



一种基于表达式树的 gadget 语义分析技术

蒋 楚,王永杰

(国防科技大学 电子对抗学院,合肥 230037)

摘 要: 代码重用攻击的实施过程较为繁杂,通常需要一些工具辅助人工来完成 gadget 序列的构建,但现有的自动化构建工具效率较低。在分析 Ropper、angrop 和 BOPC 等典型开源 gadget 工具语义分析内容的基础上,总结 gadget 语义分析应包含的要素,提出一种基于表达式树的 gadget 语义分析方法。通过表达式树变体描述寄存器和内存读写的表达式信息,提高 gadget 语义分析的效率。实现一个 gadget 搜索与语义分析工具 SemExpr,针对现有 gadget 工具难以进行对比分析的问题,设计能对多种 gadget 工具进行效率和效能分析的实验系统 gadgetAnalysis。基于该系统进行实验,结果表明,SemExpr 工具能够权衡效率和效能,取得较好的语义分析效果。

关键词: 代码重用攻击; gadget 序列自动化构建; 语义分析; 语义摘要; gadget 工具

开放科学(资源服务)标志码(OSID):



中文引用格式:蒋楚,王永杰.一种基于表达式树的 gadget 语义分析技术[J].计算机工程,2021,47(1):109-116.

英文引用格式:JIANG Chu, WANG Yongjie. A technique of gadget semantic analysis based on expression tree[J]. Computer Engineering, 2021, 47(1): 109-116.

A Technique of gadget Semantic Analysis Based on Expression Tree

JIANG Chu, WANG Yongjie

(College of Electronic Countermeasure, National University of Defense Technology, Hefei 230037, China)

[Abstract] Due to the complexity of implementing Code Reuse Attack(CRA), some tools are required to construct the gadget sequence, but the existing automation build tools are inefficient. Based on the semantic analysis of typical open-source gadget tools such as Ropper, angrop and BOPC, this paper abstracts the elements that should be included in the semantic analysis of gadget, and puts forward a kind of semantic analysis method for gadget based on expression tree. The method improves the efficiency of semantic analysis of gadget by using an variant of expression tree to describe the expression information of reading or writing registers and memory. On this basis, a gadget search and semantic analysis tool named SemExpr is implemented. As it is difficult to compare and analyze the existing tools, an experimental system named gadgetAnalysis is designed to analyze and compare the efficiency and performance of several gadget tools. Experimental results shows that SemExpr can balance the efficiency and performance, and achieve excellent semantic analysis effect.

[Key words] Code Reuse Attack(CRA); automated gadget sequence generation; semantic analysis; semantic summary; gadget tool

DOI: 10. 19678/j. issn. 1000-3428.0056671

0 概述

自1988年第一例缓冲区溢出漏洞攻击——莫里斯蠕虫爆发以来,针对内存漏洞的攻击方法便层出不穷,相应的防护手段也不断推陈出新^[1],内存漏洞的攻防博弈成为网络空间安全领域的热点问题之一。

在早期的代码注入攻击中,渗透测试人员利用溢出漏洞向内存中注入恶意代码并篡改保存在内存中的

跳转地址,使程序的控制流转向所注入的恶意代码中。为了解决这一问题,现代操作系统开始引入一种名为数据执行保护(Data Execution Prevention, DEP)的保护机制,其将内存中的数据和代码进行严格区分,渗透测试人员注入的恶意代码只能被当作数据解析而无法被执行,从而使得常规的代码注入攻击难以发挥作用。

随着攻防技术的相互促进与发展,代码重用攻击(Code Reuse Attack, CRA)开始兴起^[2-4],渗透测试

基金项目:国家部委基金。

作者简介:蒋 楚(1995—),男,硕士研究生,主研方向为软件安全;王永杰,副教授。

收稿日期:2019-11-21 修回日期:2019-12-31 E-mail: 1532173962@qq.com

人员利用内存漏洞改变程序的执行流程,通过将内存空间中已有的分散代码片段(称为 gadget)进行链接,构造出具备特定功能的程序逻辑从而实现攻击的目的,能够达到与传统代码注入攻击相同的效果。

代码重用攻击在实施过程中较为繁杂,通常需要一些工具辅助人工来完成 gadget 序列的构建,学术界也提出了很多自动化的构建方案,但都存在诸多问题使其难以得到广泛应用。短期内,代码重用攻击在实施过程中存在的问题将使得攻击成本增加,恶意攻击者不得不付出大量的时间来构建攻击载荷,从长远来看,这也不利于防御机制的进一步发展。因此,加快代码重用攻击的实现过程,快速构建攻击载荷,能够方便渗透测试人员对系统进行评估测试,从而构建更加安全有效的防御系统。

本文提出一种基于表达式树的 gadget 语义分析方法,通过表达式树变体描述 gadget 修改寄存器和内存读写的情况,从而提高 gadget 语义分析的效率,加快 gadget 序列的构建过程。

