基于 ARMv8 架构 ROP 自动构造框架

赵利军 彭 城
(徐州工程兵学院作战实验中心 江苏 徐州 221000)

摘 要 为了在 ARM 公司最新发布的首款支持 64 位处理器的 ARMv8 架构上实现 ROP(Return_Oriented Programmig)技术,提高 ROP 攻击效力,设计了 ARMv8 架构上的 ROP 自动构造工具。首先对已有的 ARM 架构下 gadget 搜索工具进行扩展,使之支持 ARMv8 架构下可用指令序列的搜索,并把这些指令序列存储在 gadget 库中,其次采用优化策略对 gadget 库进行优化。然后利用语义等价性找出与 shellcode 语义相同的 gadget 链,并利用寄存器连通性解决语义断层和寄存器冲突等问题。最后完成 gadget 的自动串联形成具有特定行为的 ROP 链。通过利用 ROP 自动构造工具对网站"exploit. db"中大量的 shellcode 进行自动构造,证明了工具具有良好的攻击效力,增强了 ROP 攻击的实用性。

关键词 ROP 指令序列 ARMv8 寄存器连通性 语义断层 寄存器冲突

中图分类号 TP309.1 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2017.08.057

ROP AUTOMATIC CONSTRUCTION FRAMEWORK BASED ON ARMV8 ARCHITECTURE

Zhao Lijun Peng Cheng
(*Operational Experiment Center, Xuzhou Engineering Corps College, Xuzhou 221000, Jiangsu, China*)

Abstract To support Return Oriented Programming technology on 64 bit ARMv8 architecture, we designed the ROP automatic construction tool. This tool can improve the effectiveness of the ROP attack. First, we extend the gadget searching tools, so the instruction sequence search can be used in the ARMv8 architecture. And we store the instruction sequence into gadget library. Secondly, we use the optimization strategy to optimize the gadget library. Then we use semantic equivalence to find the same gadget chain as the shellcode semantics. In addition, we use register connectivity to solve semantic faults and register conflicts and other issues. Finally, we need to complete the gadget automatic series to form a specific behaviour of the ROP chain. By using the ROP automatic construction tool to construct shellcode in the website "exploit. db", it proves that the tool has good attack effectiveness and enhances the practicability of ROP attack.

Keywords ROP Instruction sequence ARMv8 Register connectivity Semantic fault Register conflict

0 引 言

数据执行保护和签名等技术有效地阻止了代码注入式攻击。2005 年 Krahmer 等提出了一种借用代码攻击方法,该方法不再需要向内存中注入恶意代码,而是利用代码段中的代码实现攻击。return-into-libc[1] 技术是一种典型的借用代码攻击,通过劫持控制流,跳转到 C 语言函数库 libc,复用 libc 中已有的函数。但是 return-into-libc 攻击只能顺序调用函数,不能实现图灵完备的行为,如分支操作、循环操作等。

为了弥补 return-into-libc 攻击的局限性,2007 年 Shacham[2] 第一次提出了 X86 平台上的返回导向编程技术 ROP。返回导向编程攻击的方式不再局限于将漏洞程序的控制流跳转到库函数中,而是利用库函数或可执行文件的指令代码片段实现攻击,将复用的代码粒度从 return-to-libc 的函数级别缩小到指令序列。自 2007 年 ROP 概念被提出后,关于 ROP 攻击的研究引起了研究人员的广泛关注,他们先后在各个平台上对 ROP 攻击进行而来实验。例如 ARM 平台,SPARC 平台和 AVR 平台。Kornau[3] 首次在 ARM 架构上实现了 ROP 攻击并验证了图灵完整性,而且提出了一套基于 REIL 语言的

收稿日期:2016-08-29。**赵利军**,讲师,主研领域:软件开发,信息安全。**彭城**,副教授。

自动构建 ROP 链的算法。2008 年,Roemer[4] 等在 SPARC 平台上验证了 ROP 图灵完整性。卢森堡大学的 Ralf-Philipp Weinmann 利用 ROP 技术实现了 iOS 系统的入侵[5]。

ROP 技术至关重要的一环就是可用指令序列的自动搜索,本文中将这些指令序列定义为 gadget。由于库文件中的代码的数量非常巨大,所以手工搜索 gadget 将会非常的耗时,这将严重降低 ROP 技术的效率。2011 年,Schwartz[6] 等提出了一种 X86 平台上 gadget 自动搜索算法,其定义了内部语言 QooL 来实现漏洞程序到 ROP payload 之间的转换,但是其缺陷是定义的语义只有内存操作和逻辑操作,没有涉及到条件执行和循环。然而这些技术只适用于 X86 架构,并不适用于其他架构,特别是 RISC 架构。2010 年,Kornau 等在 ARM 架构上实现一种基于中间语言 REIL 的 ARM gadget 自动搜索算法,首先该算法定位到分支指令,然后根据语法树算法把一条指令翻译成 REIL 语言,最后组合信息成为有用的 gadgets。Schwartz 等提出的自动构造 ROP 攻击的框架 Q 是基于短指令序列的,不能绕过 Chen[7] 等提出的防御机制。2014 年,Yang[8] 等在 Schwartz 基础上提出了基于长指令序列的自动构造框架,成功地绕过了 Ping、Chen 等人的防御机制。

