1. 分析基本逻辑

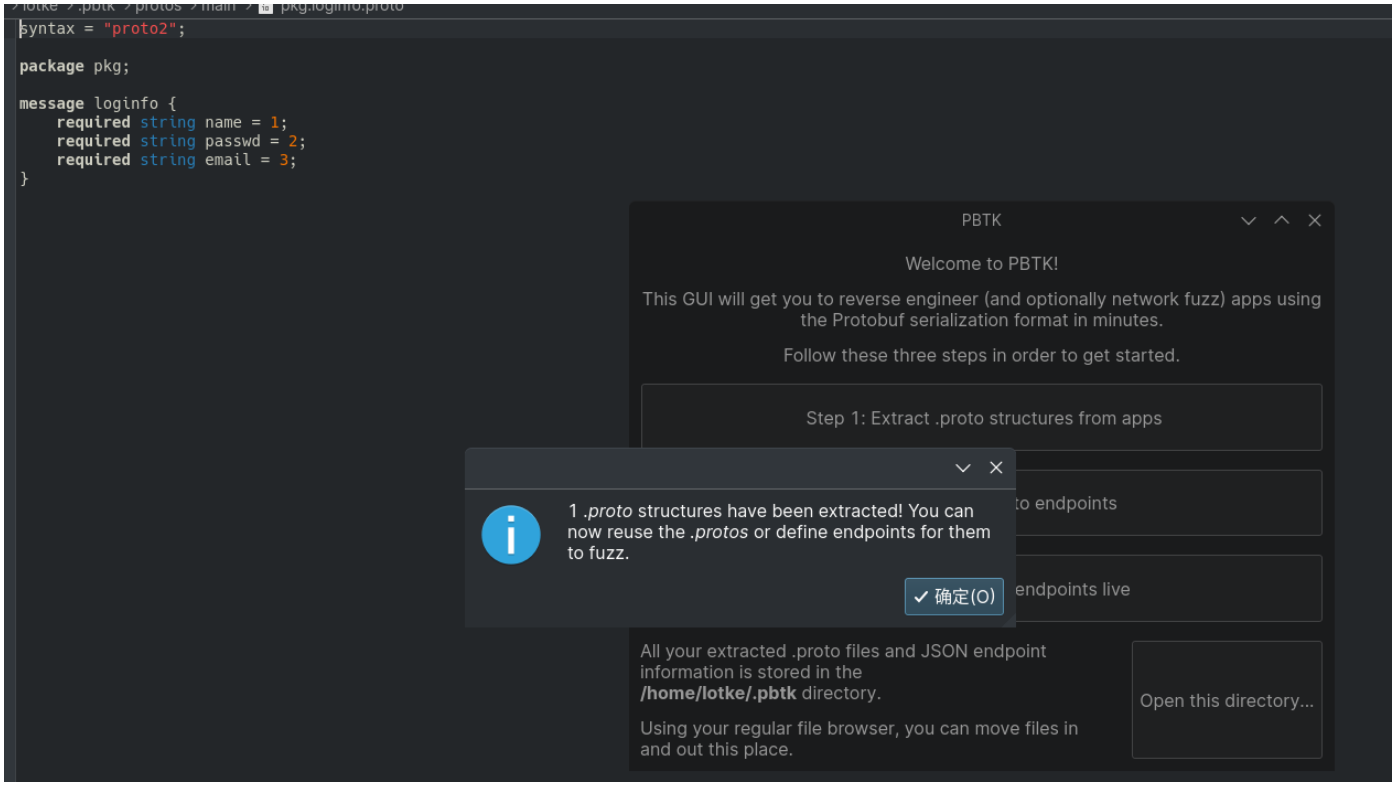
查看main函数，发现它起了个socket来监听本地8848端口读取字符串

```
IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums
{
  socklen_t addr_len; // [rsp+0h] [rbp-A0h] BYREF
  int i; // [rsp+4h] [rbp-9Ch]
  int fd; // [rsp+8h] [rbp-98h]
  int v7; // [rsp+Ch] [rbp-94h]
  struct sockaddr addr; // [rsp+10h] [rbp-90h] BYREF
  struct sockaddr v9; // [rsp+20h] [rbp-80h] BYREF
  char s[104]; // [rsp+30h] [rbp-70h] BYREF
  unsigned __int64 v11; // [rsp+98h] [rbp-8h]

  v11 = __readfsqword(0x28u);
  fd = socket(2, 1, 0);
  addr = 0LL;
  addr.sa_family = 2;
  *(_DWORD *)&addr.sa_data[2] = htonl(0);
  *(_WORD *)addr.sa_data = htons(8848u);
  bind(fd, &addr, 0x10u);
  listen(fd, 20);
  puts("waiting......");
  addr_len = 16;
  v7 = accept(fd, &v9, &addr_len);
  memset(s, 0, 0x64uLL);
  for ( i = read(v7, s, 0x64uLL); i; i = read(v7, s, 0x64uLL) )
  {
    sub_5827((__int64)s);
    memset(s, 0, 0x64uLL);
  }
  puts("closed..");
  close(v7);
  return 0LL;
}
```

