[misc]智能车协议分析：2020网鼎杯青龙组misc_teslaaaaa_wp

原创

breezeO_o 于 2020-05-16 11:47:31 发布

分类专栏： ctf # ctf-misc # 逆向 文章标签： ctf 智能汽车 UDS协议 misc 协议分析

本文链接：https://blog.csdn.net/Breeze_CAT/article/details/106156567

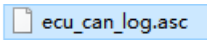
[misc]智能车协议分析：2020网鼎杯青龙组misc_teslaaaaa_wp

文章目录

[misc]智能车协议分析：2020网鼎杯青龙组misc_teslaaaaa_wp

teslaaaaa这道题是青龙组被做出来的题目中做出来人数最少的，直到比赛快结束才有第一个人做出来，怎么说，难并不是因为脑洞大，讲道理这道题是一道好题，我当时也是肝了一天没做出来，我倒是把文件提取出来了，奈何比赛时间已经接近尾声，而且脑子也是一团浆糊，没有想到是刷的固件，还以为是配置文件等等。怎么说，还是太菜了。昨天看到 看雪论坛的HHHso大佬发布的wp，于是恍然大悟，自己复现一番。

拿到题目，解压出来是一个.asc文件：



并没接触过这种类型，搜索了一番也没搜到什么，是一个文本文件，打开之后可以看到类似数据包的抓包日志：

```
date Thu Apr 2 10:37:15.950 am 2020
base hex  timestamps absolute
internal events logged
// version 8.2.1
   0.005921 CAN 1 Status:chip status error active
   1.005921 CAN 1 Status:chip status error active
   2.005922 CAN 1 Status:chip status error active
   3.005922 CAN 1 Status:chip status error active
   4.000621 1  7DF             Tx   d 8 02 3E 80 00 00 00 00 00  Length = 0 BitCount = 124 ID = 2015  // 1  OTP(01) At
   4.005922 CAN 1 Status:chip status error active
   5.005922 CAN 1 Status:chip status error active
   6.005923 CAN 1 Status:chip status error active
   7.005923 CAN 1 Status:chip status error active
   8.000537 1  7DF             Tx   d 8 02 3E 80 00 00 00 00 00  Length = 0 BitCount = 124 ID = 2015  // 1  OTP(02) At
   8.005923 CAN 1 Status:chip status error active
   9.005924 CAN 1 Status:chip status error active
   9.498709 1  7DF             Tx   d 8 02 10 02 AA AA AA AA AA  Length = 0 BitCount = 116 ID = 2015  // 1  OTP(03) At
   9.499693 1  7B0             Rx   d 8 06 50 02 00 32 01 F4 00  Length = 235910 BitCount = 122 ID = 1968
   9.740585 1  730             Tx   d 8 02 27 05 AA AA AA AA AA  Length = 222015 BitCount = 114 ID = 1840
   9.741697 1  7B0             Rx   d 8 06 67 05 11 22 33 44 00  Length = 223910 BitCount = 116 ID = 1968
   9.782739 1  730             Tx   d 8 06 27 06 EE DD CC BB AA  Length = 226244 BitCount = 116 ID = 1840
   9.783703 1  7B0             Rx   d 8 02 67 06 00 00 00 00 00  Length = 235910 BitCount = 122 ID = 1968
   9.788131 1  730             Tx   d 8 10 0D 31 01 FF 00 44 08  Length = 232000 BitCount = 119 ID = 1840
   9.788431 1  7B0             Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
   9.788947 1  730             Tx   d 8 21 00 00 00 00 00 20 00  Length = 244244 BitCount = 125 ID = 1840
   9.789707 1  7B0             Rx   d 8 05 71 01 FF 00 00 00 00  Length = 233910 BitCount = 121 ID = 1968
   9.791765 1  730             Tx   d 8 10 0B 34 00 44 08 00 00  Length = 236244 BitCount = 121 ID = 1840
   9.792061 1  7B0             Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
   9.792625 1  730             Tx   d 8 21 00 00 00 20 00 AA AA  Length = 234244 BitCount = 120 ID = 1840
   9.793715 1  7B0             Rx   d 8 04 74 20 01 02 00 00 00  Length = 233910 BitCount = 121 ID = 1968
```

