KCon

KCon 2019 無界

2019

滴滴出行

基于符号执行的反混淆方法研究

演讲人：糜波

目录
CONTENTS

01 PART 01 混淆框架简介

02 PART 02 混淆技术原理

03 PART 03 反混淆技术原理

04 PART 04 后续的工作

KCon

PART.01

CLICK ADD RELATED TITLE TEXT, AND CLICK ADD RELATED TITLE TEXT, CLICK ADD RELATED TITLE TEXT, CLICK ON ADD RELATED TITLE WORDS.

混淆框架简介

- 混淆技术是基于OLLVM开源代码， https://github.com/obfuscator-llvm/obfuscator

- 是瑞士西北应用科技大学于2010年6月份发起的一个项目，该项目旨在提供一套开源的基于LLVM的代码混淆工具，以增加逆向工程的难度。

- LLVM是开源的编译器框架，LLVM出现也是为了替换与系统紧耦合的GCC编译器。
- LLVM分为三个独立模块，高级语言解析、中间语言（IR）处理、目标机器语言生成。三个模块都具备可扩展性。
- 高级语言支持C/C++，OC等，目标指令支持x86、ARM、mips等，混淆是基于IR进行的扩展。

PART.02

CLICK ADD RELATED TITLE TEXT, AND CLICK ADD RELATED TITLE TEXT, CLICK ADD RELATED TITLE TEXT, CLICK ON ADD RELATED TITLE WORDS.

混淆技术原理

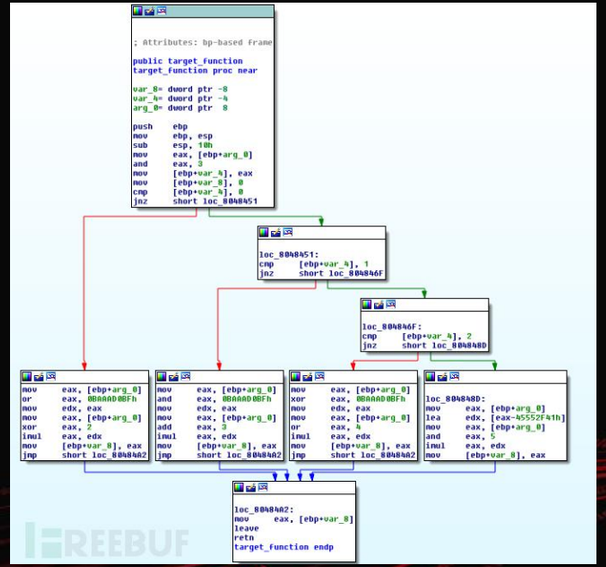
- OLLVM4.0主要支持三种混淆特性:

