20211023-第四届浙江省大学生网络与信息安全竞赛（预赛）-Crypto方向WP

原创

4XWi11 于 2021-10-29 14:50:15 发布 316 收藏 1

分类专栏： 树哥让我天天写之Crypto 文章标签： 网络

本文链接：https://blog.csdn.net/m0_49109277/article/details/121034579

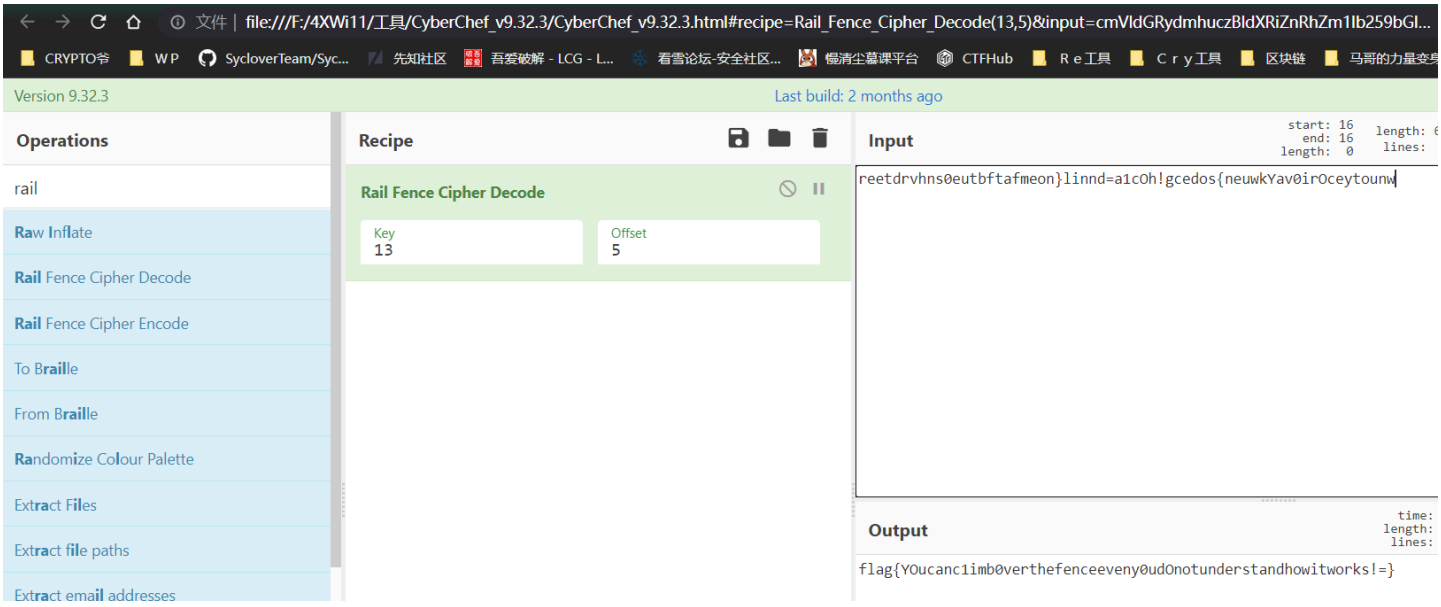
浙江省赛复现

Crypto

Easy Railfence

尝试分析逻辑失败，或者说要花的时间不值得

题目提示Rail，key和offset未知，那么直接手撕



> flag{YOucanc1imb0verthefenceeveny0udOnotunderstandhowitworks!=}

```
import hashlib

m = b'flag{YOucanc1imb0verthefenceeveny0udOnotunderstandhowitworks!=}'

flag = hashlib.md5(m).hexdigest()

print(flag)
```

> a88dd8d7894a7a65dda2d4c6d44357b9

EasyCrypto（recuring）

> ip: 152.136.122.197
> port: 52503
> protocol: tcp

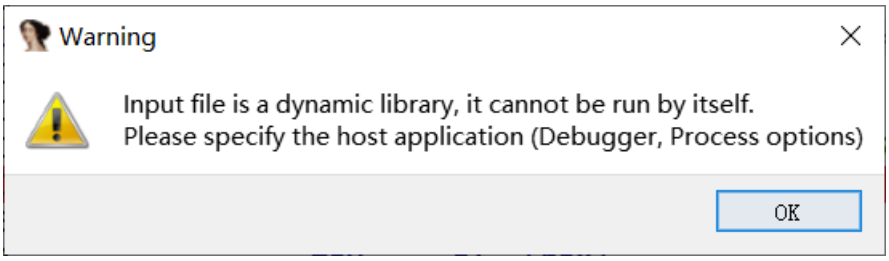
nc链接，好隐晦不知道干嘛，不会又和上次WM一样吧

没看到公告的附件，太卡了，拿到附件让逆向队友upx脱下壳

要达到的攻击效果就是，要在login的时候，把`adminadmin`生成的`token`给发过去，但是在register阶段，不能加密任何包含`admin`字段的字符串