1 研究背景

1.1 代码重用攻击

代码重用攻击是一种利用漏洞改变程序在内存中的原有代码执行流程以实现攻击目的的漏洞利用技术。以一个存在栈溢出漏洞的程序为例,在传统的攻击方法中,渗透测试人员需要向栈中注入恶意代码,并修改程序的返回地址,使程序的控制流转移到恶意代码的起始地址,这种方法称为代码注入攻击。但是,现代操作系统已经广泛采用数据不可执行(DEP)等防御机制,代码注入攻击对此难以实现攻击目的,因此,代码重用攻击成为当前的主流攻击方式。

代码重用攻击和代码注入攻击之间既有一些共性特征也存在差异,它们都是利用计算机内存漏洞的攻击方式,均会篡改一些影响程序执行的关键数据,如栈中的返回地址、函数指针等,以此实现超出程序设计者预期的功能逻辑。两者的区别在于渗透测试人员执行恶意代码的方法不同,代码重用攻击无需向内存中注入外部代码,只需重用程序中已经存在的指令片段,这些指令片段能够完成一定的计算和赋值等功能,因此也被称为 gadget,通过多个 gadget 的拼接,渗透测试人员可以实现一些特定的意图。代码重用攻击通常会使程序的控制流转向特定的 gadget,并按设定的顺序依次执行 gadget,其实现过程如图1所示。

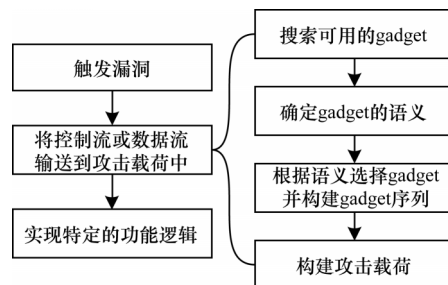


图1 代码重用攻击的实现过程

Fig.1 Implementation of code reuse attack

对于现代操作系统中某个存在缓冲区溢出问题的程序,为了绕过 ASLR 和 DEP 保护,渗透测试人员需要进行以下3个步骤:

1)利用内存信息泄露漏洞得到需要调用的函数的地址和存在问题的缓冲区位置。

2)构建 gadget 序列以调用该函数。

3)根据 gadget 序列构造攻击载荷并传递给程序。

上述过程的核心是选择合适的 gadget 序列。渗透测试人员在实现某些功能前需要初始化一些寄存器,例如,在 64 位 Windows 操作系统中进行函数调用前需要在 RCX、RDX、R8 和 R9 寄存器以及栈中设置参数。对于每一个需要初始化的寄存器,都要找到一个能够对其进行修改的 gadget,因此,在典型的代码重用攻击中要调用某个函数,构造的 gadget 序列通常要包含:

1)寄存器写 gadget,向 RCX、RDX、R8 和 R9 寄存器中写入指定的值。

2)能够调用特定函数的 gadget。

要从程序中找到这些 gadget,需要确定 gadget 的部分语义,例如能否修改栈顶指针、能否修改特定的寄存器、是否有函数调用以及调用了哪个函数等信息。gadget 序列的构建过程还会受到一些因素的影响,其中,最主要的影响是多个 gadget 使用同一寄存器或同一内存空间的情况,这会导致寄存器或某一内存空间在赋值和使用时出现数据不一致的问题,这也被称为 gadget 的副作用。由于 gadget 副作用的存在,手工构造 gadget 序列的过程极为繁琐复杂。自 SHACHAM 提出 ROP^[3]以后,自动化构建 gadget 序列的研究就开始展开,但截至目前,gadget 工具的自动化效果并不理想,多数工具只实现了自动化搜索 gadget 并进行语义分析的功能。对于 ROP 及其变种,gadget 搜索主要通过指令特征定位到特定的指令位置,再通过切片的方式实现,该方法的优化存在局限性。

1.2 gadget 语义分析技术研究现状

由于机器指令的指令集过于复杂,并且与硬件结合过于紧密,直接对其进行分析时难度较大。因此,现有工具通常会将 gadget 的指令转换成 IR

(Intermediate Representation),再以形式化的方法来确定 gadget 所执行的操作。

SCHWARTZ 等人设计了能够对二进制文件自动生成 ROP 载荷的工具 Q^[5-6],其主要思想是将搜索到的 gadget 以一种更容易处理的中间形式表示,使用编译器编译时的指令匹配思路,结合中间形式在 gadget 集合中选择合适的 gadget 来实现用户描述功能。Q 将 gadget 按功能进行分类,并设计一种 Qool 语言,与 gadget 的类别相对应。Q 在判定某个 gadget 是否属于特定功能的 gadget 类别时,采用程序验证领域中程序最弱前置条件的计算方法^[7-9],具体如下: gadget \mathcal{L} 和后置条件 \mathcal{B} 的最弱前置条件 $WP(\mathcal{L}, \mathcal{B})$ 是一个能够判断 gadget 在执行后能否满足条件 \mathcal{O} 的布尔值,如果 $WP(\mathcal{L}, \mathcal{B})$ 恒为真,则 gadget 属于后置条件 \mathcal{B} 对应的类别。这种将语义进行分类的思想也被许多 gadget 工具所采用。