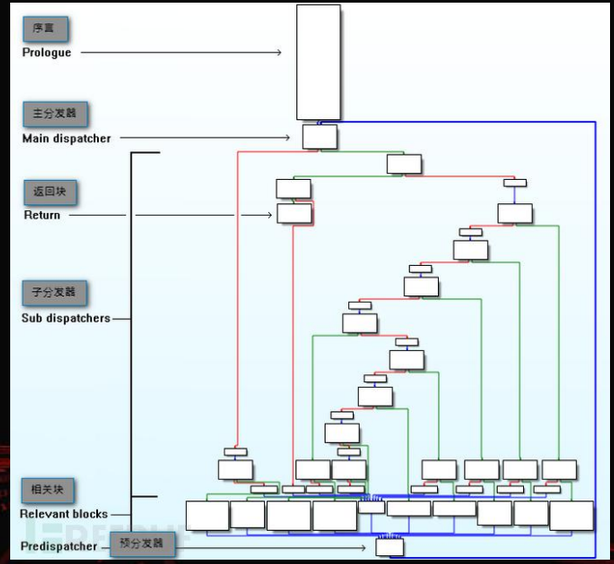
a. 控制流平坦化

b. 虚假控制流

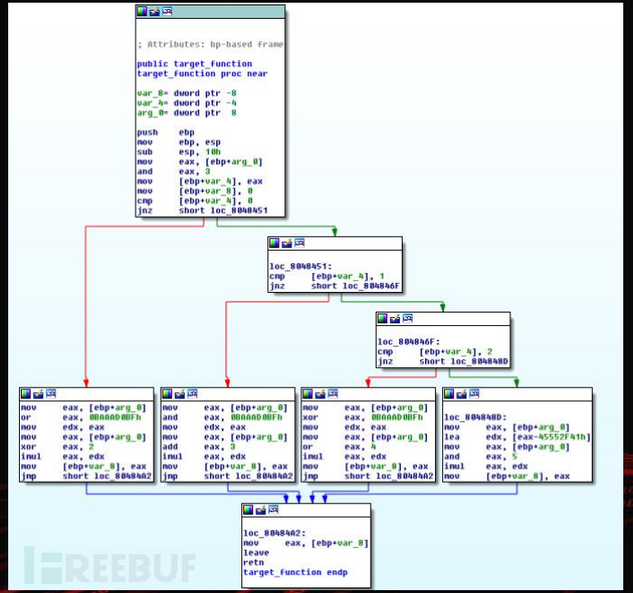
c. 指令替换

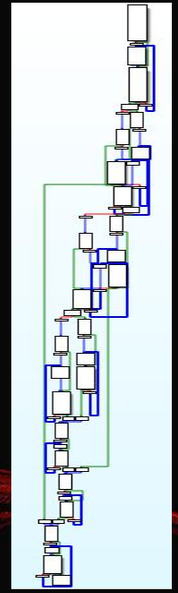
- 控制流平坦化（引自freebuf)



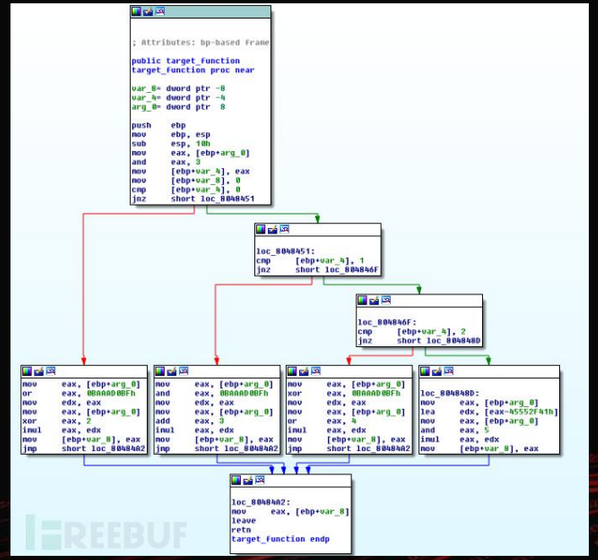


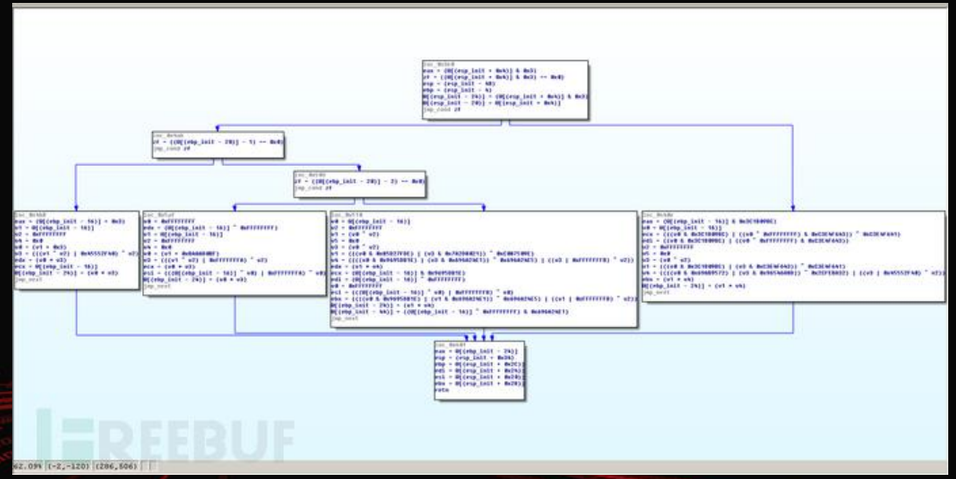
- 虚假控制流（引自freebuf)





- 指令替换（引自freebuf)





- 混淆技术原理——虚假控制流
  - 引入不透明谓词
    - (y <10 || x *(x + 1)%2 == 0)
- https://github.com/obfuscator-llvm/obfuscator/wiki/Bogus-Control-Flow

- 混淆技术原理——指令替换

加法 a = b + c：

a = b - (-c)

a = -(-b + (-c))