结合文件名ecu_can_log，通过搜索，可以确定这是个智能汽车can总线的抓包log。

首先了解一下can总线，can总线控制器局域网络(Controller Area Network, CAN)的简称，也是智能汽车里面的总线，智能汽车中各个部件、传感器都会通过can总线链接到车辆的主控板（ECU）上组成一个局域网（对智能汽车不是很了解，简单描述一下），我们只需要知道这个抓包log是来自智能汽车就可以了，那么也要明白，这里面的协议也是智能汽车中独有的协议。

分析log文件格式

通过搜索，我们得到了这个log文件里面的数据格式：

CAN数据格式-ASC

从左到右依次是：时间戳、CAN通道编号、帧ID（16进制）、帧方向（发送或接收）、d。之后跟的DLC、数据。我们只需要知道**数据流向和数据内容**即可：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-AEKbWdUo-1589599664736)(teslaaaa_wp.assets/image-20200515162832992.png)]

分析协议组成

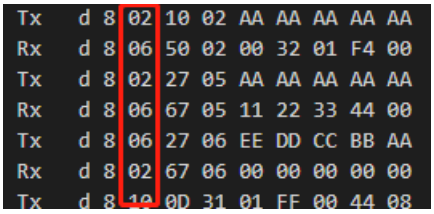
接下来我们要分析数据报文究竟是什么协议，根据我们搜索的内容和最后官方给的提示：**UDS诊断协议（ISO 15765-2， ISO 14229-1）**，我们确定了协议内容为**UDS网络传输层和UDS统一诊断服务**（虽说官方提示前我就知道了是什么协议，但并不影响我做不出来）。

这两个协议的关系就类似传输层和应用层的关系，UDS网络传输层协议在UDS统一诊断服务的外层，我们可以像分析网络协议栈数据包一样分析这个log文件。

UDS网络传输层

关于UDS网络传输层：ISO15765-2(UDS网络传输层)

关于UDS网络传输层不用知道太多，只需知道这个协议有两种模式，分别为单帧传输和多帧传输，协议内容很短，只需要看每条数据的第一个字节：



第一个字节的含义：

byte1定义	7-4bit		3-0bit	
	0	单帧SF	SF_DL	数据长度
	1	首帧FF	FF_DL	数据长度
	2	连续帧C	SN	连续帧编号
	3	流控FC	FS	0 继续发送
				1 等待
				2 溢出

如`02 10 02 AA AA AA AA AA`这条数据就代表单帧(0)，数据长度2(2)，然后`10 02`就是协议的数据，后面的AA 都么用。

如

```
Tx   d 8 10 0D 31 01 FF 00 44 08
Rx   d 8 30 08 00 00 00 00 00 00
Tx   d 8 21 00 00 00 00 00 20 00
```

代表**发送方**要发送连续帧(1)，数据长度13(0 0D)，然后后面的6个是数据

然后**接收方**返回控制FC(3)，含义是继续发送(0)，后面08是啥不用管，因为我看这个数据包里后面所有的继续发送都是30 08 00 00…

然后**发送方**继续发送连续帧(2)的第二帧(1)，后面是接首帧的内容。

这两组例子差不多能概括这个数据包里面所有的UDS网络传输层报文了，这个数据包里基本只有单帧和连续帧。

UDS统一诊断服务

接下来分析UDS统一诊断服务

关于UDS统一诊断服务也不用知道太多，我们只需要知道这个协议是测试人员向ECU（车辆主控板）上发送的协议，协议结构也非常简单：

发送方（测试人员）发送：SID+具体数据

接收方（ECU）回复：肯定：（SID+0x40这里代表数学加法）+具体数据

否定：7F+SID+一字节NRC

协议也是很简单，以下面的内容为例：

```
Tx   d 8 02 10 02 AA AA AA AA AA
Rx   d 8 06 50 02 00 32 01 F4 00
```

发送方发送的内容为（去掉外层UDS网络传输层）：`10 02`，10代表协议SID为10，对应的是诊断会话服务，可以看出这里是诊断的开始，发送方发起诊断请求，02是对应的协议数据，可能是一些类别啥的，没啥用，我们是要解题而不是完全掌握这个协议

然后返回方的内容为（去掉外层UDS网络传输层）：`50 02 00 32 01 F4`，这里50代表的就是对之前发送方发起的10诊断服务的肯定（SID+0x40），后面的是数据吧，这里不用管是啥，只需知道**请求放发起了诊断服务，ECU给予了确定的回复。**这时单帧的情况，连续帧同理，比如：

```
Tx   d 8 10 0D 31 01 FF 00 44 08
Rx   d 8 30 08 00 00 00 00 00 00
Tx   d 8 21 00 00 00 00 00 20 00
Rx   d 8 05 71 01 FF 00 00 00 00
```

