为有效汇编代码。将伪汇编代码的基本块进行符号化(跳转目标地址/偏移量标签化)，对于条件跳转指令实际执行无条件跳转的情况，将不可能被执行到的分支目标设置为预先插入的错误处理代码的起始地址。第三，从符号化后的有效汇编代码出发，生成机器码，作为裁剪后的二进制代码。设置二进制文件的原代码节(.text 节)为不可执行，在原始二进制中新增一个代码节，将生成的机器码复制到新代码节中并设置其执行权限，将该节作为裁剪后二进制程序的实际代码节，得到裁剪后的应用程序可执行文件。

4 实验与结果分析

将本文 RBdeb 与现有的 Razor 二进制代码裁剪方案进行对比实验。首先，从路径覆盖率、代码裁剪量、裁剪结果健壮性 3 方面说明 RBdeb 的裁剪健壮性；其次，分析说明 RBdeb 能够达到与 Razor 相当的安全性，说明 RBdeb 的健壮性

改进未造成安全性下降；再次，分析 RBdeb 的裁剪结果运行时开销，说明运行时开销增长不影响裁剪结果的可用性；然后，分析 RBdeb 对现实世界应用程序 SQLite，Nginx，FFmpeg，httpd 的裁剪效果，验证方案的适用性；最后，分析 RBdeb 的库函数分类方法的可扩展性和分类粒度，并评估库函数分类方法对裁剪结果健壮性的影响。

4.1　实验环境与数据集

本文实验环境硬件为：Intel(R) Core(TM) i5-7400 CPU@3.00GHz，8G RAM，软件环境为 Ubuntu18.04 64 位操作系统，gcc 4.8.3/g++4.8.5，glibc 2.27，make 4.1。本文收集了 4 个数据集，用于对 RBdeb 裁剪方案进行实验评估，如表 1 所列。数据集 S1 为 SPEC CPU 2006 基准测试集中的 C/C++ 程序；数据集 S2 为 Razor 采用的基准数据集；数据集 S3 为现实世界应用程序，包括 SQLite 3.36.0，Ngnix 1.21.6，FFmpeg（版本号为 N-105374=gb2421c4f26），Apache httpd 2.4.7；数据集 S4 为共享库文件。对于数据集中的每个程序，采用 gcc 编译的 x64 ELF 可执行文件作为裁剪对象，编译优化选项为 -O2。

表 1　数据集信息

Table 1　Datasets information

数据集	样本数量	样本来源	样本用途
S1	10	SPEC CPU 2006	健壮性分析，性能分析
S2	10	GNU coreutils	健壮性分析，安全性分析，库函数调用点分析
S3	4	实际应用	适用性测试
S4	11	共享库文件	库函数分类有效性分析

4.2　健壮性分析

本节使用数据集 S1 和 S2 作为基准程序集，对其中基准程序的可执行文件，分别使用本文提出的 RBdeb 和现有二进制裁剪框架 Razor 进行裁剪，得到裁剪后的基准程序。在数据集 S1 和 S2 上，首先对比 RBdeb 与 Razor 的输入样本集路径覆盖率，验证 RBdeb 基于模糊测试构造输入样本集合的效果；然后评估 RBdeb 与 Razor 的代码裁剪量差异；最后对比 RBdeb 和 Razor 的裁剪结果二进制程序的健壮性，通过对具体裁剪结果的分析说明本文路径发现算法的优势。

1）路径覆盖率

对于数据集 S1 中的程序，RBdeb 采用 SPEC CPU 2006 为基准程序提供的标准测试用例集合作为初始种子集合，Razor 则采用该测试用例集合作为输入样本集合；在数据集 S2 上，本文为每个基准程序收集包含 50 个测试用例的测试用例集合，作为 RBdeb 的初始种子集合和 Razor 的输入样本集合。

首先使用 RBdeb 构造输入样本集，然后使用基于 DynamoRIO 的轨迹收集方法分别收集 RBdeb 输入样本集和 Razor 输入样本集的执行轨迹。统计并对比两个输入样本集各自的执行轨迹所覆盖的基本块数量，如图 4 所示。前 10 个用例为 S1 的用例，后 10 个用例为 S2 的用例。结果表明，与 Razor 的输入样本集相比，RBdeb 使用模糊测试构造的输入样本集能够覆盖更多的基本块，反映出 RBdeb 的输入样本集的路径覆盖率更高。