BARFgadget^[10]用 2 种方式实现 gadget 分类和验证的功能,其仿照 Q 将 gadget 按功能分类,并为每一个类别定义一些判断条件。在分类功能中,创建一个随机化的初始状态,通过 BARF 中的 IR 模拟器执行二进制代码对应的 REIL 指令以得到执行 gadget 后的状态,确定 gadget 读写了哪些寄存器以及是否修改了内存和标志位,并根据这些信息对 gadget 的功能进行初步分类;在验证功能中,根据预先定义的约束条件,通过约束求解的方式进一步验证 gadget 是否具有前一阶段确定的功能类别。

PSHAPE^[11]使用 VEX 作为中间语言,并结合 VEX 指令的特点,将 PUT、ST 和 GET 指令中的寄存器进行展开,用由寄存器构成的表达式替换临时变量,展开的过程中根据寄存器的传递关系,每次更新寄存器的最终值并且记录寄存器解引用的情况。Ropper 和 PSHAPE 的思路基本一致,针对 PUT、ST 和 GET 3 种操作寄存器和内存的指令,提取指令所依赖的表达式,最后使用约束求解器求解栈指针的偏移。angrop 基于 angr 框架开发,其充分利用了 angr 提供的功能,将寄存器符号化,创建一个空白的初始状态,使用符号执行得到执行 gadget 后的状态,根据新状态中寄存器和内存的值获取 gadget 读写寄存器和内存的情况。

ROPgenerator^[12]使用 REIL 作为中间语言,在将 gadget 的指令转换成 REIL 指令后,会对每一条指令进行分析,所有对寄存器和内存的操作会被作为有向图的一个节点,节点中包含操作的表达式,通过节点之间的边来表示寄存器之间的依赖关系,在遇到条件指令时,在边上添加条件的约束信息。最后,从有向图中提取出条件和表达式的键值对以得到 gadget 的语义摘要。

BOP(Block Oriented Programming)^[13]是一种非控制数据攻击,其 gadget 是遵循程序原有执行流的

代码块,但若要实现特定意图,还需程序能够执行到具有特定语义的基本块上,并且在执行时满足一些特定的约束。自动化工具 BOPC 对程序的基本块进行了抽象,其将寄存器符号化,在 angr 框架上进行符号执行,通过对符号执行后状态的判断来得到 gadget 的关键信息。

1.3 问题分析

代码重用攻击的实施过程较为繁杂,通常需要一些工具辅助人工来完成 gadget 链的构建,当前已经有很多自动化的构建工具,本文总结 6 种经典 gadget 工具的所用技术和实现方法,如表 1 所示。

表 1 6 种现有工具的 gadget 语义分析技术实现原理

Table 1 The implementation principle of six existing tools' gadget semantic analysis technology

| 工具 | 中间语言 | 所用技术 | 具体实现 |
|--------------|------|---------|-------|
| BARFgadget | REIL | IR 模拟执行 | — |
| ROPgenerator | REIL | 静态分析 | 有向图 |
| PSHAPE | VEX | 静态分析 | 表达式替换 |
| Ropper | REIL | 静态分析 | 表达式替换 |
| angrop | VEX | 动态符号执行 | — |
| BOPC | VEX | 动态符号执行 | — |

以上 gadget 工具采用的 gadget 语义分析技术主要基于静态分析和动态符号执行,一般而言,前者具有更高的效率,后者更容易实现。实际中在进行 gadget 语义分析时,主要面临如下 2 个问题:

1) gadget 语义分析主要服务于 gadget 链的构建,其关注的核心内容与其他领域不同,而在当前学术界公开发表的论文中,与 gadget 语义分析相关的文献资料较少,因此,分析确定 gadget 的语义是首先要解决的问题。

2) 在多数情况下, gadget 的数量可能达到数十万,而单个 gadget 也可能是大段的代码块,需要尽可能地优化 gadget 的语义分析方法,因此,对现有 gadget 语义分析方法进行改进,加快语义分析的进程,也是 gadget 语义分析时需要解决的问题。

针对上述第一个问题,本文分析现有 gadget 工具的源码,根据各工具分析的 gadget 语义,定义 gadget 语义摘要的概念,以明确 gadget 语义分析的内容;针对第二个问题,本文提出一种基于表达式树的 gadget 语义分析技术,用一种表达式树变体描述寄存器和内存读写的表达式信息,以提高语义分析的效率。

gadget 序列的自动化生成需要完成 gadget 的搜索、gadget 语义摘要的计算与 gadget 的选择和拼接等功能。gadget 搜索已经广泛应用于工程实践中,对于 ROP 及其变种^[14-16],采用的算法大多基于 Galileo 方法^[3],很难有优化的空间^[17];对于 gadget 的选择和拼接,由于某一阶段 gadget 的选择可能会影响之后

所有的 gadget 选择,并且使用不同的 gadget 选择策略会产生不同的效果,对 gadget 长度的影响难以估量,因此,很难用一种通用且有效的方法实现自动化;对于 gadget 的语义分析,其内容不够明确,当前 gadget 工具使用的语义分析方法或是设计的结构过于复杂,依托的框架也过于庞大,因此存在优化的可能性。

本文明确 gadget 语义分析的内容,提出一种基于表达式树的方法,以描述寄存器和内存读写的表达式信息。在该方法中,对描述表达式信息的数据结构进行优化,使其能够快速得到表达式所依赖的寄存器。在各个工具实现的搜索功能和语义摘要的计算功能各不相同的情况下,设计一个实验系统进行对比与分析。