发送方发送请求内容（去掉外层UDS网络传输层连续帧）：`31 01 FF 00 44 08 00 00 00 00 00 20 00`，请求的服务是0x31例行程序控制，协议数据01位启动程序，FF 00为擦除数据（一般用于请求APP数据下载（34服务）之前，这也和后文34服务照应上了），擦除地址为0x8000000 。

接收方返回内容为（去掉外层UDS网络传输层连续帧）：`71 01 FF 00 00`71代表肯定回复（0x31+0x40），然后01和之前请求保持相同，FF 00 00代表擦除成功？？不太确定猜的。

附：在知乎上找到一个大佬，较为详细的讲解了大部分的服务：

https://www.zhihu.com/people/zoe-white/posts

或者百度“UDS诊断之XX服务”

协议分析

```
Tx   d 8 02 10 02 AA AA AA AA AA  10服务 诊断会话控制
Rx   d 8 06 50 02 00 32 01 F4 00  请求成功
Tx   d 8 02 27 05 AA AA AA AA AA  27服务 安全访问-5
Rx   d 8 06 67 05 11 22 33 44 00  请求成功，返回秘钥
Tx   d 8 06 27 06 EE DD CC BB AA  27服务 安全访问-6，计算秘钥返回结果
Rx   d 8 02 67 06 00 00 00 00 00  结果校验无误，通过安全校验
Tx   d 8 10 0D 31 01 FF 00 44 08  31服务 例行程序控制，擦除数据0x8000000
Rx   d 8 30 08 00 00 00 00 00 00  继续发送
Tx   d 8 21 00 00 00 00 00 20 00  31服务 例行程序控制接上文，擦除数据0x8000000
Rx   d 8 05 71 01 FF 00 00 00 00  擦除成功
Tx   d 8 10 0B 34 00 44 08 00 00  34服务 请求下载，下载到0x8000000
Rx   d 8 30 08 00 00 00 00 00 00  继续发送
Tx   d 8 21 00 00 00 20 00 AA AA  34服务 请求下载接上文，下载到0x8000000
Rx   d 8 04 74 20 01 02 00 00 00  成功，接受下载请求
Tx   d 8 10 82 36 01 28 04 00 20  36服务 开启数据传输，长度0x82，第一次传输
Rx   d 8 30 08 00 00 00 00 00 00  继续发送
Tx   d 8 21 45 01 00 08 21 03 00  接下来是持续发送数据直到发送结束（编号21-2f之后从21
Tx   d 8 22 08 23 03 00 08 27 03                                    重新开始）
Tx   d 8 23 00 08 2B 03 00 08 2F
        ...    ...    ...    ...
Tx   d 8 22 08 5F 01 00 08 AA AA  第一次数据传输结束
Rx   d 8 03 7F 36 78 00 00 00 00  返回阻塞？？这里不用管，因为下一个数据包就是成功返回
Rx   d 8 02 76 01 00 00 00 00 00  第一次数据传输成功
Tx   d 8 10 82 36 02 5F 01 00 08  第二次数据传输开始，长度还是0x82,下面相同
Rx   d 8 30 08 00 00 00 00 00 00  继续发送
Tx   d 8 21 5F 01 00 08 5F 01 00
        ...  一共传输了0x40次  ...
Rx   d 8 02 76 40 00 00 00 00 00  最后一次传输成功
Tx   d 8 02 37 01 AA AA AA AA AA  37服务 传输结束
Rx   d 8 06 77 01 C6 B6 5E 10 00  传输结束成功
Tx   d 8 04 31 01 DF FF AA AA AA  31服务 例行程序控制，一些操作
Rx   d 8 03 7F 31 78 00 00 00 00  阻塞（不用管，下面就成功了）
Rx   d 8 05 71 01 DF FF 00 00 00  成功
Tx   d 8 04 31 01 FF 01 AA AA AA  31服务 例行程序控制，执行刚刚传输的东西
Rx   d 8 05 71 01 FF 01 00 00 00  成功
Tx   d 8 02 11 01 AA AA AA AA AA  11服务 ECU复位，重启ECU
Rx   d 8 03 7F 11 78 00 00 00 00  阻塞（不用管，下面就成功了）
Rx   d 8 02 51 01 00 00 00 00 00  成功
Tx   d 8 02 3E 80 00 00 00 00 00  3E服务 回话保持心跳包
Tx   d 8 02 3E 80 00 00 00 00 00  3E服务 回话保持心跳包
```