ARMv8 架构为了获得低功耗高效率的 64 位计算优势,引入了一个全新的指令字长为 32 位的 64 位指令集 A64。上文论述的 gadget 搜索算法或是针对 ARMv7 架构,或是针对 X86 架构,并不适用于 ARMv8 架构。并且目前针对 ROP 攻击的大部分研究仍停留在 gadget 自动搜索阶段,还需要很多的人工参与,严重降低了 ROP 攻击的效率。为了解决以上技术缺陷,本文设计了一种 ARMv8 ROP shellcode 自动构造工具,该工具首先搜索到 ARMv8 架构下的可用指令序列,然后在可用指令序列中搜索到与 shellcode 语义相同的 gadget,自动完成 gadget 的串联,提高了 ROP 攻击的效率。

1　背　景

1.1　ARMv8 架构的差异性分析

ARMv8 架构在指令集设置、寄存器使用和函数调用等方面和我们熟知的架构(X86、ARM)截然不同。ARMv8 架构有两种工作状态:AArch64 和 AArch32。在这两种工作状态下指令集仍然都是 32 位的,但是指令的寻址范围不同了:AArch64 状态下支持 64 bit 的地址空间,AArch32 状态下支持 32 bit 的地址空间 ,且在每种工作状态下 ARMv8 都有 31 个通用寄存器[9]。本文将主要对 AArch64 状态下 ARMV8 架构的差异性。

1.1.1　子函数调用规则

ARMv8 架构摒弃了 ARMv7 架构下的 7 种工作模式:1 个用户模式和 6 个特权模式。ARMV8 架构下采用 4 种工作模式:EL0 – EL3,EL0 相当于用户模式,下面我们将主要对非特权模式进行分析。

ARMv8 架构下当发生子函数调用时,程序将 PC 中下一条指令的地址保存到寄存器 X30 中,然后跳转到 PC 中的地址去执行,当函数返回时程序将会跳转到 X30 寄存器中的地址处执行。当子函数的参数的个数不大于 8 个的时候,则参数由寄存器 X0 – X7 进行参数传递,如果子函数的参数的个数大于 8 个,则大于 8 个参数之外的参数通过栈进行传递。根据 ARMv8 架构的子函数的调用规则,在 ARMv8 架构中的寄存器分配规则如表 1 所示。

表 1　函数调用时寄存器分配规则

寄存器	作用	ARMv7 对比	X86 对比
X30	存储函数的返回地址	R14(LR)	–
X29	栈帧寄存器	R11(FP)	EBP
PC	保存下一条待执行的指令	R15(PC)	EIP
SP|WSP	栈指针寄存器	R13(SP)	ESP
X28 – X8	存放本地变量	R10 – R4	EAX. EBX.....
X7 – X0	子函数参数传递寄存器	R3 – R0	栈传递

从表 1 中可以看出 ARMv7 架构为 15 个通用寄存器,其中 LR 用来存放函数的返回地址,当函数从被调函数返回时从 LR 寄存器读取返回地址跳转到调用函数继续执行。ARMv8 架构下为 31 个通用寄存器,栈指针寄存器 SP 独立于 31 个通用寄存器之外,其中 X30 相当于 ARMv7 架构下的 LR 寄存器,当函数返回时,程序将会跳转到 X30 寄存器中的地址执行。

ARMv8 架构下函数调用时的内存布局相比于 ARM 以前的版本有了较大的改变,函数调用时的栈结构如图 1 所示。

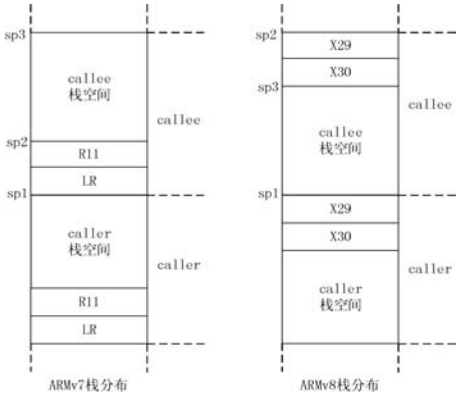


图 1　函数调用栈的分布情况对比图

左图为 ARMv7 的情形。caller 调用 callee 之前，栈指针位于 sp1 位置；进入 callee 后，将栈基址寄存器 R11 和程序链接寄存器 LR 分别压栈，LR 寄存器保存的是 callee 的返回地址。接着将栈指针减去一个常数（由编译器根据具体情况而定），假设其位于 sp3 位置。sp2 - sp3 之间的空间为 callee 的栈空间。

右图为 ARMv8 的情形。caller 调用 callee 之前，栈指针位于 sp1 位置；紧挨着 sp1 位置的两个寄存器 X29 和 X30 分别存放的是 caller 栈基址和返回地址。进入 callee 函数后，首先会将栈指针减去一个常数（由编译器根据具体情况而定），假设位于 sp2 位置，然后将 X29 和 X30 寄存器分别压栈，此时栈指针位于 sp3 位置，X30 寄存器保存的是 callee 的返回地址；sp1 - sp3 之间的空间为 callee 的栈空间。

1.1.2 ARMv8 架构分支指令

通过对 ARMv8 架构指令系统的研究[9]，总共有 5 类跳转指令可以影响程序的执行流。其功能如表 2 所示，当处理器在 AArch64 位的工作状态下时，表中的寄存器就写成 Xm，当工作在 AArch32 状态下时寄存器就写成 Wm。

