

qinless (<https://www.qinless.com/>)

sgmain 6.4.x xminiwua 加密算法分析研究

android 逆向 (<https://www.qinless.com/category/android-reverse>) · 2022-04-15 · 2764 次浏览

文章目录[隐藏]

- android 逆向 36 / 37
- 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
- 0x1 读取文件
- 0x2 AES 解密 SGSAFETOKEN_IN
- 0x6 base64 encode && aes encrypt
- 0x5 x2 = xor(x1)
- 0x4 aes decrypt sdfsd

android 逆向 (<https://www.qinless.com/series/androidnixiang>) 36 / 37

app OAuth api_sign 分析 (<https://www.qinless.com/139>)

app sig 参数加密分析 unidbg 模拟黑盒调用 (<https://www.qinless.com/141>)

app sign so 加密参数破解 | unidbg (<https://www.qinless.com/137>)

sgmain x-sign 分析 - unidbg (<https://www.qinless.com/179>)

androidAsync fridaManager sgmain 70102 rpc 远程调用 (<https://www.qinless.com/134>)

app edata 参数 so aes 加密分析破解 | unidbg (<https://www.qinless.com/341>)

frida 加载 sekiro dex 文件 实现与服务端交互 (<https://www.qinless.com/387>)

frida sekiro 实现 sgmain 70102 远程 rpc 调用 (<https://www.qinless.com/400>)

xposed sekiro hook 获取 wx 万能 key (<https://www.qinless.com/420>)

unidbg console debugger 使用 (<https://www.qinless.com/470>)

unidbg hook inline hook 使用 (<https://www.qinless.com/472>)

app 公众号文章列表 so 加解密算法分析还原 | 简单分析 (<https://www.qinless.com/483>)

app 公众号文章列表 so 加解密算法分析还原 | 加密 rsa base64 分析
(<https://www.qinless.com/485>)

app 公众号文章列表 so 加解密算法分析还原 | 加密 zip aes 分析
(<https://www.qinless.com/488>)

app 公众号文章列表 so 加解密算法分析还原 | response 内容解密分析
(<https://www.qinless.com/554>)

app sign so 加密算法分析还原 | 简单分析 (<https://www.qinless.com/576>)

app sign so 加密算法分析还原 | so 算法分析 (<https://www.qinless.com/584>)

app sign so 加密算法分析还原 | so sub_126AC 函数算法还原
(<https://www.qinless.com/588>)

app so signkeyV1 参数分析 (<https://www.qinless.com/752>)

ida 动态调试 android so 文件 | 基础入门环境搭建 (<https://www.qinless.com/729>)

app so newSign 参数分析破解 (<https://www.qinless.com/745>)

app tzRgz52a 参数分析破解 (<https://www.qinless.com/914>)

app sign-v2 签名算法 aes 加解密分析 (<https://www.qinless.com/922>)

app so 加密参数分析 | protocbuf 分析 (<https://www.qinless.com/942>)

mxtakatak android app 加解密分析 (<https://www.qinless.com/1138>)

android app so 加密算法分析破解 | mtgsig unidbg (<https://www.qinless.com/1033>)

android app so 加密算法分析破解 | siua unidbg (<https://www.qinless.com/1038>)

android app nsign so 加密算法分析 (<https://www.qinless.com/1281>)

android app sig 参数 so 加密逻辑逆向分析 (<https://www.qinless.com/1429>)

android app so sig 加密参数 unidbg (<https://www.qinless.com/1432>)

狗狗音乐登陆协议加密参数逆向分析 (<https://www.qinless.com/1436>)

android sign so 加密参数分析 | unidbg (<https://www.qinless.com/1442>)

android app X-SS-QUERIES 参数分析 (<https://www.qinless.com/1445>)

unidbg android app xgorgon 加密参数 leviathan (<https://www.qinless.com/1631>)

sgmain 6.4.x xsign 加密算法分析研究 (<https://www.qinless.com/1708>)

sgmain 6.4.x xminiwua 加密算法分析研究

某 app mas 算法分析还原 cms so (<https://www.qinless.com/1879>)

