

从零打造简单的 SODUMP 工具

Author: ThomasKing

最近翻看之前的帖子，发现基于 linker init_array 加密的 SO 文件的静态分析稍微麻烦。虽然原理很清楚，但是需要 dump 之后再 section 修复才能放入 ida。可以看到，上述两步骤其实很机械。那么应该可以实现一个自动化工具，帮助我们解决上述问题，让我们可以精力专注于其他地方，提高效率。实现上述工具需要解决两个问题：1> 应对各种加密算法 2> section 重建。Section 重建在 <http://bbs.pediy.com/showthread.php?t=192874> 已经讨论，就不再赘述，仅仅改进一些不足之处。

一、应对各种加密算法策略

在 <http://bbs.pediy.com/showthread.php?t=192020&viewgoodness=1&prefixid=> 等帖子中提到的 SO 文件都基于 init_array 实现。为了静态分析，不得不分析出算法，并实现对 SO 解密，以便静态分析。而且，不同的 SO 采用不同的加密算法。这个步骤时间的花费，取决于分析人员对各种常用的加密算法 RC4、TED 等了解程度，这是比较纠结的事情。

那如何应对各种加密算法，成为工具实现的关键。直接实现各种算法，这个显然是不行的。那只能通过某种方法来绕过。想了很久，也没有很好的解决策略。不过有一点，熟悉 SO 机制的读者都知道，linker 可是应对自如！那看来只能借 linker 之手，来帮助我们实现。

Linker 除了在程序运行初加载外，还有就是通过 dl 函数会调用，即 dlopen、dlsym、dlerr 和 dlclose。啰嗦下 dlopen 和 dlsym 函数原型：

```
void * dlopen( const char * pathname, int mode);
```

```
void*dlsym(void*handle,constchar*symbol);
```

通常，使用 dlopen 打开加载某 SO 后，会返回一个 handle 对象，然后根据 handle 对象调用 dlsym 查找某函数符号，实现调用。仔细分析可以发现，这个过程和程序在运行初是类似的；另外，linker 只会加载某一 SO 一次，那么 linker 应该维护了一个数据表来记录。进一步分析还可以发现，这个 void *handle 对象应该指向了当前打开 SO 的数据对象。有了这个思路，翻看 dl 函数源码(位于：\bionic\linker\dlfcn.c，不是在 bionic\libdl\libdl.c)，dlopen 源码：

```
void *dlopen(const char *filename, int flag)
{
    soinfo *ret;
    pthread_mutex_lock(&dl_lock);
    ret = find_library(filename);
    if (unlikely(ret == NULL)) {
        set_dlerror(DL_ERR_CANNOT_LOAD_LIBRARY);
    } else {
        ret->refcount++;
    }
    pthread_mutex_unlock(&dl_lock);
    return ret;
}
```

标红部分可以看到，ret 就是 handle。翻看 linker 源码，其实际为：soinfo 结构体(截取部分结构)

```

struct soinfo
{
    const char name[SOINFO_NAME_LEN];
    Elf32_Phdr *phdr; //Elf32_Phdr 实际内存地址
    int phnum;
    unsigned entry;
    unsigned base; //SO 起始
    unsigned size; //内存对齐后占用大小

    int unused; // DO NOT USE, maintained for compatibility.
    unsigned *dynamic; //.dynamic 实际内存地址

    unsigned wrprotect_start; //mprotect 调用
    unsigned wrprotect_end;

    soinfo *next; //下一个 soinfo
    unsigned flags;

    const char *strtab; //.strtab 实际内存地址
    Elf32_Sym *symtab; //.symtab 实际内存地址

    //hash 起始位置: bucket - 2 * sizeof(int)
    unsigned nbucket; //size = nbucket * sizeof(int)
    unsigned nchain; //size = nchain * sizeof(int)
    unsigned *bucket;
    unsigned *chain;

    unsigned *plt_got; //对应.dynamic: DT_PLTGOT

    Elf32_Rel *plt_rel; //函数重定位表
    unsigned plt_rel_count;

    Elf32_Rel *rel; //符号重定位表
    unsigned rel_count;

    ....
};

```

从这个结构中，已经可以得到各种信息，直接用于后续重建。另外，还有一点：`dlopen` 返回 `soinfo` 时，`linker` 已经执行了 `init_array` 中的函数。换句话说，已经实现了自解密，直接 `dump` 就是 OK。