表 2 ARMv8 架构分支指令描述表

指令类型	汇编形式		功能描述
条件跳转指令	B. cond label		如果 cond 条件为真，跳转到 label 处执行；
	CBNZ Xn|Wn, label		如果寄存器中值不为 0，跳转到 label 处执行；
	CBZ Xn|Wn, label		如果寄存器中值为 0，跳转到 label 处执行；
	TBNZ Xn|Wn, #uimm6, label		如果寄存器中的第 uimm6 位不为 0，跳转到 label 执行；
	TBZ Xn|Wn, #uimm6, label		如果寄存器中的第 uimm6 位为 0，跳转到 label 执行
非条件跳转	直接跳转	B label	跳转到 label 处执行；
		BL label	跳转到 label 处执行，并将返回地址保存到 X30 寄存器中；
	间接跳转	BLR Xm	跳转到寄存器地址处执行，将返回地址保存到寄存器 X30；
		BR Xm	跳转到寄存器地址处执行，提示 CPU 不是子函数的返回；
		RET {Xm}	跳转到寄存器地址处执行，提示 CPU 是子函数的返回

ARMv8 架构上的可用指令序列都是以支持寄存器间接跳转指令为结尾的。从表 2 中可以看出，这些指令主要包括三类：BLR Xm、BR Xm 和 RET {Xm}。其中，ARMv8 中 BLR Xm 和 BR Xm 主要用于分支跳转或函数调用，BLR 指令时需要将返回地址保存到寄存器 X30 中，而 BR 指令不需要将返回地址保存到寄存器 X30 中；RET {Xm} 为函数返回指令，当 Xm 省略时默认使用 X30。本文中主要使用以 RET {Xm} 指令为结尾的指令序列。

1.2 ARMv8 ROP 攻击原理概述

所谓的 ROP 攻击就是利用库文件或可执行文件中的短指令序列完成的攻击，ARMv8 架构一次 ROP 攻击的原理如图 2 所示。

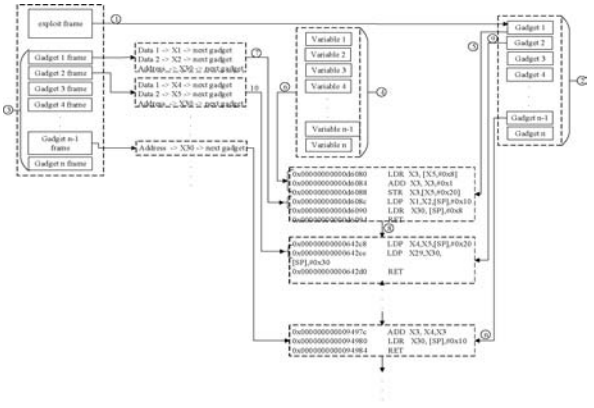


图 2 ARMv8 ROP 攻击概况图

其中各部分完成的功能如下：

① 攻击者通过一次漏洞利用将 gadget 框架注入到用户控制的栈中，并控制程序使之跳转到 gadget 1 执行。

② 所有的位于漏洞程序运行映像内存中的 gadget 序列，运行时映像包括漏洞程序本身和被漏洞程序加载的所有的库文件。

③ gadget 框架通过栈将变量传递给下一个 gadget，每个 gadget 框架包括该 gadget 需要的源数据和下一个被期望执行的 gadget 的首地址。

④ 被存储在专用内存段中的可以变多个 gadget 访问的变量，这个内存区域可以被每个 gadget 进行读写访问。

⑤ 一个 gadget 示例，本文把其命名为 gadget1。

⑥ 程序跳转到 gadget 1 执行后，gadget 1 的前两条指令访问专用内存区域[X5 + 0x8]和[X5 + 0x20]内存位置。

⑦ gadget 框架为 gadget1 中的 LDP 指令提供输入，X1 和 X2 寄存器被设置为保存在 gadget 框架中的值，X30 寄存器也被设置为下一个期望执行的 gadget 的首地址。

⑧ gadget1 通过 RET 执行使程序跳转到 gadget2 执行。

⑨ gadget2 的一个示例。

⑩ 寄存器 X4 和 X5 被设置为 gadget2 框架中的值,X30 寄存器被设置为 gadget3 的首地址,然后程序跳转到 gadget3 执行。依次循环下去直到所有的 gadget 执行完毕,即完成一次 ROP 攻击。

2　ARMv8 ROP shellcode 自动构造工具

本文中设计的 ROP shellcode 自动构造框架如图 3 所示。其主要包括三个部分:shellcode 分析框架、gadget 自动搜索框架和 gadget 自动构建框架。Shellcode 分析框架完成 shellcode 语义的提取和优化,gadget 自动搜索框架实现 ARMv8 架构下可用指令序列的搜索,gadget 自动构建框架通过分析 shellcode 的语义,在 gadget 库中搜索与 shellcode 具有语义的 gadget,以完成 gadget 的自动串联。

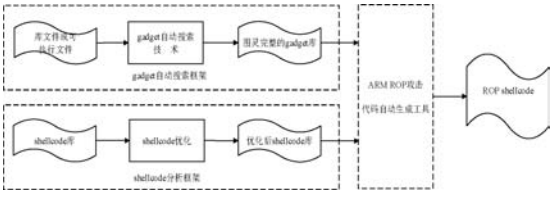
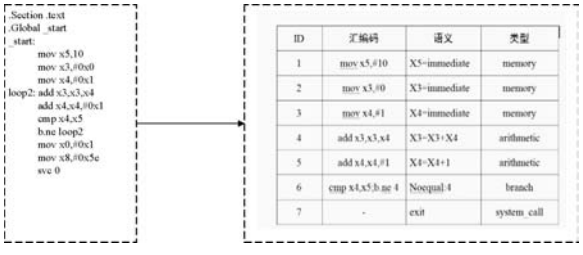