« 上一篇文章 (<https://www.qinless.com/1708>)

下一篇文章 (<https://www.qinless.com/1879>) »

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-mini-wua 参数加密算法研究分析
样本 https://www.wandoujia.com/apps/38221/history_v1843

本文主要使用 ida + unidbg + frida 动静态分析

样本 unidbg 参考文章

- sgmain x-sign 分析 - unidbg (<https://www.qinless.com/179>)

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

```
1、SGSAFETOKEN_IN = READFILE("file/app_SGLib/SG_INNER_DATA")
2、json = AES_DECRYPT(SGSAFETOKEN_IN)
3、sdfsd = BASE64_DECODE(json.getString("sdfsd"))
4、x1 = AESDECRYPT(sdfsd)
5、x2 = XOR(x1)
6、result = "HHnB" + BASE64_ENCODE(AES_ENCRYPT(x2))
```

本文基于上面的流程，简单说一下，不会扩展到其他细节上

Tips: 因为 unidbg 跑出来的 x-mini-wua 是短的，所以只能辅助分析大概的算法流程，具体细节博主是使用 frida hook 的方式来验证的

0x1 读取文件

SG_INNER_DATA 文件在手机目录下就能找到。读取出来获取 SGSAFETOKEN_IN 字段

0x2 AES 解密 SGSAFETOKEN_IN

这个如何定位有两种方法（假设你已经分析过 xsign 参数了，那对 sgmain 的 aes 会有所了解）

- 1、直接去 hook aes decrypt 函数
- 2、通过 traceRead 看看 SGSAFETOKEN_IN 在哪里读的

```

74      if ( v5[3] < 0x100 || v19 & 0xf )
75          goto LABEL_41;
76      if ( v5[2] )
77          _aeabi_memcpy();
78      _aeabi_memclr8();
79      sub_9EF50(&v25, v3, v4);           // 密钥编排 0x999B4
80      v1 = 0;
81      v24 = 0;
82      v23 = 0;
83      v22 = 0;
84      v21 = 0;
85      if ( v19 >> 4 )
86      {
87          v15 = v8;
88          v4 = &v21;
89          v3 = 0;
90          v17 = v10;
91          do
92          {
93              _aeabi_memcpy();
94              sub_A00B4(&v25, v15, v2);   // cbc 0x999FA
95              v16 = 0;
96              do
97              {
98                  v15[v16] ^= v26[v16];
99                  ++v16;
100             }
101             while ( v16 != 16 );
102             ++v3;
103             v15 += 16;
104             *v26 = v21;
105             v27 = v22;
106             v28 = v23;
107             v29 = v24;
108             v2 += 4;

```

(<https://www.qinless.com/wp-content/uploads/2022/04/62599031f2efc.png>)

博主这里就直接说答案了，在这里就是 aes cbc decrypt 逻辑，通过 inlinehook 可以直接获取到 data key iv result 等数据

```

[13:36:50 201] aes decrypt res , md5=64481e2ca9a2ed39dbfb26b7113b7578, hex=7b2
size: 640
0000: 7B 22 36 63 37 30 39 63 31 31 64 32 64 34 36 61 {"6c709c11d2d46a
0010: 37 62 22 3A 22 30 30 38 26 64 62 30 62 61 33 63 7b":"008&db0ba3c
0020: 32 6A 78 48 53 6E 54 72 54 44 4C 47 46 30 4E 51 2jxHSnTrTDLGF0NQ
0030: 63 42 56 38 30 53 75 49 4C 34 61 50 67 36 53 33 cBV80SuIL4aPg6S3
0040: 54 2B 42 47 33 67 56 76 50 65 4B 77 45 62 6E 67 T+BG3gVvPeKwEbng
0050: 42 5A 49 75 68 6D 64 34 4A 55 76 65 37 55 71 42 BZIuhmd4JUve7UqB
0060: 63 2F 43 49 66 45 77 69 4C 58 48 33 57 56 71 4D c/CIfEwiLXH3WVqM
0070: 57 50 57 54 64 77 66 43 64 6F 30 6A 32 75 46 58 WPWTdwfCdo0j2uFX
0080: 47 57 46 74 54 70 44 73 4E 68 37 36 43 2F 66 38 GWFTtpDsNh76C/f8
0090: 36 35 47 66 31 44 77 59 34 59 69 52 78 62 38 7A 65Gf1DwY4YiRxb8z
00A0: 42 22 2C 22 30 33 35 37 39 39 35 65 39 38 35 38 B","0357995e9858
00B0: 64 38 62 30 22 3A 22 30 30 38 26 64 62 30 62 61 d8b0":"008&db0ba
00C0: 33 63 32 33 66 58 79 39 4A 49 4C 44 6A 61 6E 34 3c23fXy9JILDjan4
00D0: 61 6F 52 41 72 48 6E 4C 38 55 70 5A 74 50 74 76 aoRArHnL8UpZtPtv
00E0: 4B 77 30 78 4C 6D 67 38 34 5A 38 32 30 50 50 4C Kw0xLmg84Z820PPI
00F0: 69 2B 64 6B 35 55 35 65 78 67 53 68 6C 79 2B 71 i+uL5U0exj8N1y+q
0100: 50 71 38 22 2C 22 38 64 64 32 33 36 65 33 31 32 Pq8","8dd236e312