这时赛后进行的完整分析，在比赛的时候我没对每个服务的内容进行详细分析，导致不知道传输的数据后来被执行了。这个31服务的操作数特斯拉厂商自己规定的。

所以我们可以看出整个数据报文就是，先进行例行的身份认证，然后擦出了单板上0x8000000内存区域的数据，然后向这块区域下载了一个文件并执行。

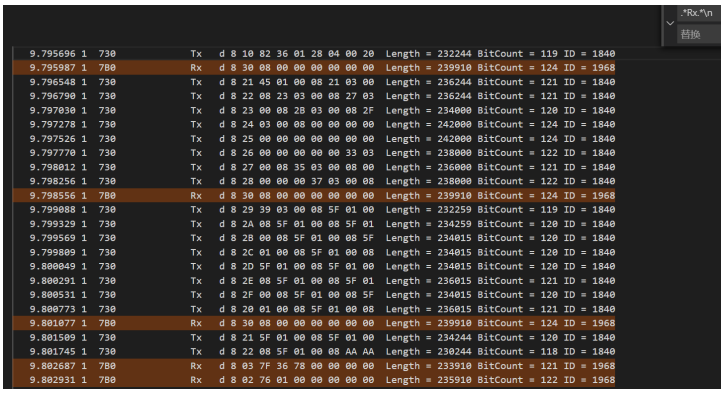
文件提取

那么根据我们之前分析的内容，我们把下载相关的报文提取出来。

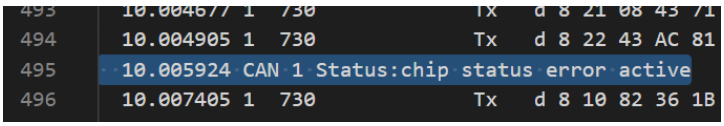
```
Tx   d 8 10 82 36 01 28 04 00 20  数据传输Length = 232244 BitCount = 119 ID = 1840
Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
Tx   d 8 21 45 01 00 08 21 03 00  Length = 236244 BitCount = 121 ID = 1840
Tx   d 8 22 08 23 03 00 08 27 03  Length = 236244 BitCount = 121 ID = 1840
Tx   d 8 23 00 08 2B 03 00 08 2F  Length = 234000 BitCount = 120 ID = 1840
Tx   d 8 24 03 00 08 00 00 00 00  Length = 242000 BitCount = 124 ID = 1840
Tx   d 8 25 00 00 00 00 00 00 00  Length = 242000 BitCount = 124 ID = 1840
Tx   d 8 26 00 00 00 00 00 33 03  Length = 238000 BitCount = 122 ID = 1840
Tx   d 8 27 00 08 35 03 00 08 00  Length = 236000 BitCount = 121 ID = 1840
Tx   d 8 28 00 00 00 37 03 00 08  Length = 238000 BitCount = 122 ID = 1840
Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
Tx   d 8 29 39 03 00 08 5F 01 00  Length = 232259 BitCount = 119 ID = 1840
Tx   d 8 2A 08 5F 01 00 08 5F 01  Length = 234259 BitCount = 120 ID = 1840
Tx   d 8 2B 00 08 5F 01 00 08 5F  Length = 234015 BitCount = 120 ID = 1840
Tx   d 8 2C 01 00 08 5F 01 00 08  Length = 234015 BitCount = 120 ID = 1840
Tx   d 8 2D 5F 01 00 08 5F 01 00  Length = 234015 BitCount = 120 ID = 1840
Tx   d 8 2E 08 5F 01 00 08 5F 01  Length = 236015 BitCount = 121 ID = 1840
Tx   d 8 2F 00 08 5F 01 00 08 5F  Length = 234015 BitCount = 120 ID = 1840
Tx   d 8 20 01 00 08 5F 01 00 08  Length = 236015 BitCount = 121 ID = 1840
Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
Tx   d 8 21 5F 01 00 08 5F 01 00  Length = 234244 BitCount = 120 ID = 1840
Tx   d 8 22 08 5F 01 00 08 AA AA  第一次数据传输结束Length = 230244 BitCount = 118 ID
Rx   d 8 03 7F 36 78 00 00 00 00  Length = 233910 BitCount = 121 ID = 1968
Rx   d 8 02 76 01 00 00 00 00 00  成功Length = 235910 BitCount = 122 ID = 1968
Tx   d 8 10 82 36 02 5F 01 00 08  第二次数据传输Length = 234244 BitCount = 120 ID = 1
Rx   d 8 30 08 00 00 00 00 00 00  Length = 239910 BitCount = 124 ID = 1968
Tx   d 8 21 5F 01 00 08 5F 01 00  Length = 234244 BitCount = 120 ID = 1840
Tx   d 8 22 08 5F 01 00 08 5F 01  Length = 238244 BitCount = 122 ID = 1840
Tx   d 8 23 00 08 5F 01 00 08 5F  Length = 234000 BitCount = 120 ID = 1840
Tx   d 8 24 01 00 08 5F 01 00 08  Length = 234000 BitCount = 120 ID = 1840
```