图 3　ARMv8 gadget 自动搜索框架

2.1　shellcode 分析模块

Shellcode 分析框架主要用于 shellcode 的优化和语义提取。对用于输入的 shellcode,该框架首先将 shellcode 转换为 LLVM 中间语言,然后在中间语言级对 shellcode 进行优化和语义提取。其中优化主要包括冗余指令的消除、指令合并等。语义提取主要是提取 shellcode 语句的操作类型。Shellcode 分析模块将会加快 gadget 自动生成模块的效率。图 4 显示了 shellcode 分析模块的执行结果,其中左半部分为完成一个从 1 加到 10 的求和运算,右半部分为经过 shellcode 分析模块分析后的结果。



```
.Section .text
.Global _start
_start:
        mov x5,10
        mov x3,#0x0
        mov x4,#0x1
loop2: add x3,x3,x4
        add x4,x4,#0x1
        cmp x4,x5
        b.ne loop2
        mov x0,#0x1
        mov x8,#0x5e
        svc 0
```

ID	汇编码	语义	类型
1	mov x5,#10	X5=immediate	memory
2	mov x3,#0	X3=immediate	memory
3	mov x4,#1	X4=immediate	memory
4	add x3,x3,x4	X3=X3+X4	arithmetic
5	add x4,x4,#1	X4=X4+1	arithmetic
6	cmp x4,x5,b.ne 4	Noequal:4	branch
7	-	exit	system_call

图 4　shellcode 分析框架结果

2.2　gadget 自动搜索框架

gadget 链是 ROP 技术最重要的一部分,所谓 gadget,是指在一个程序中能搜索到的具有某一特定功能的可用指令序列。该指令序列必须满足一定的条件:

(1) gadget 中包含的指令条数通常很短,在个位数以内;

(2) gadget 的末尾都是一个跳转指令;

(3) 每个 gadget 完成某一功能,如 mov/add/sub/and/neg 等。

本文中的 gadget 搜索框架将主要搜索以 RET 指令结尾的指令序列。通过统计发现 ARMv8 架构通常使用 X30 寄存器作为 RET 指令的目的地址寄存器。其指令的编码为"\xc0\x03\x5f\xd6"。所以本文的搜索算法将主要在目标二进制文件对特征码"\xc0\x03\x5f\xd6"进行搜索。

当定位到分支指令之后,开始以分支指令开始逆向遍历,其搜索算法如图 5 所示,图 5 给出的自动搜索算法遍历库文件或可执行文件的整个代码段。算法的第 4 行通过对 ARMv8 架构 ELF 文件格式的分析,定位到代码段。该算法的第一层循环用来搜索整个库文件来定位分支指令,当搜索到"\xc0\\x03\x5f\xd6"特征码时用第二层循环开始逆向 4 字节遍历,并把相应的指令保存到 G. gadget_binary[ ]中,当完成一个指令流的搜索后,把该指令流的首地址保存到 G. address 中。当发生以下情况时算法停止搜索:① 当遇到返回指令 RET 时;② 当 G. gadget_binary[ ]中的指令的条数大于搜索深度 maxdepth 时。

Gadget Searching Algorithm

```
1 /*  textseg_length: the size of the text segment of the libraries or
    executables;  maxdepth: the maximum instruction number for each gadget;
    input:*.so,maxdepth   output:G (gadget_logic_struct sequences)*/
2: Search_branch_sequence(*.so,maxdepth) {
3:     int i=0;
4:     check *.so //找到库文件的代码入口点
5:     for (pos=0;pos<textseg_length-1;pos=pos+4){
6:         if( the instruction at pos is "RET")
7:           {put the instruction at pos into G[i].gadget_binary[];
8:            for (step from1 to maxdepth){
9:              if ( the instruction insn at (pos-step*4) is  not "RET"  )
10:                 put current instruction insn into G[i].gadget_binary[];
11:             else break;}
12:             put address of the instruction at (pos-maxdepth*4) into G[i].address;
14:           i=i+1;
      }}
```

图 5　ARMv8 gadget 自动搜索算法

算法中的 maxdepth 为用户预先设置好的一个阈值,这个值的设置是必要的,且其值一般设置为个位数,本文中将其值设置为 4。如果不设置阈值或者该值过大会给指令序列带来巨大的边界影响。

2.3　gadget 自动构造模块

Gadget 自动构造模块通过对 shellcode 分析模块分析结果进行分析,在 gadget 自动搜索模块搜索出的图灵完成的 gadget 库中匹配到与 shellcode 语义相同的 gadget。其框架如图 6 所示,其中语义匹配器通过分析 shellcode 的语义在 gadget 库中找出所有的与 shellcode 语义

具有相同语义的 gadget。Gadget 选择器根据不同的策略对 gadget 库进行排序,然后选择最优的 gadget。Gadget 优化器主要解决语义断层和寄存器冲突等问题。

