

基于 ROP/JOP gadgets 性质的软件多样化评估方法

迟宇宁, 郭云飞, 王亚文, 扈红超

(信息工程大学, 河南 郑州 450001)

摘要: 为应对信息化生活中的网络攻击及威胁, 降低网络系统中同质化攻击快速蔓延的风险, 增强网络和软件的安全性, 软件多样化技术被应用到系统中。软件多样化旨在生成功能等价但内部发生变化的程序变体, 从而改变单一的运行环境, 缓解同质化攻击。现有的多样化技术的评估指标 ROP (return-oriented programming) gadgets 幸存率难以直接体现安全性影响且评估方法单一, 为了更加全面有效地评估软件多样化方法的有效性, 提出基于 ROP/JOP (jump-oriented programming) gadgets 性质的软件多样化评估方法, 通过分析常见的代码重用攻击, 将抽象的量化转为具象的指标, 从空间、时间及质量 3 个方面评估多样化方法的安全增益及效果。该方法根据 gadgets 的相似性、损坏度和可用性 3 个性质探讨软件多样化技术如何影响 ROP/JOP 攻击。用指令替换、NOP 插入、控制流平坦等 9 种多样化方法对 GNU coreutils 程序集进行多样化编译生成多样化程序集。对多样化程序集进行基于 gadgets 性质的实验, 根据实验结果评估不同多样化方法的有效性以及对攻击造成的影响。实验结果表明, 该方法能够对软件多样化方法的安全增益进行准确评估, 多样化技术会导致 ROP/JOP 攻击所需的攻击链空间增大, 构造攻击链的时间变长且攻击成功率降低。不同的多样化方法产生的效果高低不一, 对后续研究具有更高安全增益的多样化技术有指导作用。

关键词: 软件多样化; ROP/JOP 攻击; gadgets 性质; 安全增益评估

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2022086

Software diversity evaluation method based on the properties of ROP/JOP gadgets

CHI Yuning, GUO Yunfei, WANG Yawen, HU Hongchao

Information Engineering University, Zhengzhou 450001, China

Abstract: In order to reduce the risk of rapid spread of homogeneous attacks in network systems, and enhance network and software security, software diversification technologies are applied widely nowadays. Software diversification aims to generate functionally equivalent but internally changed program variants, thereby alter a

收稿日期: 2021-09-28; **修回日期:** 2021-12-28

通信作者: 迟宇宁, cyn_091981@163.com

基金项目: 国家重点研发计划 (2021YFB1006200, 2021YFB1006201); 国家自然科学基金 (62072467)

Foundation Items: The National Key R&D Program of China (2021YFB1006200, 2021YFB1006201), The National Natural Science Foundation of China (62072467)

引用格式: 迟宇宁, 郭云飞, 王亚文, 等. 基于 ROP/JOP gadgets 性质的软件多样化评估方法[J]. 网络与信息安全学报, 2022, 8(6): 135-145.

Citation Format: CHI Y N, GUO Y F, WANG Y W, et al. Software diversity evaluation method based on the properties of ROP/JOP gadgets[J]. Chinese Journal of Network and Information Security, 2022, 8(6): 135-145.

single operating environment and mitigating homogenization attacks. The existing diversified technical evaluation index ROP gadgets survival rate is difficult to directly reflect the safety impact and the evaluation method is single. In order to evaluate the effectiveness of the diversification method more comprehensively and effectively, a software diversification evaluation method based on the properties of ROP/JOP gadgets is proposed, by analyzing common code reuse attacks, and turns abstract quantification into concrete indicators evaluates the security gain and effect of diversified methods from three aspects of space, time and quality. The method first discusses how diversification techniques affect ROP/JOP attacks according to the three properties of gadgets similarity, damage degree and availability. Nine kinds of diversification methods, such as instruction replacement, NOP insertion, and control flow flattening, are used to diversify the GNU coreutils assembly to generate diversification assembly. Experiments based on the property of gadgets are carried out on the diverse assemblies, and the effectiveness of different diversification methods and the impact on attacks are evaluated according to the experimental results. The experimental results show that this method can accurately evaluate the security gain of software diversification methods, the diversification technology will lead to the increase of the attack chain space required by the ROP/JOP attack, the longer time to construct the attack chain and the lower the attack success rate. The effects of different diversification methods are different, it has a guiding role for the follow-up research on diversified technologies with higher safety gains.

Keywords: software diversification, ROP/JOP attack, gadgets properties, safety gain evaluation

0 引言

信息技术及互联网的飞速发展,使得大众生活越来越依赖于网络。种类繁多、功能各异的应用软件为人们提供生活便利的同时,也引入了安全风险,威胁人们的隐私和财产安全。究其原因,是大多数的软件在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,如在内存中限制可执行页面来抵抗大多数代码注入攻击^[1],控制流完整性检测检测程序是否进入非预期的执行路径^[2],动态污点分析追踪信息流识别破坏内存的攻击^[3]。在这些防御方法中,大多数只能针对某一种漏洞进行防御,且容易被攻破^[4]。

软件多样化技术为防御威胁提供了新的思路,其核心思想是应用软件多样化技术动态地改变攻击面以提高攻击成本。软件多样性的概念可以追溯到容错。多样性将一个给定组件的多个实现方法产生错误结果的可能性降到最低。软件多样化着眼于代码布局随机化,可以在不同地方用自动化的方式实现,如通过编译器实现软件多样化^[5]。在加载时对程序进行反汇编,替换 loader

加载器^[6]实现多样化等。现代操作系统通过随机化程序中对象的基址,如堆栈、堆、程序代码和动态库,实现了地址空间布局随机化(ASLR, address space layout randomization)。