根据协议，这些东西就是传输的内容，所以我们先把整个log文件“掐头去尾”，只剩下0x40次日志传输相关报文，然后将所有RX的返回报文去掉（直接批量搜索删除即可），类似下面这样：

正则语法：`.*Rx.*\n`



中间还有一行这个，别忘记删掉：

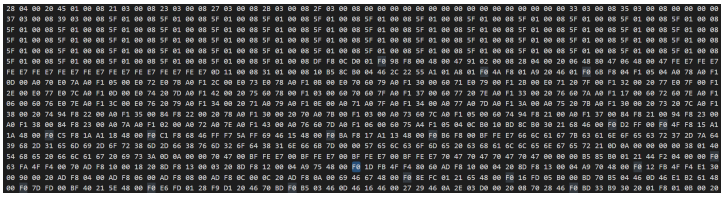


然后对得出来的结果执行下面的代码，提取出传输数据的值（ascii形式）：

```
f = open('ecu_can_log.asc', 'r')
f1 = open('result.txt', 'w')

list1 = f.readlines()
data=[]
count=1
for i in list1:
    if(count%19==0):    #19行是数据传输的最后一行，后面有两个AA是不要的
        data=i[43:57]
    elif(count%19==1):  #每次传输的第一行要去掉传输协议头
        data=i[52:63]
    else:
        data=i[43:63]   #其他中间传输的报文，直接去掉UDS网络层即可
    f1.write(data+' ')
    count+=1
```

得到的结果差不多是这样：



最开始我还以为是什么压缩文件或者图片啥的，还在找这个文件头，发现不认识这个文件头，而且binwalk直接跑也没结果。。。。唉。

然后我们需要把这个文件转化为二进制形式，通过如下脚本实现：

```
f = open('result.txt', 'r')
f1 = open('result', 'w')

strings=f.read()
i=0
while(i<len(strings)):
    data=int(strings[i:i+2],16)
    f1.write(chr(data))
    i+=3
```

输出了二进制文件之后通过strings可以发现里面有个flag：

```
breeze@DESKTOP-FO89N1Q:/mnt/d/share/ctf/wangding/wp/misc_teslaaaaa/teslaaaaa$ strings ./result
F,"U
u x
w
v z
r
s |
0!hF
flag{canoecr7-zd9h-lemi-or8m-f8vm2od81nfk}
Welcome challenger!
@The flag is:
pGpGpGpG
iFgH
```

