2019KCTF 晋级赛Q1 | 第二题点评及解题思路

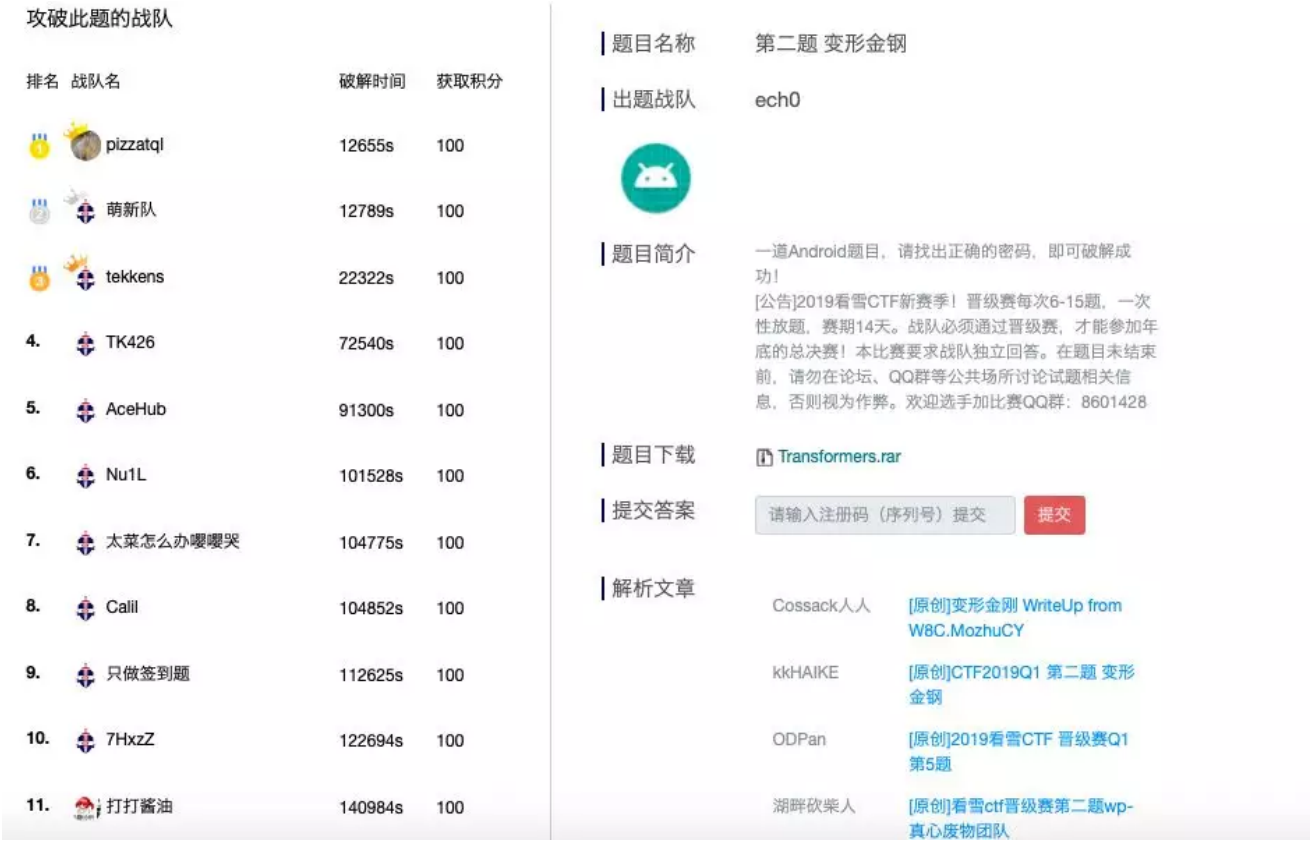
小雪 看雪学院 1周前

transformers，这个80年代一度风靡全球的名词，是否唤起了你热血澎湃的记忆呢？

谈到《变形金刚》，你脑海中出现的是大黄蜂Sam？还是擎天柱？

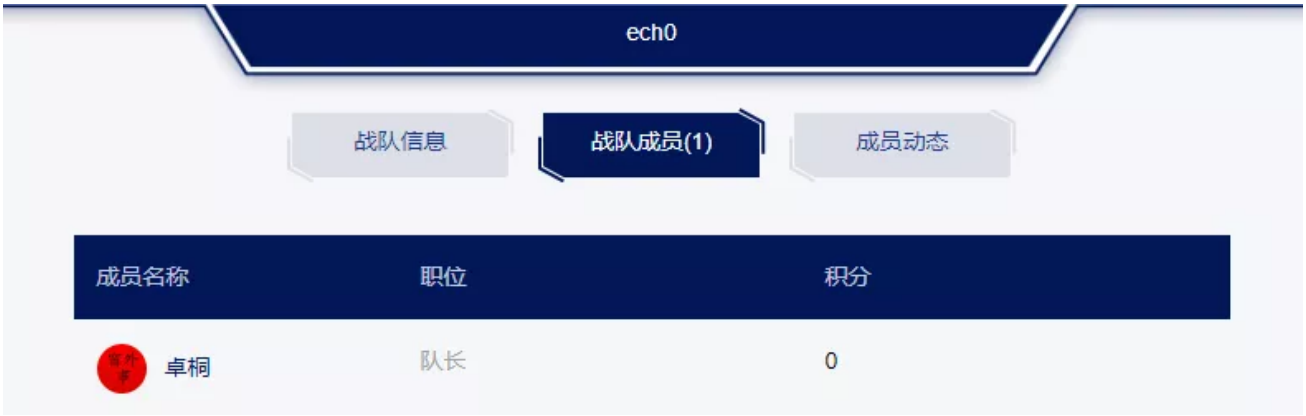
当汽车超人遇到CTF，会迸发出怎样的火花？

接下来，让我们一起来看看
KCTF 晋级赛Q1第二题——《变形金钢》



这道题目仅有**44**支战队破解，围观人数截止目前达到**3506**人。

ech0



战队成员看雪ID：卓桐

个人主页：https://bbs.pediy.com/user-670707.htm

野生的程序员，从移植Android rom到自学Android应用开发，进而深入Android系统源码、Android黑科技，沉迷开发hook框架。

现就职某安全公司主攻移动安全。