二、改进 PLT 和 GOT 重建

在 <http://bbs.pediy.com/showthread.php?t=192874> 帖子中，PLT 和 GOT section 并不能直接从 `.dynamic` 中获取。重建时，通过 section 之间的排列关系，间接的进行修正。如果想对位置变化，这种重建是无意义的。影响相对位置变化主要有：链接脚本的细微区别和 SO 经

过 DIY。比如，不同版本的 ndk 存在细微差异：

```

/cygdrive/d/workplace
Thomas@Thomas-PC /cygdrive/d/workplace
$ readelf.exe -S libtestAndoird.so
There are 21 section headers, starting at offset 0x3130:

Section Headers:
[Nr] Name                Type           Addr           Off           Size         ES Flg Lk  Inf Al
[ 0]                     NULL          00000000      000000      000000      00  0  0  0  0
[ 1] .dynsym                DYNSYM        00000114      000114      0003c0      10  A  2  1  4
[ 2] .dynstr                STRTAB        000004d4      0004d4      0004ab      00  A  0  0  1
[ 3] .hash                  HASH          00000980      000980      00018c      04  A  1  0  4
[ 4] .rel.dyn               REL           00000b0c      000b0c      000058      08  A  1  0  4
[ 5] .rel.plt               REL           00000b64      000b64      000040      08  A  1  6  4
[ 6] .plt                   PROGBITS      00000ba4      000ba4      000074      00  AX 0  0  4
[ 7] .text                  PROGBITS      00000c18      000c18      00148c      00  AX 0  0  4
[ 8] .ARM.extab             PROGBITS      000020a4      0020a4      000054      00  A  0  0  4
[ 9] .ARM.exidx             ARM_EXIDX     000020f8      0020f8      0000e0      08  AL 7  0  4
[10] .rodata                 PROGBITS      000021d8      0021d8      000005      01  AMS 0  0  1
[11] .fini_array            FINI_ARRAY    00003ea8      002ea8      000008      00  WA 0  0  4
[12] .init_array            INIT_ARRAY    00003eb0      002eb0      000004      00  WA 0  0  1
[13] .dynamic                DYNAMIC       00003eb4      002eb4      0000f8      08  WA 2  0  4
[14] .got                    PROGBITS      00003fac      002fac      000054      00  WA 0  0  4
[15] .data                   PROGBITS      00004000      003000      000004      00  WA 0  0  4
[16] .bss                     NOBITS        00004004      003004      00000c      00  WA 0  0  4
[17] .comment                PROGBITS      00000000      003004      000026      01  MS 0  0  1
[18] .note.gnu.gold-ve      NOTE          00000000      00302c      00001c      00  0  0  0  4
[19] .ARM.attributes        ARM_ATTRIBUTES 00000000      003048      00002d      00  0  0  0  1
[20] .shstrtab               STRTAB        00000000      003075      0000b8      00  0  0  0  1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)

Thomas@Thomas-PC /cygdrive/d/workplace
$
    
```

图 1

```

/cygdrive/d/workplace
Thomas@Thomas-PC /cygdrive/d/workplace
$ readelf.exe -S t5.so
There are 21 section headers, starting at offset 0xfdec4:

Section Headers:
[Nr] Name                Type           Addr           Off           Size         ES Flg Lk  Inf Al
[ 0]                     NULL          00000000      000000      000000      00  0  0  0  0
[ 1] .hash                  HASH          00000f4      0000f4      000378      04  A  2  0  4
[ 2] .dynsym                DYNSYM        0000046c      00046c      0007b0      10  A  3  3  4
[ 3] .dynstr                STRTAB        00000c1c      000c1c      0004c1      00  A  0  0  1
[ 4] .rel.dyn               REL           000010e0      0010e0      000408      08  A  2  0  4
[ 5] .rel.plt               REL           000014e8      0014e8      0003a8      08  A  2  6  4
[ 6] .plt                   PROGBITS      00001890      001890      000590      04  AX 0  0  4
[ 7] .text                  PROGBITS      00001e20      001e20      0de860      00  AX 0  0  8
[ 8] .rodata                 PROGBITS      000e0680      0e0680      01600c      00  A  0  0  8
[ 9] .ARM.extab             PROGBITS      000f668c      0f668c      000288      00  A  0  0  4
[10] .ARM.exidx             ARM_EXIDX     000f6914      0f6914      000858      00  AL 7  0  4
[11] .init_array            INIT_ARRAY    000f8b74      0f8b74      000038      00  WA 0  0  4
[12] .fini_array            FINI_ARRAY    000f8bac      0f8bac      00000c      00  WA 0  0  1
[13] .data.rel.ro           PROGBITS      000f8bb8      0f8bb8      000050      00  WA 0  0  4
[14] .dynamic                DYNAMIC       000f8c08      0f8c08      000108      08  WA 3  0  4
[15] .got                    PROGBITS      000f8d10      0f8d10      0002f0      04  WA 0  0  4
[16] .data                   PROGBITS      000f9000      0f9000      004498      00  WA 0  0  16
[17] .bss                     NOBITS        000fd498      0fd498      005a8c      00  WA 0  0  4
[18] .ARM.attributes        ARM_ATTRIBUTES 00000000      0fd498      00002b      00  0  0  0  1
[19] .comment                PROGBITS      00000000      0fd4c3      000952      00  0  0  0  1
[20] .shstrtab               STRTAB        00000000      0fde15      0000ae      00  0  0  0  1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)
    
```