从头开始，先脱壳，吾爱破解吧上随便找了个工具

加壳的不能远调？



whatever脱了就行

然后尽力了，逆向做的不多，刚开始学，而且还是用C++写的，虽然比之前的伪代码好懂多（不知道是不是出题人有意为之），但是做到最后看不出来是哪一模式的AES

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  v55 = __readfsqword(0x28u);                   // 开canary，跳过不影响理解
  key_unknown = GenIV();
  iv_known = GenIV();
  std::operator<<<std::char_traits<char>>(&std::cout, "Here is IV: ");
  // 输出IV，每8位bin（8*16）转2位hex，填充为0，共32位
  for ( i = 0; i <= 15; ++i )
  {
```

```
  v3 = std::ostream::operator<<(&std::cout, std::hex);
  v4 = std::setfill<char>('0');
  v5 = std::operator<<<char,std::char_traits<char>>(v3, v4);
  v6 = std::setw(2);
  output_format = std::operator<<<char,std::char_traits<char>>(v5, v6);
  v8 = std::bitset<8ul>::to_ulong(&iv_known[i]);// 转为十进制
  std::ostream::operator<<(output_format, v8);
}
std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
while ( 1 )
{
  while ( 1 )
  {
    while ( 1 )
    {
      display();
      std::istream::operator>>(&std::cin, &choice);
      if ( choice != 1 )
        break;
      std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(name);
      v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Plz input your name: ");
      std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
      std::operator>><char>(&std::cin, name);
      // 输入的name不包含"admin"，而且长度为16的倍数
      if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find(name, "admin", 0LL) != -1
        || (std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(name) & 15) != 0 )
      {
        v11 = std::operator<<<std::char_traits<char>>(&std::cout, "no no no");
        std::ostream::operator<<(v11, &std::endl<char,std::char_traits<char>>);
        exit(0);
      }
      v12 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(name);
      if ( v12 > 1152921504606846975LL )
        __cxa_throw_bad_array_new_length();
      v13 = operator new[](8 * v12);
      v14 = v12 - 1;
      v15 = v13;
      while ( v14 >= 0 )
      {
        std::bitset<8ul>::bitset(v15);
        v15 += 8LL;
        --v14;
      }
      bin_name = v13;
      // 将name的每个字符取出来，并转成8位bin
      for ( j = 0; ; ++j )
      {
        v16 = j;
        if ( v16 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(name) )
          break;
        v17 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::at(name, j);
        std::bitset<8ul>::bitset(&v47, *v17);
        *(bin_name + 8LL * j) = v47;
      }
      len_name = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(name);
      encode(key_unknown, iv_known, bin_name, len_name);
      std::operator<<<std::char_traits<char>>(&std::cout, "Here is your token: ");
      // 输出token
      for ( k = 0; ; ++k )
      {
```