表2 6种典型开源 gadget 工具的语义信息

Table 2 Semantic information of six typical open source gadget tools

| 工具 | regw | memr | memw | expr | sp offset | transfer | | |
|--------------|------|------|------|------|-----------|----------|---------|------|
| | | | | | | syscall | libcall | cond |
| BARFgadget | √ | √ | √ | × | × | × | × | × |
| angrop | √ | √ | √ | × | √ | √ | × | × |
| ROPgenerator | √ | √ | √ | √ | √ | × | × | × |
| PSHAPE | √ | √ | √ | √ | √ | × | × | × |
| Ropper | √ | √ | √ | √ | √ | × | × | × |
| BOPC | √ | √ | √ | √ | √ | √ | √ | √ |

在表2中,regw、memr和memw分别表示寄存器写、内存读和内存写操作,expr表示能否为相应的寄存器或内存进行定量分析,生成能够被解析的表达式,sp offset表示栈指针的偏移值,transfer表示控制流转移情况,包括系统调用 syscall、库函数调用 libcall 和条件转移 cond。

由于多数 gadget 工具没有实现 gadget 链构建功能,因此在上述工具中,只有使用符号执行进行语义分析的2个工具 angrop 和 BOPC 对控制流转移情况进行了分析。在实际确定控制流转移情况的过程中,如果目的地址是一个常量,可以通过单条指令的机器码结合指令的地址计算得到;如果目的地址是一个寄存器或内存,可以通过寄存器和内存的表达式得到。

为进一步明确 gadget 语义,本文定义 gadget 的语义摘要为:用于表述 gadget 核心语义和功能的关键信息,包括寄存器和内存的读写情况以及栈指针的偏移。其中,寄存器的读写情况包括被修改的寄存器名、最后一次写入的值及其依赖的寄存器或内存空间,内存的读写情况包括被读取或写入的内存空间的位置、内存地址、写入的值及其依赖的寄存器或内存空间。

2.2 特殊表达式树设计

通过对现有 gadget 工具的源码进行分析,得出多

2 基于表达式树的 gadget 语义摘要计算

2.1 gadget 语义摘要

在实际的代码重用攻击过程中,构建一个可用的 gadget 链并不需要 gadget 蕴含的全部语义信息。事实上,自动化构建 gadget 链通常会采用启发式算法,只需考虑 gadget 副作用的影响和部分关键语义,如系统调用、控制流转移等信息,明确需要分析的 gadget 语义,并对其进行简化或提炼即能够加快语义分析的进程。为了进一步明确构建 gadget 序列过程中需要确定的语义,本文分析常用 gadget 工具的源代码,如表2所示,其中,“√”表示工具分析了语义,“×”表示未分析。

数 gadget 工具对 gadget 进行表达式分析时使用的数据结构不易于处理,尤其对于较长的 gadget,分析效率会降低,原因是现有 gadget 工具表示寄存器数值的方式过于简单,或是使用简单的表达式替换,这使得 gadget 语义分析不能很好地解析寄存器或内存的表达式。此外,现有 gadget 工具使用较为复杂的图结构,在维护寄存器和内存读写情况的过程中需要遍历多个节点,并通过多次计算来确定依赖寄存器或内存的信息,在 gadget 数量较多时其影响较为显著。

gadget 工具能从部分二进制文件(如一些大于 10 MB 的共享库)中搜索到较多的 gadget,这时需要计算大量的语义摘要,因此,要尽可能地加快语义摘要的计算速度。符号执行或模拟执行的方法将耗费较长时间,现有工具所使用的静态分析技术同样存在优化的空间。

为了能够支持多种平台,gadget 工具一般会将 gadget 翻译成中间语言,再对中间语言进行语法分析,在语法分析中较为突出的一个问题就是对数学表达式的描述和分析,通常使用表达式树来解决该问题。为了说明中间语言转换成表达式树的可行性,本文以 VEX、REIL 两类使用最为广泛的中间语言为例,对其中常用的几种指令进行表达式形式转换。

表 3 所示为 VEX 中最常见的 8 种指令及其表达式形式,将 GET、PUT、LD、ST 和 ITE 这 5 种指令转换成赋值表达式,其中,ITE 是条件赋值指令,类似于 C 语言的条件运算符。将算术指令和类型转换指令转换成算术表达式,多数运算操作都是一元或者二元运算,部分浮

点运算操作使用了 3 个或者 4 个操作数,并调用了 VEX 中的辅助函数(helper function),但通常对 gadget 的语义分析只关心通用寄存器,而辅助函数没有副作用,不会影响寄存器的值,因此,不需要精确地定义其表达式,可以将它转换成多叉树的形式。