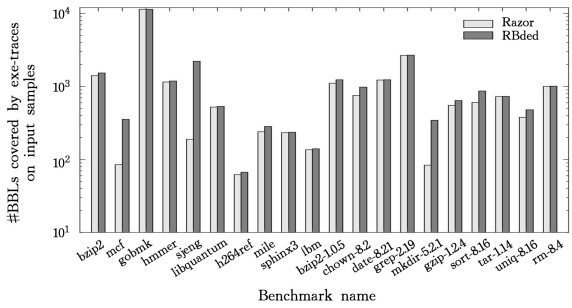


图 4　输入样本集路径覆盖率比较

Fig. 4　Path coverage comparison of input sample set

2）代码裁剪量

路径覆盖率差异反映了 RBdeb 对执行轨迹收集的结果（算法 2 的输入集合）与 Razor 方案的效果差异。执行轨迹收集的差异并不一定能反映最终的裁剪结果，因为 RBdeb 与 Razor 使用不同的路径发现方法对执行轨迹集合进行扩展。因此，我们进一步比较 RBdeb 与 Razor 的代码裁剪量差异。

由于 RBdeb 的二进制代码重写模块使用 binCFI 将原有 text 节（原始二进制代码）替换为新代码节（裁剪后二进制代码），因此，裁剪量计算方法为：

$$\text{裁剪量}=\frac{\text{原始 text 节大小}-\text{新代码节大小}}{\text{原始 text 节大小}}\times 100\%$$

裁剪方案的裁剪量越大，说明裁剪方案的裁剪能力越强。图 5 为 RBdeb 和 Razor 的裁剪量对比。总体而言，RBdeb 裁剪掉的原始二进制代码量小于 Razor 裁剪掉的代码量。这是由于 RBdeb 构造的输入样本集合以及路径扩展算法使得其得出的有效路径覆盖率比由 Razor 得出的有效路径覆盖率更高。本文进一步探讨路径覆盖率提升和裁剪量的减少是否有助于提高裁剪结果二进制的健壮性。

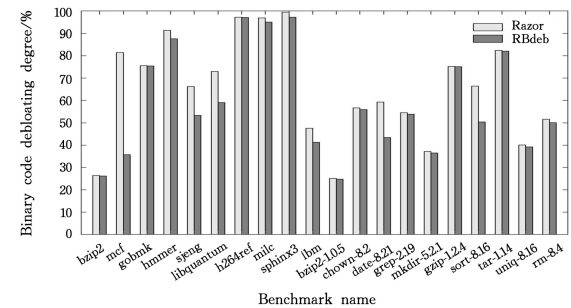


图 5　二进制代码裁剪量比较

Fig. 5　Comparison of binary code debloating degree

3）裁剪结果健壮性

为进一步说明更高的路径覆盖率对裁剪后程序健壮性的提升效果，进行如下健壮性对比实验。为保证裁剪后二进制健壮性对比的客观性，需要为每个裁剪后的二进制程序构造完全独立于裁剪过程的测试输入集。具体地，对于数据集 S1，从每个基准程序的标准测试用例集中随机选出多个标准输入，人为更改后作为初始种子集合进行输入样本集生成，从模糊测试结果中随机选出 25 个，作为 RBdeb 裁剪后二进制健壮性测试的输入；对于 Razor 的裁剪结果，也选择相同的 25 个输入作为健壮性测试的输入。因为测试输入集的选取过程完全独立于 RBdeb 和 Razor 的裁剪过程，因此在其上的

健壮性测试是有意义的。对于数据集 S2，将每个基准程序的 50 个测试用例随机分为两组，每组 25 个，其中一组用于裁剪二进制，另一组作为测试输入集。对于 RBdeb 和 Razor 分别裁剪后的二进制，使用上述测试输入集运行程序，统计运行过程中发生崩溃的测试用例数量，如表 2 所列。