```
  {
    v19 = k;
    if ( v19 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(name) )
      break;
    v20 = std::ostream::operator<<(&std::cout, std::hex);
    v21 = std::setfill<char>('0');
    v22 = std::operator<<<char,std::char_traits<char>>(v20, v21);
    v23 = std::setw(2);
    v24 = std::operator<<<char,std::char_traits<char>>(v22, v23);
    v25 = std::bitset<8ul>::to_ulong(8LL * k + bin_name);
    std::ostream::operator<<(v24, v25);
  }
  std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(name);
}
if ( choice != 2 )
  break;
std::operator<<<std::char_traits<char>>(&std::cout, "Plz input your token: ");
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(input_token);
std::operator>><char>(&std::cin, input_token);
// 长度必须是32的倍数
if ( (std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(input_token) & 31) != 0 )
{
  v26 = std::operator<<<std::char_traits<char>>(&std::cout, "no no no");
  std::ostream::operator<<(v26, &std::endl<char,std::char_traits<char>>);
  exit(0);
}
v27 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(input_token) >> 1;
if ( v27 > 0xFFFFFFFFFFFFFFFLL )
  __cxa_throw_bad_array_new_length();
init = operator new[](8 * v27);
v29 = v27 - 1;
v30 = init;
// 把token的每两个字符取出，并转成bin
while ( v29 >= 0 )
{
  std::bitset<8ul>::bitset(v30);
  v30 += 8LL;
  --v29;
}
bit_token = init;
for ( m = 0; ; ++m )
{
  v31 = m;
  if ( v31 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(input_token) >> 1 )
    break;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(
    name,
    input_token,
    2 * m,
    2LL);
  v32 = std::__cxx11::stoi(name, 0LL, 16LL);
  std::bitset<8ul>::bitset(&v47, v32);
  *(bit_token + 8LL * m) = v47;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(name);
}
len_token = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(input_token);
decode(key_unknown, iv_known, bit_token, (len_token >> 1));
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(output_name);
// 输出解密后的token
```

```
    for ( n = 0; ; ++n )
    {
      v34 = n;
      if ( v34 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(input_token) >> 1 )
        break;
      v35 = std::bitset<8ul>::to_ulong(8LL * n + bit_token);
      std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(output_name, v35);
    }
    v36 = std::operator<<<std::char_traits<char>>(&std::cout, "Hello, ");
    v37 = std::operator<<<char>(v36, output_name);
    std::ostream::operator<<(v37, &std::endl<char,std::char_traits<char>>);
    // 从output_name中取前10个作为name
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(name, output_name, 0LL, 10LL);
    tip = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::compare(name, "adminadmin") == 0;
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(name);
    if ( tip )
    {
      get_flag = std::operator<<<char>(&std::cout, &flag[abi:cxx11]);
      std::ostream::operator<<(get_flag, &std::endl<char,std::char_traits<char>>);
    }
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(output_name);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(input_token);
  }
  if ( choice == 3 )
    exit(0);
  v40 = std::operator<<<std::char_traits<char>>(&std::cout, "Wrong choice");
  std::ostream::operator<<(v40, &std::endl<char,std::char_traits<char>>);
 }
}
```

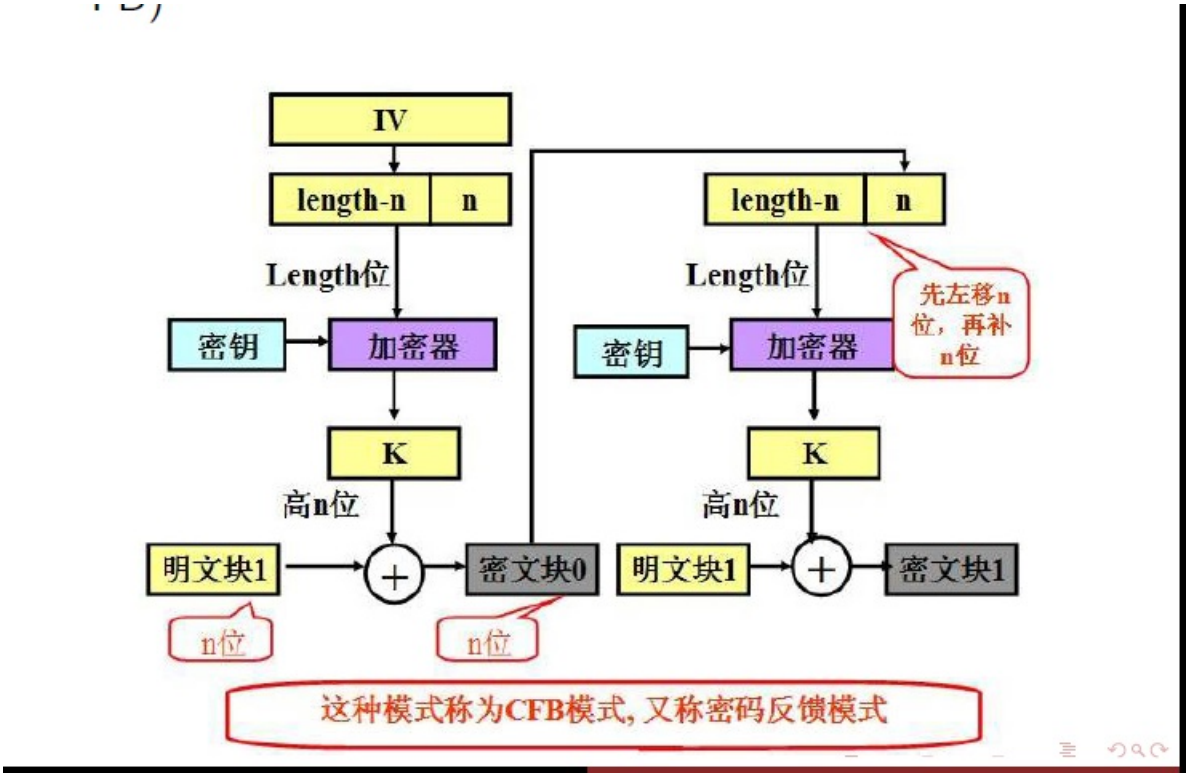
然后加密函数