大量理论和实践证明软件多样化技术能够有效提高攻击成本,增强软件安全性,但如何对软件多样化技术的有效性进行评估是一个难点。刘镇武等^[7]通过软件复杂程度的静态指标来评估软件多样化技术。Rossman 等^[8]结合香农-维纳信息熵^[9]的角度,通过利用物种多样性量化公式去分析操作系统的多样性,也有依靠逻辑认证^[10]来证明改变攻击依赖的程序属性则表明防御成功。

评估软件多样性技术有效性的一种方法是计算幸存 gadgets 的数量占多样化软件 gadgets 集合的比例。Coffman 等^[11]采用该方法,分别用相同地址的幸存 gadgets 占比和不考虑地址的幸存 gadgets 占比作为指标,对几种多样化技术进行了安全性评估。但是该方法所提到获取的 gadgets 仍足以发动攻击,即幸存 gadgets 数量的占比不能显著体现安全性影响,且幸存 gadgets 占比这个指标只考虑了多样化技术在空间角度对面向返回的编程(ROP, return-oriented programming)的部分影响。因此,本文提出了一种基于常见代码重用攻击(ROP/面向跳转的编程(JOP, jump-oriented

programming)) 的软件多样化技术有效性的评估方法, 获取程序中 ROP/JOP 攻击采用的攻击链, 分别从攻击链的相似性、损坏度和可用性 3 个方面进行定量分析, 来讨论软件多样化技术在空间角度、质量、时间角度对 ROP/JOP 的影响。

1 软件多样化评估相关研究

Coppens 等^[12]使用 BinDiff 补丁对比算法, 利用该算法研究不同补丁之间共同的指令、函数、代码等。该方法被用在衡量对抗补丁攻击^[13]的有效性上, 将补丁前和补丁后的程序视为具有多样性的程序。

Banescu 等^[14-15]用符号执行来评估各种代码混淆方法的强度。发现符号执行对执行了代码混淆的程序造成覆盖率减缓, 代表对程序进行逆向分析的困难增加。但符号执行主要关注对程序逆向的干扰程度, 受程序大小的空间限制。

Pagani 等^[16]采用模糊哈希研究了二进制文件基于局部敏感哈希 (LSH) 方法的库标识、重新编译和程序相似度。而后 Coffman 等^[17]采用了 3 种模糊哈希算法, 对多样化的二进制文件的符号表进行相似性分析, 并对比了不同模糊哈希算法的优劣。

Homescu 等^[18]采用相同地址上幸存 gadgets 占比为指标, 利用该指标衡量不同插入方案的 NOP(无操作)插入多样化技术的安全性。Coffman 等^[11]以采用了 Homescu 等的幸存 gadgets 占比为其中一个指标, 提出在不考虑地址的情况下, 将相同 gadgets 占比作为指标, 对几种多样化技术进行有效性评估。

多样化技术的多层次性使得在部署之前很难确定它的有效性, 因此多样化的目标——引入异构度, 干扰同质化攻击, 从而使攻击者的成本增加——变得难以直接测量。以上工作中, Banescu 等^[14-15]侧重于衡量代码混淆技术对抗符号执行和程序逆向的有效性。Homescu 等^[18]和文献^[11]通过 ROP gadgets 的两种幸存指标评估多样化技术的有效性, 但仅减少幸存的 gadget 占比不能增加其对 ROP 攻击的安全性, 故幸存指标难以真实反映多样化技术的有效性。通常来说, 软件多样性对攻击引入的多样化越强, 攻击者利用成本越高, 则该软件的安全性也就越高, 最理想的状态是攻

击者需要在有效攻击时间内构建一套全新的可靠的攻击链。因此, 为了量化软件多样化真正造成的攻击干扰程度以及安全性, 需要对多样化技术有基于攻击角度的更全面的评估准则和模型。

2 相关技术

2.1 代码重用攻击

攻击者利用缓冲区溢出漏洞, 在存在漏洞的内存地址处重写缓冲区外的返回指针, 重定向控制流, 以执行事先注入的代码。除了缓冲区溢出外, 整数溢出、格式化字符串等内存损坏类的漏洞大多是同一利用方法。为了应对这种利用脆弱内存漏洞达到代码注入的攻击, 在操作系统中引入了可执行保护, 该方法将内存的每个页, 根据其脆弱性, 标记为可写或可执行, 两种状态不能同时存在。若试图在不可执行页上执行代码, 大多数情况下保护机制会直接终止程序的运行。因此, 可执行空间保护使得攻击者难以执行注入的代码, 为了绕过这种保护方法, 引出了一种新的攻击类型——代码重用攻击。

代码重用攻击完全放弃了代码注入的思想, 而是将重点放在重用现有的机器码上, 以达到攻击效果。其中, 常见的代码重用攻击为面向返回的编程^[19]和面向跳转的编程^[20]。ROP 和 JOP 均利用二进制可执行文件中已有的代码片段来进行攻击。ROP/JOP 攻击的核心组件是“小工具(gadgets)”, 即紧接在返回/跳转指令之前发生的指令序列。ROP/JOP gadgets 是一个汇编指令序列, 表示返回值的字节结束, 可以解释为指令和参数(如寄存器或即时值)。

与代码注入攻击(攻击者利用漏洞, 如越界写—注入自己的注入漏洞)不同, ROP/JOP 攻击通过执行由程序和库代码构建的 gadgets 链来危害应用程序。攻击者利用 gadgets 链重定向到内存中的任意字节, 如果它们位于可执行段, 则可直接执行。ROP 使用 ret 指令来改变程序的控制流, 而 JOP 攻击利用程序间接跳转和间接调用指令(call 指令和 jmp 指令)来改变程序的控制流。

2020 年, 英特尔表示将在自己的 Tiger Lake 系列移动 CPU 上配置 CET 安全功能, 该功能旨在防范控制流劫持类攻击, 因此可以防止

ROP/JOP 攻击。但该功能目前只支持 Intel Tiger Lake 系列, 且对 Linux 系统的适配性还需要进一步整合, 即该功能的支持范围有限, 还需要时间扩大对操作系统、硬件及软件栈的普适性。故 ROP/JOP 仍将长期成为可用的攻击方法, 因此本文依旧具有较大的研究意义和价值。

