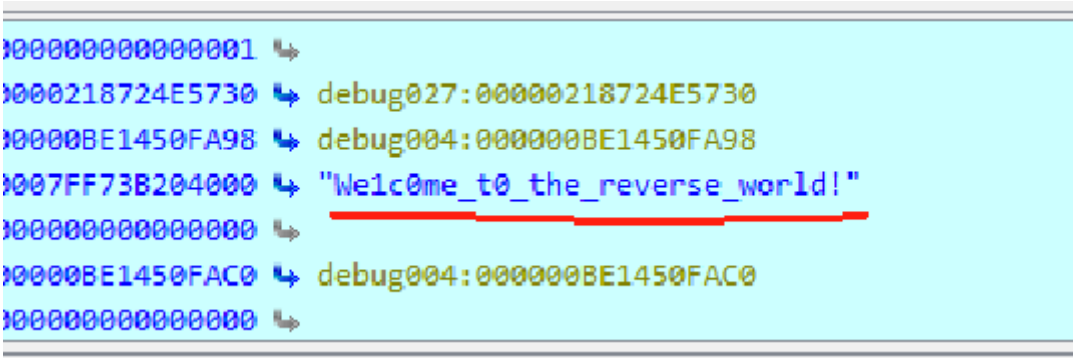
第一题 easystr

这题非常简单。

ida逆向之后进行一个调试，在strcmp处下一个断点，



于是就拿到了flag{We1c0me_t0_the_reverse_world!}

因为▶太好拿就没看程序具体过程了。

第二题xorrr

初步分析

看程序的结构，应该是有一个循环，然后发现是`for(int i=0;i<24;i++)`的循环体。

首先初始化了三个数组，对应于

```
v5[0] = (int64)"X1j3y5a7u9t;`=|";
v5[1] = (int64)";5w7n9 ;l=n?)A-";
v5[2] = (__int64)"s9?;}=j?|AxChEj";
```

里面有一个运算比较，如果失败的话就break结束程序。

判断条件是：

`(i + 8) ^ (char*)(v5[i % 3] + 2 * (i / 3))) - input_string[i] == 2`

那么正确的情况就应该是执行完整个函数都没有break对应的那个数组。

那么直接用ida反汇编的得到的程序改一下：

`input_string[i]=(i + 8) ^ (char*)(v5[i % 3] + 2 * (i / 3))) +2`

就能得到input_string对应的值,然后得到flag：

`N0w_y0u_kn0w_what_x0r_1s`

第三题maze

首先知道了是一个长度为10的数组，所以输入应该是十个字符。

其次知道了一共有四个合法字符，分别是dsaw，d为+1，s为+5，w为-5，a为-1。

发现这个标签分别对应于对某个初值为0的变量进行加减操作。

然后那里会有一个比较，会报wrong。

比较的东西是一个字符串：`'1100001100001000110*1100'`

通过分析可知，最后对应的比较的值是42，正好是`*`。

那么那个比较判断正好就是说要通过10次操作跑到`*`处，同时如果超出数组范围或者取值为0就会退出。

那么只需要通过dsaw的组合操作"跳"到`*`就可以了。

得到的dsaw操作为：

`ddsdssasaa`

对应flag:

`flag{0e6321aa4d31bfc66b83c0406885ce86}`

第四题array

首先知道是一个20位的密码。

然后发现，每次会从一个数组

'~}|{zyxwvutsrqponmlkjihgfedcba_^] `[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(',27h,'&'

中取出一个存入之后的比较数组中。

最后和字符串

82=7#+1;?50:9&?9=+%!

进行比较，如果正确的话就过了。

有个问题就是如何从对应于第一个数组中选到正确的位置?

目标字符串第一个位置对应8，试了一些值，发现当输入f的时候得到的第一个位置刚好是8.