```
unsigned __int64 __fastcall encode(__int64 KEY, __int64 known_iv, __int64 plaintext, int length)
{
  int v4; // eax
  int i; // [rsp+20h] [rbp-200h]
  int v9; // [rsp+24h] [rbp-1FCh]
  int j; // [rsp+28h] [rbp-1F8h]
  int k; // [rsp+2Ch] [rbp-1F4h]
  __int64 iv[16]; // [rsp+30h] [rbp-1F0h] BYREF
  char key[360]; // [rsp+B0h] [rbp-170h] BYREF
  unsigned __int64 v14; // [rsp+218h] [rbp-8h]

  v14 = __readfsqword(0x28u);
  memset(key, 0, 352uLL);
  KeyExpansion(KEY, key);
  memset(iv, 0, sizeof(iv));
  for ( i = 0; i <= 15; ++i )
    iv[i] = *(known_iv + 8LL * i);
  v9 = 0;
  for ( j = 0; j < length; ++j )
  {
    encrypt(iv, key);
    std::bitset<8ul>::operator^=(plaintext + 8LL * v9, iv);
    for ( k = 0; k <= 14; ++k )
      iv[k] = iv[k + 1];
    v4 = v9++;
    iv[15] = *(8LL * v4 + plaintext);
  }
  return __readfsqword(0x28u) ^ v14;
}
```

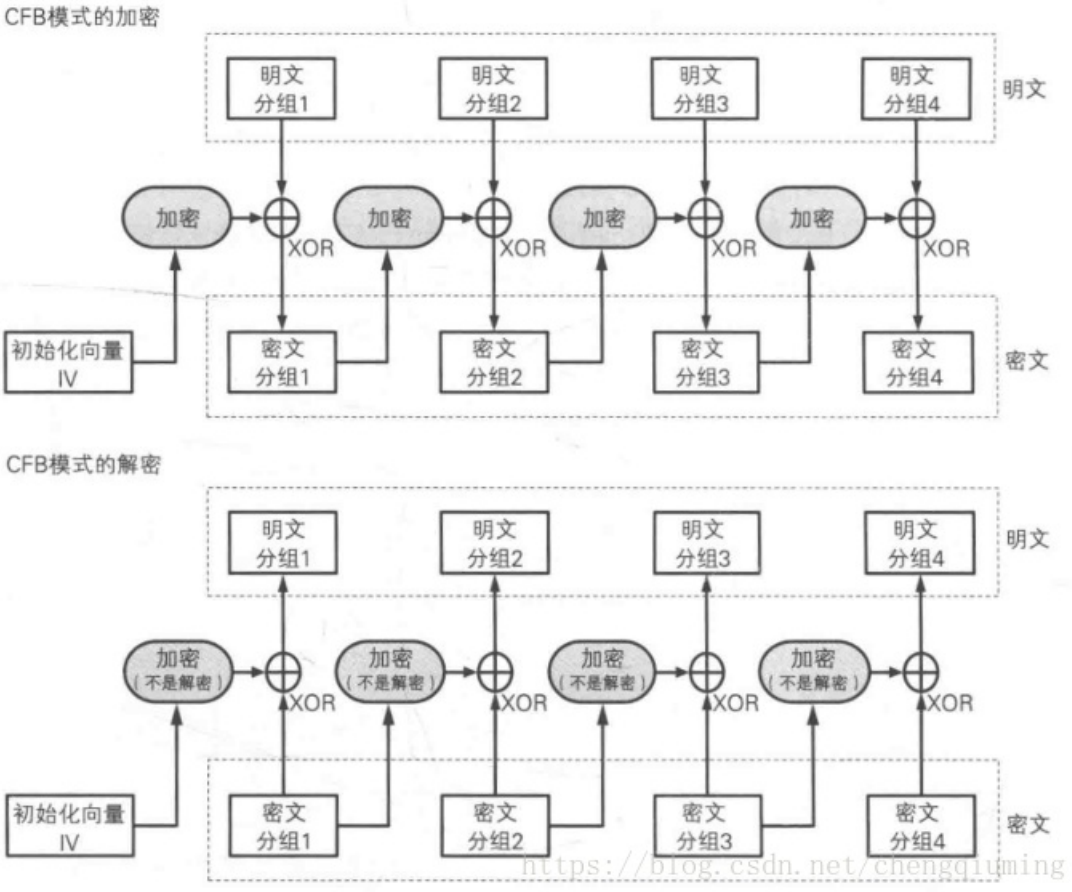
解密函数