2.2 软件多样化方法

软件多样化最初由 Cohen 等^[21]提出, Cohen 等指出程序进化是一种防御策略, 即程序“进化”成与原始程序不同, 但语义上相同的版本, 并提出了指令重排序和垃圾代码插入等程序“进化”方式。常用的软件多样化技术由编译器来实现, 本文结合 multicompile^[22]和 LLVM Obfuscator^[23]构建了具有多种软件多样化功能的编译器, 其具体功能如下。

1) 控制流平坦: 利用分发器控制^[24]基本块之间的直接跳转, 破坏基本块的跳转关系, 将整个程序的跳转逻辑变得模糊。分发器的控制让基本块都出现在同一层, 隐藏了程序的原始跳转结构。

2) 伪控制流: 通过插入不影响程序功能的无效跳转基本块^[25], 构造一些必真或必假的条件, 打乱程序原来的控制流图。

3) NOP 插入: NOP 是处理器在不修改寄存器文件或内存的情况下执行的指令, NOP 插入是随机在当前指令之前插入 0 或 1 个 NOP。在本文实验中, NOP 插入的方式为 40% 的随机选择插入 30% 的 NOP 指令。

4) 指令替换: 功能一致的指令有不同的表达方法, 如 $a-5=a+(-5)$ 。该方法在不更改功能的前提下, 用一个等价的指令序列替换数值上的二进制和布尔操作。

5) 全局变量洗牌: 随机排列全局变量, 打乱了编译器生成的目标代码中全局变量的顺序。

6) 函数随机化: 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。

7) 栈反转: 传统的栈布局为栈底为高地址, 栈顶为低地址, 栈向低地址生长, 栈反转技术将栈的生长方式反转为向高地址生长。

8) 栈填充: 往栈空间里填充无用数据, 使实际执行的栈空间增加。本文实验的填充率为 50%,

将预留的栈空间填充至利用率达到 50%。

9) 堆栈交换: 原有的可执行文件内存布局中, 栈位于高地址区域, 堆位于低地址区域。该方法将 Linux 可执行文件布局中的堆栈内存布局进行位置交换, 使栈处于低地址区域, 堆处于高地址区域。

3 评估模型

3.1 使用的数据集

本文选择了一个开源大数据集进行评估: GNU 核心实用程序^[17]。表 1 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)。GNU 核心实用程序是一组在类 Unix 操作系统上可以直接通过命令行使用的常用工具, 源代码是开源的, 方便利用多样化编译器进行编译生成变体。GNU 核心实用程序包含如 ls、base64、rm、kill 等大量 Unix/Linux 系统常用工具, 提供了源代码功能上的多样性。单个应用程序在实验中只提供一个数据点, 并采用整个数据集的平均值作为评估指标。

表 1 GNU 核心程序集
Table 1 GNU core utilities

数据集	版本	文件数	KLOC	指令数
GNU coreutils	8.3	100	60.1	700

3.2 评估方法

本文构建了用于评估多样化变体对原始程序 ROP/JOP 攻击的干扰程度的测量方法。为量化多样化技术引起的攻击差异, 最直接的方法是分析 ROP/JOP 赖以使用的 gadgets。同质化的可靠 ROP/JOP 攻击, 对 gadgets 集的相似度、攻击链质量和可用性有较高的要求, 本文从这 3 个方面进行评估。同质环境下, 未进行多样化的软件的 gadgets 应具有较高的相似性甚至完全一致, 攻击链质量高并且 gadgets 数量足够构建可靠的攻击链。多样化技术如果能够达到抗攻击或减缓攻击效果, 则对应的 gadgets 应当降低相似性, 并破坏链质量。

3.2.1 Gadgets 相似性算法

已有的软件多样性技术评估通过分析不同多

样化技术中幸存的 gadgets 的数量或百分比^[11]来量化其安全性影响。该方法被认为可靠性较低,且只考虑了多样化技术的 gadgets 集变化,未将正常执行体和多样化执行体作为一个整体进行比较。

评估软件多样化技术带来的攻击差异化程度的另一种方法是研究多样化技术引起的相似性降低程度。对于 ROP/JOP 攻击来说,从空间的角度,直观的衡量方法是对比正常软件和多样化软件 gadgets 集合的相似性。多样化软件的 gadgets 集与正常执行体的 gadgets 集相似性越低,说明多样化方法引入的 gadgets 变化越大,针对多样化软件构建的攻击链差异度越大,攻击者所需的空间成本越高。

为实现相似性评估,本文提出 gadgets 相似度计算。本文定义相似度是两个可执行程序不考虑地址的情况下共享的具有相同 gadgets 链的比例。Gadgets 集合代表攻击者能采用的攻击链集合,可用的 gadgets 链可能因为多样化方法在不同变体中呈现差异。

本文使用 Jaccard 相似系数^[17]来计算相似度。Jaccard 相似系数是一种衡量两个集合的相似度指标,定义为集合 A 与集合 B 交集的大小除以并集的大小,公式为

$$\text{Jaccard}(A, B) = \frac{A \cap B}{A \cup B}, \text{Jaccard}(A, B) \in [0, 1] \quad (1)$$

在本文实验中,固定正常执行体集合 Normal 为集合 A ,其余 9 种多样化方法生成的变体集合依次为集合 B ,并计算集合 A 与集合 B 的 Jaccard 相

似系数。如图 1 所示,正常 GNU 核心程序集中的 ls 为 gadgetsA 集合,控制流平坦程序集(CFF-GNU)的 ls 为 gadgetsB 集合, gadgetsA 和 gadgetsB 分别取交集和并集进行计算得到 Jaccard 相似系数。

