sgmain 6.4.x xsign 加密算法分析研究

原创 qinless 逆向小友 2022-04-08 10:24

收录于话题
#算法 13 #加密算法 1

仅供学习研究 。请勿用于非法用途，本人将不承担任何法律责任。

前言

> `sgmain 6.4.x` 版本的 `x-sign` 参数加密算法研究分析 样本 `tianmao-8.11.0`

使用到的工具

本文主要使用 `ida + unidbg` 动静态分析

样本 `unidbg` 参考文章

- 某宝系 tb tm sgmain x-sign 分析 - unidbg
- https://www.qinless.com/17

该算法有白盒 `aes`，会使用 `dfa` 攻击获取密钥

参考文章

- 第一讲——从黑盒攻击模型到白盒攻击模型
- https://www.qinless.com/1642
- 密码学学习记录 | aes dfa 练习样本一
- https://www.qinless.com/1698

「Tips: 博主也是 dfa 的初学者，文中有描述错误的情况，还望各位大佬轻喷」

「因为 so 对抗的原因，ida 看不到 c 代码，所以会使用 unidbg 大量 trace。trace 的过程很心酸，这一块不会写的很详细」

之前有位大佬发了篇 `x-sign` 的加密流程（已经被删），我也是基于此来的灵感，大概的流程是

「hmac_sha1 -> white_aes -> base64 -> hmac_sha1」

基于此流程开始分析

so hmac_sha1

```
IDA View-A                                                    Strings window
   1 int __fastcall sub_9AECE(_DWORD *a1)
   2 {
   3   *a1 = 0x67452301;
   4   a1[1] = 0xEFCDAB89;
   5   a1[2] = 0x98BADCFE;
   6   a1[3] = 0x10325476;
   7   a1[4] = 0xC3D2E1F0;
   8   a1[5] = 0;
   9   a1[6] = 0;
  10   a1[23] = 0;
  11   a1[24] = 0;
  12   a1[25] = 0;
  13   return _aeabi_memclr4();
  14 }
0009AECE sub_9AECE:1 (9AECE)
```

Output window

```
  bytes   pages size description
--------- ----- ---- --------------------------------------------
  2506752   306 8192 allocating memory for b-tree...
  3588096   438 8192 allocating memory for virtual array...
   262144    32 8192 allocating memory for name pointers...
-------------------------------------------------------------
  6356992            total memory allocated

Loading processor module /Users/qinjiahu/ida_pro_7.0/ida.app/Contents/MacOS/procs/arm.dylib for ARM...OK
Loading type libraries...
Autoanalysis subsystem has been initialized.
findHash (v0.1) plugin has been loaded.
Database for file 'libsgmainso-6.4.156.so' has been loaded.
Hex-Rays Decompiler plugin has been loaded (v7.0.0.170914)
  License: 56-BC5B-5634-8F Jiang Ying, Personal license (1 user)
  The hotkeys are F5: decompile, Ctrl-F5: decompile all.
  Please check the Edit/Plugins menu for more informaton.
IDAPython Hex-Rays bindings initialized.
Hexlight plugin installed Mod by Snow
-------------------------------------------------------------------------------------------------------------------------------
Python 2.7.16 (default, Jun 18 2021, 03:23:53)
[GCC Apple LLVM 12.0.5 (clang-1205.0.19.59.6) [+internal-os, ptrauth-isa=deploy
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-------------------------------------------------------------------------------------------------------------------------------
findHash (v0.1) plugin has been loaded.
**************************在二进制文件中检索hash算法常量**********************************
0x9aecf:函数sub_9AECE疑似哈希函数，包含初始化魔数的代码。
0x9b41f:函数sub_9B41E疑似哈希函数运算部分。
**************************存在以下可疑的字符串**********************************
```

这里使用龙哥的 `findhash` 跑一下，定位到了 `sha1` 特征函数

```
    {
      _aeabi_memcpy();
      _aeabi_memcpy();
    }
    else
    {
      if ( !sub_9B1E0(v1, v2, &v12, 0x14u) )
      {
        v1 = 0;
LABEL_27:
        v9 = 0;
        goto LABEL_28;
      }
      v22 = v12;
      v23 = v13;
      v24 = v14;
      v25 = v15;
      v26 = v16;
      v17 = v12;
      v18 = v13;
      v19 = v14;
      v20 = v15;
      v21 = v16;
    }
    v6 = 0;
    do
    {
      *(&v22 + v6) ^= 0x36u;
      *(&v17 + v6++) ^= 0x5Cu;
    }
    while ( v6 != 64 );
    v7 = v3[3];
    v8 = 85;
    v9 = v3[6];
    if ( v7 > 0x14 )
      v8 = v7 + 65;
    if ( v9 && v3[7] >= v8 )
    {
      v1 = 0;
    }
    else
    {
      v9 = malloc(v8);
      v1 = (&dword_0 + 1);
      if ( !v9 )
        goto LABEL_27;
    }
    _aeabi_memcpy();
    v10 = *(v3 + 1);
    _aeabi_memcpy();
    if ( sub_9B1E0(v9, v3[3] + 64, &v12, 0x14u) )
    {
      _aeabi_memcpy();
      _aeabi_memcpy();
      v3 = sub_9B1E0(v9, 84, *(v3 + 2), *(v3 + 2) >> 32);
      goto LABEL_29;
    }
LABEL_28:
    v3 = 0;
LABEL_29:
    v11 = v1 == 0;
    if ( v1 )
      v11 = v9 == 0;
    if ( !v11 )
      free(v9);
  }
  return v3;
```