表 3 VEX 的常见指令及其表达式形式

Table 3 Common instructions of VEX and their expressions

| 名称 | 指令示例 | 目标 | 操作符 | 参数 1 | 参数 2 | 参数 3 | 参数 4 |
|-------|--|------|-------------------------|---------|------|------|------|
| GET | t3=GET:I32(0) | t3 | = | eax | — | — | — |
| PUT | PUT(0)=t1 | eax | = | t1 | — | — | — |
| LD | t2=LDle:I32(t3) | t2 | = | [t3] | — | — | — |
| ST | STle(t3)=t0 | [t3] | = | t0 | — | — | — |
| ITE | t0=ITE(t1,t2,t3) | t0 | = | cond=t1 | t2 | t3 | — |
| CCall | t0=86g_calculate_eflags_c(t1,t2,t3,t4):Itt_I32 | t0 | x86g_calculate_eflags_c | t1 | t2 | t3 | t4 |
| Arth | t0=Add32(t2,t1) | t0 | Add32 | t2 | t1 | — | — |
| Cast | t3=32to8(t2) | t3 | 32to8 | t2 | — | — | — |

表 4 所示为 REIL 中最常见的 7 种指令及其表达式形式,将 STR、LDM、STM、BISZ 和 JCC 这 5 种指令转换成赋值表达式,其中,BISZ 是条件赋值指令,类似于 C 语言的条件运算符,JCC 是条件跳转指令,可以看作是对 IP 的条件赋值。将算术指令和逻辑指令转换成算术表达式,所有的运算操作都是二元运算,可以很容易地转换成树的形式。

表 4 REIL 的常见指令及其表达式形式

Table 4 Common instructions of REIL and their expressions

| 名称 | 指令示例 | 目标 | 操作符 | 参数 1 | 参数 2 |
|-------|-----------------------|----------|-------|----------|--------|
| STR | STR t1/b2,,t2/b2 | t2 | = | t1 | — |
| LDM | LDM 413800/b4,,t1/ b2 | t1 | = | [413800] | — |
| STM | STM t2/b2,,415280/b4 | [415280] | = | t2 | — |
| BISZ | BISZ t0/b4,,t1/b1 | t1 | = | cond=t0 | — |
| JCC | JCC t0/b1,,401000/b4 | ip | = | cond=t0 | 401000 |
| Arth | ADD t1/b4,t2/b4,t0/b8 | t0 | ADD | t1 | t2 |
| Logic | t3=32to8(t2) | t3 | 32to8 | t2 | — |

在传统的表达式树中,每一个叶节点对应一个操作数,每一个非叶节点对应一个数学运算,整个表达式树会对应一个算数表达式,而无需考虑特定的子表达式树。但是,gadget 的语义分析可能涉及多个寄存器或中间变量,多数情况下需要考虑中间变量的表达式,还需要记录表达式对应的目标变量,因此,本文对传统的表达式树进行一些修改,设计一种特殊的表达式树以表示 gadget 的语义,具体为:树中有 Val 和 Expr 两类节点,Val 节点可以是寄存器或内存的初值、常量值和运算符,Expr 节点是一棵子表达式树,每一棵子表达式树对应一条 IR 指令,其根节

点必定是一个代表运算符的 Val 节点,Expr 节点中保存了目标变量的信息,这样一个 gadget 就转化成一棵或多棵表达式树。

由于一个 gadget 可能多次修改某一寄存器或内存空间,因此需要区分每一次修改寄存器的情况,将寄存器或内存的初始值标记为如同“eax_0”的形式,即在寄存器名或内存地址后加下划线“_”和修改次数“0”。以指令“add eax,ebx”对应的 VEX IR 为例,如图 2 所示,可以得到如图 3 所示的表达式树,其中,圆圈代表 Val 节点,方框代表 Expr 节点。

t1 = GET:I32(0) # 取出eax中的值,它是一个32位的整数
t2 = GET:I32(12) # 取出ebx中的值,它是一个32位的整数
t3 = Add32(t3,t2) # 进行32位的加法运算
PUT(0) = t3 # 写回eax

图 2 “add eax,ebx”对应的 VEX 指令

Fig.2 VEX instruction corresponding to “add eax,ebx”

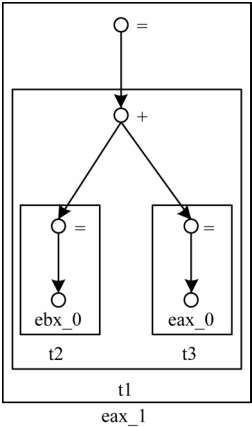


图 3 “add eax,ebx”的 VEX 指令对应的表达式树

Fig.3 The expression tree corresponding to VEX instruction of “add eax, ebx”

与传统的表达式树相同,本文表达式树能够通过访问叶节点来确定表达式的操作数,进而得到 gadget 的副作用信息,获取表达式所依赖的寄存器或内存空间。与传统表达式树的不同之处在于,该表达式树维护了表达式对应的目标变量,并能通过 Expr 节点方便地提取某个中间变量对应的表达式值,这为获取部分变量的约束提供了便利。特别地,当 gadget 的长度较长时,构造的表达式树深度也会增加,在极端情况下会近似成为一条链表,这时确定依赖寄存器的效率会大幅降低。为了能够快速得到依赖寄存器的信息,本文采用以空间换时间的思路,在将 IR 指令转化成表达式树的过程中,为每一个 Expr 节点维护依赖寄存器的信息。