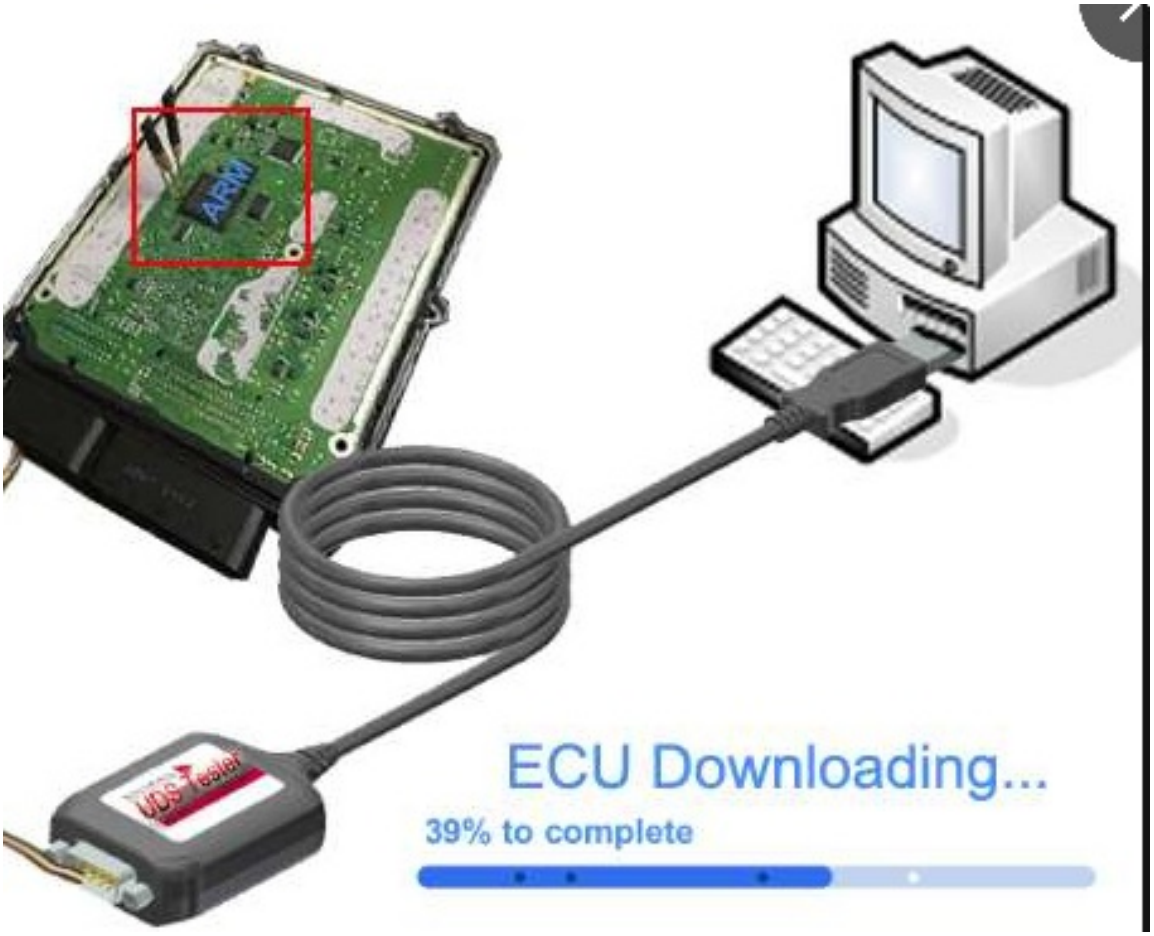
提交发现并不是对的。。。。。

当时为什么就没有想到是个二进制文件呢，好后悔，赛后复盘的时候队内大佬也再说是不是arm的可执行文件啥的。我才恍然大悟，再加上后来看到看雪的wp。

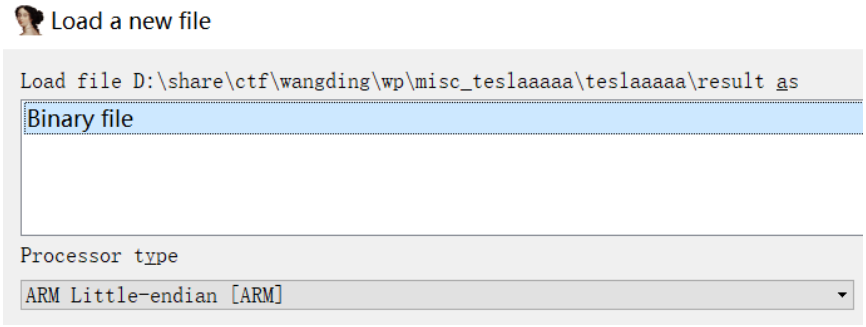
文件逆向

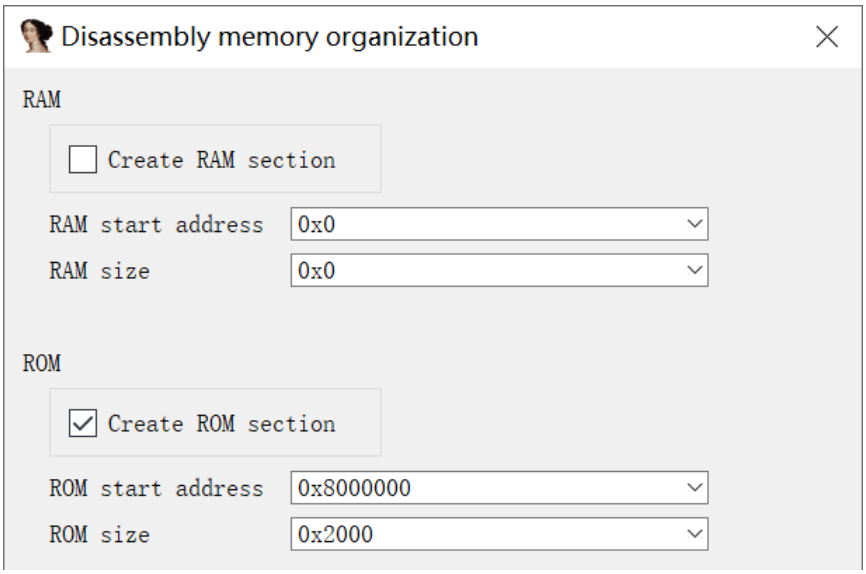
题目给的图片也是一个提示，图片上可以看到单板是arm的

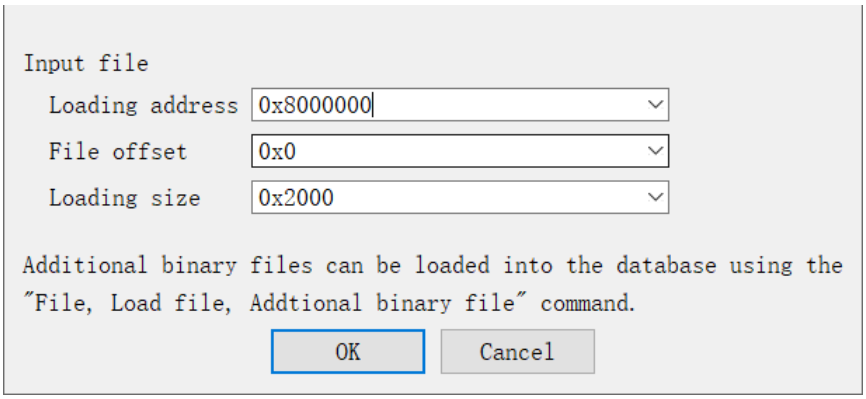
题目给的图片也是一个提示，图片上可以看到单板是arm的：



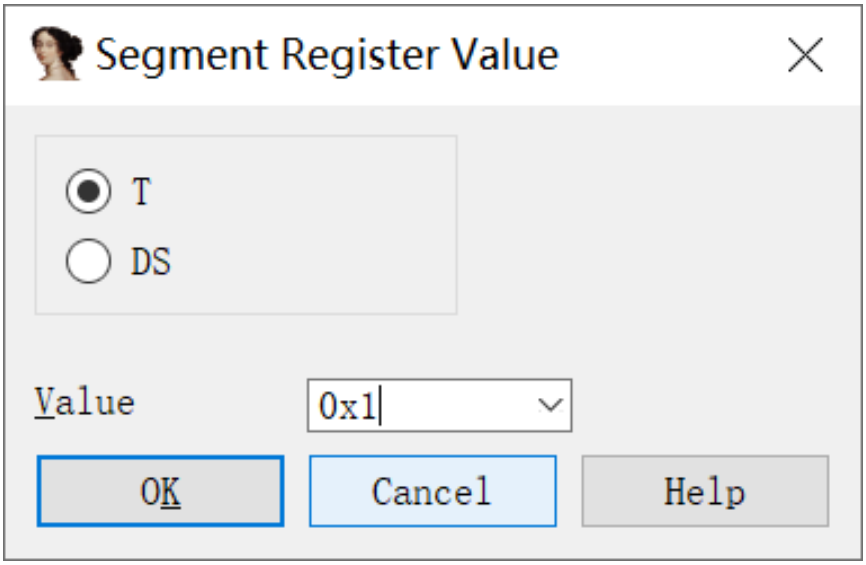
然后通过IDA打开它，arm小端，加载到内存0x8000000地址处：



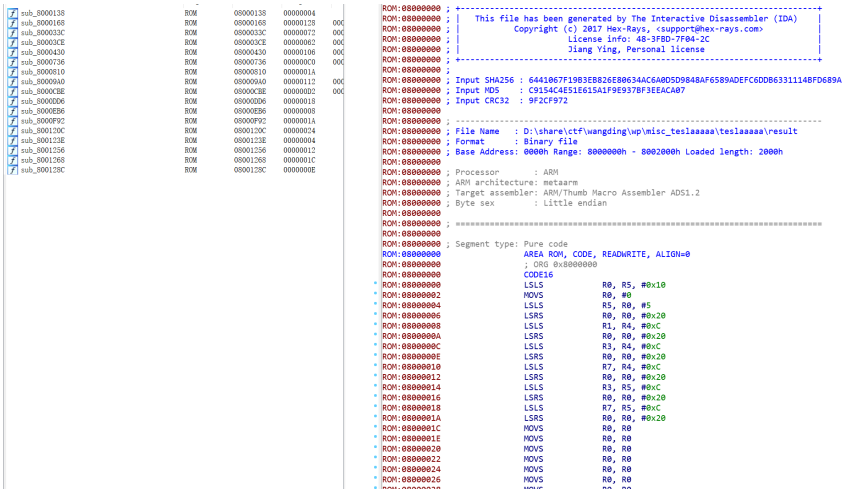




然后将汇编模式改为thumb，按alt+G，将T值设为1即可。



接下来按C便可以反汇编了：



主要函数：

```
void sub_8000138()
{
  int v0; // r0
  int v1; // [sp-30h] [bp-30h]

  v0 = sub_800123E(&v1, 48);
  sub_800033C(v0);
  sub_8000430(1073821696, "Welcome challenger!\r\n");
```

```
  sub_8000430(1073821696, "The flag is:\r\n");
  sub_8000168(&v1);
  sub_8000430(1073821696, &v1);
  sub_8000430(1073821696, "\r\n");
  while ( 1 )
    ;
}
```

计算flag：

```