查看函数的交叉引用，往上跟踪，来到这个函数。这里就是 hmac 函数了，稍后在这里 debugger 一下，看看输入跟 key

unidbg hmac_sha1

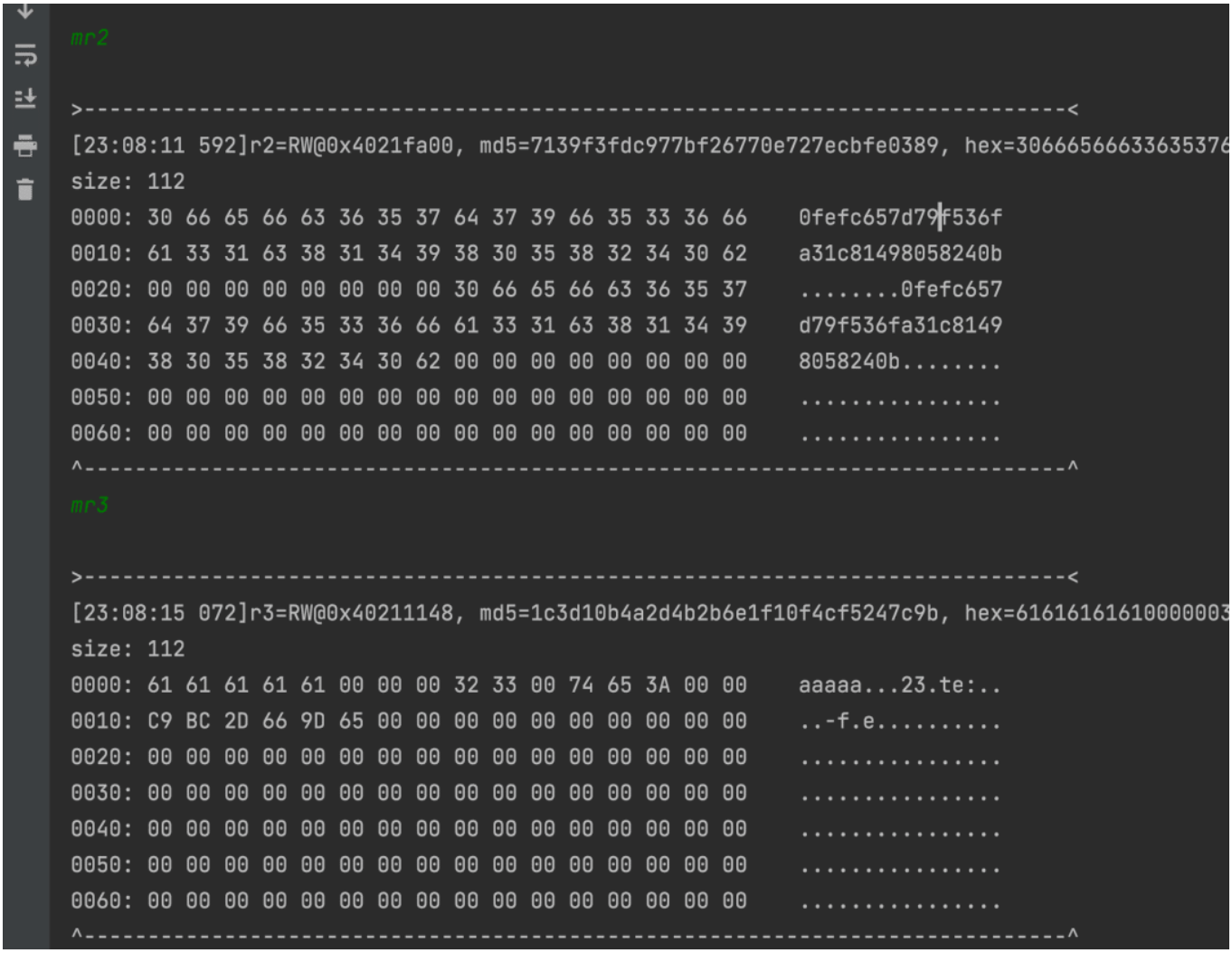
```
public void getXSign() {
    Map<String, String> map = new HashMap<>();
      map.put("INPUT", "&&&23181017&1c9d79ea8
    map.put("INPUT", "aaaaa");
    DvmObject<?> ret = JNICLibrary.callStatic
            emulator, methodSign, 10401
```

这里先把输入随便改一下

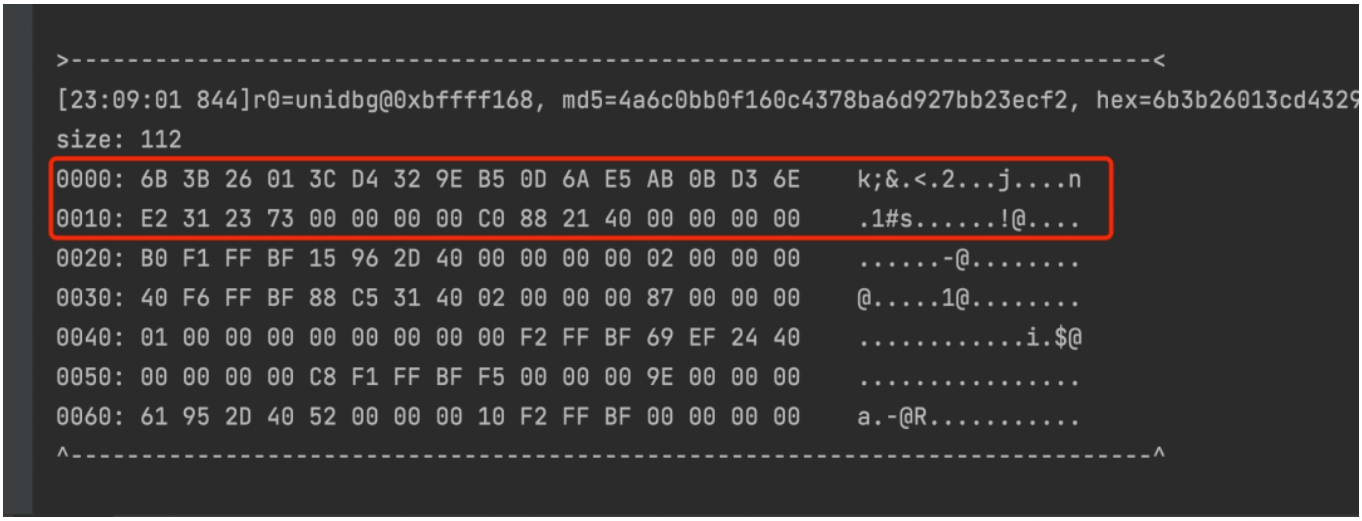
```
public void consoleDebugger() {
    Debugger debugger = emulator.attach(DebuggerType.CONSOLE);

    debugger.addBreakPoint(base + 0x9A0E0);
}
```