那么肯定就猜前四位是`flag`，试了一下果然前四位都对的上。

虽然不太搞得懂为什么f就对应8了，但是通过前两个字符就能发现规律，注意到数组是按照ascii码逆序排的，对应规则刚好是`x+y=158`，其中x是输入字符对应的ascii，y是拿到的字符对应的ascii。

那么为了方便写个c程序跑一下：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    s="82=7#+1;?50:9&?9=+%!";
    for (int i=0; i<20;i++)
    {
        printf("%c",158-s[i]);
    }//输出为flag{smc_index_easy}
    return 0;
}
```

所以对应flag：

`flag{smc_index_easy}`

第五题

首先根据循环条件知道是一个16位的字符串。

根据结构可知，会对输入的i的取值进行加减，再与特定的字符串比较，该字符串为：
`Zxb3xo_qe4_Dob@q`

那么类似于第二题，逆向考虑，根据这个值来倒推应该有的输入即可。

通过纸上一步一步的倒推，错了好多次之后，判断好各个条件对应的逆向操作，写个c程序:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    s="Zxb3xo_qe4_Dob@q";
    for (int i=0; i<16;i++)
    {
       int t=int(s[i]);
        if(t<=87&&t>=65)
        {
            t=t+3;
        }
        else if(t<91&&t>=88)
        {
            t=t-23;
        }
        else if(t<97||t>122)
        {
            t=t;
        }
        else if(t>=120&&t<122)
        {
            t=t-23;
        }
        else if(t<=119&t>=97)
        {
            t=t+3;
        }
       printf("%c",t);//输出为Cae3ar_th4_Gre@t
    }

    return 0;
}
```

于是flag为：

`Cae3ar_th4_Gre@t`

第六题equation

一个32位的密码。

首先前四位是解方程，因为过于复杂，不想倒推，那么既然只有四位，穷举空间也不大，一秒钟就完事，就写个c穷举一下。

为了方便写，用int格式来写，最后按%c来输出。

得到答案为QTEM

```
#include <bits/stdc++.h>
using namespace std;
int main()
{

    int Str[4] = {0};
    for (int i1 = 32; i1 < 126; i1++)
    {Str[0]=i1;
        for (int i2 = 32; i2 < 126; i2++)
        {Str[1]=i2;
            for (int i3 = 32; i3 < 126; i3++)
            {Str[2]=i3;
                for (int i4 = 32; i4 < 126; i4++)
                {   Str[3]=i4;
                int dword_1400040A8[5]={0};
                    dword_1400040A8[4] += 16;
                    dword_1400040A8[3] += Str[3] * dword_1400040A8[4];
                    dword_1400040A8[4] += 3;
                    dword_1400040A8[3] += Str[2] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 10;
                    dword_1400040A8[0] += Str[3] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 2;
                    dword_1400040A8[1] += Str[2] * dword_1400040A8[4];
                    dword_1400040A8[4] += 13;
                    dword_1400040A8[0] += Str[1] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 8;
                    dword_1400040A8[2] += Str[1] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 7;
                    dword_1400040A8[2] -= 3481;
                    dword_1400040A8[4] += 3;
                    dword_1400040A8[1] -= 2422;
                    dword_1400040A8[4] += 9;
                    dword_1400040A8[0] += Str[2] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 2;
                    dword_1400040A8[2] += Str[2] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 6;
                    dword_1400040A8[0] -= 4518;
                    dword_1400040A8[4] += 7;
                    dword_1400040A8[3] -= 5006;
                    dword_1400040A8[4] -= 9;
                    dword_1400040A8[1] += Str[0] * dword_1400040A8[4];
                    dword_1400040A8[4] += 5;
                    dword_1400040A8[0] += Str[0] * dword_1400040A8[4];
                    dword_1400040A8[4] ++;
                    dword_1400040A8[2] += Str[0] *dword_1400040A8[4];
                    dword_1400040A8[4] -= 8;
                    dword_1400040A8[2] += Str[3] * dword_1400040A8[4];
                    dword_1400040A8[4] += 14;
                    dword_1400040A8[3] += Str[0] * dword_1400040A8[4];
                    dword_1400040A8[4] -= 11;
                    dword_1400040A8[1] += Str[3] * dword_1400040A8[4];
                    dword_1400040A8[4] += 3;
                    dword_1400040A8[3] += Str[1] * dword_1400040A8[4];
```

```
                    dword_1400040A8[4] -= 2;
                    dword_1400040A8[1] += Str[1] * dword_1400040A8[4];
                    for (int i = 0; i < 4; ++i)
                    {
                        if (dword_1400040A8[i])
                            break;
                        else if(i==3)
                        {
                            printf("%c %c %c %c",Str[0],Str[1],Str[2],Str[3]);
                            break;
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

然后解后面二十八位，还是要倒着分析。

观察发现可以利用性质：
`x^y^y=x`

那么就，可以写c代码了。

不知道为什直接按照*取地址的方式来写会出错，所以选择换成数组来整。

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    char Str[32]={0};
    Str[0]='Q';
    Str[1]='T';
    Str[2]='E';
    Str[3]='M';
    long long qword_140004040[8];
    qword_140004040[0]= 0x7C007C00100064;
    qword_140004040[1] = 0x34002F00170022;
    qword_140004040[2]= 0x3B000B00130013;
    qword_140004040[3] = 0xF001B001F;
    qword_140004040[4] = 0x3C00720005001E;
    qword_140004040[5] = 0x40036001F0062;
    qword_140004040[6] = 0x3F0032002E001F;
    qword_140004040[7] = 0;
    for (int  j = 0; j < 28; ++j )
    {
       cout<< (char)(Str[j % 4] ^ (*((char *)&qword_140004040 + 2 * j) ));

        }

    return 0;
}
```