2.3 语义分析的执行流程

本文提出一种基于表达式树的 gadget 语义摘要计算方法,在将二进制转化成中间语言 IR 后,根据中间语言需要满足 SSA(Static Single Assignment)的特点构造 2.2 节所述的表达式树,根据该表达式树生成 gadget 语义摘要,执行流程如图 4 所示。

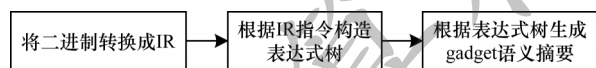


图 4 gadget 语义摘要计算流程

Fig.4 Calculation procedure of semantic summary of gadget

在生成表达式树以后,为每一个寄存器和中间变量构建一个键为名称、值为表达树的键值对,为每一个寄存器构建一个修改次数的计数器,此时获取特定寄存器或中间变量的值只需要先通过键值对访问对应的表达式树,然后解析表达式树即可。在解析树时先通过叶节点访问该寄存器或中间变量的依赖寄存器信息,再将对应的 Expr 节点转换回数学表达式作为约束条件进行求解即可得到对应的寄存器值。对于栈寄存器的偏移值,从栈寄存器对应的 Expr 节点出发,自顶向下遍历 Expr 节点并收集约束条件,即可通过约束求解器求出偏移值。

2.4 SemExpr 的设计与实现

当前有多种方案能够实现 IR 翻译和分析功能,如 S2E、BARF 和 angr 等框架。其中,S2E 使用 QEMU 进行整个计算机模拟,但这需要较多的系统资源;BARF 使用 REIL^[18]作为中间语言,当前 REIL 难以支持很多指令,其中包括 gadget 中比较常见的指令,如 retf,因此,BARF 在功能上略显不足;angr 使用 VEX IR^[19]作为中间语言,相对于 REIL,VEX 支持

的指令集更多,相对于汇编语言,VEX 提取语义信息更快捷,因此,其更适合计算 gadget 的摘要信息,并且已经有 angrop、Ropper 和 PSHAPE 等开源工具,这使得 angr 的实现和评估更为简单。

本文在 Ropper 的基础上构建原型系统 SemExpr,如图 5 所示,其中主要包括 pyvex 和 Z3 两个开源项目。pyvex 是 python 版本的 IR 翻译工具,能够将二进制文件转换成 VEX IR;Z3 是一个高效的 SMT(Satisfiability Modulo Theories)求解器,集成了多种约束求解算法。

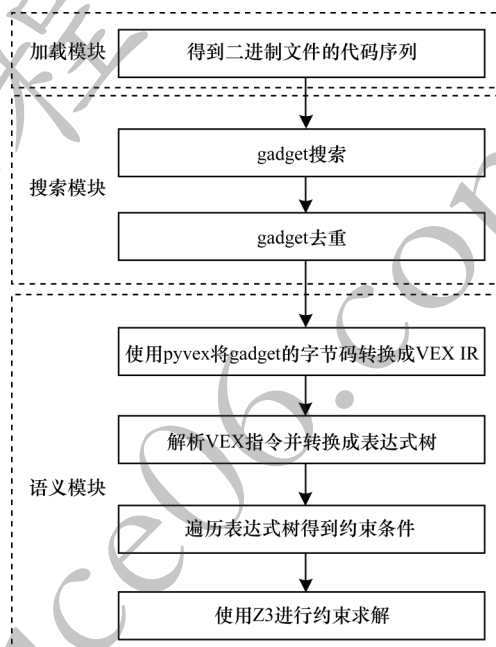


图 5 SemExpr 的结构及执行流程

Fig.5 Structure and execution procedure of SemExpr

SemExpr 由 3 个模块构成,加载模块使用了 Ropper 原有的加载代码,能够读取 PE 文件和 ELF 文件的代码段;搜索模块会根据指定的命令行参数,搜索以 ret、jmp 指令或系统调用结尾的 gadget,并去掉重复的 gadget;语义模块会对 gadget 进行分析,得到 gadget 的表达式树以及各寄存器的表达式,生成约束条件,根据约束条件计算 sp 的偏移值。系统执行流程如下:

1) 加载二进制文件,解析文件头,得到代码段中的字节序列和对应的地址。

2) 根据命令行参数,从第 1 步所得字节序列中搜索具有特定机器码的位置,通过 Galileo 算法得到 gadget,将所有的 gadget 放到一个集合中以去除相同的 gadget。

3) 为每个 gadget 创建一个图结构,利用 pyvex 得到 gadget 的 VEX IR 形式,根据 VEX IR 指令具有 IRStmt、IRExpr 和 IROp 3 层结构的特点,在 IROp 中得到能够被 Z3 解析的表达式形式,在 IRExpr 中构建

单个的表达式树结构并根据表达式形式构建约束条件,在 IRStmt 中将构建的单个表达式树添加到 gadget 对应的图中。

4)找到 sp 对应的表达式树,遍历该表达式树,得到树中的全部约束条件。

5)将约束条件作为输入,使用 Z3 进行求解,得到 sp 的偏移值。

6)返回 gadget 的语义摘要信息。

3 实验与评估

3.1 实验系统的设计与实现

现有 gadget 工具基于 Galileo 算法均实现了各自的 gadget 搜索功能,但其搜索参数,如指令长度限制和字节数限制各不相同,并且在搜索后还定义了不同的筛选策略,因此,它们搜索得到的 gadget 数量也不尽相同。此外,各个工具计算的语义摘要内容略有不同,导致难以对各工具进行对比实验和评估。