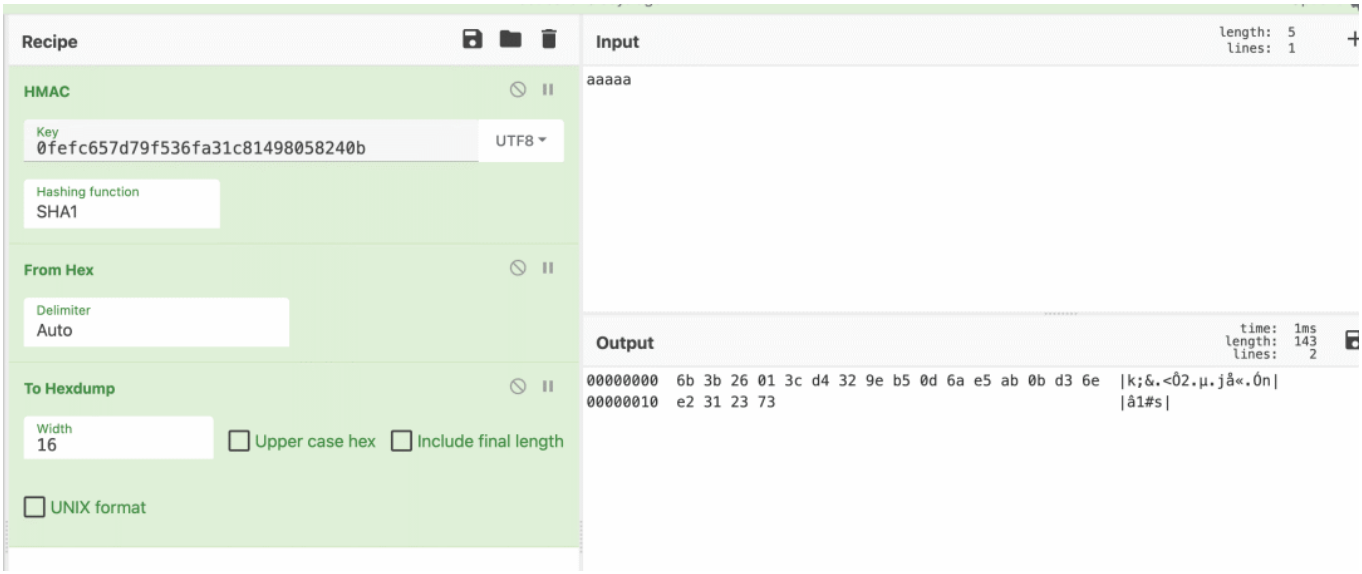
然后在 `hmac` 函数处下个断点



成功断下来，参数四是输入，参数三看起来是 `hmac key` 稍后试一下



函数执行完查看输出，这里就是 `hmac sha1` 的加密结果了



使用逆向之友加密，结果相同，说明是标准的 `hmac sha1` 函数了

「`hmac sha1` 算法分析出来之后，就要开始使用 `unidbg trace` 大法，追踪了」

unidbg trace 分析白盒 aes

```
public void trace() {
    emulator.traceRead(0xbffff168L, 0xbffff168L + 20);
}
```

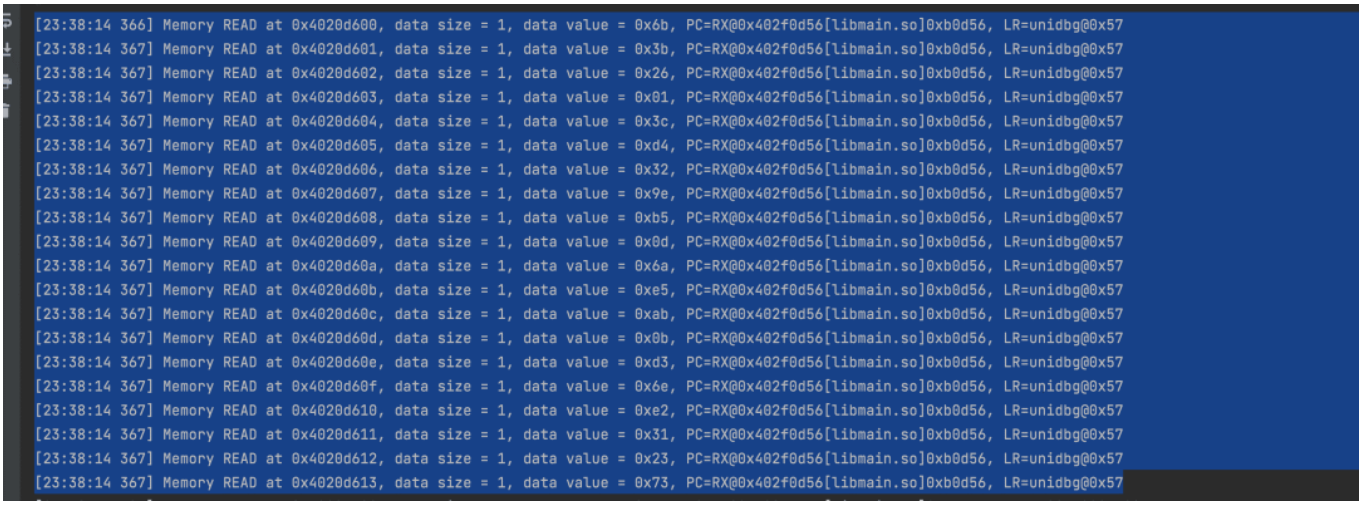
直接 `traceRead` 这个地址的 20 个字节数据，代码跑起来发现没读取

`unidbg 0xbfff` 地址开头是默认是栈，所以数据可能在堆里，去搜索一下

```
shr 6b3b26
Search heap matches 1 count
Heap matches: RW@0x4020d600
m0x4020d600

>------------------------------------------------------------------------<
[23:33:51 607]RW@0x4020d600, md5=efd5f15ece321bbc815df942bc089429, hex=6b3b26013cd4329eb50d6ae5a
size: 112
0000: 6B 3B 26 01 3C D4 32 9E B5 0D 6A E5 AB 0B D3 6E    k;&.<.2...j....n
0010: E2 31 23 73 00 00 00 00 E8 A7 62 70 8D 36 FD 71    .1#s......bp.6.q
0020: 57 59 91 2F B7 51 CF FB EA 56 62 10 00 00 00 00    WY./.Q...Vb.....
0030: 44 63 4F 2F 6C 63 4B 2B 68 3F 6D 33 63 2A 71 40    DcO/lcK+h?m3c*q@
0040: 00 69 6E 74 00 00 00 00 00 00 00 00 00 00 00 00    .int............
```

使用 `shr` 搜索可读堆（注意后面会经常用，就不再解释了），发现数据在 `0x4020d600` 地址里也有的



这里改改地址，跑起来，看到有 `trace` 结果了

```
2 {
3   unsigned __int8 *v3; // r3
4   int result; // r0
5   _BYTE *v5; // r2
6   unsigned int v6; // r4
7   unsigned int v7; // t1
8   char v8; // r5
9   char v9; // r6
10  int v10; // r5
11
12  v3 = a1;
13  result = 0;
14  if ( v3 )
15  {
16    if ( a2 )
17    {
18      if ( a3 )
19      {
20        result = 2 * a2;
21        if ( a2 >= 1 )
22        {
23          v5 = (a3 + 1);
24          do
25          {
26            v7 = *v3++;
27            v6 = v7;
28            v8 = v7 & 0xF;
29            v9 = v7 & 0xF | 0x30;
30            if ( (v7 & 0xF) > 9 )
31              v9 = v8 + 87;
32            v10 = (v6 >> 4) | 0x30;
33            *v5 = v9;
34            if ( v6 > 0x9F )
35              v10 = (v6 >> 4) + 0x57;
36            --a2;
37            *(v5 - 1) = v10;
38            v5 += 2;
39          }
40          while ( a2 );
41        }
42      }
43    }
44  }
45  return result;
46 }
```