本文编写脚本自动对比两个 gadgets 集合文件,搜索并统计相同的 gadgets 数量得到交集,用 gadgets 总数减去交集得到并集,最后实现 jaccard 相似系数的计算。

3.2.2 Gadgets 链质量

一些 gadgets 具备图灵完整性的能力,这类 gadgets 是攻击时采用的核心攻击指令,此处定义为图灵完整 gadgets(TC gadgets)^[26]。其中,ROP 中有加载内存 LM、存储内存 SM、算术操作 AM、栈操作 SP、加载寄存器 LR,移动寄存器 MR 等为图灵完备的。JOP 额外有跳转操作 JMP 和函数调用 CALL 为图灵完备的。具体分类方法如表 2 所示。

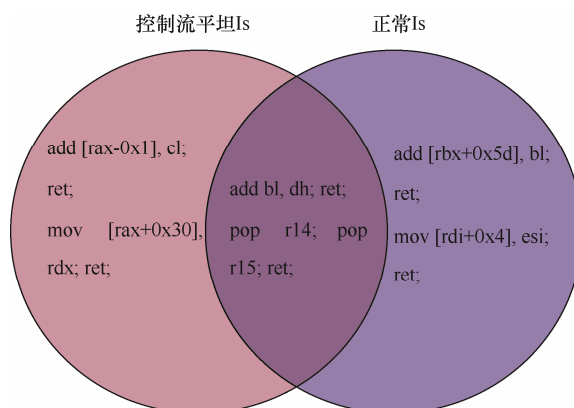


图 1 控制流平坦和正常编译的 ls 程序 gadgets 交并集
Figure1 Gadgets intersection and union of control flow flattening and normal compilers ls program

表 2 用于 ROP/JOP 攻击的 TC gadgets
Table2 TC gadgets for ROP/JOP attack

gadget 类型	描述	形式	示例
移动寄存器 MR	将 a 寄存器参数赋给 b 寄存器	mov reg1, reg2	mov rax, edx
加载寄存器 LR	将参数加载进寄存器	pop reg	pop r15
加载内存 LM	将内存存放的参数给寄存器	mov reg1, [reg2]	mov eax, [rax]
存储内存 SM	将寄存器参数存入内存	mov [reg1], reg2	mov [rax], esi
算术操作 AM	两个寄存器之间的算术运算/加载存储	add reg1, reg2	add ebx, rsi
		add [reg1], reg2	add [rcx], al
		add reg, const	add eax, 0x208d8e
栈操作 SP	设置栈顶指针	xchg rsp, reg	xchg rsp, rax
跳转操作 JMP	设置指令指针	jmp reg	jmp qword ptr [rsi+0x41]
函数调用 CALL	通过寄存器跳转到某函数	call reg	call qword ptr [rsi+0x20]

攻击者需要使用 gadgets 改变原有的寄存器相关调用和存储值, 所以用于生成攻击链的 gadgets 链质量至关重要。多样化技术可能会改变指令对寄存器的调用和寄存器的存储值, 导致在 ROP/JOP 的攻击中, 寄存器无法正常发挥原来的作用。为度量 gadgets 链的整体质量, 本文参考 Ahmed 等^[26]评估不同粒度 ASLR 方案采用的 JIT-ROP gadgets 损坏率指标, 并针对 JOP 扩展了 TC gadgets, 针对 ROP 和 JOP, 对原方法采用的指令进行了修改, 对具有图灵完备性的每一条 gadgets 链进行损坏分析。

通常, 一个 gadgets 链包含一个核心指令。如“mov eax, 0x6083e0; cmove rdi, rax; mov eax, [rdi]; ret;”包含 mov eax,[rdi]作为核心指令, 以及在核心指令之前/之后的一些附加指令。核心指令是攻击者必要的指令, 附加指令可能会修改核心指令的源寄存器或目标寄存器, 如果发生这种情况, 则视该 gadgets 已损坏。核心指令被修改的情况有以下 3 种。

1) 被核心指令前的指令修改: “and [rax], al; mov rbx, [rip+0x205bd7]; pop rbx; ret;”, “pop rbx”为核心指令, rbx 被前面的“mov rbx, [rip+0x205bd7]”修改。

2) 被核心指令后的指令修改: “add [eax], rax; add rax, cl; ret;”, 核心指令“add [eax],rax”中 rax 被后面的“add rax,cl”修改。

3) 核心指令前/后的指令均可修改: “and eax, ebx; mov eax, [r9+0x4]; mov [r9+0x4], esi; ret;”, 核心指令“mov [r9+0x4], esi”中 esi 分别被前后的指令修改。

本文通过分析附加指令是否修改小工具的核心指令来测量寄存器损坏率。根据以上 3 种场景判断 gadgets 损坏情况。确定一组可以修改核心指令的源寄存器或目标寄存器的附加指令 (mov、lea、add、sub、cmov、pop、inc、dec、xchg、and、or、xor、shl 和 shr), 将这类指令使用的寄存器视为冲突寄存器。

如果在有一组有以上附加指令和表 2 中核心指令的 gadgets 链中, 附加指令修改了核心指令寄存器, 则该条 gadgets 链视为被损坏。

通过以上方法, 获得被损坏的 gadgets 链数

量, 用损坏的 gadgets 链数量除以所有 gadgets 链的数量来测量寄存器损坏率。

3.2.3 Gadgets 收集效率

多样化技术对 ROP/JOP 的重要影响是改变 gadgets 的数量和收敛时间。多样化技术可能带来 gadgets 数量的减少或增加, 并增长或缩短收集完整 gadgets 集的收敛时间。存在一种情况: 尽管多样化技术使 gadgets 数量减少, 由于扰乱了各种层次的布局, 反而增加了收集 gadgets 的时间, 提高了攻击者的时间成本。构建一个完整可用的攻击链需要有足够数量的 gadgets, 而在真实环境中每一次攻击受到有效攻击时间的约束, 减少软件中可用的 gadgets 总数, 延长收集有效 gadgets 的时间, 增加了构建攻击链的难度和成本, 因此可以减少其受到有效攻击的次数。为了评估以上情况, 本文提出 gadgets 收集效率这一指标。