int __fastcall sub_8000168(_BYTE *a1)
{
  _BYTE *v1; // r4
  int result; // r0
  char v3; // [sp+4h] [bp-34h]

  v1 = a1;
  sub_800120C(&v3, "flag{canoecr7-zd9h-1emi-or8m-f8vm2od81nfk}", 44);
  sub_8001256(v1, &v3);
  v1 += 5;
  v1[2] -= 13;
  v1[11] -= 5;
  v1[15] -= 44;
  v1[3] -= 11;
  v1[5] -= 48;
  v1[7] += 43;
  v1[28] += 50;
  v1[31] += 46;
  v1[19] -= 13;
  v1[20] -= 66;
  v1[1] += 3;
  v1[29] -= 55;
  v1[24] -= 51;
  v1[9] -= 23;
  v1[25] -= 6;
  v1[27] -= 60;
  v1[4] -= 52;
  v1[6] -= 14;
  v1[30] -= 52;
  v1[22] -= 58;
  v1[12] -= 48;
  v1[16] -= 56;
  v1[34] -= 53;
  *v1 -= 48;
  v1[14] += 3;
  v1[17] -= 5;
  v1[33] -= 55;
  v1[35] -= 56;
  v1[10] -= 2;
  v1[26] -= 67;
  result = (unsigned __int8)v1[21] - 6;
  v1[21] = result;
  return result;
}
```

flag计算

直接将上述代码拷贝到python上执行一遍：

```
temp="canoecr7-zd9h-1emi-or8m-f8vm2od81nfk"
v1=[]
j=0
