手机软件反分析

IDA篇

没有F5的日子

- 没有Hexrays, 还可以肉眼反编译啊
- 再加上没有Thumb2, 16bit指令包含的立即数还有包含移位, GCC频繁使用EOR来生成有效bit较多的立即数, 而不优先用text pool的话

分析点参数是没问题的

```
.text:8010E264                MOV     R0, #0
.text:8010E268                MCR     p15, 0, R0,c7,c7
.text:8010E26C                BL      System_EnableICache
.text:8010E270                LDR     R0, =0x70000013 ; Base Addres : 0x70000000, Size : 256 MB (0x13)
.text:8010E274                MCR     p15, 0, R0,c15,c2, 4
.text:8010E278                LDR     R0, =0x71200014 ; VIC0INTENCLEAR
.text:8010E27C                MOVL    R1, 0xFFFFFFFF
.text:8010E280                STR     R1, [R0]
.text:8010E284                LDR     R0, =0x71300014 ; VIC1INTENCLEAR
.text:8010E288                MOVL    R1, 0xFFFFFFFF
.text:8010E28C                STR     R1, [R0]
.text:8010E290                LDR     R0, =0x7E004000 ; WTCON
.text:8010E294                MOV     R1, #0
.text:8010E298                STR     R1, [R0]
.text:8010E29C                LDR     R0, =0x7E00F900 ; OTHERS
.text:8010E2A0                LDR     R1, [R0]
.text:8010E2A4                AND     R1, R1, #0x40   ; OTHERS[6] = 0:AsyncModde 1:SyncMode
.text:8010E2A8                CMP     R1, #0x40
.text:8010E2AC                BNE     System_SetSyncMode
.text:8010E2B0                LDR     R3, =0x83FF3F07 ; Mask for APLL_CON/MPLL_CON
.text:8010E2B4                LDR     R4, =0x80FF3F07 ; Mask for EPLL_CON0
.text:8010E2B8                LDR     R5, =0xFFFF     ; Mask for EPLL_CON1
.text:8010E2BC                LDR     R6, =0x3FF17    ; Mask for CLKDIV0
.text:8010E2C0                LDR     R0, =0x7E00F00C ; Check APLL
.text:8010E2C4                LDR     R1, [R0]
.text:8010E2C8                AND     R1, R1, R3
.text:8010E2CC                LDR     R2, =0x810A0301 ; APLL_CON value to configure
.text:8010E2D0                CMP     R1, R2
.text:8010E2D4                BNE     PLL_NeedToConfigure ; CLK_SRC
.text:8010E2D8                LDR     R0, =0x7E00F010 ; Check MPLL
.text:8010E2DC                LDR     R1, [R0]
.text:8010E2E0                AND     R1, R1, R3
.text:8010E2E4                LDR     R2, =0x810A0302 ; MPLL_CON value to configure
.text:8010E2E8                CMP     R1, R2
.text:8010E2EC                BNE     PLL_NeedToConfigure ; CLK_SRC
.text:8010E2F0                LDR     R0, =0x7E00F014 ; Check EPLL_CON0
.text:8010E2F4                LDR     R1, [R0]
```

分析逻辑就比较头大

```
                CMP     R0, #0
                BEQ     loc_410030AC
                LSRS    R3, R0, #2
                LSLS    R2, R3, #2
                MOV     R12, R2
                BEQ     loc_410030B2
                ADDS    R6, R5, #0
                ORRS    R6, R4
                MOVS    R2, #3
                CMP     R2, R0
                SBCS    R1, R1
                ANDS    R2, R6
                NEGS    R1, R1
                NEGS    R6, R2
                ADCS    R2, R6
                ANDS    R2, R1
                ADDS    R6, R4, #4
                ADDS    R1, R5, #4
                CMP     R6, R5
                SBCS    R6, R6
                CMP     R1, R4
                SBCS    R1, R1
                NEGS    R6, R6
                NEGS    R1, R1
                ORRS    R1, R6
                TST     R1, R2
                BEQ     loc_410030B2
                ADDS    R6, R4, #0
                ADDS    R1, R5, #0
                MOVS    R2, #0

loc_4100308E:                           @ CODE XREF: func_appenc
                LDMIA   R6!, {R7}
                ADDS    R2, #1
                STMIA   R1!, {R7}
                CMP     R3, R2
                BHI     loc_4100308E
                ADD     R5, R12
```