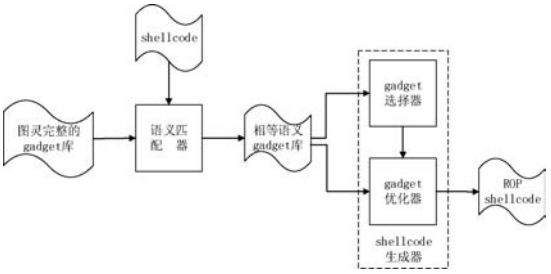


图 6 gadget 自动构造模块

2.3.1 gadget 选择器

本文中主要是根据栈指针 SP 偏移值的大小进行排序。语义匹配器生成的 gadget 中的所有的 gadget 都具有相同的语义,但是其栈指针 SP 的偏移却是不同的。例如 gadget"add x3,x2,x5;ldr x30,[sp],#0x8;ret"和 gadget"add x3,x2,x5;ldp x29,x30,[sp],#0x10;ret",这两个 gadget 完成的功能都是将寄存器 X2 和寄存器 X5 中内容相加,结果保存到寄存器 X3 中,但是第一个 gadget 中 SP 调整了 8 个字节,而第二个 gadget 中 SP 调整了 16 个字节,本文中将会保留第一个 gadget。本文采用线性扫描的方法,从 gadget 集的开始位置遍历整个 gadget 集,根据 SP 偏移的大小采取冒泡排序算法对 gadget 进行排序,当遍历结束时,gadget 库中的 gadget 将是按照栈指针 SP 偏移值从小到大排序,然后选取那个 SP 调整最小的 gadget 作为 gadget 链中的一个。这样在进行 ROP 攻击时,会节省非常多的栈空间,提高了 ROP 攻击的效率。将来本文还会对排序策略进行扩展,从而构造出更优的 gadget 链。

2.3.2 gadget 优化器

虽然 2.3.1 节中已经选择除了最优的 gadget,但是其仍然存在以下两个问题。第一,语义分析器有可能搜索不到与 shellcode 语义相同的 gadget;而且 2.3.1 结构搜索出的 gadget 可能存在叶子 gadget,这将影响 gadget 的串联;这些都将导致 gadget 语义断层。第二,gadget 选择器生成的 gadget 链中存在寄存器冲突。Gadget 优化器主要解决这两个问题。

1)语义断层

语义断层主要包括两个方面:叶子 gadget 和 shellcode 语义丢失。

叶子 gadget:所谓的叶子 gadget 就是指那些无法控制目的地址寄存器的 gadget,例如 gadget{add x0,x2,x4;ret}就是一个叶子 gadget,因为在 RET 指令之前没有一条汇编指令能够对 X30 寄存器进行控制,这将影响该 gadget 与下一条期望执行的 gadget 的串联。本文将使用一个跳板 gadget 进行解决,跳板 gadget 的主要功能是对 X30 寄存器进行设置,然后使用通用寄存器来作为目的地址寄存器实现 gadget 的串联,例如 gadget1{ldr x6,[sp],#0x8;ldr x30,[sp],#0x10;bx r6}就是一个跳板 gadget。当遇到叶子 gadget 时,就在该 gadget 之前插入跳板 gadget1,将 R6 寄存器设置为叶子 gadget 的首地址,将 X30 寄存器设置为叶子 gadget 执行完毕后期望执行的 gadget 的地址。这样当跳板 gadget1 执行完毕后,将会使用 BX R6 指令跳转到叶子 gadget 执行,叶子 gadget 最后的 RET 指令完成与后续 gadget 的串联。但是在使用跳板 gadget 时,一定要注意不要破坏 R6 寄存器中的值,必须存放叶子 gadget 的首地址。

shellcode 语义丢失:shellcode 语义丢失是指在 gadget 库中找不到与 shellcode 具有相同语义的 gadget。本文通过分析 gadget 自动搜索模块生成的 gadget 库,证明了寄存器的连通性,即寄存器间是两两可达的。然后通过利用寄存器的连通性实现寄存器的替换,在替换过程中,本文将尽量替换最少的寄存器以解决 shellcode 语义丢失问题。例如假设在 2.1 节的求和例子中,本文假设语句"add x3,x3,x4"将会造成 shellcode 语义丢失,即在 gadget 库中搜索不到与该语句语义相同的 gadget。那么本文将会按照寄存器最少替换原则,首先对目的寄存器 X3 寄存器进行替换,直到在 gadget 库中找到语义相同的 gadget,当使用寄存器 X1 对 X3 进行替换时,则在 gadget 库中找到了与其语义相等的 gadget1"add x1,x3,x4;ldp x29,x30,[sp],#0x20;ret",然后在 gadget1 之后增加 gadget2"mov x3,x1;ldr x30,[sp],#0x8;ret",将 gadget1 与 gadget2 串联到一起将会完成与语句"add x3,x3,x4"语义相同的 gadget,解决语义断层问题。如果对目的寄存器替换完毕后,仍然无法发现 gadget,那么就对源寄存器进行替换,按照寄存器替换由少到多的原则一直替换下去,直到找到语义相等的 gadget。

2)寄存器冲突的解决

寄存器冲突指存在两个 gadgets:G1 和 G2,G1 存储临时数据到寄存器 A,G2 需要初始化寄存器 A 以完成相应的操作,G1 和 G2 之间就存在寄存器冲突。本文将首先寻找一个栈安全位置,该位置不会影响 gadget 的控制结构,然后在两个 gadget 之间增加一个内存存储 gadget 将临时数据保存到栈安全位置,等到使用时再将其取出。例如"add x0,x1,x3;ldr x30,[sp],#0x10;ret"和"mov x0,#0x8;ldp x29,x30,[sp],#0x20;ret"这两个 gadget,第一个 gadget 的功能是将寄存器 X1 和 X3 相加之后的结果保存到寄存器 X0,这将影响