ida 跳转过来，也看不懂是个啥，下面写个代码 hook 一下看看

通过胡乱一通调试，修改后最终的 `inlinehook` 代码如上

```
[23:45:12 423] Memory READ at 0x4020d600, data size = 1, data value = 0x6b, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:36, res2:36, r4:6b, r6:62, r2:402150c1, r12 :30
[23:45:12 424] Memory READ at 0x4020d601, data size = 1, data value = 0x3b, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:33, res2:33, r4:3b, r6:62, r2:402150c3, r12 :30
[23:45:12 424] Memory READ at 0x4020d602, data size = 1, data value = 0x26, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:32, res2:32, r4:26, r6:36, r2:402150c5, r12 :30
[23:45:12 424] Memory READ at 0x4020d603, data size = 1, data value = 0x01, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:30, res2:30, r4:1, r6:31, r2:402150c7, r12 :30
[23:45:12 424] Memory READ at 0x4020d604, data size = 1, data value = 0x3c, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:33, res2:33, r4:3c, r6:63, r2:402150c9, r12 :30
[23:45:12 424] Memory READ at 0x4020d605, data size = 1, data value = 0xd4, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:3d, res2:3d, r4:d4, r6:34, r2:402150cb, r12 :30
[23:45:12 424] Memory READ at 0x4020d606, data size = 1, data value = 0x32, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:33, res2:33, r4:32, r6:32, r2:402150cd, r12 :30
[23:45:12 424] Memory READ at 0x4020d607, data size = 1, data value = 0x9e, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:39, res2:39, r4:9e, r6:65, r2:402150cf, r12 :30
[23:45:12 424] Memory READ at 0x4020d608, data size = 1, data value = 0xb5, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:3b, res2:3b, r4:b5, r6:35, r2:402150d1, r12 :30
[23:45:12 424] Memory READ at 0x4020d609, data size = 1, data value = 0x0d, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:30, res2:30, r4:d, r6:64, r2:402150d3, r12 :30
[23:45:12 424] Memory READ at 0x4020d60a, data size = 1, data value = 0x6a, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:36, res2:36, r4:6a, r6:61, r2:402150d5, r12 :30
[23:45:12 424] Memory READ at 0x4020d60b, data size = 1, data value = 0xe5, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:3e, res2:3e, r4:e5, r6:35, r2:402150d7, r12 :30
[23:45:12 424] Memory READ at 0x4020d60c, data size = 1, data value = 0xab, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:3a, res2:3a, r4:ab, r6:62, r2:402150d9, r12 :30
[23:45:12 424] Memory READ at 0x4020d60d, data size = 1, data value = 0x0b, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:30, res2:30, r4:b, r6:62, r2:402150db, r12 :30
[23:45:12 424] Memory READ at 0x4020d60e, data size = 1, data value = 0xd3, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:3d, res2:3d, r4:d3, r6:33, r2:402150dd, r12 :30
[23:45:12 424] Memory READ at 0x4020d60f, data size = 1, data value = 0x6e, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
ORR.W R5, R12, R4,LSR#4 --- res1:36, res2:36, r4:6e, r6:65, r2:402150df, r12 :30
[23:45:12 424] Memory READ at 0x4020d610, data size = 1, data value = 0xe2, PC=RX@0x402f0d56[libmain.so]0xb0d56, LR=unidbg@0x57
```

`trace` 之后结果又写入到 `r2` 的地址了

```
m0x402150c1

>--------------------------------------------------------------------------------<
[23:45:27 045]RW@0x402150c1, md5=cd42a3f8de08299363615f7cfc99966e, hex=62336232363031336364
size: 112
0000: 62 33 62 32 36 30 31 33 63 64 34 33 32 39 65 62    b3b26013cd4329eb
0010: 35 30 64 36 61 65 35 61 62 30 62 64 33 36 65 65    50d6ae5ab0bd36ee
0020: 32 33 31 32 33 37 33 00 00 41 3D 00 00 00 00 00    2312373..A=.....
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
^--------------------------------------------------------------------------------^
```

看看 `r2` 的地址，好吧这就是 `hmac sha1` 的结果转成了 `hex`，继续 `trace`

```
shr 623362
Search heap matches 6 count
Heap matches: RW@0x402150c1
Heap matches: RW@0x402240f1
Heap matches: RW@0x40224141
Heap matches: RW@0x4038c0a1
Heap matches: RW@0x403ba0c1
Heap matches: RX@0x40ad4d8b[libsecuritybody.so]0x14d8b
```

好消息 `trace` 又没结果，堆找下，结果还挺多

```
m0x403ba0c1

>------------------------------------------------------------------------<
[23:52:22 662]RW@0x403ba0c1, md5=79bc5037a4044b88fe0d95f5bc982455, hex=623362323630313363643
size: 112
0000: 62 33 62 32 36 30 31 33 63 64 34 33 32 39 65 62    b3b26013cd4329eb
0010: 35 30 64 36 61 65 35 61 62 30 62 64 33 36 65 65    50d6ae5ab0bd36ee
0020: 32 33 31 32 33 37 33 26 30 66 65 66 63 36 35 37    2312373&0fefc657
0030: 64 37 39 66 35 33 36 66 61 33 31 63 38 31 34 39    d79f536fa31c8149
0040: 38 30 35 38 32 34 30 62 07 07 07 07 07 07 07 00    8058240b........
0050: D0 DE 7A 00 00 00 00 00 00 00 00 00 00 00 00 00    ..z.............
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
^------------------------------------------------------------------------^
```

这里发现了有趣的地方，这不就是 `hmac_sha1 + & + hmac_sha1_key` 吗

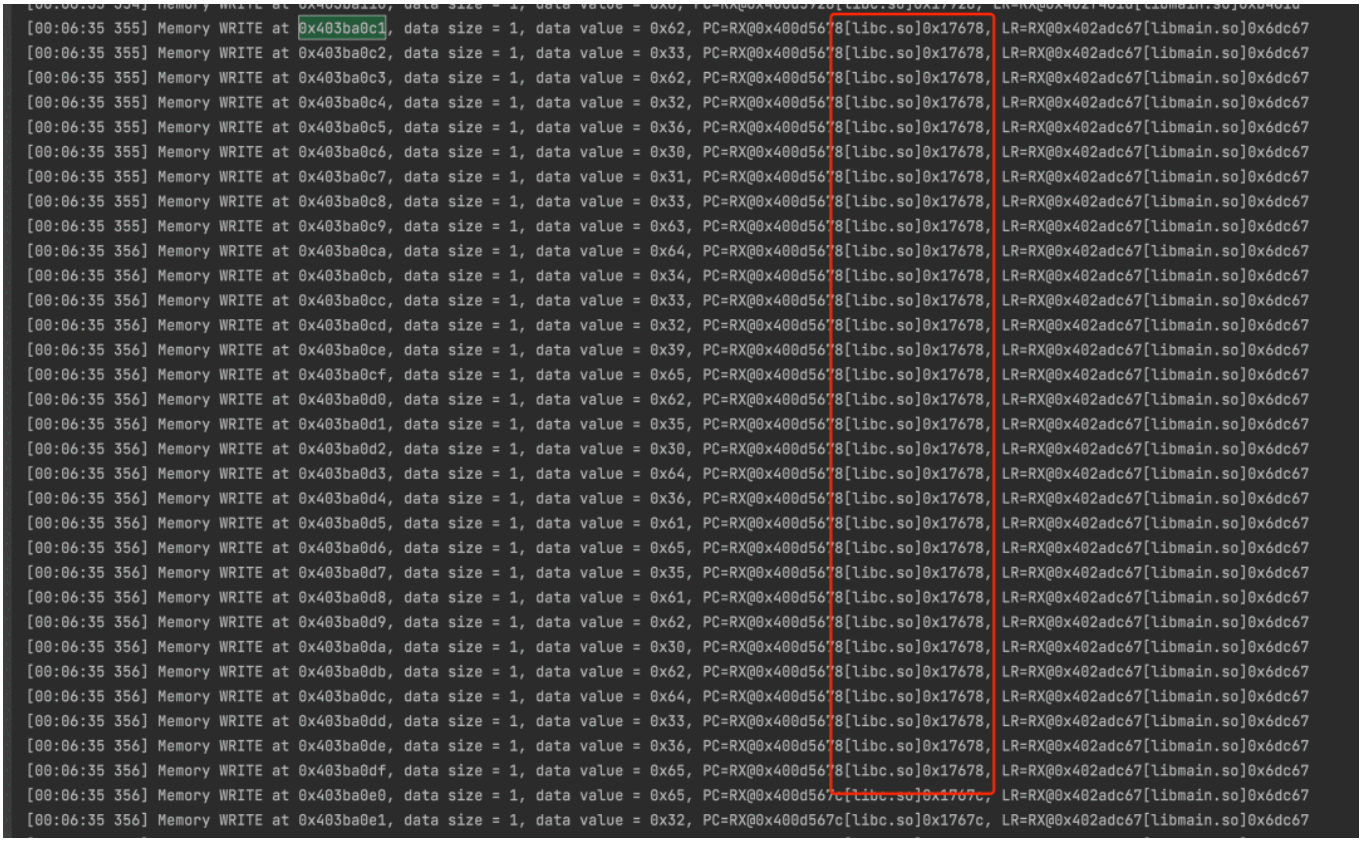
重要的是后面的 `07` 这是啥 ？，这不妥妥的就是 `pkcs7` 填充的特征吗，很明显是 `aes` 的输入