Epi/prolog都诡异起来

```
loc_41003C48:                                   @ CODE XREF: func_printcharon
                LDR     R3, =0xE7BA             @ 41012408 - (41003C4A + 4)
                ADD     R3, PC
                LDR     R2, [R3]
                ADDS    R2, #1
                STR     R2, [R3]

loc_41003C52:                                   @ CODE XREF: func_printcharon
                                                @ func_printcharonframebuffer
                LDR     R3, =0xE7B8
                ADD     R3, PC
                LDR     R3, [R3]
                CMP     R3, R2
                BEQ     loc_41003C86
                LDR     R3, =0xE7AA
                ADD     R3, PC
                LDR     R2, [R3]

loc_41003C62:                                   @ CODE XREF: func_printcharon
                LDR     R3, =0xE7AC
                ADD     R3, PC
                LDR     R3, [R3]
                CMP     R2, R3
                BEQ     loc_41003CB0

loc_41003C6C:                                   @ CODE XREF: func_printcharon
                POP     {R2-R5}
                MOV     R8, R2
                MOV     R9, R3
                MOV     R10, R4
                MOV     R11, R5
                POP     {R4-R7}
                POP     {R0}
                BX      R0
@ ---------------------------------------------------------------------------

loc_41003C7C:                                   @ CODE XREF: func_printcharon
                LDR     R3, =0xE784
```

鸡肋鸡肋食之无味

- 除非有很多自由时间的人, 或者睡不着觉的, 才会很开心地去手动分析
- 0x1234 可以分解为34|12<<8, 1010+0202+22, 看编译器们或者我们当时的心情

IDA的癖好

- BLX享有高优先级, 连data都无法幸免

```
__text:00012D9C dword_12D9C     DCD 0                   ; DATA )
__text:00012DA0 dword_12DA0     DCD 0                   ; DATA )
__text:00012DA4                 CODE32
__text:00012DA4 off_12DA4       DCD 0                   ; DATA )
__text:00012DA4                                         ; fake_s
```

- 只要POP列表含有PC, 优先当成函数末尾. 手动定义函数末尾后Hexrays依然不认账

```
.text:00014134 sub_14134
.text:00014134
.text:00014134 arg_84           =  0x84
.text:00014134
.text:00014134                  LDR             R6, [SP,#arg_84]
.text:00014136                  MOV.W           R0, #0xFFFFFFFF
.text:0001413A                  POP.W           {R7,PC}
.text:0001413A ; End of function sub_14134
.text:0001413A
.text:0001413A ; ---------------------------------------------------
.text:0001413E                  DCW 0x47A0
```

手动定义后的结局

```
.text:00014134 sub_14134                               ; CODE XRE
.text:00014134
.text:00014134 arg_50          =  0x50
.text:00014134 arg_84          =  0x84
.text:00014134
.text:00014134                 LDR             R6, [SP,#arg_84]
.text:00014136                 MOV.W           R0, #0xFFFFFFFF
.text:0001413A                 POP.W           {R7,PC}
.text:0001413E ; ---------------------------------------------
.text:0001413E                 BLX             R4
.text:00014140                 BLX             loc_14154
.text:00014144                 BLX             sub_14134
.text:00014148                 STR             R7, [SP,#arg_50]
```

```
1 int __fastcall sub_14134(int a1, int a2, int a3, int a4, int a5, int (__fastcall *a6)(_DWORD))
2 {
3     return ((int (__fastcall *)(signed int, int, int, int))a6)(-1, a2, a3, a4);
4 }
```

xrefs to sub_14134

Direction	Typ	Address	Text	
Do...	p	sub_14134+10	BLX	sub_14134

IDA的癖好

- panic之类函数不手动标为noreturn的话,在末尾调用这类函数的caller很容易保持散β代码状态, 不会识别为功能.
- 只要发生BL/BLX均认为r0~r3一定会丢失
- b一个blx r4/rx的指令, 会标记该指令为函数
- 不认可同一个函数内应该有ARM/Thumb状态翻转
- 把一些公共调用认成function thunks而且在反汇编状态下标记的参数偏移量会歪掉