为度量 gadgets 收集效率, 本文对所有执行体运行 gadgets 收集函数, 记录收敛到完整 gadgets 集所需的时间, 并计算每秒可收集 gadgets 个数, 得出收集效率, 该指标不仅反映多样化技术对 gadgets 收集造成的时间成本影响, 也反映了数量的变化。对于攻击者来说, 越低的时间成本内收集的 gadgets 越多, 收集效率越高, 则 gadgets 的可用性越高, 攻击的成功率越高。

4 实验结果与分析

本文实验在使用 Ubuntu 18.04 LTS 64 位操作系统的 Linux 机器上进行, 采用内生安全软件开发环境 (ES2, endogenous security studio) 生成执行体。ES2 是集软件代码开发、异构化编译、多样化执行体生成等功能于一体的软件开发环境, 具备业界领先的进程级细粒度多样化编译性能。

本文采用基于 Python 的开源 gadgets 收集工具——ropper, 并针对实验需求对源码进行改进。改造后的 ropper 添加了计算收集完整 gadgets 时间的功能函数, 增加了只收集 ROP 或 JOP gadgets 的功能。

本文采用 9 种多样化技术 (控制流平坦 (CFF)、虚假控制流 (FCF)、NOP 插入 (NI)、指令替换 (IR)、全局变量洗牌 (GVS)、函数随机化 (FR)、栈反转 (RS)、栈填充 (FS) 和堆栈

交换 (ESH)) 以及一种正常编译方法 Normal。对于每种多样化技术, 利用 GNU 核心程序集, 用 ES2 集中编译连接。其中, 每个核心程序均使用不同的多样化技术进行编译, 生成 900 个多样化变体, 加上正常编译未使用多样化技术的 100 个执行体, 共计 1 000 个执行体, 10 个不同的 GNU 核心程序集。

利用 3.2 节所述的各类方法, 得到每个 GNU 核心程序集的每个程序对应数据, 并取该程序集的平均值, 作为实验的最终数据。

4.1 Gadgets 相似度的评估

假设随着 gadgets 的共同集减少, 攻击者的成本增加。理想情况下, 两个多样化的二进制文件应该完全没有共同的 gadgets, 攻击者必须为每个多样化的二进制文件构建一个新的 ROP 链。

本实验计算了正常执行体和其他所有多样化变体的 gadgets 集相似度。对于每一对执行体和变体, 使用相同 gadgets 来确定每个可执行文件的交集, 使用 Jaccard 算法得到相似度, 并取程序集的平均值。

对于每种多样性技术, 图 2 显示了每个 GNU 核心程序集中 ROP gadgets 的平均相似度, 即 Jaccard 相似系数。在采用的多样化技术中, 发现 NOP 插入效率最高, 相似度不到 5%, 说明相同的 gadgets 的数量最少, 因为额外的 NOP 使 gadgets 的数量变化较大。控制流平坦技术的相似度仅为 6.73%。除了 NOP 插入和控制流平坦外, 其他的多样化技术最终达到高于 7% 低于 8% 的相似度。

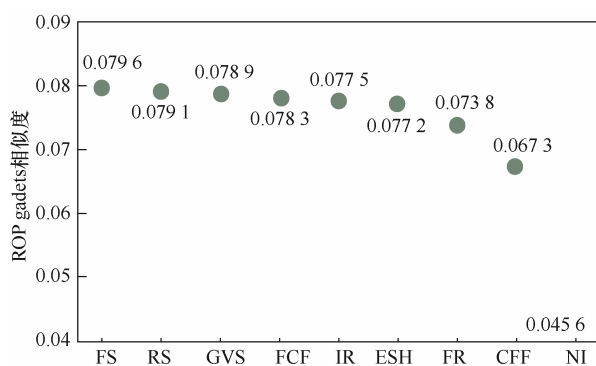


图2 ROP gadgets 的相似度
Figure 2 The similarity of ROP gadgets

函数随机化排序因打乱了函数的顺序, 干扰

了内存中函数存放的位置, 相似度较小。而细粒度指令替换和全局变量洗牌以及虚假控制流较为接近。因为 ROP 攻击以 ret 指令为结尾, 需要多次调用栈达到实现执行 gadgets 链的目的, 堆栈交换引入的堆栈布局多样性优于栈反转和栈填充。

Coffman 等^[11]采用不考虑地址的幸存 gadgets 占比这一指标从空间角度上度量多样化技术。Coffman 的实验结果中, NOP 插入、控制流平坦、指令替换、伪控制流这 4 种方法的指标依次升高, 这与本文的实验结果是吻合的, 也侧面证明了本文采用指标的可信度。

图 3 是每个 GNU 核心程序集中 JOP gadgets 的平均相似度。JOP gadgets 的平均相似度整体高于 ROP gadgets。与 ROP 的情况类似, NOP 插入依旧是相似度最低的多样化技术, 因其带来了较多的无用过程。控制流平坦技术相似性仅为 6.66%。其他的多样化技术则维持在 8.5%~9%。