所以果断放弃前面跟踪的结果，拥抱新欢，开始跟踪这个

先别高兴的太早，输入有 `73 byte`，填充 `7 byte`（太长了 = =），使用 `dfa` 攻击太麻烦了，所以需要先把输入改成一组也就是 `16 byte` 以下（`dfa` 不了解的去看看前面推荐文章，到这里默认了解）

unidbg 修改 aes 输入长度

继续需要 `trace` 这些数据哪来的



`trace` 发现在 `libc.so` 里写入的，猜测是 `memcpy` 函数，有兴趣的小伙伴可以去看看这个 `so` 的函数，这里是在 `libc.so` 操作的那我们咋整呢，直接 `hook` 也不太现实，那没办只好去 `hook` 后面的 `lr` 地址了

```
bt
[0x40240000][0x402aa4f5][        libmain.so][0x6a4f5]
[0x40240000][0x402aa5fb][        libmain.so][0x6a5fb]
[0x40240000][0x4024ef67][        libmain.so][0x0ef67]
[0x40240000][0x402a1b65][        libmain.so][0x61b65]
[0x40240000][0x4029d769][        libmain.so][0x5d769]
[0x40240000][0x4024ef67][        libmain.so][0x0ef67]
[0x40240000][0x40269e79][        libmain.so][0x29e79]
[0x40240000][0x4024ef67][        libmain.so][0x0ef67]
[0x40240000][0x402503d3][        libmain.so][0x103d3]
```

这里 `hook` 成功，查看调用栈

IDA View-A Strings w

```
1  void *__fastcall sub_6A5B8(__int64 *a1, int *a2)
2  {
3    int *v2; // r10
4    __int64 v3; // kr00_8
5    __int64 v4; // kr08_8
6    int v5; // r4
7    void **v6; // r8
8    void **v7; // r5
9    void **v8; // r6
10   _WORD *v9; // r0
11   void *v10; // r1
12   void *v11; // r4
13   int v12; // r1
14   void *result; // r0
15   int v14; // [sp+0h] [bp-30h]
16   void **v15; // [sp+4h] [bp-2Ch]
17   void **v16; // [sp+8h] [bp-28h]
18   int v17; // [sp+Ch] [bp-24h]
19   int v18; // [sp+10h] [bp-20h]
20   int v19; // [sp+14h] [bp-1Ch]
21
22   v2 = a2;
23   v19 = *off_D97C4;
24   v3 = *a1;
25   v4 = a1[1];
26   v5 = *(a1 + 4);
27   v6 = sub_B46A8((*a1 >> 32));
28   v7 = sub_B46DC(*HIDWORD(v4), *HIDWORD(v4) >> 32);
29   v14 = v3;
30   v15 = v6;
31   v16 = v7;
32   v17 = v5;
33   v18 = 1;
34   v8 = sub_6A4C4(&v14, v2);
35   if ( v8 && *v8 && v8[1] )
36   {
37     v9 = malloc(0xAu);
38     if ( v9 )
39     {
40       v9[4] = 0;
41       *(v9 + 1) = 0;
42       *v9 = 0;
43     }
44     *v4 = v9;
45     v10 = *v8;
46     _aeabi_memcpy();
47     v11 = malloc(v8[1] - 8);
48     if ( v11 )
49       _aeabi_memclr();
50     v12 = *v8 + 9;
51     _aeabi_memcpy();
52   }
53   else
54   {
55     v11 = 0;
56   }
57   sub_B4710(v7);
58   sub_B4710(v6);
59   sub_B4710(v8);
60   result = (*off_D97C4 - v19);
61   if ( *off_D97C4 == v19 )
62     result = v11;
63   return result;
64 }
```