之后将字符串传入处理函数，用protobuf的api进行反序列化，格式错误则输出报错

```
char v16[32]; // [rsp+70h] [rbp-F0h] BYREF
char msg[48]; // [rsp+90h] [rbp-D0h] BYREF
char v18[96]; // [rsp+C0h] [rbp-A0h] BYREF
char v19[16]; // [rsp+120h] [rbp-40h] BYREF
__int64 v20[3]; // [rsp+130h] [rbp-30h]
unsigned __int64 v21; // [rsp+148h] [rbp-18h]

v21 = __readfsqword(0x28u);
std::allocator<char>::allocator(&v11);
makestring(str, input, &v11);
std::allocator<char>::~allocator(&v11);
initstring(msg);
if ( (unsigned __int8)google::protobuf::MessageLite::ParseFromString(msg, str) != 1 )
{
  v1 = std::operator<<<std::char_traits<char>>(&std::cerr, "input format error!");
  std::ostream::operator<<(v1, &std::endl<char,std::char_traits<char>>);
}
else
{
```

使用pbtk工具解析题目的二进制文件，即可提取出.proto文件




得到.proto中定义的数据格式。继续分析逻辑，反序列化出数据之后，会对email字段进行验证，如果email字符串中不含字符“@”，则报错并返回，并且如果email中不含字符“@”且不含字符“."且含有字符串”114“，程序就会抛出一个异常。

```
{
  v2 = sub_7C6E(msg);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v18, v2);
  v3 = IsEmailValid(v18) ^ 1;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v18);
  if ( v3 )
  {
    v4 = std::operator<<<std::char_traits<char>>(&std::cerr, "email format error!");
    std::ostream::operator<<(v4, &std::endl<char,std::char_traits<char>>);
```

```
std::allocator<char>::allocator(&v4);
makestring(v8, "@", &v4);
std::allocator<char>::~allocator(&v4);
std::allocator<char>::allocator(&v4);
makestring(v9, ".", &v4);
std::allocator<char>::~allocator(&v4);
std::allocator<char>::allocator(&v4);
makestring(v10, "114", &v4);
std::allocator<char>::~allocator(&v4);
v5 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find(a1, v8, 0LL);
v6 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find(a1, v9, 0LL);
v7 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find(a1, v10, 0LL);
if ( v5 == -1 )
{
  if ( v6 == -1 && v7 != -1 )
  {
    exception = __cxa_allocate_exception(8uLL);
    *exception = "something wrong";
    __cxa_throw(exception, (struct type_info *)&`typeinfo for'char const*, 0LL);
  }
  v2 = 0;
}
else
{
  v2 = v6 != -1;
}
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v10);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v9);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v8);
return v2;
}
```

邮箱验证经过之后，程序将name字段和passwd字段进行结合，并进行md5运算，之后与现有数据进行比对，若数据一致则通过。显然md5不可逆，别的地方肯定还有其他验证逻辑。

2. 编写交互脚本

首先用.proto文件生成.h文件和.cc文件--> `protoc -I=./ --cpp_out=./ ./pkg.loginfo.proto`，再用c++语言引用头文件并新建protobuf对象，将对应字段赋值为想要输入的字符串之后调用函数将其序列化为字符串，并输出到标准输出。