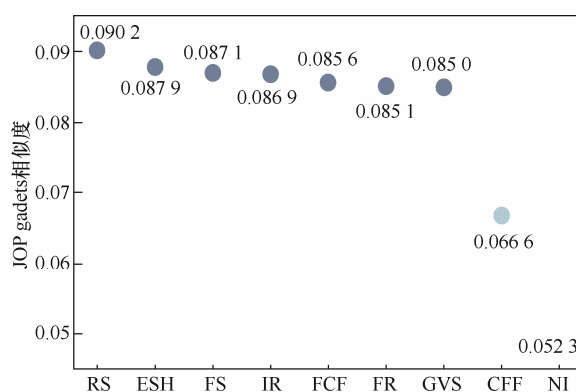


图3 JOP gadgets 的相似度
Figure 3 The similarity of JOP gadgets

与 ROP gadgets 相似度不同的是, 全局变量洗牌、函数随机化和虚假控制流比指令替换引入了更多的多样性。因为 JOP 以跳转为主, 函数和变量的变化对跳转地址的干扰更大。但是 JOP 攻击通过修改栈内容来修改寄存器值, 实现对栈操作, 达到跳转的目的, 所以栈多样化技术基本不分上下。

Gadgets 相似度为多样性技术的有效性提供了一些见解。不同的多样化对 ROP/JOP gadgets 产生不同程度的影响, 均未使得相似度趋近于 0。这一结果说明即便采用了多样化技术, 执行体的 gadgets 中始终有共同存在的部分。这个部分或许

是一些初始指令，或许是完成攻击的核心指令。并且不同技术对 ROP 和 JOP 产生的多样性效果也不相同，取决于该多样化技术对 ROP 和 JOP 主要利用的地址和内存布局有怎样程度的改变。

4.2 Gadgets 链质量的评估

这个实验的目的是分析 gadgets 链的质量和寄存器的可用性。根据 3.2.2 节描述的方法，通过对单个 gadget 的寄存器损坏分析来衡量 gadget 的质量。本文测量了图灵完备的 gadgets 集的寄存器损坏情况，并通过将损坏的小工具的数量除以所有小工具的数量来计算 gadgets 的寄存器损坏率。

图 4 为图灵完备的 ROP gadgets 的寄存器损坏率分析结果。控制流平坦导致损坏率高达 34%，NOP 插入也造成了 gadgets 质量的较大损失。细粒度的指令替换和全局变量洗牌整体损坏率略高于粗粒度的函数随机化和虚假控制流（约为 4%）。这个略高的寄存器损坏率表明，细粒度多样化技术中 gadget 链的形成和使用比粗粒度多样化技术略难。因为栈反转只是将栈的生长方向对调，ROP 对栈顶指针的调用受方向的干扰较小，故栈反转的效果较差。

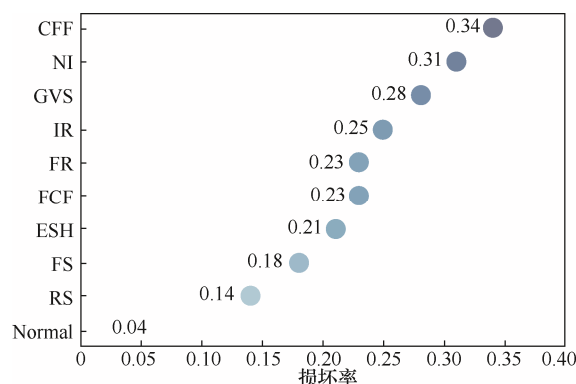


图 4 图灵完备的 ROP gadgets 寄存器损坏率
Figure 4 Register damage rate of TC ROP gadgets

图 5 为图灵完备的 JOP gadgets 的寄存器损坏率分析结果。JOP gadgets 的整体寄存器损坏率比 ROP gadgets 的略低，最高不超过 30%。细粒度的指令替换和全局变量洗牌与粗粒度的函数随机化和虚假控制流损坏率接近，表明对于 JOP 攻击，这两类技术中 gadgets 链的形成和使用成本接近。JOP 攻击对堆栈的依赖较小，对寄存器的调用也

没有 ROP 严格，因此无论是堆栈多样性技术还是其他的多样性技术，JOP gadgets 总体损坏率都略低于 ROP gadgets 的损坏率。

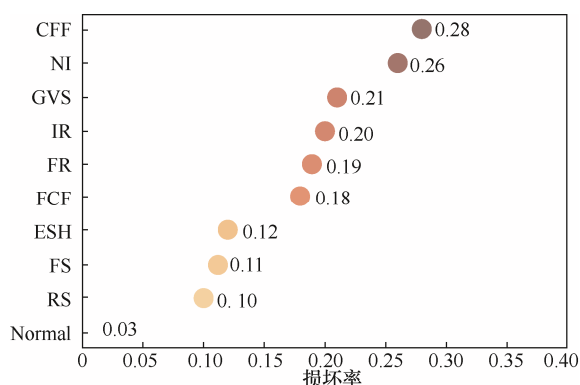


图 5 图灵完备的 JOP gadgets 寄存器损坏率
Figure 5 Register damage rate of TC JOP gadgets

通过衡量 gadgets 的寄存器损坏率，可以发现 ROP 和 JOP 对寄存器及堆栈的依赖程度不同。但多样化技术都带来不同程度的寄存器损坏，使 gadgets 链质量下降，可靠性降低。未来的多样化技术发展方向，可以考虑以增加 gadgets 中的寄存器损坏率为研究目标，因为越高的寄存器损坏率让攻击变得越不可靠，攻击能正常使用的寄存器越少。

4.3 Gadgets 可用性的评估

为了度量 gadgets 收敛时间和收集效率，本文采用 3.2.3 节描述的方法，收集了 ROP 和 JOP 各自所有的 gadgets，并获得了收集完整 gadgets 的收敛时间，再计算得到平均每秒能收集的 gadgets 个数。

图 6 中列出了各种生成的 GNU 程序集的 ROP gadgets 平均每秒收集的 gadgets 个数。从该指标可看出，多样化技术让 ROP gadgets 数量有减少的趋势，降低了 gadgets 的收集效率和可用性。NOP 插入带来的减少程度是倍数级的。NOP 插入让软件膨胀，且插入的 NOP 指令是无效的，很大限度增加了收敛完整 gadgets 的时间，导致每秒收集数量急剧降低。而堆栈级和函数指令级多样化技术整体表现相似。因为理想情况下，这两类的多样化技术不会破坏 gadgets，只会将 gadgets 从一个位置转移到另一个位置。由于程序执行流的混乱和寄存器分配的随机化，控制流平