图 2

另外，GOT 结构也不同，相应链接脚本中也存在顺序不一致。比如：`.got : { *(.got.plt)`

*(.igot.plt) *(.got) *(.igot) } 就是函数符号在前,其他符号在后,和之前讨论的是相反的。由于存在上述原因,对 PLT 和 GOT 的重建进行如下改进。

2.1 PLT section

由于优化,PLT 结构前四项是固定代码。

```
AREA .plt, CODE
; ORG 0xEB8
CODE32
STR    LR, [SP,#-4]!
LDR    LR, =(_GLOBAL_OFFSET_TABLE_ - 0xEC8)
ADD    LR, PC, LR ; _GLOBAL_OFFSET_TABLE_
LDR    PC, [LR,#8]!
```

图 3

借鉴 window 内核中搜索未导出符号的思路,通过搜索前 16 个字节来确定 plt。恰好这里刚好够 16 个字节(真是无巧不成书),即:

```
static unsigned g_plt_start[] = {
```

```
    0xe52de004, 0xe59fe004, 0xe08fe00e, 0xe5bef008 };
```

这样就可以得到 plt section 的起始地址。Size 前面帖子中已经说明,这里就不在赘述。

2.2 GOT section

前面已经提到 GOT 表存在两种结构。另外,.got 与 .rel 的对应关系并不像 .got.plt 与 .rel.plt 的对应关系那么稳定,受到诸如指针间址寻址影响。但始终有一点:.got 中的项一定出现在 .rel section 中。那么重建的思路就是:

Step1: 读取 DT_PLTGOT, 获取 __global_offset_table__ 地址, 记为: plt_got

Step2: 读取 plt_got - 4 地址的数值, 在 .rel 表中进行搜索。如果匹配, 说明 GOT 结构式: { .got, .got.plt }, 转 3.1; 否则为 { .got.plt, .got } 转 3.2

Step3.1: 继续向前搜索, 直到搜索到起始位置。

Step3.2: 调整搜索起始位置: $plt_got + 4 * (3 + rel_plt_count)$, 向后搜索, 搜索到末尾。这里, 存在一种情况: 若下面 .data 也有重定位项且处于 .data 起始连续位置, 也会被搜索到。但对于 .got 和 .data 都对应到 .rel 重定位中, 重定位方式相同, 统一处理也是合理的, 实测不存在问题。

通过搜索, 即可准确获取 GOT 起始地址和长度, 完成重建。

另外, 对于 .data 和 .rodata 的重建目前并没有很好的思路, 仅只通过相对位置重定位。如果有更好的思路, 请告知我学习学习, 衷心感谢。

三、SODUMP 出炉

经过上面讨论, SODUMP 工具基本成型, 剩下的就是添砖添瓦, 做成一个 apk。通过 EditText 和 Button 配合, 将待 dumpSO 进行处理, 记得使用 dlclose 关闭, 最好使用 dlerr 把错误信息打印出来便于定位问题。废话不多说, 上几张测试图:

此贴 <http://bbs.pediy.com/showthread.php?t=190384&viewgoodnees=1&prefixid=> SO 文件脱壳如下, <http://bbs.pediy.com/showthread.php?t=188793&viewgoodnees=1&prefixid=> 帖子类似, 都为某公司一种模式, 就不贴了。


```

LOAD:0000DC34 ; End of function _Unwind_GetTextRelBase
LOAD:0000DC34
LOAD:0000DC34 ; -----
LOAD:0000DC38 EXPORT __gnu_armfini_31
LOAD:0000DC38 __gnu_armfini_31 DCD 0x97A29A3E, 0x57E9111A, 0x39D872CC, 0x53427AE8, 0x84D519ED ; DATA XREF: LOAD:0FF_FEF4↓o
LOAD:0000DC38 DCD 0x27F9B5F2, 0x7093CB09, 0x93A233E3, 0x27663FB4, 0xE1044159
LOAD:0000DC38 DCD 0x46C5B7A7, 0x91C9FC7C, 0x2EF2827F, 0xC4E6B07A, 0x54E97938
LOAD:0000DC38 DCD 0xC7C95244, 0xCA11296F, 0x7D218BA0, 0xD4BE3C30, 0x363F07B4
LOAD:0000DC38 DCD 0x7EC9AF28, 0x392921BC, 0x25ABA7CF, 0xF01B222, 0xA806AF84
LOAD:0000DC38 DCD 0x753E511F, 0xA4175F42, 0xEBE832B3, 0xCCCE4E9F, 0xD53495C
LOAD:0000DC38 DCD 0xFE9382A1, 0x9762A449, 0xAD08E8F4, 0x2C335457, 0x5FD025E2
LOAD:0000DC38 DCD 0x564EA3C0, 0x6BE0D39A, 0xB60B4AF3, 0x9F285CC2, 0x120D1415
LOAD:0000DC38 DCD 0xD541F0C9, 0x4146DD1D, 0x61636776, 0x3228CCEE, 0x67D833CC
LOAD:0000DC38 DCD 0x714485C7, 0xD51BAF41, 0x55EA2285, 0xD682A46B, 0x3E22779F
LOAD:0000DC38 DCD 0xF8A64A7A, 0x55F8DCDE, 0x1E7981D6, 0xFF09C2EB, 0x7DD0C92C
LOAD:0000DC38 DCD 0xDFECC6F5, 0x8F609589, 0xE85AF8E6, 0x2B97C5DC, 0xE2B06377
LOAD:0000DC38 DCD 0x9138443D, 0x42297A4F, 0x44C82712, 0xFC1988F3, 0x5DC2A716
LOAD:0000DC38 DCD 0x6916B296, 0xC75E98E1, 0x8D05DE66, 0xBAD8A873, 0x5B979916
LOAD:0000DC38 DCD 0xF1E6913C, 0x66A07538, 0x3C321209, 0x4794AF28, 0x433C8051
LOAD:0000DC38 DCD 0x3B98AE9, 0xEBDEB0AE, 0xFEE33EBE, 0xB380F84E, 0x3C50E819
LOAD:0000DC38 DCD 0xF9C8CE0E, 0x4C97DD2A, 0x5837CB33, 0x40903834, 0xFFFD8052
LOAD:0000DC38 DCD 0xF682E42A, 0xABDAF27, 0x6A0B2421, 0x92A89191, 0x195914CE
LOAD:0000DC38 DCD 0x629F215F, 0x5EE403BB
LOAD:0000DA8 ; -----

```

图 4 脱壳前

```

.text&_ARM.extab:0000DC38
.text&_ARM.extab:0000DC38
.text&_ARM.extab:0000DC38 EXPORT __gnu_armfini_31
.text&_ARM.extab:0000DC38 __gnu_armfini_31 ; CODE XREF: __gnu_arm_fini_05+58↓p
.text&_ARM.extab:0000DC38 STMFD SP!, {R4,LR}
.text&_ARM.extab:0000DC3C MOV R4, R0
.text&_ARM.extab:0000DC40 MOV R0, #2
.text&_ARM.extab:0000DC44 BL __gnu_armfini_09
.text&_ARM.extab:0000DC48 LDR R3, =(off_FEB4 - 0xDC54)
.text&_ARM.extab:0000DC4C LDR R3, [PC,R3] ; off_FEB4
.text&_ARM.extab:0000DC50 STR R3, [R4]
.text&_ARM.extab:0000DC54 LDMFD SP!, {R4,PC}
.text&_ARM.extab:0000DC54 ; End of function __gnu_armfini_31
.text&_ARM.extab:0000DC54
.text&_ARM.extab:0000DC54
.text&_ARM.extab:0000DC54 ; -----
.text&_ARM.extab:0000DC58 off_DC58 DCD off_FEB4 - 0xDC54 ; DATA XREF: __gnu_armfini_31+10↓r
.text&_ARM.extab:0000DC5C