for i in temp:
    v1.append(ord(i))
    j+=1
v1[2] -= 13
v1[11] -= 5
v1[15] -= 44
v1[3] -= 11
v1[5] -= 48
v1[7] += 43
v1[28] += 50
v1[31] += 46
v1[19] -= 13
v1[20] -= 66
v1[1] += 3
v1[29] -= 55
v1[24] -= 51
v1[9] -= 23
v1[25] -= 6
v1[27] -= 60
v1[4] -= 52
v1[6] -= 14
v1[30] -= 52
v1[22] -= 58
v1[12] -= 48
v1[16] -= 56
v1[34] -= 53
v1[0]-= 48
v1[14] += 3
v1[17] -= 5
v1[33] -= 55
v1[35] -= 56
v1[10] -= 2
v1[26] -= 67
flag="flag{"
for i in v1:
    flag+=chr(i)
flag+='}'
print flag
```

```
breeze@DESKTOP-FO89N1Q:/mnt/d/share/ctf/wangding/wp/misc_teslaaaaa/teslaaaaa$ python ./exp.py
flag{3dad13db-cb48-495d-b083-3231d80f1713}
```

flag为：flag{3dad13db-cb48-495d-b083-3231d80f1713}

这道题真的是一道很好的题目，之前都没接触过智能汽车协议分析等相关的东西，学习了很多。

参考链接：

看雪学院HHHso大佬wp

知乎养只喵叫天狼星大佬的UDS诊断服务详解

CSDN GavinChen-GuiGan大佬的UDS网络传输层讲解

CSDN zhyongquan大佬的CAN数据格式

CSDN qfmzhu大佬的UDS汽车诊断协议讲解