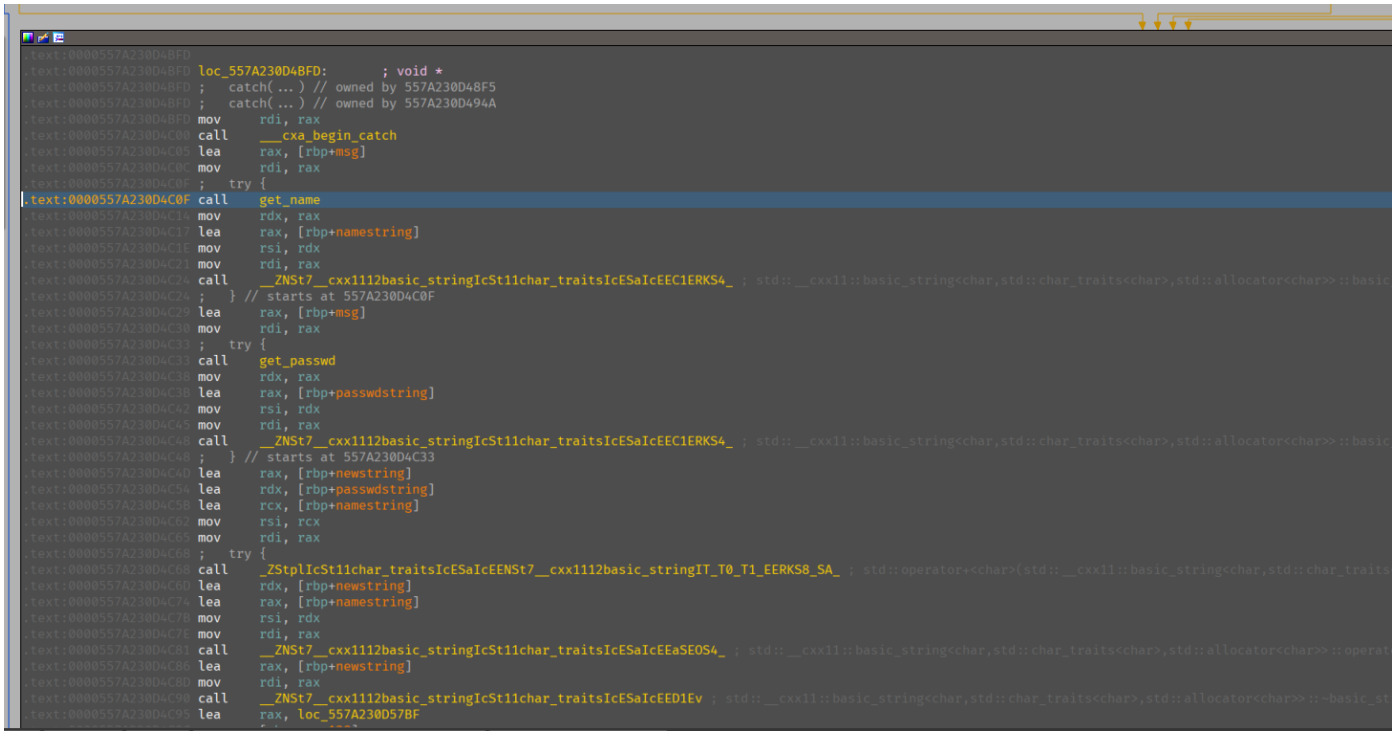
```
#include <iostream>
#include <stdio.h>
#include "pkg.loginfo.pb.h"
#include "pkg.loginfo.pb.cc"
using namespace std;
int main(){
    pkg::loginfo msg1;
    // msg1.set_age(61);
    msg1.set_email("email");
    msg1.set_name("name");
    msg1.set_passwd("passwd");
    string str1;
    msg1.SerializeToString(&str1);
    // int length=str1.length();
    // for(int i=0;i<length;i++){
    //     printf("%#x ",(unsigned char)str1[i]);
    // }
    cout << str1;
}
```