表 2 裁剪后二进制健壮性比较

Table 2 Robustness comparison of binaries after debloating

数据集	程序名称	测试用例崩溃的数量	
		RBdeb	Razor
S1	bzip2	6	6
	mcf	0	25
	hmmer	3	25
	sjeng	0	8
	h264ref	7	8
	sphinx3	4	4
	lbm	2	3
	Others	0	0
S2	grep-2.19	0	1
	gzip-1.2.4	0	1
	sort-8.16	0	25
	Others	0	0

表 2 简单列出了 RBdeb 和 Razor 在 25 个测试输入上均不会崩溃的测试用例(表 2 中 Others 所示)。表 2 结果表明，在数据集 S1 上，RBdeb 裁剪的二进制的崩溃率明显低于 Razor 裁剪的二进制；在数据集 S2 上，RBdeb 裁剪的二进制发生崩溃的数量为 0。总体而言，RBdeb 导致的崩溃率为 4.4%，相比 Razor 的 21.2%，崩溃率显著降低，证明 RBdeb 的裁剪结果具有高健壮性。

对于 S1 中的 hmmer，mcf 以及 S2 中的 sort-8.16，RBdeb 裁剪的二进制的崩溃数量接近于 0，而 Razor 裁剪的二进制在测试输入上全部崩溃。对于 S1 的 sjeng，崩溃数量也有显著差异。进一步结合图 5 的裁剪量情况可以发现，mcf，sjeng 和 sort-8.16 的裁剪量差异相对比较明显，说明裁剪量的差异显著影响了裁剪后二进制的健壮性；而 hmmer 的裁剪量差异较小，说明存在少量关键基本块的差异，显著影响了程序的健壮性。通过 IDA PRO 逆向分析两种方法对 mcf，sjeng 和 sort-8.16 的裁剪结果发现，与 Razor 裁剪结果相比，RBdeb 的裁剪结果包含的函数调用更多，且函数的控制流图更加完整(控制流图包含的基本块数量更多)，说明本文路径发现算法相比 Razor 更能有效扩展到程序的有效功能。逆向分析两种方法对 hmmer 的裁剪结果可以发现，对于 hmmer 中用于解析输入参数的 getopt 函数，RBdeb 的路径发现算法能有效发现该函数的主要处理流程，而 Razor 的启发式方法仅保留了该函数的 3 个基本块，其他基本块均被裁剪，这是两种裁剪方法在 hmmer 上崩溃数量呈现差异的原因。

4.3 安全性分析

RBdeb 在获得高健壮性裁剪结果的同时，总体上也保留了原始程序中更多的代码，因而存在增大潜在攻击面的可能。因此，本节进一步分析代码裁剪量的减少是否会导致 RBdeb 裁剪结果的安全性下降。

本节采用数据集 S2 中的程序验证 RBdeb 裁剪结果的安全性。首先收集与 S2 中程序的各版本对应的典型安全漏洞，如表 3 所列。然后分别使用 RBdeb 和 Razor 对基准程序进行裁剪，使用 IDA PRO 逆向分析裁剪结果程序和原始程序，比较二者的 text 节，分析裁剪结果二进制程序是否仍存在安全漏洞，从而衡量裁剪方案的安全性。若裁剪结果的 text 节中不包含原始程序被漏洞利用的代码，则说明裁剪方案裁剪掉了原始程序的安全漏洞，提升了程序安全性(表 3 中 Yes)。

表 3 基准程序裁剪对漏洞的缓解效果

Table 3 Vulnerability mitigation of benchmark program dedebloating

程序名称	CVE	裁剪结果是否避免 CVE 漏洞	
		RBdeb	Razor
bzip2-1.0.5	CVE-2010-0405	No	No
chown-8.2	CVE-2017-18018	Yes	Yes
date-8.21	CVE-2014-9471	No	No
grep-2.19	CVE-2015-1345	Yes	Yes
gzip-1.2.4	CVE-2005-1228	Yes	Yes
	CVE-2009-2624	Yes	Yes
	CVE-2010-0001	Yes	Yes
tar-1.14	CVE-2016-6321	No	No

