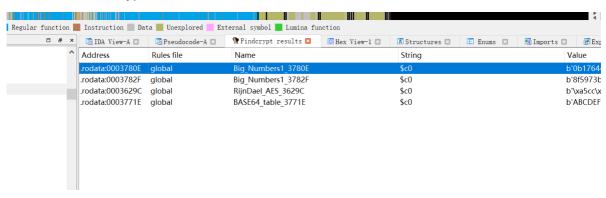
```
function testsiua(plainText){
   Java.perform(function () {
       var datacollect =
Java.use('com.meituan.android.common.datacollection.DataCollectionJni');
       var ByteString = Java.use("com.android.okhttp.okio.ByteString");
       var Base64 = Java.use("java.util.Base64");
       // 参数1是上下文
       var currentApplication=
Java.use("android.app.ActivityThread").currentApplication();
       var context = currentApplication.getApplicationContext();
       // 参数2是明文字节数组
       // 使用String类将我们传入的字符串转成数组
       var JavaString = Java.use("java.lang.String");
       var bytesStr = JavaString.$new(plainText).getBytes();
       // 参数3是明文的长度
       var textLength = plainText.length;
       // 主动调用
       var result = datacollect.packData(context, bytesStr, textLength)
       // 返回base64编码后的结果,使用系统类转换成十六进制
       var base64result = ByteString.of(result).utf8()
       var base64before = Base64.getDecoder().decode(base64result);
       // console.log("base64 result:"+base64result);
       console.log("result:"+ByteString.of(base64before).hex());
   });
}
```

观察输入和输出

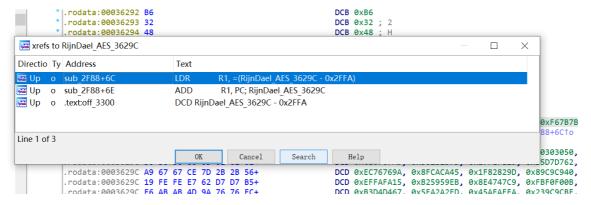
发现呈现出分组的情态,且分组大小为32字节,疑似AES。

进入SO,使用Findcrypt



有一处AES

查找其交叉引用



如果熟悉AES算法, 你会知道这就是T表

交叉引用指向2F88函数,即为包含AES运算的函数

查找2F88上级,因为包含AES具体运算的部分,很可能要到外层才能看到业务逻辑。

这儿需要有对分组加密的经验和知识

```
qmemcpy(v30, a5, 0x10u);
if ( a6 )
 if ( a3 >= 0x10 )
   v25 = a3 - 16;
   v23 = (a3 - 16) \& 0xFFFFFFF9;
   v27 = &_stack_chk_guard;
   v12 = v7;
  v26 = &v6[v23 + 16];
   do
   {
     qmemcpy(v30, v12, 0x10u);
     a3 -= 16;
     v12 += 16;
     v6 += 16;
   while ( a3 > 0xF );
   v7 += v23 + 16;
   v6 = v26;
   v9 = v27;
   a3 = v25 - v23;
 if ( a3 )
   v14 = 0;
   do
```

```
*((_BYTE *)v29 + i) = v6[i] ^ *((_BYTE *)v30 + i);
    sub_2F88(v29, v12, v28);
                                           // 出现位置1
    qmemcpy(v30, v12, 0x10u);
    a3 -= 16;
    v12 += 16;
    v6 += 16;
  while ( a3 > 0xF );
  v7 += v23 + 16;
  v6 = v26;
  v9 = v27;
  a3 = v25 - v23;
if ( a3 )
  v14 = 0;
  do
  {
    *((_BYTE *)v29 + v14) = v6[v14] ^ *((_BYTE *)v30 + v14);
    ++v14;
  while ( a3 != v14 );
  if ( a3 <= 0xF )
    qmemcpy((char *)v29 + a3, (char *)v30 + a3, 16 - a3);
                                          // 出现位置2
  sub_2F88(v29, v29, v28);
  qmemcpy(v7, v29, 0x10u);
  v30[0] = v29[0];
  \sqrt{30[1]} = \sqrt{29[1]};
  \sqrt{30}[2] = \sqrt{29}[2];
  \sqrt{30}[3] = \sqrt{29}[3];
}
```

位置1是明文长度大于一个分组的情况

位置2是明文长度只需要一个分组去加密的情况

显然位置2代码量更小,但逻辑是一样的,所以看2.

这个代码的理解依赖于对AES 实现源码的熟悉

```
void AES_CBC_encrypt_buffer(struct AES_ctx *ctx, uint8_t* buf, uint32_t length)

uintptr_t i;
uint8_t *Iv = ctx->Iv;
for (i = 0; i < length; i += AES_BLOCKLEN)

XorWithIv(buf, Iv);
Cipher((state_t*)buf, ctx->RoundKey);
Iv = buf;
buf += AES_BLOCKLEN;

/* store Iv in ctx for next call */
memcpy(ctx->Iv, Iv, AES_BLOCKLEN);

}
```

对2F88进行静态分析或者动态调试

都可以发现这三个参数的意义

```
1 unsigned int *_fastcall sub_2F88(unsigned int *text, _BYTE *buffer, int key)
2 {
3    int v3; // r1
4    int v4; // r3
5    int v5; // r1
```

AES 中重要的是这些内容

1.明文

可以发现明文和我们输入的不同,但没关系,先不去考虑它

2.密钥

暂时还不知道

3.IV

我们已经知道IV了

4.密钥是128还是256还是192

不知道(其实继续往上追溯也可以知道,但我们后续可以用一种密码学的、通用的方式去分析)

5.工作模式

CBC

所以接下来的重点就是在密钥上。

根据对AES的理解以及程序的分析,这个key有问题,不是种子密钥

往上层找

找到2B34函数,Hook发现确实是它做了密钥的编排工作。

根据传进去的种子密钥的长度,发现是128位密钥,也知道了密钥的值

进行校验,发现不对,无法加密出相同结果。

接下来开始排验AES是否被魔改。

这个过程讲究的是一步一步

第一步就是密钥的编排

对比Clion发现编排有问题

那么有两个思路

1是对编排算法进行还原

2是先不还原,使用它继续往下走

选择2

对照往下走

发现addRoundKey环节不同,类似于大小端序的处理

可以修改subkeys表,也可以修改程序逻辑。

发现AES 加密成功。

总结: siua 对AES 做了魔改,改了密钥相关的处理,包括密钥编排和密钥异或两个环节。