```
unsigned __int64 __fastcall decode(__int64 a1, __int64 a2, __int64 a3, int a4)
{
  int v4; // eax
  int i; // [rsp+24h] [rbp-20Ch]
  int v9; // [rsp+28h] [rbp-208h]
  int j; // [rsp+2Ch] [rbp-204h]
  int k; // [rsp+30h] [rbp-200h]
  int v12; // [rsp+34h] [rbp-1FCh]
  __int64 v13; // [rsp+38h] [rbp-1F8h] BYREF
  __int64 v14[16]; // [rsp+40h] [rbp-1F0h] BYREF
  char v15[360]; // [rsp+C0h] [rbp-170h] BYREF
  unsigned __int64 v16; // [rsp+228h] [rbp-8h]

  v16 = __readfsqword(0x28u);
  memset(v15, 0, 0x160uLL);
  KeyExpansion(a1, v15);
  memset(v14, 0, sizeof(v14));
  for ( i = 0; i <= 15; ++i )
    v14[i] = *(a2 + 8LL * i);
  v9 = 0;
  for ( j = 0; j < a4; ++j )
  {
    encrypt(v14, v15);
    v12 = std::bitset<8ul>::to_ulong(8LL * v9 + a3);
    v4 = v9++;
    std::bitset<8ul>::operator^=(a3 + 8LL * v4, v14);
    for ( k = 0; k <= 14; ++k )
      v14[k] = v14[k + 1];
    std::bitset<8ul>::bitset(&v13, v12);
    v14[15] = v13;
  }
  return __readfsqword(0x28u) ^ v16;
}
```

可以看出是AES，感觉是CFB？



借一下上次的图



我觉得已知IV，根据重排攻击，搞出 E_KEY(IV) ，应该不难，然后另起炉灶，用IV和KEY加密以 adminadmin 开头的明文，再send过去就好了

吧

之前字节CTF就做到类似的，看V神的博客知道怎么攻击，可惜没实现

然后网上搜到翻车鱼师傅的博客

https://blog.shi1011.cn/ctf/1700