```

(<https://www.qinless.com/wp-content/uploads/2022/04/6259903976e3c.png>)

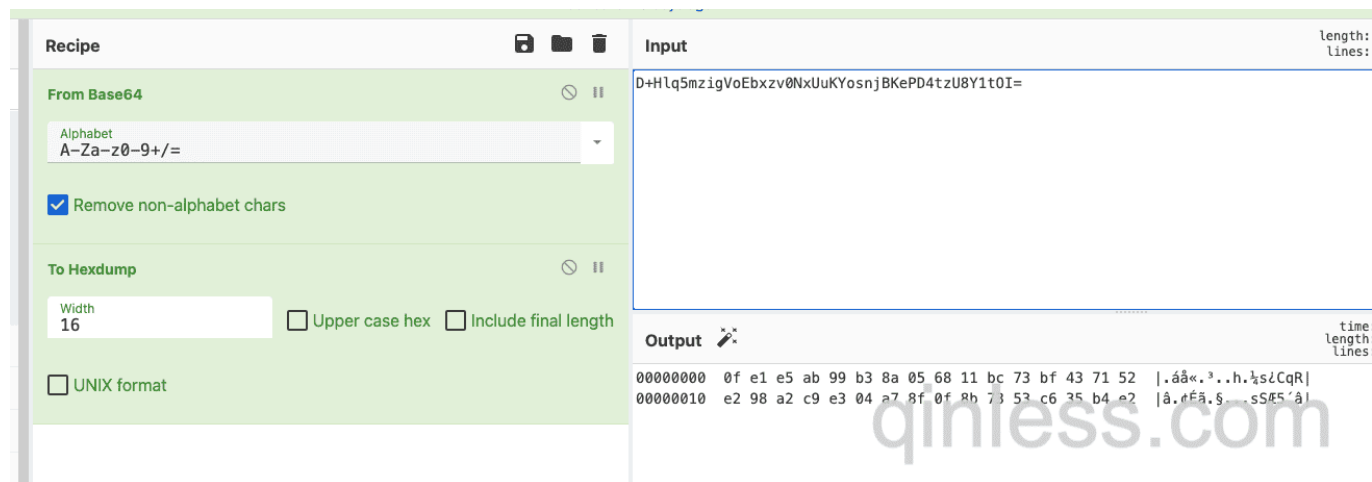
分析到这里，就卡住了，因为 `SGSAFETOKEN_IN` 解密出来的结果是个 `json` 有四个字段，但是没有 `sdfsd`，可能是这个版本名称不一样

然后经过 `n` 多次的 `trace` 也还是没有找到想要的答案，这里各位读者大佬可以自己尝试呀，过程很心酸

最后还是决定从后往前追（当然也是遇到了 `n` 多个问题，有些问题卡了很久，这个后面在说）

0x6 base64 encode & aes encrypt

最后的结果是 "HHnB" + BASE64_ENCODE(AES_ENCRYPT(x2))