针对上述问题,本文设计一个如图6所示的实验系统 gadgetAnalysis,以评估本文设计的 gadget 语义摘要计算方法的性能。该实验系统由搜索模块、转换模块和语义模块构成,其中,搜索模块用于搜索 gadget,在实现时对 gadget 搜索工具 ROPgadget^[20]进行略微修改,使用户能够设定 gadget 长度的上下限;转换模块用于将 ROPgadget 的搜索结果转换成各工具能够分析的数据结构,部分工具会在确定语义前对 gadget 进行筛选,为了提高对比实验的准确性,本文将筛选部分全部去除;语义模块能够对搜索到的 gadget 进行分析,确定 gadget 的语义,实现时将各 gadget 工具的语义分析部分进行提取和修改,只计算本文定义的语义摘要,但是对于 PSHAPE,其使用基于字符串替换的语义分析,最后只能得到嵌套的 VEX 形式的表达式,不能用于 gadget 序列构建。

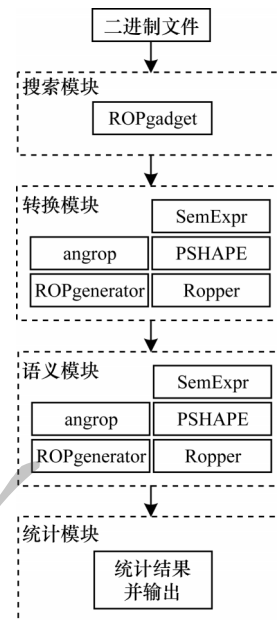


图6 gadgetAnalysis 系统结构

Fig.6 System structure of gadgetAnalysis

3.2 结果与分析

选取 Windows 和 Linux 下常见的共享库和软件,设定 gadget 的指令数为 2~10,统计各工具计算语义摘要的时间以及能够处理的 gadget 数量,进行 10 次实验取平均值得到如表5所示的结果。根据表5的实验结果,从计算时间、能处理的 gadget 数量和能否生成可用的表达式3个方面衡量各工具的性能,结果如表6所示。从表6可以看出,angrop 计算时间最长,由于其使用了符号执行,计算的语义最多,面对一些特殊的指令时反而无法处理,因此能处理的 gadget 数量最少;ROPgenerator 虽然计算时间最短,但受限于 REIL 指令,其能处理的 gadget 数量较少;PSHAPE 计算时间较短,能处理的 gadget 数量也最多,但其生成的表达式不能被解析;Ropper 能处理的 gadget 数量较多,但其计算时间较长;相较于 Ropper, SemExpr 在计算时间上取得了明显改进,其能取得较好的效果。

表5 各 gadget 工具计算语义摘要的时间和能够处理的 gadget 数量

Table 5 The time that each gadget tool requires to calculate the semantic summary and the number of gadgets that can be processed

| 文件 | 数量 | angrop | | ROPgenerator | | PSHAPE | | Ropper | | SemExpr | |
|---------------|--------|-------------|--------|--------------|--------|-------------|--------|-----------|--------|-----------|--------|
| | | 时间/s | 数量 | 时间/s | 数量 | 时间/s | 数量 | 时间/s | 数量 | 时间/s | 数量 |
| kernel32.dll | 10 908 | 1 074.478 6 | 7 923 | 109.622 | 7 076 | 133.799 0 | 9 794 | 564.677 | 10 885 | 370.366 | 10 882 |
| msvcrt.dll | 14 304 | 968.265 6 | 5 778 | 101.928 | 10 221 | 199.313 0 | 14 290 | 720.646 | 14 219 | 462.436 | 14 219 |
| AcroRd32.exe | 1 251 | 118.242 9 | 721 | 30.312 | 885 | 17.055 0 | 1 224 | 64.025 | 1 251 | 41.574 | 1 251 |
| libc6_2.19.so | 81 430 | 2 380.168 5 | 13 823 | 766.803 | 46 581 | 1 118.321 0 | 77 614 | 4 099.074 | 80 788 | 2 735.171 | 80 680 |
| libm-2.23.so | 27 295 | 833.192 8 | 6 961 | 177.370 | 8 674 | 268.325 0 | 27 235 | 1 045.432 | 26 625 | 685.817 | 26 554 |
| proftpd | 13 523 | 1 124.404 1 | 6 415 | 140.934 | 8 918 | 166.875 0 | 13 396 | 630.223 | 13 383 | 403.489 | 13 380 |
| sudo | 1 558 | 140.423 0 | 815 | 26.632 | 1 193 | 22.153 1 | 1 558 | 80.745 | 1 539 | 50.800 | 1 539 |

表6 各gadget工具性能对比

Table 6 Performance comparisons of each gadget tool

| 工具 | 计算时间 | 能处理的 gadget数量 | 能否生成可 解析的 表达式 |
|--------------|------|------------------|---------------------|
| angrop | 最长 | 最少 | √ |
| ROPgenerator | 最短 | 少 | √ |
| PSHAPE | 短 | 最多 | × |
| Ropper | 长 | 多 | √ |
| SemExpr | 中 | 多 | √ |