flag:

`QTEM5D91sCjyBGNvNOJMOQ7q3KsINzwr`

第七题click

点了好几个1000次，得到的结果有点随机。

想调参debug跳过验证，但是跳不过去。

后来想到搜索一下输出的flag存在哪里，然后搜半天也没搜到。后来想到搜一下`Click`，然后就找到了对应于界面输出的地方，对应找到了好几个字符串。

好几个字符串都试了下，直接提交没法直接通过，那么大概率是加密了，于是需要解一下。

因为在写了第八题`junkcode`之后才写的这个题，正好这几个字符串位数也是4的倍数，所以就猜它是base64。

于是解一下码，还真是，就得到了：

“YWJjZGUtMzg0ODMta2RraHly” -> “abcde-38483-kdkhyr”

“aHFrcWstMjE4MjMtamNoZGts” -> “hqkqk-21823-jchdkl”

“eHh4eXktMTIzNDUtamtvcG1w” -> “xxxyy-12345-jkopmp”

“UXRmdW4tMTAwODYtR1VJdG9v” -> “Qtfun-10086-GUItoo”

把最后一个带进去，竟然就行了。

`flag{Qtfun-10086-GUItoo}`

第八题 junkcode

最开始就是愣头青，找到一个函数就开始点，于是发现了一个函数可以点进去，就点了，弄了好久知道了这个是base64。

还以为junkcode就这？不过是把函数藏起来找一下罢了。硬猜那个base64的函数就是最终结果，后来发现这样子搞不出来，解码得到的玩意儿一大堆非法字符，看起来果然不止一次加密。

投机取巧失败，于是考虑别的。

通过百度知道了红色的这玩意儿叫做花指令，于是看了一些相关的视频，但是不知道为什么一直不成功。

但还是去掉了几个花指令，但不知道哪里有问题一直没法把主函数反编译成功。

于是不得不硬着头皮看了很久的汇编来分析，幸好因为nop了一些，大概也知道了主函数的流程是什么。

然后幸好两个子函数比较清晰明了，有清晰的结构。

首先知道最后要比较的串是：

P1Ekb1UxW9ErWC6ZVUKiKgMaLSEgS5gpyZOrSQG

过程中首先经过了一次对输入字符串的base64编码，然后再对它进行凯撒密码加密，每11位为一组，移动不同的位数。

于是只需要根据程序本身，把凯撒加密逆过来，修改一下正负号，再用base64解码就可以了。

逆凯撒加密的程序：

```
#include<bits/stdc++.h>
using namespace std;
int __cdecl sub_401000(char *Str, int a3)
{
  signed int v4; // [esp+0h] [ebp-Ch]
  int k; // [esp+4h] [ebp-8h]
  int j; // [esp+4h] [ebp-8h]
  signed int i; // [esp+8h] [ebp-4h]
  if ( !Str || a3 <= 0 )
    return -1;
  v4 = 11;
  char x;
  if ( v4 <= 0 )
    return -1;
  for ( i = 0; i < v4; ++i )
  {
    if ( Str[i] < 65 || Str[i] > 90 )
    {
      if ( Str[i] < 97 || Str[i] > 122 )
      {
        if ( Str[i] < 48 || Str[i] > 57 )
          x=Str[i];
        else
         x= (Str[i] - 48 + a3 + 10) % 10 + 48;
      }
      else
      {
        for ( j = Str[i] - 97 + a3; j >=26; j -= 26 )
          ;
        x = j + 97;
      }
    }
    else
    {
      for ( k = Str[i] - 65 + a3; k >= 26; k -= 26 )
        ;
      x = k + 65;
    }
    printf("%c",x);
  }
  return 0;
}
int main(){
  char s[100]="ZOrSQG3tP8g";//为了方便，直接根据反编译结果改的，简化了输入输出，每次输入11个来直接输出原本字符串。
  sub_401000(s, 7);//分别是1，3，5，7位加密
 return 0;
}
```