(<https://www.qinless.com/wp-content/uploads/2022/04/62599028c6142.png>)

这里直接进行 base decode 结果可以正常出来，然后就可以进行 unidbg trace 了

- 1、在最后的结果下断点，使用 shr 搜索堆
- 2、搜索到之后就可以使用 traceWrite 了
- 3、不出意外的情况下，trace 到的地址就是 aes encrypt
- 4、然后根据代码接口，就可以判断出这是个 aes cbc 模式的加密，因为 密钥编排、iv ^ data 的特征都很明显，还有 pkcs7 的填充标志

以上都是博主亲测出来的流程，只不过以文字的方式描述，而没有加上图片，各位读者大佬如果没有思路可根据以上的思路尝试一下

$$0x5 \times 2 = \text{xor}(x1)$$

这一步最复杂，为何复杂，因为博主的 unidbg 没有跑出全的 x-mini-wua 所以使用 unidbg 不好去分析，这一步也是卡了很久

```

>-----<
[13:54:56 132] aes encrypt data , md5=319019a5e82ac4a5865f1727c125b2a4, hex=000000036
size: 20
0000: 00 00 00 03 6B 1D 00 24 02 EB 1D 62 1D 7A 1C 0A ....k..$.b.z..
0010: 35 E4 1E 6B
^-----^

```

(<https://www.qinless.com/wp-content/uploads/2022/04/625990497d60f.png>)

这里就是最后 aes encrypt 加密的数据

```

[16:14:52 485] Memory WRITE at 0xbffff214, data size = 4, data value = 0x40aa48d7, PC=RX@0x400d56dc[libc.so]0x176dc, LR=unidbg@0x1d
[16:14:52 485] Memory WRITE at 0x401cd660, data size = 1, data value = 0x6b, PC=RX@0x400d56dc[libc.so]0x176dc, LR=unidbg@0x1d
[16:14:52 485] Memory WRITE at 0x401cd661, data size = 1, data value = 0x1d, PC=RX@0x400d56e0[libc.so]0x176e0, LR=unidbg@0x1d
[16:14:52 485] Memory WRITE at 0x401d87e4, data size = 4, data value = 0x2, PC=RX@0x40aa48da[libsecuritybody.so]0x248da, LR=unidbg@0x1d
[16:14:52 486] Memory WRITE at 0xbffff220, data size = 4, data value = 0x0, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af49
[16:14:52 486] Memory WRITE at 0xbffff224, data size = 4, data value = 0x4, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af49
[16:14:52 486] Memory WRITE at 0xbffff228, data size = 4, data value = 0xbffff438, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af49
[16:14:52 486] Memory WRITE at 0xbffff22c, data size = 4, data value = 0x40a8af49, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af49
[16:14:52 486] Memory WRITE at 0x401cd662, data size = 1, data value = 0x0, PC=RX@0x40aa4a46[libsecuritybody.so]0x24a46, LR=RX@0x40a8af49
[16:14:52 486] Memory WRITE at 0x401d87e4, data size = 4, data value = 0x3, PC=RX@0x40aa4a4c[libsecuritybody.so]0x24a4c, LR=RX@0x40a8af49
[16:14:52 490] Memory WRITE at 0xbffff220, data size = 4, data value = 0x0, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af5c
[16:14:52 490] Memory WRITE at 0xbffff224, data size = 4, data value = 0x4, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af5c
[16:14:52 490] Memory WRITE at 0xbffff228, data size = 4, data value = 0xbffff438, PC=RX@0x40aa4a22[libsecuritybody.so]0x24a22, LR=RX@0x40a8af5c
[16:14:52 490] Memory WRITE at 0xbffff22c, data size = 4, data value = 0x40a8af5d, PC=RX@0x40aa4a46[libsecuritybody.so]0x24a46, LR=RX@0x40a8af5c
[16:14:52 491] Memory WRITE at 0x401cd663, data size = 1, data value = 0x24, PC=RX@0x40aa4a4c[libsecuritybody.so]0x24a4c, LR=RX@0x40a8af5c
[16:14:52 491] Memory WRITE at 0x401d87e4, data size = 4, data value = 0x4, PC=RX@0x40aa4a4c[libsecuritybody.so]0x24a4c, LR=RX@0x40a8af5c