这么多，一个一个看，其中一个地址在这里，抱着怀疑心情 `hook` 一下这里，发现该函数的参数并没有我们想要的数据，那可能就不是在这里，那后面咋整呢

博主也是卡了挺久，这里就直接说答案了，大家可以自行测试分析一下，入口就在上图的 `sub_b46dc` 函数，`hook` 一下

`r0` 正确，`r1` 刚好是 `r0` 的长度，直接在这里修改即可

具体咋修改参考龙哥的 `unidbg hook 大全`

修改完成咋看有没有修改成功呢，第一可以通过内存地址，第二可以通过 `hmac sha1` 参数

在第二次执行到 `hmac sha1` 查看第三个参数，就是 `aes -> base64` 之后的结果，正常应该是这么长的

修改之后的结果就只有这么长了，所以以上的逻辑是可行的

修改了输入就需要确定 `aes` 有没有 `iv` 干扰了，还要确定循环的轮数在哪

unidbg 分析 aes iv

这个如何分析呢，正常 `aes cbc` 模式，输入会先跟 `iv` 异或，所以直接 `trace` 输入

修改输入之后发现之前的 `trace` 没有了，那我继续搜索堆

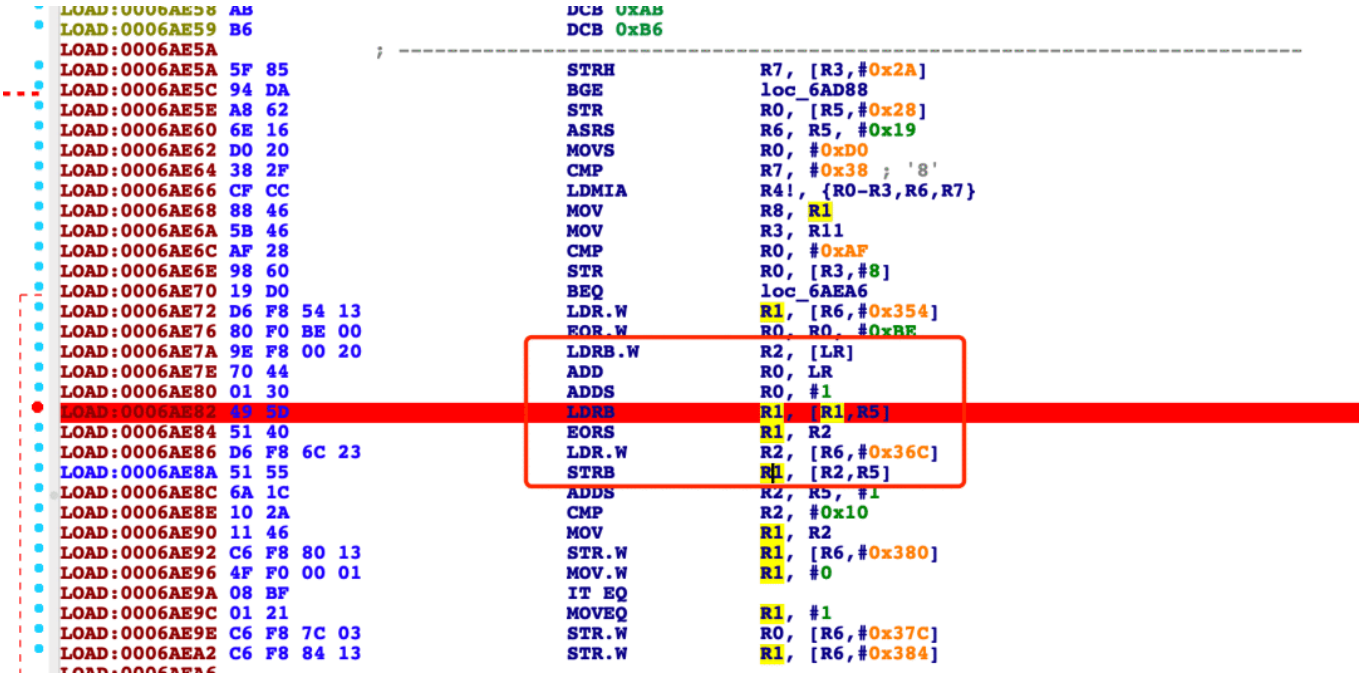
```
^-------------------------------------------------------------------------^
shr 090806
Search heap matches 3 count
Heap matches: RW@0x4020d618
Heap matches: RW@0x40211160
Heap matches: RW@0x402240f0
m0x4020d618

>-------------------------------------------------------------------------<
[00:42:23 324]RW@0x4020d618, md5=fe664d4b35e51cb89fa14defab264fbd, hex=0908060d0d0d0d0d
size: 112
0000: 09 08 06 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D    ...............
0010: 00 56 62 10 00 00 00 00 C2 6A 1D 2E 68 A6 D1 B5    .Vb......j..h...
0020: FE 97 A4 D3 59 5A ED 43 00 69 6E 74 00 00 00 00    ....YZ.C.int....
0030: D6 6C 69 09 89 2F B4 29 AA C4 66 F5 55 AC 4A 07    .li../.)..f.U.J.
0040: 00 00 00 00 00 00 00 00 80 22 98 11 F5 3A 3B D5    ........."...:;.
0050: 48 F4 54 5F A9 5C 8B 8B 00 00 00 00 00 00 00 00    H.T_.\..........
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ...............
^-------------------------------------------------------------------------^
```

这里的 `090806` 是我修改的，发现在 `0x4020d618` 地址里，继续 `traceRead`

```
[00:46:08 641] Memory READ at 0x4020d618, data size = 1, data value = 0x09, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d600
[00:46:08 641] Memory READ at 0x4020d619, data size = 1, data value = 0x08, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d601
[00:46:08 641] Memory READ at 0x4020d61a, data size = 1, data value = 0x06, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d602
[00:46:08 641] Memory READ at 0x4020d61b, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d603
[00:46:08 641] Memory READ at 0x4020d61c, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d604
[00:46:08 641] Memory READ at 0x4020d61d, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d605
[00:46:08 641] Memory READ at 0x4020d61e, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d606
[00:46:08 642] Memory READ at 0x4020d61f, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d607
[00:46:08 642] Memory READ at 0x4020d620, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d608
[00:46:08 642] Memory READ at 0x4020d621, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d609
[00:46:08 642] Memory READ at 0x4020d622, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60a
[00:46:08 642] Memory READ at 0x4020d623, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60b
[00:46:08 642] Memory READ at 0x4020d624, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60c
[00:46:08 642] Memory READ at 0x4020d625, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60d
[00:46:08 642] Memory READ at 0x4020d626, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60e
[00:46:08 642] Memory READ at 0x4020d627, data size = 1, data value = 0x0d, PC=RX@0x402aae82[libmain.so]0x6ae82, LR=RW@0x4020d60f
resolve.pathname-11111: /data/user/0/com.tmall.wireless/files
```

`trace` 成功， `ida` 跳过去看看



这里看下 `arm` 就可以了，取值异或存储，下面使用 `unidbg inlinehook` 看看

```
emulator.getBackend().hook_add_new(new CodeHook() {
    @Override
    public void hook(Backend backend, long address, int size, Object user) {
        if (address == (base + 0x6AE84)) {
            Arm32RegisterContext ctx = emulator.getContext();

            int r1 = ctx.getR1Int();
            int r2 = ctx.getR2Int();
            int res = r1 ^ r2;

            System.out.println(
                    "eors r1, r2 --- " +
                            "r1:" + Long.toHexString(r1) + ", " +
                            "r2:" + Long.toHexString(r2) + ", " +
                            "res:" + Long.toHexString(res)
            );
        }
    }

    @Override
    public void onAttach(UnHook unHook) {
    }

    @Override
    public void detach() {
    }
}, base + 0x6AE84, base + 0x6AE84, null);
```

```
eors r1, r2 --- r1:9, r2:36, res:3f
eors r1, r2 --- r1:8, r2:7a, res:72
eors r1, r2 --- r1:6, r2:69, res:6f
eors r1, r2 --- r1:d, r2:38, res:35
eors r1, r2 --- r1:d, r2:74, res:79
eors r1, r2 --- r1:d, r2:65, res:68
eors r1, r2 --- r1:d, r2:79, res:74
eors r1, r2 --- r1:d, r2:34, res:39
eors r1, r2 --- r1:d, r2:33, res:3e
eors r1, r2 --- r1:d, r2:32, res:3f
eors r1, r2 --- r1:d, r2:38, res:35
eors r1, r2 --- r1:d, r2:54, res:59
eors r1, r2 --- r1:d, r2:63, res:6e
eors r1, r2 --- r1:d, r2:55, res:58
eors r1, r2 --- r1:d, r2:68, res:65
eors r1, r2 --- r1:d, r2:31, res:3c
resolve.pathname-11111: /data/user/0/com.tmall.wireless/files
```