X0 寄存器的正常使用,那么本文就在这两个 gadget 之间增加一个 store gadget{str x0,[x5];ldp x29,x30,[sp],#0x20;ret}。首先将 X0 寄存器中的值保存到栈安全位置,等到使用时再从相应的位置取出,以消除寄存器冲突。其中 X5 寄存器保存的是栈安全位置的基址。

3　实验与评估

3.1　ARMv8 shellcode 实例

实验环境: Linaro ARMv8 Linux,其版本号为 Linux 3.6.0-1-Linaro-vexpress64。Linaro 是 ARM 公司授权商,其为 ARMv8 开发的工具链和快速模型虚拟平台能从 ARM 官网下载[11]。

本节将给出 Linaro ARMv8 Linux 上实现的 ROP 实例。该实例的 ROP shellcode 欲实现的功能是通过复用 libc.so.6 中的 write 函数在终端输出字符串"ARMv8 exploit",然后复用 libc.so.6 中的 exit 函数使程序离开。

ROP shellcode 对应的内存布局及工作原理如图 7 所示。图的左侧给出 ROP 的内存安排和内存地址的后 12 位,图的右侧的虚线代表指令序列的执行顺序。

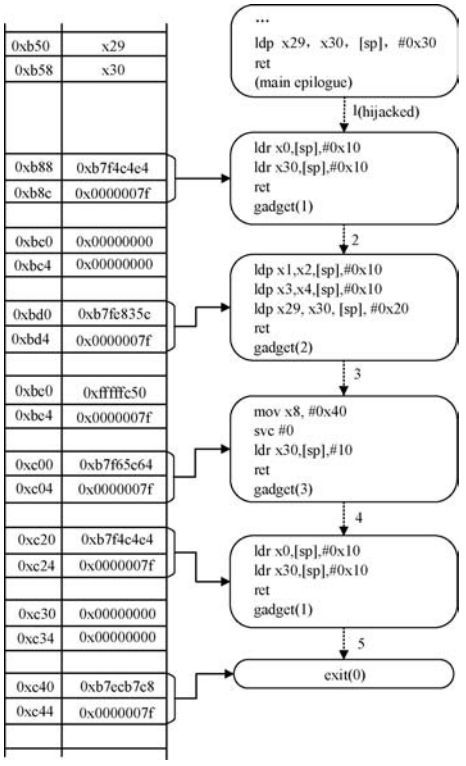


图 7　ROP 攻击实例原理图

通过一次漏洞利用,将返回地址用 gadget(1)的返回地址改写,程序返回时将会跳转到本文设定的指令序列,程序依次执行 gadget(1)和 gadget(2)。在 gadget(1)和 gadget(2)执行完毕之后,X0、X1 和 X2 三个寄存器已经被设置为 write 系统调用所需要的三个参数,gadget(3)将 write 系统调用号 0x40 传递给寄存器 0x40,然后执行 write 系统调用在终端输出字符串"ARMV8 exploit"。通过控制 write 系统调用的返回地址,write 系统调用执行完毕后程序将会跳转到 gadget(1)执行。gadget(1)将 exit 系统调用的参数和入口地址分别传递给寄存器 X0 和 X30,最后通过执行 exit 系统调用使程序离开。在内存中,本文没有用到的闲置内存全部用字符 A 填充。

```
0x7fb7f4c4e4 F84107E0 ldr x0,[sp],#0x10
0x7fb7f4c4e8 F84107FE ldr x30,[sp],#0x10
0x7fb7f4c4ec D65F03C0 ret
gadget(1)
0x7fb7fe835c A8C10BE1 ldp x1,x2,[sp],#0x10
0x7fb7fe8360 A8C113E3 ldp x3,x4,[sp],#0x10
0x7fb7fe8364 A8C27BFD ldp x29, x30, [sp], #0x20
0x7fb7fe8368 D65F03C0 ret
gadget(2)
0x7fb7f65c64 D2800808 mov x8, #0x40
0x7fb7f65c68 D4000001 svc #0
0x7fb7f65c6c F84027FE ldr x30,[sp],#0x10
0x7fb7f65c70 D65F03C0 ret
gadget(3)
```

通过对上面 gadget 链的分析,生成的 ARMv8 ROP shellcode 的控制结构内容如下:

```
char shellcode[] =
"\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41"
"\x41\x41\x41\x41\x41\x41\x41\x41\xe4\xc4\xf4\xb7\x7f\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x41\x41\x41\x41\x41\x41\x41\x41"
"\x5c\x83\xfe\xb7\x7f\x00\x00\x00\x41\x41\x41\x41\x41\x41\x41\x41"
"\x41\x41\x41\x41\x41\x41\x41\x41\xd0\xc4\xf4\xb7\x7f\x00\x00\x00"
"\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41"
"\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41"
"\x50\xfc\xff\xff\x7f\x00\x00\x00\x0d\x00\x00\x00\x00\x00\x00\x00"
"\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41"
"\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41"
"ARMv8 exploit;
```

3.2 gadget 自动搜索框架搜索结果

本文利用该工具在 Linaro ARMv8 Linux 常用的库文件 libc. so. 6 和 ld-linux-aarch64. so. 1 中进行搜索，实验结果如图 8、图 9 所示。