坦这种基本块级的技术对 ROP gadgets 的破坏程度较大。

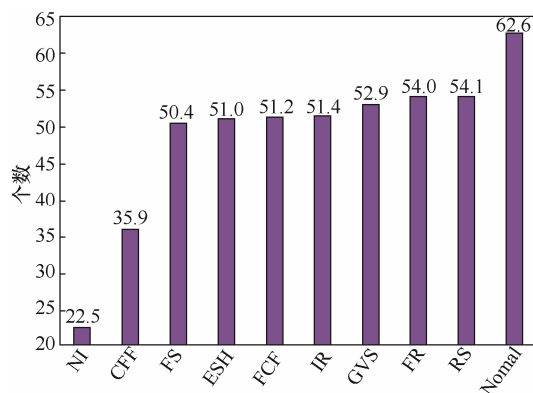


图 6 平均每秒收集的 ROP gadgets 数量
Figure 6 Average of ROP gadgets collected per second

JOP 的情况与 ROP 不太一样。由图 7 可知,除了 NOP 插入带来 JOP gadgets 的减少外,其他多样化技术均展现出令人意外的收敛情况。包括在其他指标中表现优越的控制流平坦技术,其指标与其余多样化技术接近。这个情况揭示了大多数多样化技术在一定限度上增加了 JOP gadgets 数量。因为 JOP 不用 ret 来连接原始操作和指令,所以需要收集更多可用的以 jmp 和 call 结尾的 gadgets 来达到攻击目标。而多样化技术对 ret 结尾的 gadgets 有削减的作用,因此在 JOP 攻击中需要收集更多的 jmp 和 call gadgets 来弥补这一缺失。

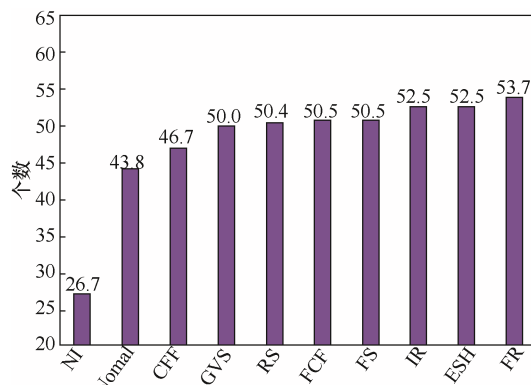


图 7 平均每秒收集的 JOP gadgets 数量
Figure 7 Average of JOP gadgets collected per second

通过分析 gadgets 的收敛情况可知,对于 ROP 来说,多样性越大的技术使得 gadgets 的收集效率越低,造成的可用性越低,并且增加越多的攻击时间成本。但是 JOP 受到相关影响,要完成图灵完备攻击则需要收集更多的 JOP gadgets。总的来说,多样化技术让 gadgets 的收敛速度和数量有改变,在实际情况中应该综合影响两种攻击的平衡性去考虑多样化技术的有效性。

4.4 评估总结

通过对 9 种多样化技术和正常编译产生的 GNU 核心程序集的 ROP/JOP gadgets 不同角度的分析发现,多样化技术均对 ROP/JOP 攻击产生了不同程度的干扰,会导致 ROP/JOP 攻击所需的攻击链空间增大,构造攻击链的时间发生变化且攻击成功率降低。综合而言, NOP 插入和控制流平

表 3 多样化技术评估结果
Table 3 Results of diversified technology assessment

变体生成技术	Gadgets 相似度		Gadgets 损坏率		Gadgets 收集效率 (个/s)	
	ROP	JOP	ROP	JOP	ROP	JOP
CFF	6.73%	6.66%	34%	28%	35.9	46.7
NI	4.56%	5.23%	31%	26%	22.5	26.7
FCF	7.63%	8.56%	23%	20%	51.2	50.4
GVS	7.69%	8.50%	28%	18%	52.8	50.0
IR	7.75%	8.69%	25%	19%	51.4	52.5
FR	7.38%	8.51%	23%	21%	54.0	53.7
RS	7.91%	9.02%	14%	10%	54.1	50.4
FS	7.96%	8.71%	18%	11%	50.4	50.5
ESH	7.82%	8.79%	21%	12%	51.0	52.5
Normal	—	—	4%	3%	62.6	43.8

坦都发挥出色,使 3 个指标有了较为明显的变幅,具有更高的安全增益。其他多样化技术根据攻击的需求和多样化的粒度,对攻击呈现不同的限制效果。堆栈多样化技术总体对执行体的改变局限于堆栈区域,带来的安全增益也较弱。

本文的评估方法在 ROP/JOP 攻击层面,对多个影响攻击的 *gadgets* 指标进行分析,从同质化攻击的不同角度对各类软件多样化技术进行定量比较,给出多样化效果的安全增益和有效性的综合评估结果,并能更加深入地挖掘 ROP/JOP *gadgets* 的性质,日后可根据这些指标和性质研究更加具有安全性的多样化技术。

5 结束语

现有的 ROP *gadgets* 幸存率指标存在难以直接体现多样化技术的安全性影响且评估方法单一的问题。为了更加全面地从攻击视角评估多样化技术的有效性,本文根据常见的代码重用攻击 ROP 和 JOP 及其 *gadgets* 的不同性质来确定多样化技术对网络攻击产生的影响,建立评估体系。本文利用 GNU 核心程序数据集开展实验,评估了 9 种不同的多样化技术对 *gadget* 相似度、链质量、可用性等方面的影响,从 *gadgets* 的空间角度、时间角度和质量可靠角度 3 个方面进行量化和比较,验证了所提方法在软件多样化领域的适用性以及有效性。