得到的base64编码值：

`Q2Flc2VyX0FuZF9CYXNlNjRfQXJlX0ludGVyZXN0aW5n`

解码得到flag：

`Caeser_And_Base64_Are_Interesting`

(但是base64和caeser真不有趣(T⌓T))

第九题multithread

首先去除一大堆花指令。

但是有些花指令不太理解，就把会搞的搞了。

最重要的是通过nop去除花指令得到了一个看起来很重要的函数：StartAddress。

然后观察经过初步处理去掉一些花指令后反编译得到的主函数：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // eax
  char v5; // [esp+0h] [ebp-14h]
  HANDLE Handles[2]; // [esp+8h] [ebp-Ch] BYREF

  sub_401020("plz input your flag:", v5);
  sub_401050("%42s", (char)byte_40336C);
  Handles[0] = CreateThread(0, 0, StartAddress, 0, 0, 0);
  Handles[1] = CreateThread(0, 0, sub_401200, 0, 0, 0);
  CreateThread(0, 0, sub_401240, 0, 0, 0);
  WaitForMultipleObjects(2u, Handles, 1, 0xFFFFFFFF);
  for ( i = 0; i < 42; ++i )
  {
    if ( byte_40336C[i] != byte_402150[i] )
    {
      sub_401020("error", (char)Handles[0]);
      exit(0);
    }
  }
  sub_401020("win", (char)Handles[0]);
  getchar();
  return 0;
}
```

搜了一下createthread什么意思，不太懂，但是我猜是好几个函数一起调用，就不管了。

第二个和第三个线程看起来都怪怪的，看不懂有什么用，其中有一个的输出是叫我debug，但不知道什么情况下会遇到，调试的时候没发现。还有一个好像是和那个debug的输出有关，也没看懂。

但是自己反编译搞出来那个函数比较好看。

```
DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
{
  int v2; // [esp+4h] [ebp-14h]
  int i; // [esp+14h] [ebp-4h]

  CreateThread(0, 0, hHandle, 0, 0, 0);
  WaitForSingleObject(hHandle, 0xFFFFFFFF);
  for ( i = 0; i < 42; ++i )
  {
    byte_40336C[i] = (byte_40336C[i] << 6) ^ ((int)(unsigned __int8)byte_40336C[i] >> 2);
    byte_40336C[i] ^= 0x23u;
    Sleep(6u);
    v2 += NtCurrentPeb()->BeingDebugged + 9;
    byte_40336C[i] += 35;
  }
  return sub_401200(lpThreadParameter);
}
```

把所有函数点进去看了一下，好像那几个线程和函数都没有影响到主函数中比较用到的那两个字符串，那么就直接根据自己解出来的玩意儿写一个反向的解密函数试试。

本来打算完全逆推，后来发现第一步没法推，就只能用循环遍历穷举了。