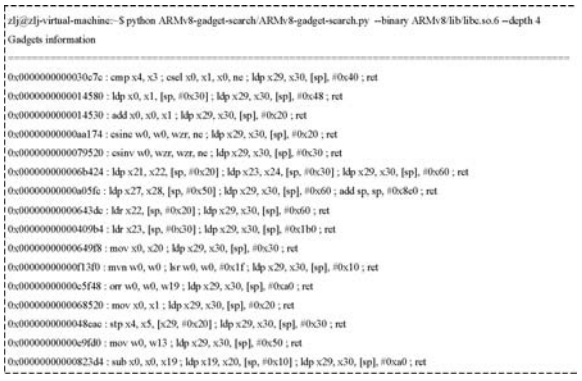


图 8 libc. so. 6 库文件中搜索深度为 4 的部分搜索结果

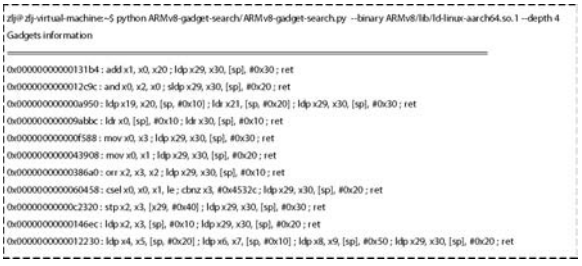


图 9 ld-linux-aarch64. so. 1 库文件中搜索深度为 4 的部分搜索结果

通过利用搜索到的结果统计出以 RET 指令结尾的 gadget 的数目如表 3 所示。

表 3 gadget 统计结果

库文件	gadget 数目
libc. so. 6	4 168
ld-linux-aarch64. so. 1	479

Buchanan[12] 等首次在 X86 架构上对 gadget 的图灵完整性进行了证明，如果一个 gadget 集可以满足基本的赋值操作、算术和逻辑运算、控制流和函数调用四个条件，那么该 gadget 集就是图灵完整的，可以执行任何操作。通过统计，本文中搜索工具搜索到的 gadget 同样可以满足上面四个条件，因此本文工具搜索到的 gadget 也是图灵完备的，可以执行任意的操作。

3.3 gadget 自动生成框架测试

本文从网站 exploit. db 中选取了 10 个 shellcode，然后利用本文的自动构造工具进行自动的构造。本文所使用的环境为 Linaro ARMv8 Linux，其版本号为 Linux 3. 6. 0-1-Linaro-vexpress64。Linaro 是 ARM 公司授权商，其为 ARMv8 开发的工具链和快速模型虚拟平台能从 ARM 官网下载。转化后的 ARMv8 shellcode 大小和 ARMv8 ROP shellcode 大小如表 4 所示。

表 4 ARMv8 ROP shellcode

功能	ARMv8 shellcode 大小	ARMv8 ROP shellcode 指令序列数目	ARMv8 ROP shellcode 大小
execve(“/bin/sh”)	30	10	188
exit(0)	12	8	124
chmod(“/etc/passwd”,777)	48	12	368
write(1,”arm exploit.” ,13)	76	14	472
kill all processs	16	9	246
set system time to 0 and exit	18	12	274
edit /etc/sudoers	84	32	524
chmod(“/etc/shadow”,666)	52	12	376
dup2(0,0),dup2(0,1),dup2(0,2)	36	24	348
setreuid(getuid(),setuid()),execve(“/bin/sh”,0,0)	52	28	498

在表 4 所示的 shellcode 中，有的涉及了复杂的 shellcode 的设计，包括使用多个系统调用和条件跳转等。例如，“setreuid(getuid(),setuid()),execve(“/bin/sh”,0,0)”ROP shellcode 中需要涉及 4 个系统调用，“dup2(0,0),dup2(0,1),dup2(0,2)”ROP shellcode 中需要涉及条件跳转。从表 4 中可以看出，随着 shellcode 的复杂性的增加，构造 ARMv8 ROP shellcode 的复杂度也在增大。shellcode 的长度越大，构造的 ARMv8 ROP shellcode 的 gadget 的数目和大小也越大。

4 结 语

为了实现 ARMv8 架构上的 ROP 技术，提高 ROP 攻击的效率，增加其实用性。本文首次对已有的 gadget 搜索框架进行扩展，实现了 ARMv8 架构上 ROP gadget 自动搜索的框架。它首先在库文件搜索出所有的以 RET 指令结尾的短指令序列，并把这些指令序列存储在 gadget 库中；然后利用语义分析器找到与 shellcode 语义相同的 gadget，gadget 选择器对这些 gadget 进行优化排序，选择出最优的 gadget；最后利用 gadget、优化器解决语义断层和寄存器冲突问题，完成 gadget 的自动构造。利用实验对本文的工具进行了测试，实验结果表明本文的工具具有良好的攻击效力。但是本文中的 shellcode 的控制结构是通过人工分析进行的，并且其不具有隐蔽性。下一步主要研究 ROP shellcode 控制结构的自动构造和控制结构的隐蔽性，以此提高 ROP 攻击能力和抗检测能力。

（下转第 328 页）

[J]. 软件学报,2014(9):1889 - 1908.
[5] 李动周. 大型关系型数据库优化探讨[J]. 办公自动化,2007(2):32 - 34.
[6] 岑巍. 数据库优化在海量数据下的研究与应用[J]. 计算机时代,2015(2):33 - 35.
[7] 李振国,郑惠中. 网络流量采集方法研究综述[J]. 吉林大学学报(信息科学版),2014(1):70 - 75.
[8] 袁梅宇. 高效率多线程网络流量采集算法研究及实践[J]. 昆明理工大学学报(理工版),2006(1):32 - 36.
[9] 王冬梅,张素青,王硕. IP 城域网网络安全分析及流量过滤技术[J]. 信息通信,2014(10):253 - 254.
[10] 窦衍旭. 高速网络流量内容还原系统的设计与实现[D]. 兰州大学,2014:1 - 65.