编译之后编写python脚本，调用pwntools运行上面编写的程序，并读取其输出并保存，之后再连接本机的8848端口发送此数据，即可完成交互。

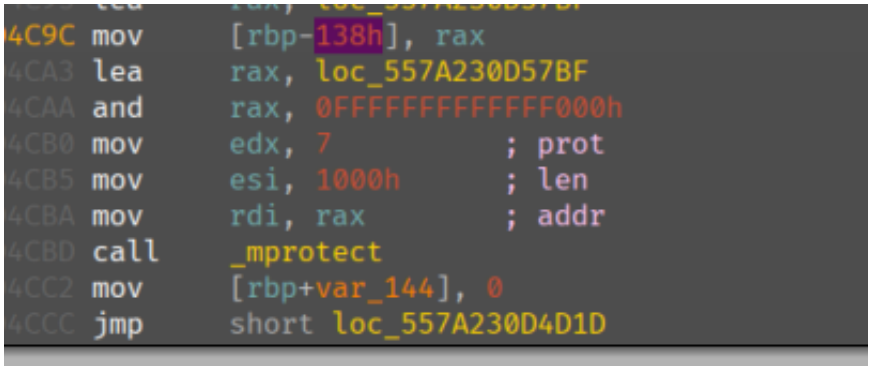
```
from pwn import *
import os
context.log_level="debug"
os.system("g++ writer.cpp -o writer -l protobuf")
sh=process("./writer")
data=sh.recvall()
print(data)
sh.close()
sh=remote("127.0.0.1",8848)
sh.send(data[0:])
sh.interactive()
```

3. 动态调试发现真正逻辑

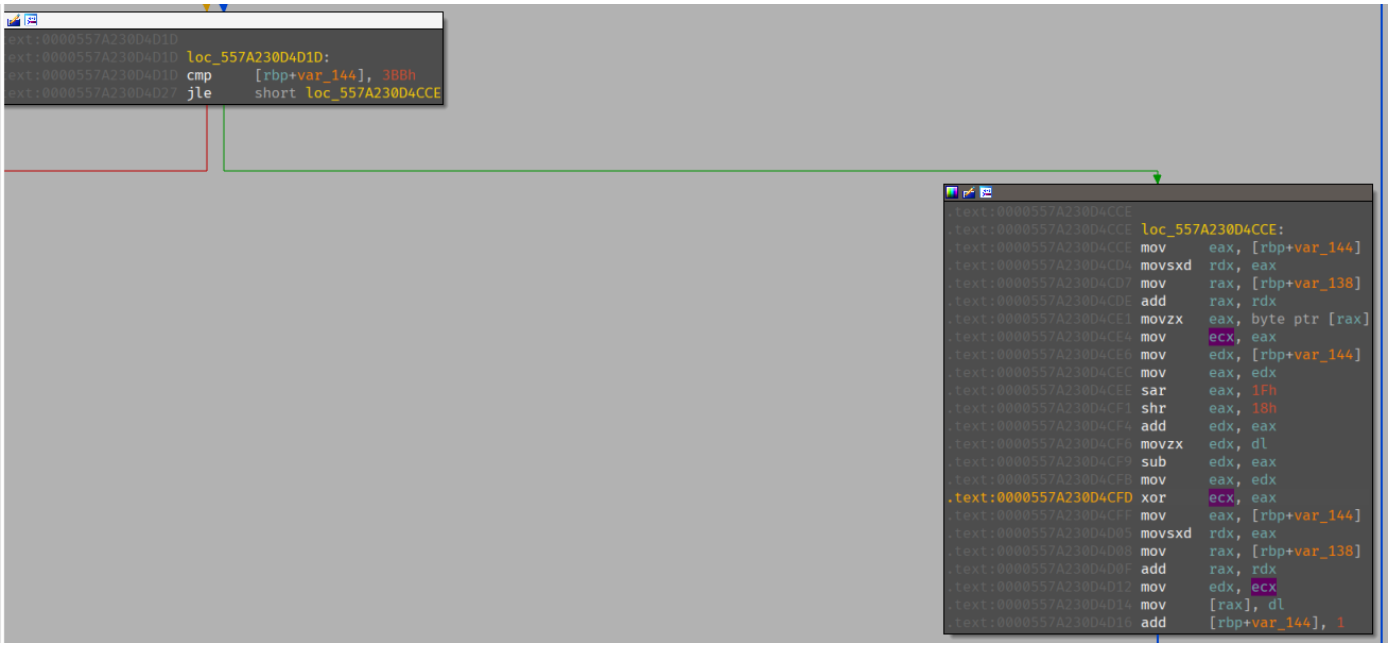
在邮箱验证阶段，在抛出异常的时候打下断点，可以跟踪到后续的catch块中的逻辑，首先取出了name字段和passwd字段，然后将它们结合。