IDA的癖好

- 堆栈配不平影响很多参数识别, 强行中途假配平后Hexrays仍会出现问题
- 如果一个函数经过F5错误识别, 错误识别的参数会享有比未识别前默认的正确参数更高的优先权, 会影响到调用方的分析
- 关闭IDA前F5的某些函数, 打开IDA后会再次F5一遍但是不显示结果.

仅仅针对IDA软件本身的变形

- 使用NEON的额外用法 (复制清空, 隐藏调用参数, 隐藏字符串)
- 操作堆栈推入参数和返回地址后pop {r0, r1, pc}取代call
- 插入复杂的, 不改动r0~r3的函数, 调用后造成F5中变量传递过程识别错误 (断链)
- 插入SVC, 同时干扰反汇编和反编译
- 用复杂的函数实现lo寄存器向hi寄存器的传值, 调用后一举两得

接上页

- 调整LR的值到后继地址后, 调用B, 实现BL, 可以造成Hexrays在调用处中断分析或者识别为JMPOUT
- 在程序中嵌入优化版的memcpy/strcat等函数, 在这些函数中识别LR, 对特定的调用完成额外功能.
- 违规使用SP正偏移的caller区空间, 试图造成无法配平干扰F5, 甚至被识别为function thunk无法单独F5.

验证下r0~r3和lo hi传值效果

- 演示效果 handle和index被弄混了, printf的参数被空函数吸走了

```
200         ++index;
201         if ( (signed int)index >= (signed int)v45 )
202           break;
203         v5 = v56[3];
204         v6 = v56[4];
205       }
206     }
207     handle = dlopen(file, 1);
208     mov_r4_from_r0(handle);
209     fakeformat = (const char *)keepr0_to_r3("\"%s\": 0x%08X\n", file, index);
210     printf(fakeformat);
211     if ( index )
212     {
213       v35 = dlsym((void *)index, "JNI_OnLoad");
214       v36 = v35;
```

```
1 int __fastcall mov_r4_from_r0(unsigned int a1)
2 {
3   return operator new[](a1);
4 }
```

用三条组合下

- 加多一个ADD LR, F5就强制认为到此结束了

```
215         }
216         handle = dlopen(file, 1);
217         mov_r4_from_r0(handle);
218         result = j__keepr0_to_r3("\"%s\": 0x%08X\n", file, index);
219     }
220     else
221     {
222         printf("Can't find or process ELF file %s\n", v3);
223         ELFIO::elfio::~elfio((ELFIO::elfio *)&v56);
224         result = 4;
225     }
226     return result;
227 }
```

```
.text:00009BB0 loc_9BB0                                ; CODE XREF: main+274↑j
.text:00009BB0                 ADD             LR, LR, #8
.text:00009BB4                 B               j__keepr0_to_r3
.text:00009BB8 ; ---------------------------------------------------------------------------
.text:00009BB8                 CODE16
.text:00009BB8                 BLX             printf
.text:00009BBC                 CBZ             R4, loc_9C02
.text:00009BBE                 LDR             R0, =(_GLOBAL_OFFSET_TABLE_ - 0x9BC6)
.text:00009BC0                 LDR             R1, =(aJni_onload - 0x44ED0)
.text:00009BC2                 ADD             R0, PC ; _GLOBAL_OFFSET_TABLE_
```

针对IDA新手的变形

- 部分加壳 (ios上需要越狱环境, android不需)
- 假导出函数 (dlopen时候填充caller的plt)
- 抠系统函数 (SVC或者_internal)
- 返回前篡改dyld/linker的寄存器

通过栈中linker的返回地址...