但是本文的评估方法主要评估了 Linux 下的 X86-64 架构软件多样化方法,未来需要进一步对 Windows 系统和其他架构(如 ARM 的软件多样化技术)进行评估验证。此外,可以创建新的指标,对正常软件和多样化软件的函数、指令、基本块等粒度进行差异分析,提出一套不依赖特定攻击方法的评估体系。

参考文献:

- [1] LITCHFIELD D. Buffer Underruns, DEP, ASLR and improving the exploitation prevention mechanisms (XPMs) on the Windows platform[EB].
- [2] ABADI M, BUDI M H, ERLINGSSON Ú, et al. Control-flow integrity[C]//Proceedings of the 12th ACM conference on Computer and Communications Security CCS '05. 2005: 340-353.
- [3] LIVSHITS V B, LAM M S. Finding security vulnerabilities in Java applications with static analysis[J]. 14th USENIX Security Symposium, 2005: 271-286.
- [4] 姚东, 张铮, 张高斐, 等. 多态执行安全防御技术研究综述[J]. 信息安全学报, 2020, 5(5): 77-94.
YAO D, ZHANG Z, ZHANG G F, et al. A survey on multi-variant execution security defense technology[J]. Journal of Cyber Security, 2020, 5(5): 77-94.
- [5] GIUFFRIDA C, KUIJSTEN A, TANENBAUM A S. Enhanced operating system security through efficient and fine-grained address space randomization[C]//Proceedings of the 21st USENIX Conference on Security symposium. 2012: 40.
- [6] HISER J, NGUYEN-TUONG A, CO M, et al. ILR: where'd my gadgets go[C]//Proceedings of 2012 IEEE Symposium on Security and Privacy. Piscataway: IEEE Press, 2012: 571-585.
- [7] 刘镇武, 隋然, 张铮, 等. 基于信息熵与软件复杂度的软件多样性评估方法[J]. 信息工程大学学报, 2020, 21(2): 207-213.
LIU Z W, SUI R, ZHANG Z, et al. Software diversity evaluation method based on information entropy and software complexity[J]. Journal of Information Engineering University, 2020, 21(2): 207-213.
- [8] HERNANDEZ-CASTRO J, ROSSMAN J. Measuring software diversity, with applications to security[J]. arXiv: 1310.3307, 2013.
- [9] SHANNON C E. A mathematical theory of communication[J]. Bell System Technical Journal, 1948, 27(3): 379-423.
- [10] COHEN F B. Operating system protection through program evolution[J]. Computers & Security, 1993, 12(6): 565-584.
- [11] COFFMAN J, KELLY D M, WELLONS C C, et al. ROP gadget prevalence and survival under compiler-based binary diversification schemes[C]//Proceedings of the 2016 ACM Workshop on Software Protection. 2016: 15-26.
- [12] COPPENS B, DE SUTTER B, MAEBE J. Feedback-driven binary code diversification[J]. ACM Transactions on Architecture and Code Optimization, 2013, 9(4): 1-26.
- [13] BRUMLEY D, POOSANKAM P, SONG D, et al. Automatic patch-based exploit generation is possible: techniques and implications[C]//Proceedings of 2008 IEEE Symposium on Security and Privacy. 2008: 143-157.
- [14] 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.
- [15] SEBASTIAN B, CHRISTIAN C, AND 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.
- [16] FABIO P, MATTEO D'A, DAVIDE B. Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis[C]//Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18). 2018: 354-365.
- [17] 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.
- [18] 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). Piscataway: IEEE Press, 2013: 1-11.

- [19] ROEMER R, BUCHANAN E, SHACHAM H, et al. Return-oriented programming[J]. ACM Transactions on Information and System Security, 2012, 15(1): 1-34.
- [20] BLETSCH T, JIANG X X, FREEH V, et al. Jump-oriented programming: a new class of code-reuse attack[C]//Proceedings of ASIACCS '11. 2011.
- [20] BLETSCH T, JIANG X X, FREEH V W, et al. Jump-oriented programming: a new class of code-reuse attack[C]//Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security - ASIACCS '11. 2011: 30-40.
- [21] COHEN F B. Operating system protection through program evolution[J]. Computers & Security, 1993, 12(6): 565-584.
- [22] CRANE S, LIEBCHEN C, HOMESCU A, et al. Readactor: practical code randomization resilient to memory disclosure[C]//Proceedings of 2015 IEEE Symposium on Security and Privacy. Piscataway: IEEE Press, 2015: 763-780.
- [23] JUNOD P, RINALDINI J, WEHRLI J, et al. Obfuscator-LLVM: software protection for the masses[C]//Proceedings of 2015 IEEE/ACM 1st International Workshop on Software Protection. 2015: 3-9.
- [24] L'ASZLÓ T, KISS A. Obfuscating C++ programs via control flow flattening[C]//Processing of Annales Universitatis Scientiarum 25 Budapestinensis de Rolando Eötvös Nominatae, Sectio. 2009.
- [25] 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 - POPL '98. 1998: 184-196.
- [26] AHMED S, XIAO Y, SNOW K Z, et al. Methodologies for quantifying (re-) randomization security and timing under JIT-ROP[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 1803-1820.

[作者简介]



迟宇宁（1995- ），女，云南楚雄人，信息工程大学硕士生，主要研究方向为网络主动防御、软件多样化。



郭云飞（1963- ），男，河南郑州人，信息工程大学教授、博士生导师，主要研究方向为网络空间安全、云安全和电信网安全。



王亚文（1990- ），男，河南郑州人，信息工程大学助理研究员，主要研究方向为拟态防御和云计算。



扈红超（1982- ），男，河南商丘人，信息工程大学教授、博士生导师，主要研究方向为网络空间安全和主动防御技术。