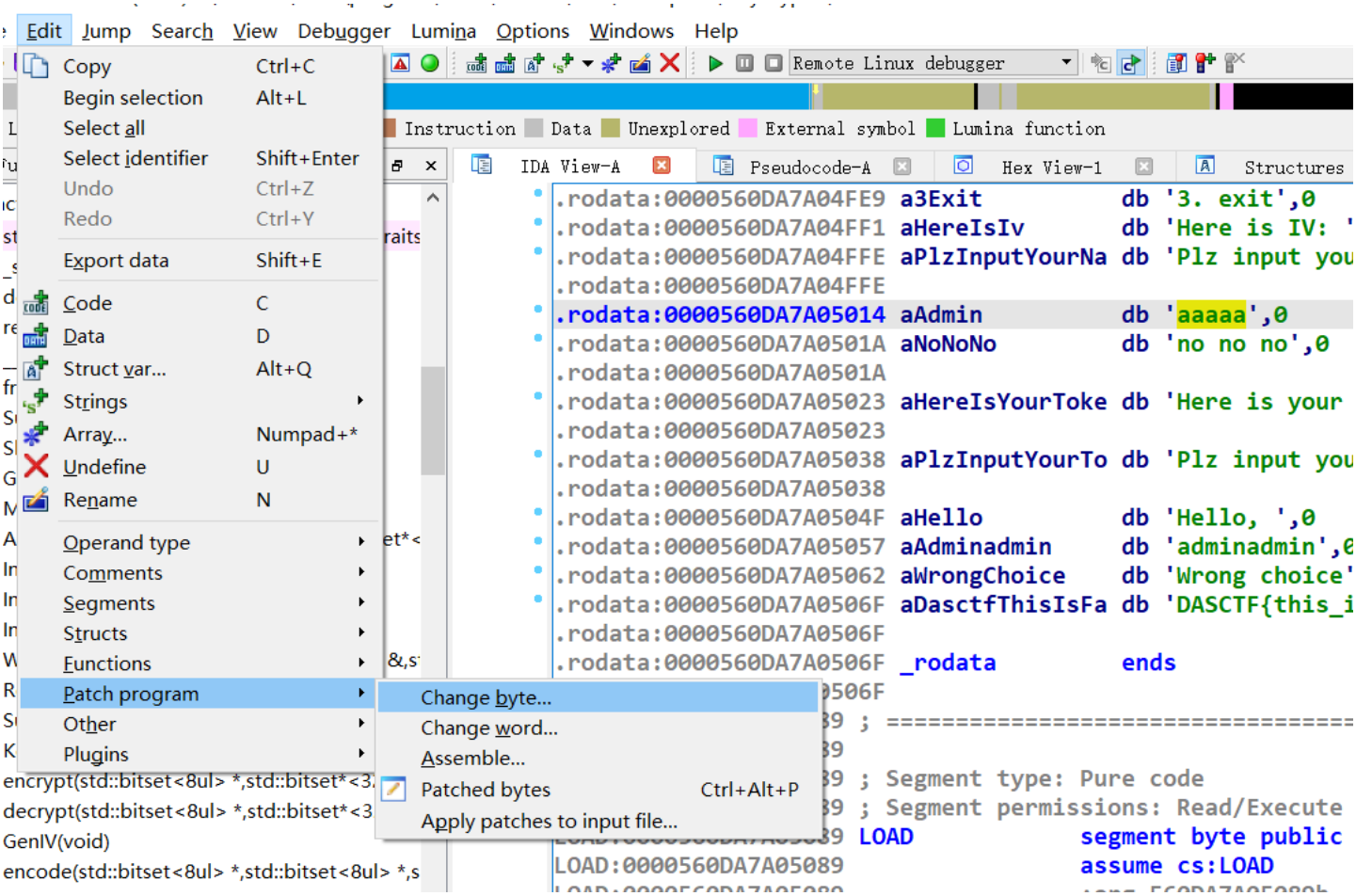
虽然比赛的想到绕过 if，让逆向队友帮 nop 了一下，但是机智的我想改了本地有毛用，服务器没变啊

。。。忘了可以在动调的时候直接改内存，把本地的IV改成服务器跑的IV，然后加密 adminadmin 填充成16长度的字符

emmmmmm逆向的基本素质不够，归结于这方面做的不多

看着师傅的博客一边复现一边学

绕过加密的 if，将 admin 字符串改成别的



再在 GenIV 的时候下个断点，远程调试，步入 GenIV

这里还有一点二进制方向的点，就是IV和KEY其实是同一个，动调也可以看出在加密的时候两个寄存器的值其实是一样的

原因主要是和这个有关系吧

```
14    std::bitset<8ul>::bitset(v4, v1 % 255);
15    GenIV(void)::ret[i] = v4[0];
16  }
17  return GenIV(void)::ret;
18}
000035A5 _Z5GenIVv:1 (561885C035A5)
```

Hex View-1

```
.bss:0000561885E08300 ; _QWORD GenIV(void)::ret[16]
.bss:0000561885E08300 _ZZ5GenIVvE3ret dq 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.bss:0000561885E08300                                         ; DATA XREF: GenIV(void)+84↑o
.bss:0000561885E08300                                         ; GenIV(void):loc_561885C0363E↑o
```

返回的这个是一个全局变量；所以我们一定要步入，在 return 之前把 0x561885e08300 处的值给改成服务器的IV

可以像师傅那样写个脚本放进ida，也可以。。。emmmmm按理说也可以手动patch掉啊，没成功，以后再说

偷下脚本

```
addr = 0x55DF1EE08300  # patch address
test = "2beb18d821cc340659a730a1ac571bb3"  # patch hex data
ps = [i for i in b''.fromhex(test)]
for i, v in enumerate(ps):
    ida_bytes.patch_qword(addr+i * 8, v)
```

反正就是把这些数据给换成连上服务器的IV