(上接第 285 页)

了基于云模型的猫群优化算法的码书设计。将整个猫群分为搜寻组和跟踪组。在搜寻组中运用云发生器建立个体变异程度和适应值大小的关系,实现克隆变异个数的自适应调节;在跟踪组中通过正态云算子实现猫群的进化和变异,自适应控制猫群的搜索范围。实验结果表明,云猫群优化算法能较好地应用到码书设计中,减少对初始码书的依赖性,更好地实现全局最优的收敛。

参 考 文 献

[1] Linde Y, Buzo A, Gray R M. An Algorithm for Vector Quantizer Design[J]. IEEE Trans on Communications, 1980, 28(1):84 - 95.
[2] 李霞,罗雪晖,张基宏. 基于人工蚁群优化的矢量量化码书设计算法[J]. 电子学报,2004,32(7):1082 - 1085.
[3] 李殷,李飞. 基于量子粒子群优化算法的矢量量化码书设计[J]. 电视技术,2012,36(17):26 - 29.
[4] 赵梦玲,刘红卫,刘若辰. 基于遗传模拟退火算法的矢量量化码书设计[J]. 数学的实践与认识,2015,45(1):209 - 218.
[5] Chu S C, Tsai P W, Pan J S. Cat Swarm Optimization[C]// Proc of the 9th Pacific Rim International Conference on Artificial Intelligence. Guilin, China, 2006:854 - 858.
[6] 许波,彭志平,余建平. 一种基于云模型的改进型量子遗传算法[J]. 计算机应用研究,2011,28(10):3684 - 3686.
[7] 吴伟林,周永华. 基于差分演化与猫群算法融合的群体智能算法[J]. 计算技术与自动化,2014,33(4):78 - 83.
[8] 叶飞. 基于智能优化算法的优秀序列的研究[D]. 上海大学,2014.
[9] 张英杰,邵岁锋. 一种基于云模型的云变异粒子群算法[J]. 模式识别与人工智能,2011,24(1):90 - 96.
[10] 刘争艳,李絮,王慧玲. 基于云模型的自适应蚁群算法改进研究[J]. 计算机工程与应用,2016,52(19):68 - 71.
[11] 马颖,田维坚,樊养余. 基于云模型的自适应量子免疫克隆算法[J]. 计算物理,2013,30(4):627 - 632.
[12] 杨淑莹,刘旭鹏,陶冲,等. 基于免疫猫群优化算法的矢量量化的码书设计及语音识别[J]. 模式识别与人工智能,2014,27(7):577 - 583.
[13] Pu Lingmin, Hu Hongmei. Codebook design using improved particle swarm optimization based on selection probability of artificial bee colony algorithm[J]. Journal of Chongqing University(English Edition), 2014, 13(3):90 - 98.

(上接第 322 页)

参 考 文 献

[1] Krahmer S. x86 - 64 buffer overflow exploits and the borrowed code chunks exploitation technique[OL]. 2005-09-28. http://users. suse. com/ ~ krahmer/no - nx. pdf.
[2] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)[C]//Proceedings of the 14th ACM conference on Computer and communications security. ACM, 2007:552 - 561.
[3] Kornau T. Return oriented programming for the ARM architecture[D]. Ruhr-Universitat Bochum, 2010.
[4] Roemer R, Buchanan E, Shacham Hl. Return-oriented programming: Systems, languages, and applications[J]. ACM Transactions on Information and System Security (TISSEC), 2012, 15(1):2.
[5] Vincenzo Iozzo. ROP and iPhone[OL]. 2010-04-16. http://blog. zynamics. com/2010/04/16/rop - and - iphone/.
[6] Schwartz E J, Avgerinos T, Brumley D. Q: Exploit Hardening Made Easy[C]//USENIX Security Symposium, 2011.
[7] Chen P, Xiao H, Shen X, et al. DROP: Detecting return-oriented programming malicious code[M]//Information Systems Security. Springer Berlin Heidelberg, 2009:163 - 177.
[8] Yang C, Zheng T, Lin Z. AR Exploit: An Automatic ROP Exploit Based on Long Sequence[C]//Software Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on. IEEE, 2014:50 - 56.
[9] https://silver. arm. com/download/ARM_and_AMBA_Architecture/AR100-DA-70501-r0p0-00eac5/ARMv8 _ ISA _ PRD03-GENC-010197-30-0. pdf.
[10] https://silver. arm. com/download/ARM_and_AMBA_Architecture/AR10 0-DA-70501-r0p0-00eac5/DDI0487A _ a _ armv8_arm_errata. pdf.
[11] Linaro. Linaro ARMv8 Project[OL]. http://www. Linaro. org/projects/armv8/.
[12] Buchanan E, Roemer R, Shacham H. When good instructions go bad: generalizing return-oriented programming to RISC[C]//Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008:27 - 38.