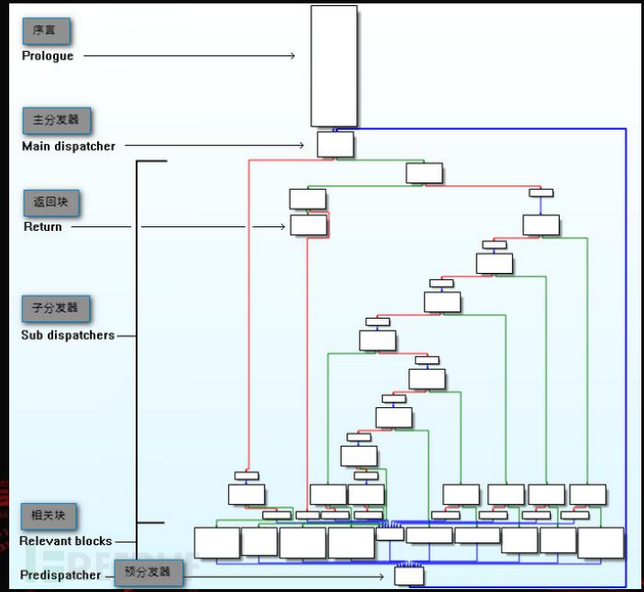
r = rand (); a = b + r; a = a + c; a = a - r

r = rand (); a = b - r; a = a + b; a = a + r

减法、与、或、异或运算：

https://github.com/obfuscator-llvm/obfuscator/wiki/Instructions-Substitution

- 混淆技术原理——控制流平坦化
  - 有块变量（可能是堆栈或寄存器变量）
  - 初始化后的块变量，经过二分搜索执行块
  - 当前块修改块变量，决定下一步执行哪个块
  - 原始代码中的分支，会影响块变量赋值



PART.03

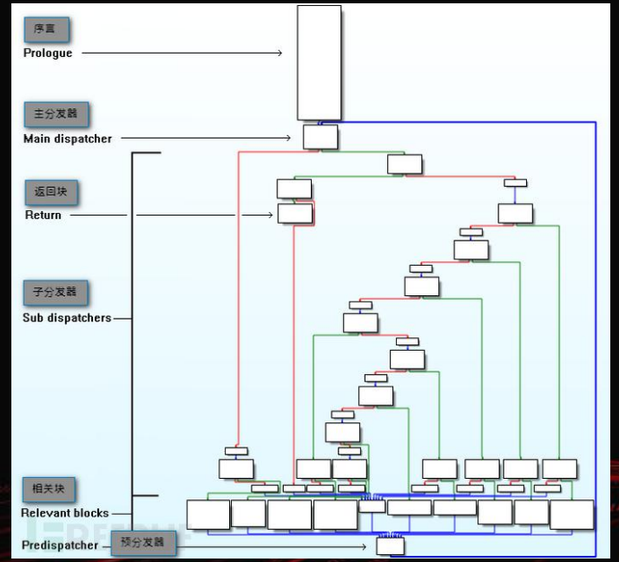
CLICK ADD RELATED TITLE TEXT, AND CLICK ADD RELATED TITLE TEXT, CLICK ADD RELATED TITLE TEXT, CLICK ON ADD RELATED TITLE WORDS.

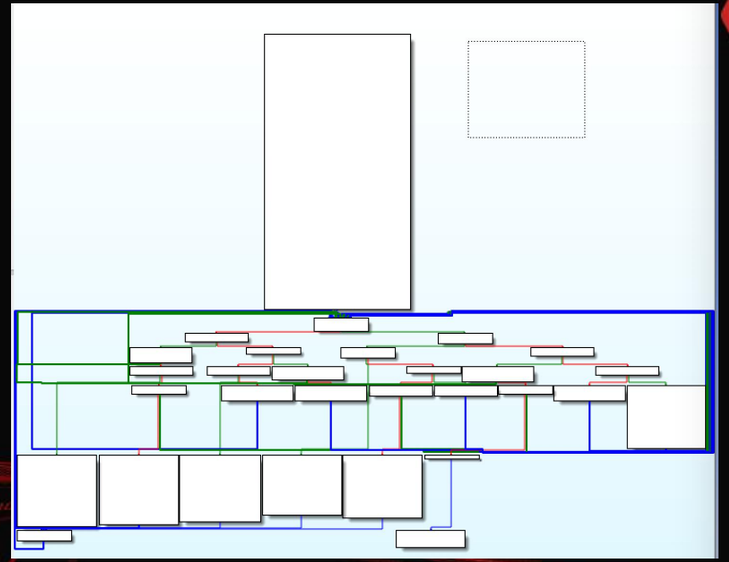
反混淆技术原理

- 认识符号执行引擎
  - 用符号代替变量
  - 模拟程序执行
  - 约束求解
  - 我使用的是符号执行引擎释angr
    https://github.com/angr/angr