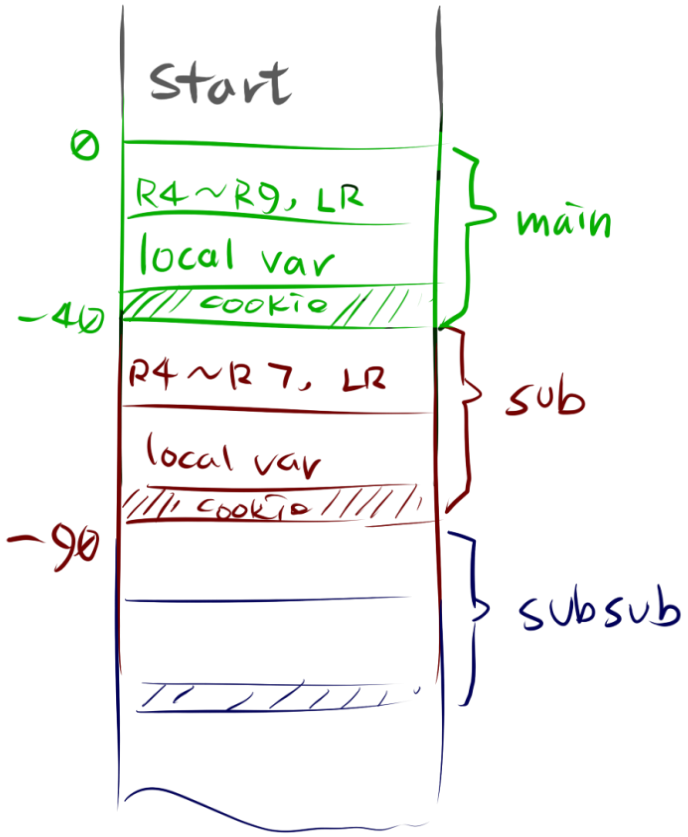
- 返回前操作R5, 操作寄存器LR, R4都可以实现特殊操作.
- 收集所有linker来分析, 很好兼容

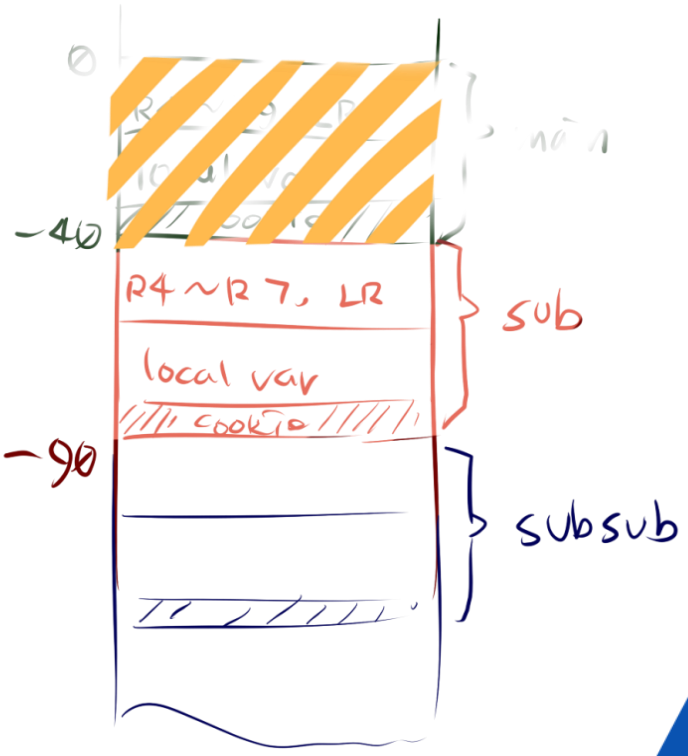
```
.text:00003748
.text:00003748 invokeinitarray                 ; CODE XREF: sub_3730+28↓j
.text:00003748                 LDR             R2, [R4]
.text:0000374A                 ADDS            R4, R4, R6
.text:0000374C                 SUBS            R0, R2, #1
.text:0000374E                 ADDS            R3, R0, #3
.text:00003750                 BHI             bypassarrayitem
.text:00003752                 BLX             R2
.text:00003754
.text:00003754 bypassarrayitem                 ; CODE XREF: sub_3730+20↑j
.text:00003754                 SUBS            R5, #1
.text:00003756
.text:00003756 loc_3756                        ; CODE XREF: sub_3730+16↑j
.text:00003756                 CMP             R5, #0
.text:00003758                 BGT             invokeinitarray
.text:0000375A                 POP             {R4-R6,PC}
```