```
.bss:000055DF1EE08300 ; GenIV(void)::ret
.bss:000055DF1EE08300 _ZZ5GenIVvE3ret dq 71h, 9Ch, 8Eh, 3Ch, 2Bh, 71h, 92h, 0D7h, 5Fh, 65h, 94h, 0EBh, 5Ah, 24h
.bss:000055DF1EE08300                                         ; DATA XREF: GenIV(void)+84↑o
.bss:000055DF1EE08300                                         ; GenIV(void):loc_55DF1EC0363E↑o
.bss:000055DF1EE08300 dq 9Dh, 37h
.bss:000055DF1EE08300 _bss ends
.bss:000055DF1EE08300
```

IV和KEY都要改

可以看到

```
Looking for GNU DWARF file at "/usr/lib/debug/.build-id/31/52be
Here is IV: 2beb18d821cc340659a730a1ac571bb3
1. register
2. login
3. exit
```

```
# will @ LAPTOP-8N0K85RK in /mnt/f/4XWi11/Natal/progr
$ ./main
Here is IV: 2beb18d821cc340659a730a1ac571bb3
1. register
2. login
3. exit
```

两个IV是一样的了，现在是要加密`adminadmin`后面随便填充满16个字符

```
2021-10-29 14:24:17 [9] Accepting connection from 192.168.137.1...
Looking for GNU DWARF file at "/usr/lib/debug/.build-id/31/52be605d3a2ddbee58ca8571e9e8503b57a80e.debug"... no.
Here is IV: 2beb18d821cc340659a730a1ac571bb3
1. register
2. login
3. exit
1
Plz input your name:
adminadmin123456
Here is your token: 5f53285a68296d575e45104d819722de
1. register
2. login
3. exit
```

把加密的结果给服务器，就获得flag啦