`r1` 是我们的输入，`r2` 就是 `iv` 了

找到了 `iv` 那就需要，就需要去分析轮数在哪里，分析轮数之前最好找到 `state` 这样可以更快速的定位到轮数，正常 `state` 的第一轮，`data ^ iv` 结果会跟 `key` 进行异或，但这是白盒 `aes` 是直接查表操作，没啥好的办法 = = 还是只能硬着头皮进行 `trace` 了

unidbg 分析 aes state

```
shr 3f726f
Search heap matches 1 count
Heap matches: RW@0x40212d80
m0x40212d80

>------------------------------------------------------------------------<
[00:57:55 237]RW@0x40212d80, md5=2053d5e96077c78b867b3a427c6eedd3, hex=3f726f35796874
size: 112
0000: 3F 72 6F 35 79 68 74 39 3E 3F 35 59 6E 58 65 3C    ?ro5yht9>?5YnXe<
0010: 00 2C AA 82 AB 87 BB D7 AB 88 BE E4 D4 56 2A 00    .,...........V*.
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

根据 `data ^ iv` 的结果，搜索堆，有结果，开始 `trace`

```
[01:01:51 994] Memory READ at 0x40212d80, data size = 1, data value = 0x3f, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=unidbg@0x1
[01:01:51 994] Memory READ at 0x40212d84, data size = 1, data value = 0x79, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 994] Memory READ at 0x40212d88, data size = 1, data value = 0x3e, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 994] Memory READ at 0x40212d8c, data size = 1, data value = 0x6e, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d81, data size = 1, data value = 0x72, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=unidbg@0x1
[01:01:51 995] Memory READ at 0x40212d85, data size = 1, data value = 0x68, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d89, data size = 1, data value = 0x3f, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d8d, data size = 1, data value = 0x58, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d82, data size = 1, data value = 0x6f, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=unidbg@0x1
[01:01:51 995] Memory READ at 0x40212d86, data size = 1, data value = 0x74, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d8a, data size = 1, data value = 0x35, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d8e, data size = 1, data value = 0x65, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d83, data size = 1, data value = 0x35, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=unidbg@0x1
[01:01:51 995] Memory READ at 0x40212d87, data size = 1, data value = 0x39, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d8b, data size = 1, data value = 0x59, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
[01:01:51 995] Memory READ at 0x40212d8f, data size = 1, data value = 0x3c, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=null
```

`ida` 跳过去看看

```
[01:00:16 375] Memory READ at 0x40212d80, data size = 1, data value = 0x3f, PC=RX@0x402aafd0[libmain.so]0x6afd0, LR=unidbg@0x1
debugger break at: 0x402aafe2 @ Function32 address=0x40250269, arguments=[unidbg@0xfffe12a0[libandroid.so]0x2a0, -2030840821, 10401, 9685140
>>> r0=0xbffff028(-1073745880) r1=0x40212d80 r2=0x0 r3=0xbffff028 r4=0x402aafcd r5=0x3f r6=0xbfffea90 r7=0xbffff170 r8=0x31 sb=0x94 sl=0xbff
>>> SP=0xbfffea80 LR=unidbg@0x1 PC=RX@0x402aafe2[libmain.so]0x6afe2 cpsr: N=1, Z=0, C=0, V=0, T=1, mode=0b10000
>>> d0=0xd0d0d0d0d0d0d0d0(8.309872195179385E-246) d1=0x3168556354383233(1.1017839574267251E-70) d2=0x909bc2c7c2949792(-1.1443957316916304E-22
>>> d8=0x0(0.0) d9=0x0(0.0) d10=0x0(0.0) d11=0x0(0.0) d12=0x0(0.0) d13=0x0(0.0) d14=0x0(0.0) d15=0x0(0.0)
=> *[libmain.so        *0x6afe3]*[9d54    ]*0x402aafe2:*"strb r5, [r3, r2]" [0xbffff028] => 0x0
    [libmain.so         0x6afe5] [4ff00005] 0x402aafe4: "mov.w r5, #0"
    [libmain.so         0x6afe9] [02f10102] 0x402aafe8: "add.w r2, r2, #1"
    [libmain.so         0x6afed] [daf80830] 0x402aafec: "ldr.w r3, [sl, #8]"
    [libmain.so         0x6aff1] [08bf    ] 0x402aaff0: "it eq"
    [libmain.so         0x6aff3] [0125    ] 0x402aaff2: "moveq r5, #1"
    [libmain.so         0x6aff5] [572b    ] 0x402aaff4: "cmp r3, #0x57"
    [libmain.so         0x6aff7] [18bf    ] 0x402aaff6: "it ne"
    [libmain.so         0x6aff9] [ae46    ] 0x402aaff8: "movne lr, r5"
    [libmain.so         0x6affb] [1ef0010f] 0x402aaffa: "tst.w lr, #1"
    [libmain.so         0x6afff] [c8d0    ] 0x402aaffe: "beq #0x402aaf92"
    [libmain.so         0x6b001] [d6f87033] 0x402ab000: "ldr.w r3, [r6, #0x370]"
    [libmain.so         0x6b005] [c6f888e3] 0x402ab004: "str.w lr, [r6, #0x388]"
    [libmain.so         0x6b009] [581c    ] 0x402ab008: "adds r0, r3, #1"
    [libmain.so         0x6b00b] [0428    ] 0x402ab00a: "cmp r0, #4"
    [libmain.so         0x6b00d] [7ff48faf] 0x402ab00c: "bne.w #0x402aaf2e"
```

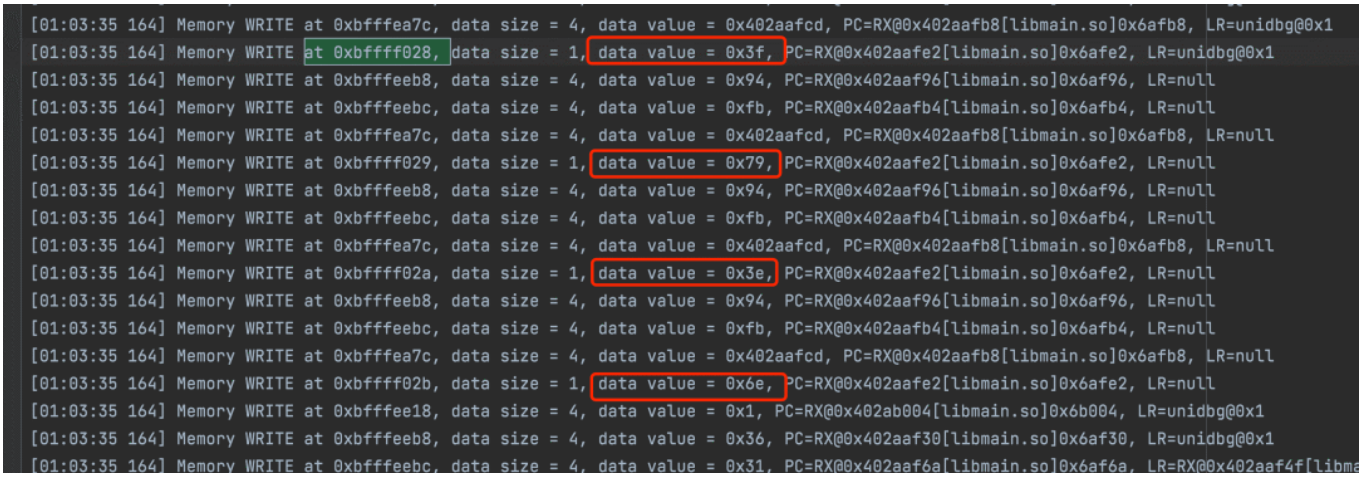
仔细看看就会发现，数据取出来 给到 `r5` 啥都没干，在存起来，有点离谱， `debugger` 看一下