使用caller的栈空间

- push寄存器, 编译器分配局部变量, 分配调用超过r0~r3可传范围的函数, 都会减sp来保护后继操作不会动到大于sp的数据.
- 通过伪装可以在C级别让caller调整sp给子函数使用, 子函数中对其写入和用来调用子子函数. 如果留出足够余量, 在子函数中可以add sp, 这样一来基本可以被识别为function thunk

正常子函数不会侵犯调用方栈



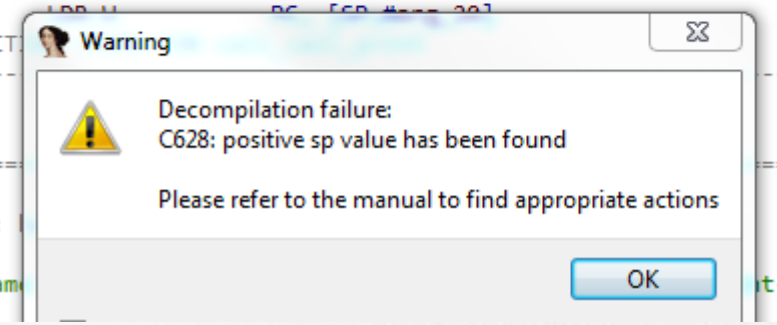


果然生成了function chunk

```
.text:0000C68C call_call_print                          ; CODE XREF: main+A↑p
.text:0000C68C
.text:0000C68C var_18           = -0x18
.text:0000C68C var_14           = -0x14
.text:0000C68C var_10           = -0x10
.text:0000C68C var_C            = -0xC
.text:0000C68C arg_10           =  0x10
.text:0000C68C arg_1C           =  0x1C
.text:0000C68C arg_20           =  0x20
.text:0000C68C
.text:0000C68C ; FUNCTION CHUNK AT .text:0000C616 SIZE 00000020 BYTES
.text:0000C68C
.text:0000C68C                  PUSH            {R7,LR}
.text:0000C68E                  MOV             R7, SP
.text:0000C690                  SUB             SP, SP, #0x28
.text:0000C692                  LDR             R0, =(_GLOBAL_OFFSET_TABLE_ - 0xC69E)
.text:0000C694                  ADD             R3, SP, #0x30+var_C
.text:0000C696                  LDR             R2, =0xFFFFADB3
.text:0000C698                  MOVS            R1, #0x50
.text:0000C69A                  ADD             R0, PC ; _GLOBAL_OFFSET_TABLE_
.text:0000C69C                  ADD             R0, R2
.text:0000C69E                  STR             R0, [SP,#0x30+var_18]
.text:0000C6A0                  STR             R1, [SP,#0x30+var_14]
.text:0000C6A2                  STR             R1, [SP,#0x30+var_10]
.text:0000C6A4                  BL              _call_my_demo2
.text:0000C6A4 ; End of function call_call_print
.text:0000C6A4
.text:0000C6A8 ; ---------------------------------------------------------------------------
.text:0000C6A8                  ADD             SP, SP, #0x28
.text:0000C6AA                  POP             {R7,PC}
.text:0000C6AA ; ---------------------------------------------------------------------------
.text:0000C6AC off_C6AC         DCD _GLOBAL_OFFSET_TABLE_ - 0xC69E
.text:0000C6AC                                          ; DATA XREF: call_call_print+6↑r
.text:0000C6B0 dword_C6B0       DCD 0xFFFFADB3          ; DATA XREF: call_call_print+A↑r
```