在 gzip-1.2.4 程序中，CVE-2010-0001 涉及 *unlzw* 函数，该函数存在整数下溢漏洞，CVE-2009-2624 涉及 *huft_build* 函数，该函数存在输入验证错误，RBdeb 裁剪结果中并不包含这两个函数。CVE-2005-1228 的触发条件是使用 gzip 的-N 选项，RBdeb 裁剪结果能够裁剪掉该选项相关的路径。grep-2.19 程序的 CVE-2015-1345 涉及 *bmexec_trans* 函数，攻击者可以利用该函数，构造特定输入造成拒绝服务，RBdeb 裁剪结果中并不包含该函数。chown-8.2 程序的 CVE-2017-18018 漏洞使得使用 POSIX-R-L 选项时触发竞争条件，RBdeb 裁剪结果能够避免程序进入-L 选项对应的执行路径，因而可以避免此漏洞。Razor 的裁剪结果也能够达到类似的裁剪效果。

在 bzip2-1.0.5 程序中，CVE-2010-0405 涉及 *BZ2_decompress* 函数，该函数存在整数溢出漏洞；在 date-8.21 程序中，CVE-2014-9471 中涉及 *parse_datetime* 函数，攻击者可以通过构造特殊输入的方式利用该函数造成攻击；在 tar-1.14 程序中，CVE-2016-6321 涉及 *safer_name_suffix* 函数，攻击者可利用该漏洞绕过安全机制进行攻击。RBdeb 和 Razor 的裁剪结果程序中，均含有上述函数，虽然在裁剪结果中上述函数的函数体可能被裁剪掉一部分路径(如 RBdeb 对 *safer_name_suffix* 函数体的裁剪)，但仍然存在脆弱性。

综合表 3 中裁剪结果的安全性，可见 RBdeb 的裁剪结果的安全性与 Razor 相当，均能删除原始程序中的相同脆弱性函数或选项，使二进制程序的安全性得到提升，进而说明 RBdeb 相比 Razor 的代码裁剪量减少并未导致 RBdeb 裁剪结果安全性下降。

4.4 性能分析

本节采用数据集 S1 对 RBdeb 和 Razor 裁剪结果程序的运行时开销进行比较，统计每个程序运行多个测试用例的平均时间开销，如表 4 所列。Razor 裁剪得到的 hmmer 和 mcf 程序均不能运行，因此无相应实验结果。由于 RBdeb 的二进制代码重写过程使用了 binCFI 的二进制代码重写模块，而该模块引入了额外的运行时跳转目标检查，因此裁剪结果相比原始程序运行时性能有所降低。表 4 结果显示，RBdeb 和

Razor 的裁剪均给二进制程序带来了性能开销，但开销增长及增长差异均不明显，运行时开销增长不影响裁剪结果二进制的可用性。

表 4　数据集 S1 上的性能测试结果

Table 4　Performance tests on dataset S1

(s)

程序名称	原始程序运行时间	裁剪结果运行时间	
		RBdeb	Razor
bzip2	4.23	5.81	5.70
mcf	1.86	1.88	—
gobmk	13.50	13.80	13.40
hmmer	1.67	1.68	—
sjeng	2.86	3.15	3.06
libquantum	0.04	0.05	0.05
h264ref	8.83	8.84	8.86
milc	4.85	4.95	4.90
sphinx3	1.08	1.10	1.09
lbm	1.93	2.01	1.97

4.5　实际应用程序测试

为评估 RBdeb 的适用性，使用 RBdeb 裁剪数据集 S3 中的现实世界应用程序。首先，获取 SQLite，FFmpeg，Nginx 和 httpd 的源代码，在本文实验环境下使用 gcc 4.8.3 编译生成可执行文件。然后，根据本文的初始种子生成方法，生成 25 个 SQL 文件、MP3 文件和 HTTP 请求，分别作为 SQLite，FFmpeg，Nginx 和 httpd 的初始种子集合。使用 RBdeb 从初始种子集合构造输入样本集。从输入样本集中选择 25 个输入样本作为测试输入，选择标准是在避开初始种子的前提下随机选择。从输入样本集中排除掉 25 个测试输入，用剩余输入样本对原始程序进行裁剪，得到裁剪后的二进制程序。最后，用 25 个测试输入验证裁剪结果二进制程序的可运行性，确认裁剪结果可运行。裁剪结果的裁剪量如表 5 所列。可以看到，对于现实应用程序，仅构造 25 个初始种子触发裁剪显然无法覆盖应用程序的大部分功能，实验结果仅能说明 RBdeb 在程序的特定功能侧面上的裁剪结果的可用性。正因如此，实验结果的裁剪量更大。