4 结束语

本文针对当前自动化方案构造 gadget 序列时效率较低的问题,对主流 gadget 工具计算的语义信息进行提炼,并定义 gadget 语义摘要的概念,提出一种基于表达式树的 gadget 语义摘要计算方法。以 VEX、REIL 2 种中间语言为例,说明中间语言转换成表达式树的可行性,并实现一种原型系统 SemExpr。在此基础上,本文设计用于测量 gadget 语义摘要计算效率的实验系统 gadgetAnalysis,仿真结果验证了 SemExpr 在计算 gadget 语义摘要时具有较好的性能。在对 gadget 进行语义分析后,如何根据语义分析的信息来拼接 gadget 以构建具有特定功能的 gadget 链,是代码重用攻击实现自动化的瓶颈之一。对于该问题,目前只能通过一些启发式算法来构建具备简单功能的 gadget 序列,设计算法构建具有指定复杂逻辑的 gadget 序列将是下一步的研究方向。

参考文献

- [1] LÁSZLÓ ERDDI, JSANG A. Exploit prevention, quo vadis? [C]//Proceedings of International Workshop on Security and Trust Management. Washington D. C., USA; IEEE Press, 2017: 15-23.
- [2] NERGA L. Advanced return-into-lib(c) exploits (PaX case study) [EB/OL]. [2019-10-10]. <http://phrack.org/issues/58/4.html>.
- [3] SHACHAM H. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86) [C]//Proceedings of ACM Conference on Computer and Communications Security. New York, USA; ACM Press, 2007: 552-561.
- [4] CHECKOWAY S, SHACHAM H. Escape from return-oriented programming: return-oriented programming without returns (on the x86) [EB/OL]. [2019-10-10]. <https://hovav.net/ucsd/dist/noret.pdf>.
- [5] SCHWARTZ E J, AVGERINOS T, BRUMLEY D. Q: exploit hardening made easy [C]//Proceedings of USENIX Security Symposium. San Diego, USA; USENIX Association, 2011: 25-41.
- [6] BRUMLEY D, SCHWARTZ E J. BAP: a binary analysis platform [C]//Proceedings of International Conference on Computer Aided Verification. Washington D. C., USA; IEEE Press, 2011: 463-469.
- [7] FLANAGAN C, SAXE J B. Avoiding exponential explosion [J]. ACM SIGPLAN Notices, 2001, 36(3): 193-205.
- [8] JAGER I, BRUMLEY D. Efficient directionless weakest preconditions [EB/OL]. [2019-10-10]. https://www.cylab.cmu.edu/_files/pdfs/tech_reports/CMUCyLab10002.pdf.
- [9] DIJKSTRA E. Dijkstra on hamming's problem [M]. Berlin, Germany; Springer, 1976.
- [10] HEITMAN C. BARF: a multiplatform open source binary analysis and reverse engineering framework [EB/OL]. [2019-10-10]. <https://raw.githubusercontent.com/programa-stic/barf-project/master/doc/papers/barf.pdf>.
- [11] FOLLNER A, BARTEL A, PENG H, et al. PSHAPE: automatically combining gadgets for arbitrary method execution [C]//Proceedings of International Workshop on Security and Trust Management. Washington D. C., USA; IEEE Press, 2016: 212-228.
- [12] BOYAN-MILANOV. ROPgenerator [EB/OL]. [2019-10-10]. <https://github.com/Boyan-MILANOV/ropgenerator>.
- [13] ISPOGLOU K K, ALBASSAM B, JAEGER T, et al. Block oriented programming: automating data-only attacks [C]//Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. New York, USA; ACM Press, 2018: 1868-1882.
- [14] SNOW K Z, MONROSE F, DAVI L, et al. Just-in-time code reuse: on the effectiveness of fine-grained address space layout randomization [C]//Proceedings of 2013 IEEE Symposium on Security and Privacy. Washington D. C., USA; IEEE Press, 2013: 574-588.
- [15] BLETSCH T, JIANG 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. New York, USA; ACM Press, 2011: 30-40.
- [16] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns [C]//Proceedings of the 17th ACM Conference on Computer and Communications Security. New York, USA; ACM Press, 2010: 559-572.
- [17] SADEGHI A, NIKSEFAT S, ROSTAMIPOUR M. Pure-Call Oriented Programming (PCOP): chaining the gadgets using call instructions [J]. Journal of Computer Virology and Hacking Techniques, 2018, 14(2): 139-156.
- [18] DULLIEN T, PORST S. REIL: a platform-independent intermediate representation of disassembled code for static code analysis [EB/OL]. [2019-10-10]. <http://www.zynamics.com/downloads/csw09.pdf>.
- [19] NETHERCOTE N, SEWARD J. Valgrind: a framework for heavyweight dynamic binary instrumentation [J]. ACM SIGPLAN Notices, 2007, 42(6): 89-90.
- [20] SALWAN J. ROPgadget [EB/OL]. [2019-10-10]. <https://github.com/JonathanSalwan/ROPgadget>.

编辑 吴云芳