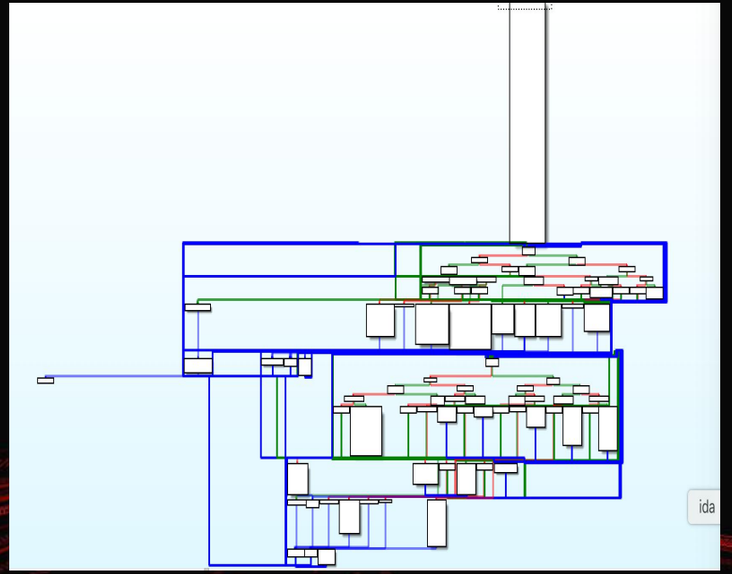
- 去流程平坦化思路
  - 找到所有真实块
  - 从序言块符号执行
  - 到第一个真实块即为序言块的后继
  - 再递归找这个真实块的后继
  - 遇到分支进入递归执行
  - 递归返回时修改分支条件继续执行
  - 输出（patch汇编 or 其他）

- 寻找基本块思路 Case 1
  - 序言块是真实块，后继是主分发器
  - 后继是主（预）分发器都是真实块
  - 没有后继的是return块（也是真实块）

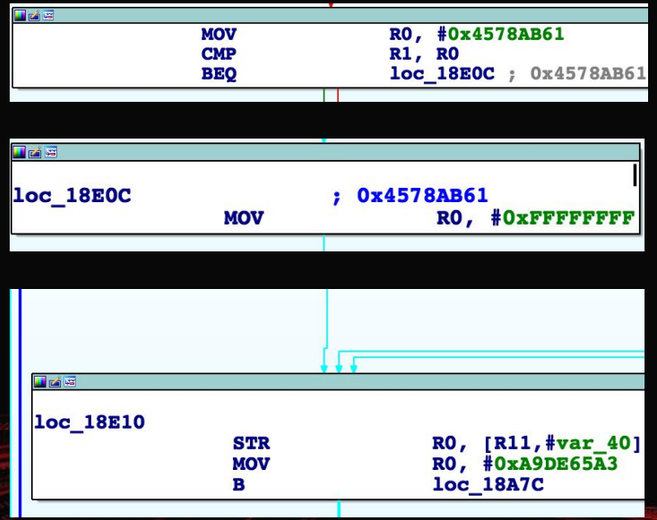




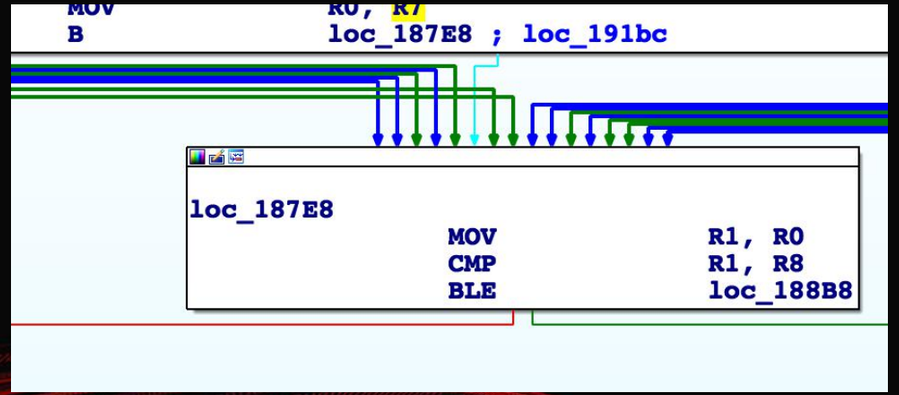
- 寻找基本块思路 Case 2
  - 两个以上主分发器
  - 手工指定主分发器地址
  - “人工的智能”往往最简单有效