表 5　RBdeb 对现实程序的裁剪量

Table 5　RBdeb's debloating degree to real-world programs

程序名称	裁剪量/%
FFmpeg N-105374	95.8
SQLite 3.36.0	90.8
Nginx 1.21.6	88.9
Apache httpd 2.4.7	87.6

4.6　库函数分类方法可扩展性和有效性分析

本节在数据集 S4 上比较 RBdeb 的库函数分类方法与 Razor 的人工库函数分类方法的差异，从而验证本文库函数分类方法的可扩展性和有效性。Razor 通过人工分类方法对 Glibc 的库函数进行分类，分类的依据是文献[32]的公开结果，且 Razor 仅分析了 Glibc 库函数。由于其他库缺乏类似文献[32]的分类结果，因而分类方法无法扩展到其他库。

RBdeb 的自动化库函数分类方法不仅可分类 Glibc 库函数，还能分类其他库函数。使用 RBdeb 的库函数分类方法对 S4 中共享库文件的函数进行分类，在分类前先统计每个共享库所包含的函数总数，在分类后统计该库的函数类别数量，分类效果如表 6 所列。结果表明，RBdeb 的库函数分类方法可对第三方库和系统库的函数进行无差别自动分类，可扩展性强。RBdeb 的库函数分类方法的平均分类粒度为 2.671。在分析 Glibc 函数时，RBdeb 能将 2 806 个函数分为 1 103 个类别，分类粒度为 2.544，而 Razor 的人工分类则将 1 636 个常用 Glibc 函数分为 297 个类别，分类粒度为 5.508。在分析 Glibc 函数时，RBdeb 的库函数方法分类粒度更细。

表 6　库函数分类效果

Table 6　Categorization effect of library functions

共享库	函数总数	类别数量	分类粒度=函数总数/类别数量
libcidn.so	57	24	2.375
libcrypt.so	71	28	2.536
libdl.so	40	13	3.077
libnsl.so	276	94	2.936
librt.so	124	32	3.875
libutil.so	45	12	3.750
libpcre.so	179	140	1.279
libevent-1.4.so.2.2.0	326	135	2.415
libanl.so	47	13	3.615
libc.so.6	2806	1103	2.544
libsidplay2.so.1.0.1	676	146	4.630

为验证库函数分类方法对裁剪结果二进制的影响，对于 RBdeb 和 Razor 裁剪的数据集 S2 中的程序，通过静态分析统计每个裁剪结果二进制使用到的 Glibc 库函数个数，以说明库函数分类对裁剪结果的影响。实验结果如表 7 所列。结果表明，在多数用例上，RBdeb 的裁剪结果二进制保留的有效 Glibc 库函数个数与 Razor 大致相当。对于 sort-8.16，RBdeb 裁剪结果使用的 Glibc 库函数个数明显多于 Razor 的裁剪结果，反汇编发现 RBdeb 的路径发现算法能更有效地将 sort-8.16 所使用的同类函数的调用引入控制流子图。因此虽然 RBdeb 对于 Glibc 库函数的总体分类粒度更小，但对于 sort-8.16 的库函数，RBdeb 的库函数分类方法能保留更多的有效 Glibc 库函数，相比 Razor 的人工分类更准确。根据 3.2 节分析，RBdeb 的裁剪结果 sort-8.16 程序的健壮性亦明显优于 Razor 裁剪结果，验证了 RBdeb 的库函数分类方法对裁剪结果健壮性的正面效果。

表 7　数据集 S1 程序 Glibc 库函数使用个数比较

Table 7　Comparison on the number of the used Glibc functions for dataset S1

程序名称	原始程序的库函数使用个数	裁剪结果库函数使用个数	
		RBdeb	Razor
bzip2-1.0.5	38	38	38
chown-8.2	68	41	41
date-8.21	72	61	63
grep-2.19	119	80	80
mkdir-5.2.1	45	29	29
gzip-1.2.4	47	21	21
sort-8.16	122	99	78
tar-1.14	133	71	72
uniq-8.16	62	58	61
rm-8.4	71	54	56