```

图 5 脱壳后

一朋友给的某手机游戏，据说是美杜莎壳。当然，这仅仅只是该壳子的一个很小方面，这壳子很生猛的。

```

LOAD:00007BEC EXPORT _dHm_iWweXTSb1xDrJH_LVxAJfJMaJYeNBuHDJKbHQbM_TttsKS_y
LOAD:00007BEC _dHm_iWweXTSb1xDrJH_LVxAJfJMaJYeNBuHDJKbHQbM_TttsKS_y ; [dHm[iWweXTSb1xDrJH_LVxAJfJMaJYeNBuHDJKbHQbM][TttsKS]y
LOAD:00007BEC USAD#8 R11, LR, R11, R12
LOAD:00007BEC ; -----
LOAD:00007BF0 DCD 0x176E0333, 0x8CBA0788, 0x3E0810F, 0xD087BF3D, 0xC327872B
LOAD:00007BF0 DCD 0x77F31DF7, 0x673FAB01, 0xA3F6C66A, 0x337EC2B7, 0x8FC6DFA1
LOAD:00007BF0 DCD 0x50570BB6, 0xB859A738, 0xEDDA6A84, 0xD78538BF, 0x81137F1F
LOAD:00007BF0 DCD 0x8D07342DF, 0x1348BACF, 0x1727E0A, 0xDBAF487F, 0xD00A177B
LOAD:00007BF0 DCD 0xEE052ABF, 0x2774D33B, 0x15D71C8A, 0x19F3C056, 0x4227B797
LOAD:00007BF0 DCD 0xCBE14238, 0x3EE7D6DA, 0x3DC928B, 0x13CD8AB4, 0xF3C0E7C2
LOAD:00007BF0 DCD 0xE51B7DDD, 0xC8138D04, 0x8E33D7AE, 0x88C2EF8A, 0x46C0A727
LOAD:00007BF0 DCD 0xE08CB7D4, 0x4A0736F3, 0x286E1309, 0xEF3AA403, 0xB1435A1B
LOAD:00007BF0 DCD 0x7B72865A, 0x3B48EF6B, 0x6EFA0A9, 0xB33986C, 0x170A09EF
LOAD:00007BF0 DCD 0x28A75826, 0x7249FFEF, 0xBFBE2B07, 0xAFF52FD, 0xCEB54A2E
LOAD:00007BF0 DCD 0xF0A06E3, 0xB51B18C6, 0xB306034C, 0x408FC373, 0x7071B3B4
LOAD:00007BF0 DCD 0xD107268B, 0xFED0AD4, 0x94336DEC, 0x40CD8381, 0xF6CD24EC
LOAD:00007BF0 DCD 0x694320F6, 0x8B8B870B, 0x2F815B66, 0xCDA06C87, 0x6AA3D0FF
LOAD:00007BF0 DCD 0xEF682AA0, 0x348E8F64, 0x1774A08C, 0x737FA510, 0xCB6B6B27
LOAD:00007BF0 DCD 0xBCC9173, 0xC4B6707F, 0xD20B1FAE, 0xB7BF683, 0xADC2C50B
LOAD:00007BF0 DCD 0x62856393, 0xC1FB0337, 0x8A536385, 0xD048A8F4, 0xAD136BB7
LOAD:00007BF0 DCD 0x968428B8, 0x1788694B, 0x85BE3627, 0x68807FC6, 0xA8AEE741
LOAD:00007BF0 DCD 0xF7BADC13, 0x13F3A5E3, 0xB036702F, 0xD38B3FC0, 0xE3026F76
LOAD:00007BF0 DCD 0x24E8F6ED, 0x69B476EB, 0xE171B0F, 0x5334EB57, 0xF79CB60A
LOAD:00007BF0 DCD 0xD886634D, 0xC23B31AA, 0x2FB6F5B7, 0xB737DC6A, 0xDDA3B6CC
LOAD:00007BF0 DCD 0xB7640B2A, 0x327A8393, 0x407F8B73, 0x139277B6, 0xAFC1BEC0
LOAD:00007BF0 DCD 0x10B9AFF, 0xCBD8556F, 0x933ADA88, 0x6F4BF678, 0x575736DB
LOAD:00007BF0 DCD 0xA176C8F, 0x23D7D64B, 0xD0B37DC, 0xAF28B797, 0xD35B2A91
LOAD:00007BF0 DCD 0x8368FA94, 0xB74CF776, 0x564B0E33, 0x716BA1E7, 0xF3BE56E9
LOAD:00007BF0 DCD 0xD36D29DC
LOAD:00007DD4 ; -----