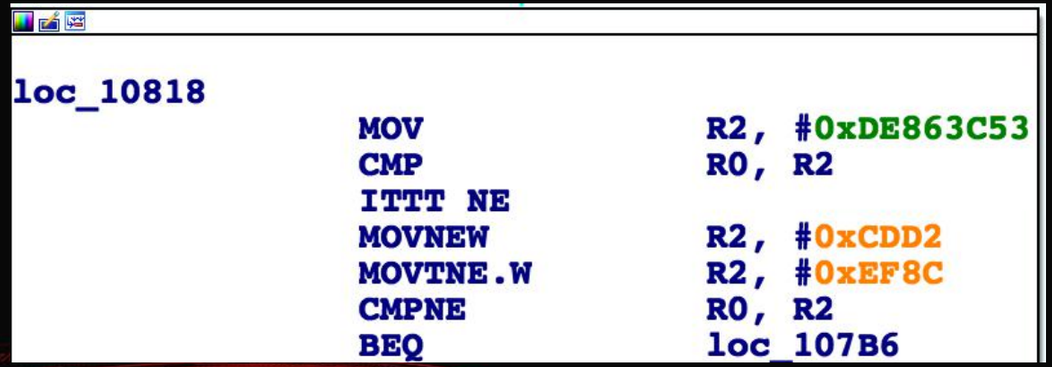
- 寻找基本块思路 Case 3
  - 真实块被优化成多个
  - 在BEQ或BNE和主（预）分发器之间的是真实块



- 为什么从函数开头符号执行
  - 和块变量比较的可能不是常量
  - 这些block value在序言块中初始化



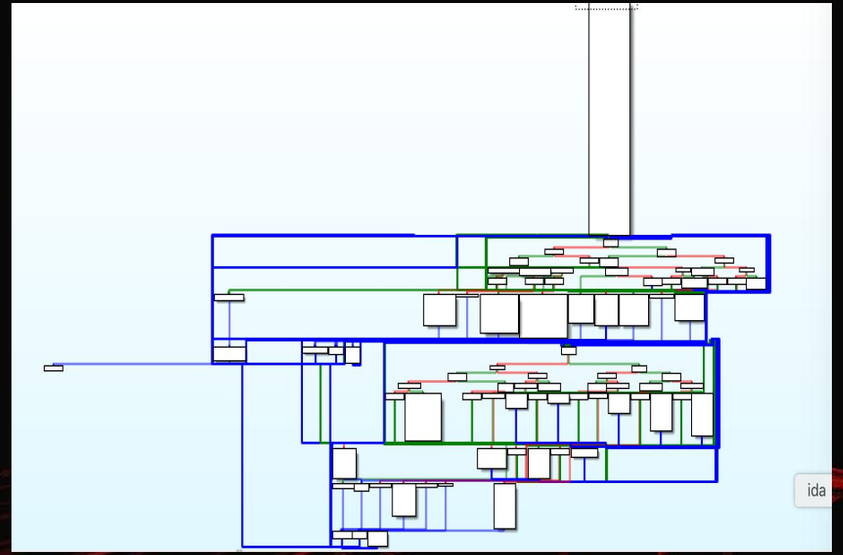
- 识别原始分支
  - 单纯控制流平坦化识别相对容易
  - ARM 32遇到ITT指令，即可认为是是原始分支
  - 加上虚假控制流，流程变得复杂，可以考虑使用约束求解，还在研究中……