结束语　本文提出并实现了一种高健壮性的二进制应用程序裁剪方法 RBdeb。RBdeb 的输入样本集合构建过程充分利用了黑盒模糊测试技术的合法输入样本生成能力，产生的

输入样本能覆盖更多的有效执行轨迹；其库函数自动分类算法能有效针对各种第三方库在二进制层面进行相似库函数分类；其路径发现算法在收集到的执行轨迹集合上结合库函数分类结果生成符合裁剪要求的控制流子图和高健壮性二进制文件。实验结果说明了以上 3 方面特性使得 RBdeb 的裁剪结果相比现有二进制裁剪方案 Razor 的裁剪结果具有更高的健壮性，并能达到与 Razor 裁剪结果相当的安全性。虽然本文方案的测试输入从理论上不可能覆盖所有的实际输入，但裁剪后二进制的执行路径集合包含在原始二进制的执行路径集合中，能够保证裁剪结果可靠性，且 RBdeb 能够裁剪 SQLite，FFmpeg，Nginx，httpd 等大规模的实际应用程序，裁剪结果是可以运行的，可以说明方案具有实用性。

RBdeb 沿用了 binCFI 的二进制代码重写模块，该重写模块虽增强了重写后二进制的安全性，但给裁剪结果二进制带来了一定的运行时开销，且无法处理精简二进制和混淆二进制。后续可采用 Egalito[33] 等二进制重写方法，提高二进制生成过程的准确性。此外，RBdeb 仅能裁剪 x86-64 位 ELF 格式文件，未来还可考虑支持更多指令集体系结构的二进制文件裁剪。

参 考 文 献

[1] QUACH A, ERINFOLAMI R, DEMICCO D, et al. A Multi-OS Cross-Layer Study of Bloating in User Programs, Kernel and Managed Execution Environments[C]// Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation. New York: ACM, 2017: 65-70.

[2] WILLIAMS C. Anatomy of OpenSSL's Heartbleed: Just Four Bytes Trigger Horror Bug [EB/OL]. (2014-04-09)[2023-05-13]. https://www.theregister.com/2014/04/09/heartbleed_explained.

[3] PENG G, LIANG Y, ZHANG H, et al. Survey on software binary code reuse technologies [J]. Ruan Jian Xue Bao/Journal of Software, 2017, 28(8): 2026-2045.

[4] XIN Q, ZHANG Q, ORSO A. Studying and Understanding the Tradeoffs Between Generality and Reduction in Software Debloating[C]// Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2022: 99: 1-99: 13.

[5] REGEHR J, CHEN Y, CUOQ P, et al. Test-Case Reduction for C Compiler Bugs[C]// Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2012: 335-346.

[6] HEO K, LEE W, PASHAKHANLOO P, et al. Effective Program Debloating via Reinforcement Learning[C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2018: 380-394.

[7] XIN Q, KIM M, ZHANG Q, et al. Subdomain-based Generality-Aware Debloating[C]// Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2020: 224-236.

[8] SHARIF H, ABUBAKAR M, GEHANI A, et al. TRIMMER: Application Specialization for Code Debloating[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. New York: ACM, 2018: 329-339.

[9] QUACH A, PRAKASH A, YAN L. Debloating Software through Piece-Wise Compilation and Loading[C]// Proceedings of the 27th USENIX Security Symposium. USENIX Association, 2018: 869-886.

[10] PORTER C, MURURU G, BARUA P, et al. BlankIt Library Debloating: Getting What You Want instead of Cutting What You don't[C]// Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2020: 164-180.

[11] BISWAS P, BUROW N, PAYER M. Code Specialization through Dynamic Feature Observation[C]// Proceedings of the 11th ACM Conference on Data and Application Security and Privacy. New York: ACM, 2021: 257-268.

[12] QIAN C, KOO H, OH C, et al. Slimium: Debloating the Chromium Browser with Feature Subsetting[C]// Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2020: 461-476.

[13] WU J, WU R, ANTONIOLI D, et al. LIGHTBLUE: Automatic Profile-Aware Debloating of Bluetooth Stacks[C]// Proceedings of the 30th USENIX Security Symposium. USENIX Association, 2021: 339-356.