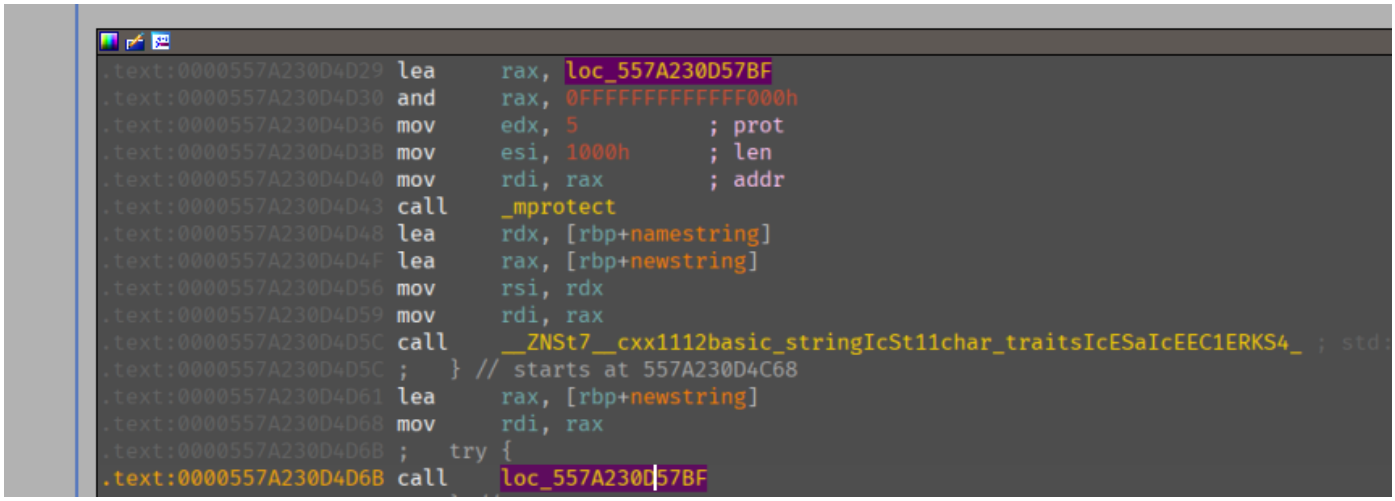

然后调用了mprotect，将.text段的某段地址(记为A)读写权限改为了可读可写可执行。



之后对A+0x3bc的数据进行了异或处理。




之后再次调用mprotect将A读写权限设置为可读可执行。并以name和passwd结合后的字符串为第一个参数，调用A地址的函数。




在动态调试中跟进A，来到真正的逻辑判断函数。

```
v21 = __readfsqword(0x28u);
inputchar = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(input);
length = strlen(inputchar);
v13 = 16;
gentable();
do
{
  sub_557A230D567D(aPmPmXhwyh, byte_557A230DC400, inputchar, length);
  for ( i = 0; i <= 15; ++i )
    aPmPmXhwyh[i] ^= byte_557A230DC400[i];
  for ( j = 0; j <= 3; ++j )
  {
    v9 = aPmPmXhwyh[4 * j];
    v10 = aPmPmXhwyh[4 * j + 1];
    v11 = aPmPmXhwyh[4 * j + 2];
    v12 = aPmPmXhwyh[4 * j + 3];
    v1 = sub_557A230D5077(v9);
    aPmPmXhwyh[4 * j] = v12 ^ v11 ^ v1 ^ v10 ^ sub_557A230D5077(v10);
    v2 = v9 ^ sub_557A230D5077(v10);
    aPmPmXhwyh[4 * j + 1] = v12 ^ v2 ^ v11 ^ sub_557A230D5077(v11);
    v3 = sub_557A230D5077(v11) ^ v10 ^ v9;
    aPmPmXhwyh[4 * j + 2] = v12 ^ sub_557A230D5077(v12) ^ v3;
    v4 = sub_557A230D5077(v9);
    aPmPmXhwyh[4 * j + 3] = sub_557A230D5077(v12) ^ v11 ^ v10 ^ v9 ^ v4;
  }
  for ( k = 0; k <= 15; ++k )
    byte_557A230DC400[k] ^= aPmPmXhwyh[k];
  --v13;
}
while ( v13 );
sub_557A230D567D(aPmPmXhwyh, byte_557A230DC400, inputchar, length);
v20[0] = 0xA7DE8D305A26C6B6LL;
v20[1] = 0xB6E9621D5F6E763DLL;
v20[2] = 0xD3ED57F726097494LL;
v20[3] = 0x475DC94A50A97F96LL;
for ( m = 0; m < (int)length; ++m )
{
  if ( inputchar[m] != *((_BYTE *)v20 + m) )
  {
    v5 = std::operator<<<std::char_traits<char>>(&std::cerr, "faild!!");
    std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
    exit(1);
  }
}
if ( m != 0x20 )
{
  v6 = std::operator<<<std::char_traits<char>>(&std::cerr, "faild!!");
```