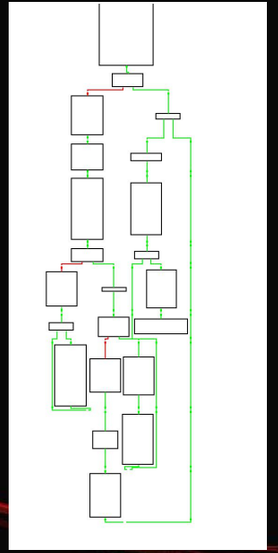


- 输出——patch汇编or其他
  - 指令空间不够
  - 输出GDL文件，用wingraph打开。

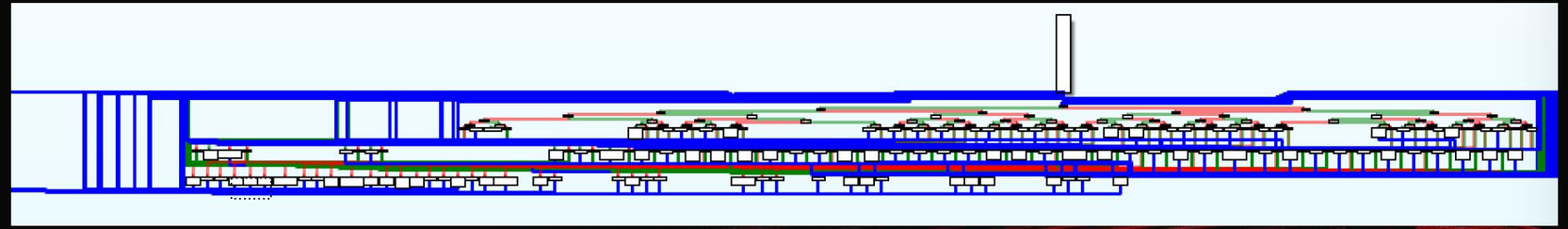
```
00011508                LDR.W           R3, =(byte_37280 - 0x1151A)
0001150C                MOVW            R11, #0xD79A
00011510                LDR             R0, [SP,#0xF4+var_B0]
00011512                MOVT.W          R11, #0x9E56
00011516                ADD             R3, PC  ; byte_37280
00011518                ADDS            R7, R0, #1
0001151A                LDRB            R4, [R3,R0]
0001151C                CMP             R7, #0x2A ; '*'
0001151E                ITT EQ
00011520                MOVEQW          R11, #0xD73
00011524                MOVTEQ.W        R11, #0xB19A
00011528                STR             R7, [SP,#0xF4+var_B0]
0001152A                EOR.W           R4, R4, #0x8A
0001152E                STRB            R4, [R3,R0]
00011530                B.W             loc_10954
```

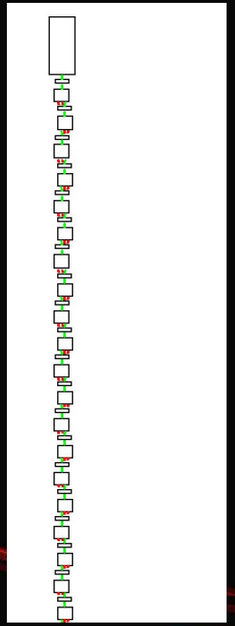
- 最终效果
  - 混淆代码vs还原的代码





- 最终效果
  - 混淆代码vs还原的代码





PART.04

CLICK ADD RELATED TITLE TEXT, AND CLICK ADD RELATED TITLE TEXT, CLICK ADD RELATED TITLE TEXT, CLICK ON ADD RELATED TITLE WORDS.

后续的工作

- GDL文件输出只是图形，不方便查找和交叉引用。
- 计划编写hex-ray的插件，修改反编译的Ctree结构。
- IDA Pro 7.2的Hex-Rays api 中microcode引入了block概念，由此可以调整block的后继。

- 另外一种输出方式--指令patch
  - 做指令迁移
    - 这里坑比较多
  - 重写SO文件

```
00011508                LDR.W           R3, =(byte_37280 - 0x1151A)
0001150C                MOVW            R11, #0xD79A
00011510                LDR             R0, [SP,#0xF4+var_B0]
00011512                MOVT.W          R11, #0x9E56
00011516                ADD             R3, PC  ; byte_37280
00011518                ADDS            R7, R0, #1
0001151A                LDRB            R4, [R3,R0]
0001151C                CMP             R7, #0x2A ; '*'
0001151E                ITT EQ
00011520                MOVEQW          R11, #0xD73
00011524                MOVTEQ.W        R11, #0xB19A
00011528                STR             R7, [SP,#0xF4+var_B0]
0001152A                EOR.W           R4, R4, #0x8A
0001152E                STRB            R4, [R3,R0]
00011530                B.W             loc_10954
```

谢谢观看

演讲人：糜波