执行视图

PT\_DYNAMIC Segment

SHT Section Header Table

readelf -S libtest.so

PHT Program Header Table

readelf -l libtest.so

.dynamic section

readelf -d libtest.so

readelf -s libtest.so

0000146c <\_Z10check\_forkv>:

146c: b580 push {r7, lr}

146e: 466f mov r7, sp

1470: b088 sub sp, #32

1472: 481d ldr r0, [pc, #116] ; (14e8 <\_Z10check\_forkv+0x7c>)

1474: 4478 add r0, pc

1476: f04f 31ff mov.w r1, #4294967295

147a: 9006 str r0, [sp, #24]

147c: 4608 mov r0, r1

147e: 9906 ldr r1, [sp, #24]

1480: f7ff ee68 **blx 1154 <dlsym@plt>**

**ARM 寄存器**

fp R11 帧指针

ip R12 内部过程调用寄存器

sp R13 栈指针

lr R14 链接寄存器

pc R15 程序计数器

00001154 <dlsym@plt>:

1154: e28fc600 add ip, pc, #0, 12 ; IP = PC + (0 << 12)

1158: e28cca02 add ip, ip, #8192 ; 0x2000

115c: e5bcfe1c ldr pc, [ip, #3612]! ; 0xe1c

PC = 0xdd4 + 8 + 0x3000 + 0x1b4 = 0x3f90

PC = 0x1154 + 8 + 0x2000 + 0xe1c = 0x3f78

readelf -r libh.so|grep dlsym

**00003f78** 00000f16 R\_ARM\_JUMP\_SLOT 00000000 dlsym@LIBC

0x3f78 got section

**Problem**

sym.imp.\_\_libc\_start\_main :

.plt:0x000082bc 00c68fe2 add ip, pc, 0, 12; after execution ip=0x82c4

.plt:0x000082c0 08ca8ce2 add ip, ip, 8, 20; after execution ip=0x102c4

.plt:0x000082c4 48fdbce5 ldr pc, [ip, 0xd48]!

**Answer**

Understanding the code

First Line:

This is simply translated into add ip, pc because rotate operations on #0 is still 0.

So it's actually IP = PC + (0 << 12) = PC + 0

Second Line:

Let's take apart the opcodes and understand the problematic line:

The opcodes should be read like this because of endianness: e28cca08

**e - always execute this instruction**

**28 - add immediate**

**c - Rd is the ip**

**c - Rn is the ip**

**a 08 - 8 right rotated by 20**

The things is, that it's not 8<<20 but instead it is 8<<(32-12) because we are on a 32-bit system and it is a Circular Shift.

Here's a C code that showing the Circular Shift based on the example from Wikipedia:

#include <stdint.h> // for uint32\_t, to get 32bit-wide rotates, regardless of the size of int.

#include <limits.h> // for CHAR\_BIT

uint32\_t rotl32 (uint32\_t value, unsigned int count) {

const unsigned int mask = (CHAR\_BIT\*sizeof(value)-1);

count &= mask;

return (value<<count) | (value>>( (-count) & mask ));

}

uint32\_t rotr32 (uint32\_t value, unsigned int count) {

const unsigned int mask = (CHAR\_BIT\*sizeof(value)-1);

count &= mask;

return (value>>count) | (value<<( (-count) & mask ));

}

int main()

{

printf("Result: 0x%x\n",rotr32(8,20));

return 0;

}

# function arc4random\_buf

# References

<http://www.cnblogs.com/twlqx/p/4339758.html>