gentable()函数对几个全局变量进行异或操作。

seudocode-A

```
unsigned __int64 gentable()
{
  int i; // [rsp+4h] [rbp-2Ch]
  int j; // [rsp+8h] [rbp-28h]
  int k; // [rsp+Ch] [rbp-24h]
  char v4[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  strcpy(v4, "naynaynaynaynayn");
  for ( i = 0; i <= 255; ++i )
    byte_557A230DC2C0[i] ^= v4[i % 16];
  for ( j = 0; j <= 9; ++j )
    dword_557A230DC3C0[j] ^= (unsigned __int8)v4[j];
  for ( k = 0; k <= 15; ++k )
  {
    aPmPmXhwyh[k] ^= v4[k];
    byte_557A230DC400[k] ^= v4[k];
  }
  return v5 - __readfsqword(0x28u);
}
```

步过之后可得这前两个全局变量分别为AES的sbox和rcon，后两个全局变量为两个字符串“1145141919810a”和"qweasdzxcrtyfghv"。查看sbox引用可知sbox被31行的函数中的一个函数调用，即为subbytes函数，合理猜测31行函数为AES加密函数，前两个参数为key和iv，即gentable()函数中解出的两个字符串。

```
__int64 __fastcall AES_CBC(unsigned __int8 *key, unsigned __int8 *iv, __int64 state, int length)
{
  __int64 result; // rax
  int i; // [rsp+24h] [rbp-Ch]
  int j; // [rsp+28h] [rbp-8h]
  int round; // [rsp+2Ch] [rbp-4h]

  if ( (length & 0xF) != 0 )
  {
    printf("length error!!!");
    exit(0);
  }
  round = length / 16;
  KeyExpansion(key, &exkey);
  for ( i = 0; ; ++i )
  {
    result = (unsigned int)i;
    if ( i >= round )
      break;
    xorstate(iv, state);
    AddRoundkey(state, key);
    for ( j = 0; j <= 8; ++j )
    {
      SubBytes((char *)state);
      ShiftRows(state);
      MixColumns(state);
      AddRoundkey(state, (char *)&exkey + 16 * j + 16);
    }
    SubBytes((char *)state);
    ShiftRows(state);
    AddRoundkey(state, &unk_557A230DC820);
    iv = (unsigned __int8 *)state;
    state += 16LL;
```

外层主要逻辑为

```
for i in range(16):
  AES(key,iv,state)
    key^=iv
    MixColumns(key)
    iv^=key
AES(key,iv,state)
```

```
gentable();
do
{
  AES_CBC(key, iv, inputchar, length);
  for ( i = 0; i <= 15; ++i )
    key[i] ^= iv[i];
  for ( j = 0; j <= 3; ++j )
  {
    s1 = key[4 * j];
    s2 = key[4 * j + 1];
    s3 = key[4 * j + 2];
    s4 = key[4 * j + 3];
    v1 = xtime(s1);
    key[4 * j] = s4 ^ s3 ^ v1 ^ s2 ^ xtime(s2);
    v2 = s1 ^ xtime(s2);
    key[4 * j + 1] = s4 ^ v2 ^ s3 ^ xtime(s3);
    v3 = xtime(s3) ^ s2 ^ s1;
    key[4 * j + 2] = s4 ^ xtime(s4) ^ v3;
    v4 = xtime(s1);
    key[4 * j + 3] = xtime(s4) ^ s3 ^ s2 ^ s1 ^ v4;
  }
  for ( k = 0; k <= 15; ++k )
    iv[k] ^= key[k];
  --v13;
}
while ( v13 );
AES_CBC(key, iv, inputchar, length);
```