```

图 6 脱壳前

```

.text@.ARM.extab:00007BEC
.text@.ARM.extab:00007BEC      EXPORT _dHm_iWueXTSb1xDrJH_LYxAJfjMaJYeNBuHdJKbHQbM_TtSKS_y
.text@.ARM.extab:00007BEC      _dHm_iWueXTSb1xDrJH_LYxAJfjMaJYeNBuHdJKbHQbM_TtSKS_y
.text@.ARM.extab:00007BEC      ; CODE XREF: sub_DB88+640lp
.text@.ARM.extab:00007BEC      ; sub_DB88+90Clp ...
.text@.ARM.extab:00007BEC      STMFDB      SP!, {R4,R5}; [dHm_iWueXTSb1xDrJH_LYxAJfjMaJYeNBuHdJKbHQbM_TtSKS_y]
.text@.ARM.extab:00007BF0      LDRB        R3, [R1,#2]
.text@.ARM.extab:00007BF4      LDRB        R5, [R1,#1]
.text@.ARM.extab:00007BF8      LDRB        R4, [R1]
.text@.ARM.extab:00007BFC      LDRB        R12, [R1,#3]
.text@.ARM.extab:00007C00      MOV         R3, R3, LSL#16
.text@.ARM.extab:00007C04      ORR        R3, R3, R5, LSL#8
.text@.ARM.extab:00007C08      ORR        R3, R3, R4
.text@.ARM.extab:00007C0C      ORR        R3, R3, R12, LSL#24
.text@.ARM.extab:00007C10      STR        R3, [R0,#4]
.text@.ARM.extab:00007C14      LDRB        R3, [R1,#6]
.text@.ARM.extab:00007C18      LDRB        R5, [R1,#5]
.text@.ARM.extab:00007C1C      LDRB        R4, [R1,#4]
.text@.ARM.extab:00007C20      LDRB        R12, [R1,#7]
.text@.ARM.extab:00007C24      MOV         R3, R3, LSL#16
.text@.ARM.extab:00007C28      ORR        R3, R3, R5, LSL#8
.text@.ARM.extab:00007C2C      ORR        R3, R3, R4
.text@.ARM.extab:00007C30      ORR        R3, R3, R12, LSL#24
.text@.ARM.extab:00007C34      STR        R3, [R0,#8]
.text@.ARM.extab:00007C38      LDRB        R3, [R1,#0xA]
.text@.ARM.extab:00007C3C      LDRB        R5, [R1,#9]
.text@.ARM.extab:00007C40      LDRB        R4, [R1,#8]
.text@.ARM.extab:00007C44      LDRB        R12, [R1,#0xB]
.text@.ARM.extab:00007C48      MOV         R3, R3, LSL#16
.text@.ARM.extab:00007C4C      ORR        R3, R3, R5, LSL#8
.text@.ARM.extab:00007C50      ORR        R3, R3, R4
.text@.ARM.extab:00007C54      ORR        R3, R3, R12, LSL#24
.text@.ARM.extab:00007C58      STR        R3, [R0,#0xC]
.text@.ARM.extab:00007C5C      LDRB        R3, [R1,#0xE]

```

图 7 脱壳后

四、参考文献

Linker.c、dlfcn.c 源码

<http://bbs.pediy.com/showthread.php?t=192874>

<http://bbs.pediy.com/showthread.php?t=190384&viewgoodnees=1&prefixid=>

<http://bbs.pediy.com/showthread.php?t=193720&viewgoodnees=1&prefixid=>