```

(<https://www.qinless.com/wp-content/uploads/2022/04/6259904e62a3a.png>)

```
[16:14:52 507] Memory WRITE at 0x401cd665, data size = 1, data value = 0xeb, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 508] Memory WRITE at 0x401cd666, data size = 1, data value = 0x1d, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 509] Memory WRITE at 0x401cd667, data size = 1, data value = 0x62, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 509] Memory WRITE at 0x401cd668, data size = 1, data value = 0x1d, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 509] Memory WRITE at 0x401cd669, data size = 1, data value = 0x7a, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 509] Memory WRITE at 0x401cd66a, data size = 1, data value = 0x1c, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 509] Memory WRITE at 0x401cd66b, data size = 1, data value = 0xa, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 510] Memory WRITE at 0x401cd66c, data size = 1, data value = 0x35, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 510] Memory WRITE at 0x401cd66d, data size = 1, data value = 0xe4, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 510] Memory WRITE at 0x401cd66e, data size = 1, data value = 0xe, PC=RX@0x40a8b1a6[libsecuritybo  
[16:14:52 510] Memory WRITE at 0x401cd66f, data size = 1, data value = 0x6b, PC=RX@0x40a8b1a6[libsecuritybo
```

(<https://www.qinless.com/wp-content/uploads/2022/04/6259905339076.png>)

博主在进行 trace 的过程, 发现这些字节是在 n 多处 write 的

而且 so 还看不到 c 代码, 很难根据 arm 分析出数据的计算逻辑, 这一步也是卡了很久, 具体分析的过程细节就不说了, 简单说下思路

- 1、使用 unidbg 进行 hook trace , 根据结果, 来猜测尝试
- 2、使用 frida hook 完成验证, hook 出一些关键数据, 进行计算

0x4 aes decrypt sdfsd

这一步就比较简单了, 没啥可说的

```
"pageSize": "20",  
"totalNum": "672",  
"totalPage": "34"
```

```
    },  
    "pageSize": "20",  
    "rn": "00d7e9213c2acefeb0191014fa128e7e",  
    "selectedFilterModule": {  
        "filterList": [],  
        "moduleName": "default"  
    },  
    "shopId": "63612646",  
    "shopTitle": "百草味旗舰店",  
    "success": "true",  
    "totalPage": "34",  
    "totalResults": "672"  
},  
"ret": [  
    "SUCCESS::调用成功"  
],  
"v": "1.0"  
}  
  
qinless.com  
  
Process finished with exit code 0
```

(<https://www.qinless.com/wp-content/uploads/2022/04/6259905b85b9e.png>)

最后放上一张测试请求成功的图片

标签: #aes, #dfa, #frida, #hmac, #native, #sgmain, #sha1, #so, #unidbg, #x-mini-wua, #x-sign, #算法分析, #算法还原 (<https://www.qinless.com/tag/suanfahuanyuan>)

 (<https://www.qinless.com/1724>)

 ([https://www.qinless.com/wp-content/themes/honey/public/qrcode?](https://www.qinless.com/wp-content/themes/honey/public/qrcode?data=https://www.qinless.com/1724)

[data=https://www.qinless.com/1724](https://www.qinless.com/1724))


 (<https://www.qinless.com/1724>)

 暂无评论 (<https://www.qinless.com/1724#respond>)

sgmain 6.4.x xsign 加密算法分析
研究

郑重声明

下一篇 

上一篇 

(<https://www.qinless.com/1708>)

(<https://www.qinless.com/1785>)



会爬山的小脑虎 (<https://www.qinless.com/author/xiayu>)

小爬爬

本文作者: 会爬山的小脑虎 (<https://www.qinless.com/author/xiayu>).

本文链接: <https://www.qinless.com/?p=1724> (<https://www.qinless.com/1724>).

版权声明: 本博客所有文章除特别声明外, 均采用 [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/)