```
LOAD:0006AFC4 B1 27                    MOVS     R7, #0xB1
LOAD:0006AFC6 C6 D7                    BVC      loc_6AF56
LOAD:0006AFC8 03 C8                    LDMIA    R0, {R0,R1}
LOAD:0006AFCA BE 9D                    LDR      R5, [SP,#0x2F8]
LOAD:0006AFCC 06 F2 24 4A              ADDW     R10, R6, #0x424
LOAD:0006AFD0 11 F8 22 50              LDRB.W   R5, [R1,R2,LSL#2]
LOAD:0006AFD4 03 2A                    CMP      R2, #3
LOAD:0006AFD6 DA F8 08 30              LDR.W    R3, [R10,#8]
LOAD:0006AFDA 83 F0 FB 03              EOR.W    R3, R3, #0xFB
LOAD:0006AFDE 00 EB 03 13              ADD.W    R3, R0, R3,LSL#4
LOAD:0006AFE2 9D 54                    STRB     R5, [R3,R2]
LOAD:0006AFE4 4F F0 00 05              MOV.W    R5, #0
LOAD:0006AFE8 02 F1 01 02              ADD.W    R2, R2, #1
LOAD:0006AFEC DA F8 08 30              LDR.W    R3, [R10,#8]
LOAD:0006AFF0 08 BF                    IT EQ
```

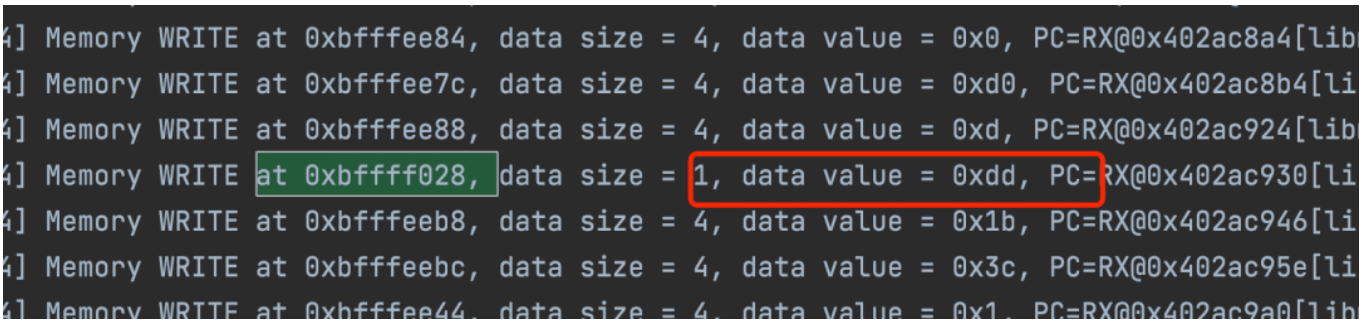
写到这个栈里了，目前不知道干啥在 `trace` 一波

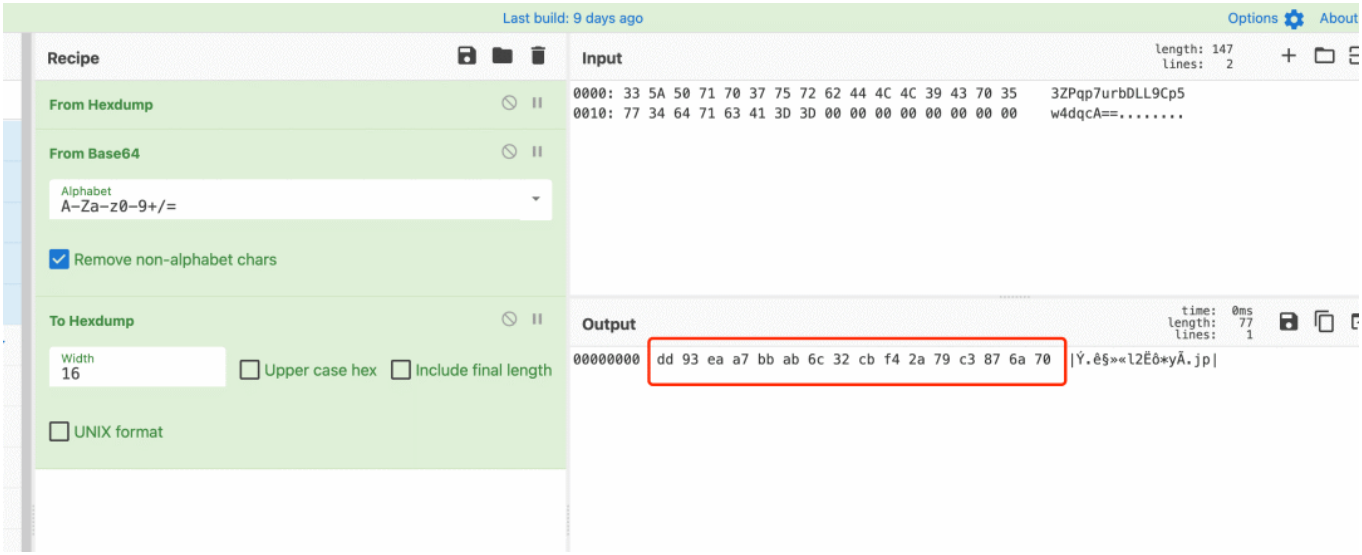
在控制台输入 `traceWrite [0xbffff028 0xbffff038]` 这个会把结果写到文件里

打开文件分析一波



搜索 `0xbffff028` 可以看到有写入 `data ^ iv` 的结果





再往下跟踪看到了写入 `aes` 的加密结果，所以猜测这个地址就是 `state`

最后

好了这一篇就到了这里，后面就是 `dfa` 注入攻击了

偷懒了（主要还是因为太多了，有点写不下去了）