F5报错，试试强行改SP？

```
.text:0000C616                ; START OF FUNCTION CHUNK FOR call_call_print
.text:0000C616
.text:0000C616                _call_my_demo2                          ; CODE XREF: call_call_print+18↓j
.text:0000C616 000 18 46                      MOV             R0, R3
.text:0000C618 000 07 99                      LDR             R1, [SP,#arg_1C]
.text:0000C61A 000 08 9A                      LDR             R2, [SP,#arg_20]
.text:0000C61C 000 01 23                      MOVS            R3, #1
.text:0000C61E 000 07 90                      STR             R0, [SP,#arg_1C]
.text:0000C620 000 CD F8 20 E0                STR.W           LR, [SP,#arg_20]
.text:0000C624 000 09 20                      MOVS            R0, #9
.text:0000C626 000 04 B0                      ADD             SP, SP, #0x10
.text:0000C628 -10 00 90                      STR             R0, [SP,#-0x10+arg_10]
.text:0000C62A -10 08 46                      MOV             R0, R1
.text:0000C62C -10 FE 46                      MOV             LR, PC
.text:0000C62E -10 0E F1 04 0E                ADD.W           LR, LR, #4
.text:0000C632 -10 05 E0                      B               demo2_printname
.text:0000C634 -10 84 B0                      SUB             SP, SP, #0x10
.text:0000C636 000 07 99                      LDR             R1, [SP,#arg_1C]
.text:0000C638 000 08 60                      STR             R0, [R1]
.text:0000C63A 000 DD F8 20 F0
.text:0000C63A 000            ; END OF FUNCT
.text:0000C63E                ; ------------
.text:0000C63E 00 47
.text:0000C640
.text:0000C640                ; ============
.text:0000C640
.text:0000C640                ; Attributes:
.text:0000C640
.text:0000C640                demo2_printnam
.text:0000C640
```



强改SP后F5, 关键参数被合并丢了

```
1 int __fastcall call_call_print(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8, int a9, int a10, int a11, int a12)
2 {
3   return demo2_printname(a12);
4 }
```

```
12  int demo2_printname(const char* name, int height, int weight, int gender, int age)
13  {
14      return printf("%s, %d, %d, %s, %d\n", name, height, weight, (gender?"male":"female"), age);
15  }
16
17  void call_call_print()
18  {
19      unsigned a1, a2, a3, a5, a6, a7, a8, a9, a10;
20      int ret;
21      _call_my_demo2(a1, a2, a3, &ret, a5, a6, a7, a8, a9, a10, "mario", 80, 80);
22  }
```

保护的目的

- 不想特有的算法被逆向使用
- 不想自己的整个代码被对手抠出来用, 或者被补丁后包装成自己的用
- 个人开发者, 秘密全在流程, 同行的一旦动态跟踪一次, 甚至不用逆算法就可以仿出来
- 自己产品要访问隐私数据, 不愿被杀毒软件拦截

一些难以发挥作用的保护

- 显式调用ptrace, DENY_ATTACH或者TRACEME可以简单F5或者通过strace发现, 在android下也受到debuggable的影响
- 某些用ndk做dex加密的方案, 虽然双进程三进程互相调试, 仍可以attach线程而dump
- 检查文件句柄, android中已经有很多被打开
- 检查父进程命令行, 通常都是zygote

针对人工的防护

- 使用MIPS, 68000或者其他代码密度低的指令集, 制作板级模拟
- 为其编写基础的guest ROM,设计一些外设的IO, 可以通过IO和host交互
- host发放虚拟的keypad操作在ROM中人工输入操作

windknown