```
Plz input your token: 5f53285a68296d575e45104d819722de
Hello, adminadmin123456
DASCTF{this_is_fake_flag}
1. register
2. login
3. exit
```

当然这是本地的

挺好的，懂一点二进制的知识，也不是很难，200分还凑合吧，因为没有涉及到密码攻击

可信计算1

> 密码体制是可信计算的基础，我国可信计算密码体制借鉴国际先进的可信计算技术框架与技术理念并自主创新,是构建我国可信计算技术规范体系的基础。非对称密码算法采用的椭圆曲线密码算法. 包括三个子算法: 椭圆曲线数字签名算法(SM2-1)、椭圆曲线密钥交换协议(SM2-2)、椭圆曲线公钥加密算法(SM2-3). 对称密码算法采用SM-4算法。差分故障分析攻击是一种强大的密码分析技术，可通过利用加密（解密）过程中的计算错误来检索密钥。我国科研人员提出了使用单一故障（single fault）对SMS4（即SM4）进行新的攻击。黑客利用此攻击获得了密钥并且再经过了一系列RSA加密。你能重新解密它吗？

给了篇论文，让我们填代码，xs

要完整一个attack脚本得到`solution()`的值，然后解RSA，得到的明文和`solution()`的值异或就是flag

离比赛结束还有一个四五十分钟左右上了提示，相当于告诉了我们`solution()`的值，所以只要解RSA就好了

（所以本来300分的题目，现在顶多200吧

第二个n很奇怪，中间有很多`0`，显然是有意构造的，尝试，然后用**Williams's p+1 光滑数**分解出来了

最后要求d

我们现在相当于知道$n\times D=1\ (mod\ lcm(p$

我的思路是和dp泄露攻击那样，将上式写成

$$n\times D=1+k\times$$

赛后给老韩看了这题，然后就出了

前导知识

改进版欧拉定理

$n=$

欧拉定理：如果$ed\equiv 1\ (mod\ \varphi(n))$，则$m\ \ \equiv m\ (mod\ n)$

改进版欧拉定理：如果$ed\equiv 1(mod\ lcm(p-1$

证明...

按理说只要把$e\times d=1+k\varphi(n)$

emmmmm验证了下$(p-1,q-1)|k$确实不一定成立，但可能

所以由

$n\times D=1\ (mod\ lcm(p$

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from Crypto.Util.number import *
import gmpy2

solution = 29416558414220086456824624126554109715 7

n1 = 4405077945973437035419287286810342680467164824994445863017287425857718058064367794640412586853078343089682257199040398279405943520831910368411272144036263290429027279134620341369472867932142748943900939370411886210476130066241817638604325934138989972586283462534982142208555273727206617309988865793455361561026132931733994626398919219203496779449010438682739786866247084252598908907212620561764983861943789642115256382514700516697279459245070020692018083897351520741894187752652555621713799960889861847955471596042981878988930182754796995540025157400222655523547124372140444854366346125175141685537518528913398362475082963023550518705007434135499146966555320343509832799572533328142914873980714083464226641803937211470059111827905761731285053449287872425538901604516967439734828113276097444621288760780510574237036765614111782882785171493934512613063976597374878482039022229525679671485667155329416043448757872980255141203521
c1 = 2040470907728559562144837701471699958842886502924143170940951920300330705469920644505358508388311496777477054764995077035235698658575174832523551852726014373980149374239009390776201109902621711966732796749320068316711336025560029911784168004125304818031813523186592291392842343696718617379290904175534035887756216695721678801280463979349357202298738101109440012007130354993844420187364636715038757505869998163074221607302922729567651885580362184785959736170342652272664233424398501795600338328896721625926411402939719604380362340818655911893820365244157271
```

```
44395839591529353018862302302030815867905767108821157063103155007434580655468908736199629974969684767894335128 36
09450756224172170448677652692971780286821516767097534780167858067007235131095286410071429519192415627585362097 42
68757767261716579352301430309346372991150370162184343597221375875594028197566677119390867721255328734438447108 37
9020485589865192228025983240 87583

n = 3361086170602976142504997268236060982692605108576474794795911687547279286733833098612106940194747346221692033
1593715285306535801153904166799677297107391746143797832181500987751495926897810354383218808665289145042438919287
7511262015629122255924118621935004557686032418498473563715457851673394055494897328272136943459570471438915020934
5475950609404181675208130514911232000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000097508979015194291
1329495147229111661913634616013838271474836758398878983027327177969644000338725830589882793080602653406999248540
4674359556733671090314522347055607288862172718362366802494892459049088248116804936507618631173486747387737790284
542610674878263408812215057547995507075546686309506221459090 9653863
c = 5985288588748106080659025016208745030149836704632666462908326350564212272057333277360575086195078311335355726
9026097225467312896894633807204529066548955643777158320462471568142938853007896343005296978988610847298472909244
0081160180316457652683648340266414640837170058066574014124614892099724102821458274691436449398719884109828222614
0070369945275924361850504840880753304621172481109192934472359348891623444071292035903868111216739676056951476490
1773763240221302324640288453581039272894642461006609020605508948956407716113877797726371435921031926756161669666
3416315333732256499607636067924154333627546883589391082646997265511574165830249101937578220829206006989141322034
4109964112991766510624072729026322502816566521959863852989376920815694864640581895157270803091311202110094736905
5121367220203494109942947407579545762183139721246795136796492525712181146482534091088615045750827959902400539422
0037658075742792005357721304591247193598733190937420575987082139043012560262600423787438345332029289463428943040
5738002792059704914509599724607650851499207913973815555818 80453992

# 3446950429128415403769916614379587157181438522407441032806400000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001 97508979015194291
1329495147229111661913634616013838271474836758398878983027327177969644000338725830589882793080602653406999248540
4674359556733671090314522347055607288862172718362366802494892459049088248116804936507618631173486747387737790284
542610674878263408812215057547995507075546686309506221459090 9653863
p = 9750897901519429113294951472291116619136346160138382714748367583988789830273271779696440003387258305898827930
8060265340699924854046743595567336710903145223470556072888621727183623668024948924590490882481168049365076186311
7348674738773779028454261067487826340881221505754799550707554668630950622145909096 53863
q = 3446950429128415403769916614379587157181438522407441032806400000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
phi = (p-1)*(q-1)
assert n == p * q
e = 0x10001
D = gmpy2.invert(e, phi)
d = pow(c, D, n)
pq = gmpy2.gcd(n1, (pow(2, n1*d, n1)-2) % n1)
P = n1 // pq
Q = n1 // (P ** 2)
assert P ** 2 * Q == n1
fake_phin1 = (P-1)*(Q-1)
d = gmpy2.invert(n1, P-1)
flag = long_to_bytes(pow(c1, d, P) ^ solution)
print(flag)
```

flag{59814ae119dc9d7c29285fde41236f77}