最后再与现有的数据进行比对，一致即可通过检验。

3. 编写解密脚本

提取出加密数据，key和iv，再用python实现MixColumns即可还原出输入

```python
from Crypto.Cipher import AES
def xtime(a):
    temp=a<<1
    temp=temp%256
    # print(temp)
    if ((a >> 7) & 0x01)==1:
        temp=temp^27
    # print(temp)
    # print("--------")
    return temp
def MixColumns(state):
    state=list(state)
    # print(state)
    for i in range(4):
        s0=state[4*i]
        s1=state[1+4*i]
        s2=state[2+4*i]
        s3=state[3+4*i]
        state[4*i]=xtime(s0) ^ (xtime(s1)^s1) ^ s2 ^ s3
        state[1+4*i]=s0 ^ xtime(s1) ^ (xtime(s2)^s2) ^ s3
        state[2+4*i]=s0 ^ s1 ^ xtime(s2) ^ (xtime(s3)^s3)
```

```
        state[3+4*i]=(xtime(s0)^s0) ^ s1 ^ s2 ^ xtime(s3)
    # print(state)
    return bytes(bytearray(state))
def bytesxor(b1,b2):
    b1=list(b1)
    b2=list(b2)
    for i in range(len(b1)):
        b1[i]=b1[i]^b2[i]
    return bytes(bytearray(b1))
key=b"1145141919810aaa"
iv=b"qweasdzxcrtyfghv"
data=
[182,198,38,90,48,141,222,167,61,118,110,95,29,98,233,182,148,116,9,38,247,87,237,211,1
50,127,169,80,74,201,93,71]
data=bytes(bytearray(data))
keyarray=[]
ivarray=[]
keyarray.append(key)
ivarray.append(iv)
for i in range(16):
    key=bytesxor(key, iv)
    key=MixColumns(key)
    iv=bytesxor(key, iv)
    keyarray.append(key)
    ivarray.append(iv)
keyarray.reverse()
ivarray.reverse()
for i in range(len(keyarray)):
    key=keyarray[i]
    iv=ivarray[i]
    aes=AES.new(key,AES.MODE_CBC,iv)
    data=aes.decrypt(data)
print(data)
```

```
b'adm1n_y000u_pick_the_true_passwd'
```

4. 验证flag

组织输入格式

```cpp
#include <iostream>
#include <stdio.h>
#include "pkg.loginfo.pb.h"
#include "pkg.loginfo.pb.cc"
using namespace std;
int main(){
    pkg::loginfo msg1;
    // msg1.set_age(61);
    msg1.set_email("114514");
    msg1.set_name("adm1n");
    msg1.set_passwd("_y000u_pick_the_true_passwd");
    string str1;
    msg1.SerializeToString(&str1);
    cout << str1;
}
```

运行题目程序&pwntools进行交互

```
❯ python exp.py
[+] Starting local process './writer' argv=[b'./writer'] : pid 61568
[+] Receiving all data: Done (44B)
[DEBUG] Received 0x1 bytes:
    b'\n'
[*] Process './writer' stopped with exit code 0 (pid 61568)
[DEBUG] Received 0x2b bytes:
    00000000  05 61 64 6d  31 6e 12 1b  5f 79 30 30  30 75 5f 70  │·adm│1n··│_y00│0u_p│
    00000010  69 63 6b 5f  74 68 65 5f  74 72 75 65  5f 70 61 73  │ick_│the_│true│_pas│
    00000020  73 77 64 1a  06 31 31 34  35 31 34                  │swd·│·114│514│
    0000002b
b'\n\x05adm1n\x12\x1b_y000u_pick_the_true_passwd\x1a\x06114514'
[+] Opening connection to 127.0.0.1 on port 8848: Done
[DEBUG] Sent 0x2c bytes:
    00000000  0a 05 61 64  6d 31 6e 12  1b 5f 79 30  30 30 75 5f  │··ad│m1n·│·_y0│00u_│
    00000010  70 69 63 6b  5f 74 68 65  5f 74 72 75  65 5f 70 61  │pick│_the│_tru│e_pa│
    00000020  73 73 77 64  1a 06 31 31  34 35 31 34              │sswd│··11│4514│
    0000002c
[*] Switching to interactive mode
$
```

成功验证

```
❯ ./main
waiting......
congratulation!
```