```
#include<bits/stdc++.h>
using namespace std;
int  main()
{
  unsigned char b[42]={0xdd,0x5b,0x9e,0x1d,0x20,0x9e,0x90,0x91,0x90,0x90,
  0x91,0x92,0xde,0x8b,0x11,0xd1,0x1e,0x9e,0x8b,0x51,
  0x11,0x50,0x51,0x8b,0x9e,0x5d,0x5d,0x11,0x8b,0x90,
  0x12,0x91,0x50,0x12,0xd2,0x91,0x92,0x1e,0x9e,0x90,
  0xd2,0x9f};
  for (int  i = 0; i < 42; ++i )
  {
    b[i]-=35;
    b[i]^=0x23;
    for(unsigned t=0;t<128;t++)
    {unsigned char x=t;
    x=(x<<6)^((unsigned char)x>>2);//这个没想到能怎么逆推，就不得不穷举了
    if(x==b[i])
    {printf("%c",t);}
    }
  }
  return 0;
}
```

然后还真就行了？？？

`flag{a959951b-76ca-4784-add7-93583251ca92}`

第十题babyvm

首先直接ida发现非常奇怪的玩意儿，肯定是有点问题。

用detect it easy看了下，是用了upx加壳，于是查了一下upx是什么，下了个工具来脱壳，就好写起来了。

首先是一个24位的输入串。

对每个位置进行一个操作。

操作是一个循环，对应于给每个位置进行一个+2再异或0x16的操作。

循环看起来很复杂，但根据纸上顺着过程走一遍之后发现，其实对字符串进行操作的就一小段，别的都是在改一些变量的值，比如置为0之类的。

实际上它的变化主要是受下面这个影响：

```
            case 10:
              result = c + a1;
              b = *(int *)(c + a1);
              break;
            case 11:
              a = b;
```

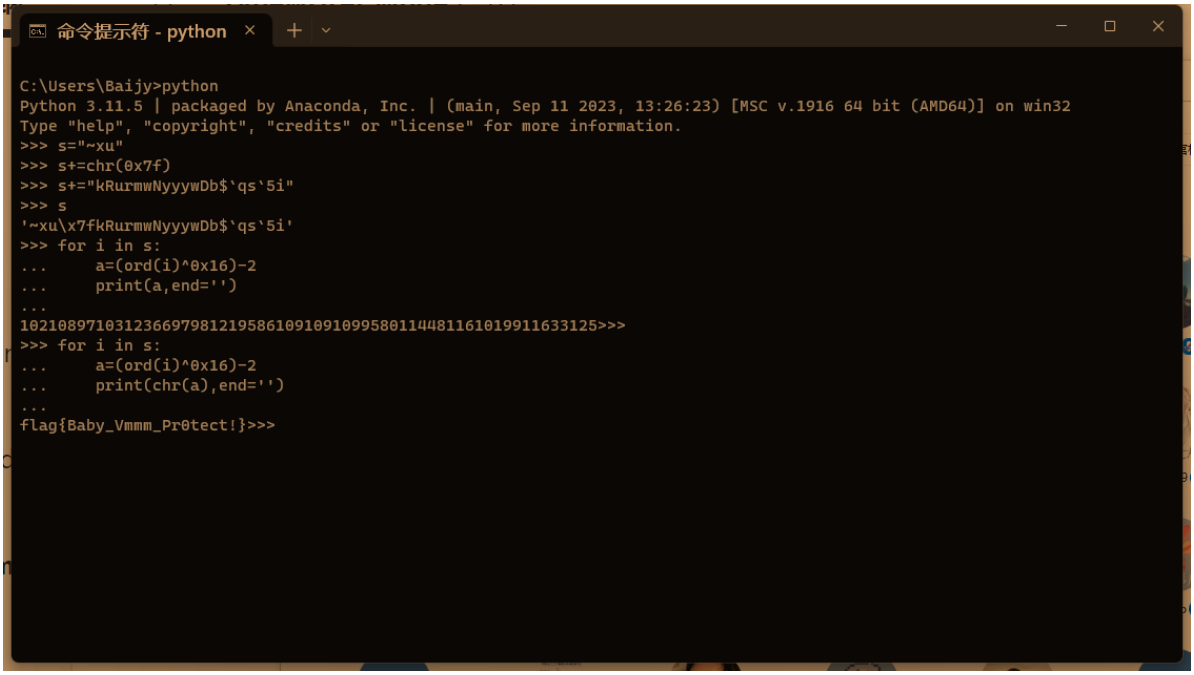
```
            break;
          case 12:
            result = (a + 2) ^ 0x16;
            a = result;
            break;
          case 13:
            b = a;
            break;
          case 14:
            result = b;
            *(int *)(c + a1) = b;
            break;
          case 15:
            ++c;
```

然后会和一个str进行比较，这个str是：

~xu(0x7f)kRurmwNyyywDb$`qs`5i

用之去倒推即可。

于是为了方便，用python写：



所以最后flag:
`flag{Baby_Vmmm_Pr0tect!}`

结束了！！！