[14] ZHANG Y, PANG C, PORTOKALIDIS G, et al. Debloating Address Sanitizer[C]// Proceedings of the 31st USENIX Security Symposium. USENIX Association, 2022: 4345-4363.

[15] AGADAKOS I, JIN D, WILLIAMS-KING D, et al. Nibbler: Debloating Binary Shared Libraries[C]// Proceedings of the 35th Annual Computer Security Applications Conference. New York: ACM, 2019: 70-83.

[16] REDINI N, WANG R, MACHIRY A, et al. BinTrimmer: Towards Static Binary Debloating through Abstract Interpretation [C]// Proceedings of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2019: 482-501.

[17] ZHANG H, REN M, LEI Y, et al. One Size does not Fit All: Security Hardening of MIPS Embedded Systems via Static Binary Debloating for Shared Libraries[C]// Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2022: 255-270.

[18] QIAN C, HU H, ALHARTHI M, et al. RAZOR: A Framework for Post-Deployment Software Debloating[C]// Proceedings of the 28th USENIX Security Symposium. USENIX Association, 2019: 1733-1750.

[19] CHRISTENSEN J, ANGHEL M, TAGLANG R, et al. DECAF: Automatic, Adaptive De-bloating and Hardening of COTS Firmware[C]// Proceedings of the 29th USENIX Security Symposium. USENIX Association, 2020: 1713-1730.

[20] HU Z, DOLAN-GAVITT B. IRQDebloat: Reducing Driver Attack Surface in Embedded Devices[C]// Proceedings of 2022 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2022: 1608-1622.

[21] ZELLER A, HILDEBRANDT R. Simplifying and isolating fai-

lure-inducing input [J]. IEEE Transactions on Software Engineering,2002,28(2):183-200.

[22] TAN J,PANG J,SHAN Z,et al. Redundant instruction optimization algorithm in binary translation [J]. Journal of Computer Research and Development,2017,54(9):1931-1944.

[23] JOSHUA P. boofuzz:Network Protocol Fuzzing for Humans. [EB/OL]. (2020-10-21) [2023-05-13]. https://boofuzz.readthedocs.io/en/stable/.

[24] DUTRA R,GOPINATH R,ZELLER A. Format Fuzzer:Effective Fuzzing of Binary File Formats[J]. arXiv:2109.11277,2021.

[25] The AFL Team. Technical Whitepaper for afl-fuzz. [EB/OL]. https://lcamtuf.coredump.cx/afl/technical_details.txt.

[26] The Preeny Team. Preeny [EB/OL]. (2021-07-04)[2023-05-13]. https://github.com/zardus/preeny.

[27] The DynamoRIO Team. DynamoRIO. [EB/OL]. (2023-06-29)[2023-06-30]. https://dynamorio.org/.

[28] CORDELLA L,FOGGIA P,SANSONE C,et al. An Improved Algorithm for Matching Large Graphs[C]// Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition. 2001:149-159.

[29] KIM S,SUN C,ZENG D,et al. Refining Indirect Call Targets at the Binary Level[C]// Proceedings of Network and Distributed Systems Security. Internet Society,2021.

[30] KIM S,ZENG D,SUN C,et al. BinPointer:Towards Precise,Sound,and Scalable Binary-Level Pointer Analysis[C]// Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. New York:ACM,2022:169-180.

[31] ZHANG M,SEKAR R. Control Flow Integrity for COTS Binaries[C]// Proceedings of the 22nd USENIX Security Symposium. USENIX Association,2013:337-352.

[32] Free Software Foundation,Inc. The GNU C Library- Function and Macro Index. [EB/OL]. https://www.gnu.org/software/libc/manual/html_node/Function-Index.html.

[33] WILLIAMS-KING D,KOBAYASHI H,WILLIAMS-KING K,et al. Egalito:Layout-Agnostic Binary Recompilation[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York:ACM,2020:133-147.

DING Duo, born in 1995, master, assistant engineer. His main research interests include software security and so on.

SUN Cong, born in 1982, Ph.D, professor, Ph.D supervisor, is a member of CCF(No. 28286M). His main research interests include software security, program analysis, and high-confidence software.

(责任编辑:何杨)