《变形金钢》这道题主要涉及两方面的知识点：Android应用基础和最基础的加解密、编码，考验了参赛者的开发、逆向、算法知识，是一道中规中矩的基础题。

主要涉及两方面的知识点。

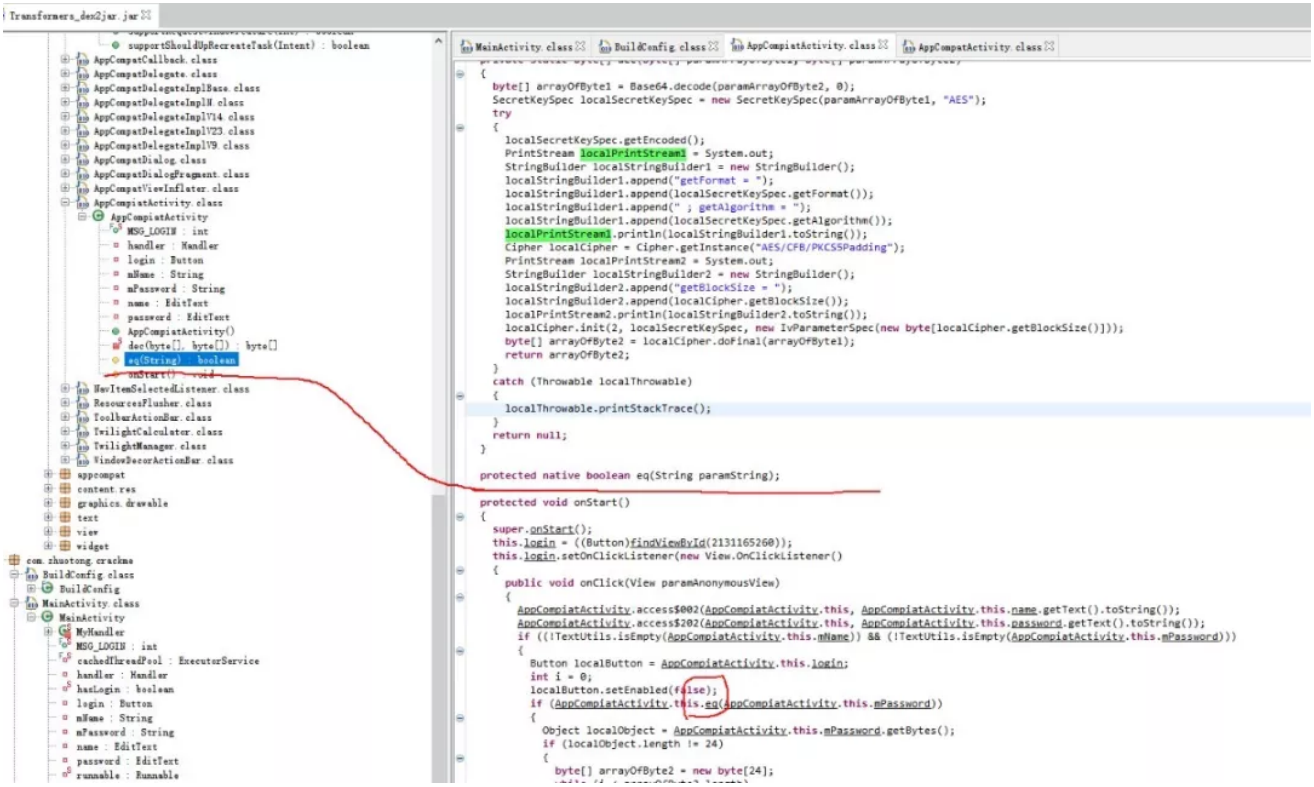
1、Android应用基础，Activity的方法和调用顺序。因为onStart在onCreate之后执行，所以重写了父类的onStart方法，并在onStart方法中覆盖了onCreate的逻辑，进而最后生效的是onStart内的逻辑。

2、最常用、基础的加解密、编码。选择了Rc4加密算法，并在加密完一个字节后对该字节进行变形的base64编码。解题的思路就是写出变形base64的解码代码，再Rc4解密，即可得出答案。

综上，这是一道基础题，旨在巩固基础的开发、逆向、算法知识。

这道题目破解思路由看雪ID：**HHHso** 提供

比较快速的方式是拿到中间运输的核心表，然后使用核心表拟合内部逻辑，相对来说比较省事，关键是避免全逻辑拟合的纰漏。
0x00 一切的关键在于native化的eq()函数



0x01 在核心Transformers\lib\armeabi-v7a\liboo000oo.so 中
导出函数datadiv_decode5009363700628197108 会通过简单的异或解密处 eq()函数的native化信息

```
char*datadiv_decode5009363700628197108()
{
  int v0;// r0
  int v1;// r0
  int v2;// r0
  char*result;// r0

  v0=0;
  do
  {
    gidstr[v0]^=0xA5u;
    ++v0;
  }
  while(v0!=37);
  v1=0;
  do
    aAbcdefghijklmn[v1++]^=0xA5u;
  while(v1!=66);
  v2=0;
  do
    aAndroidSupport[v2++]^=0x84u;
  while(v2!=42);
  result=aLjavaLangStrin;
  aEq[0]^=0xFCu;
  aEq[1]^=0xFCu;
  aEq[2]^=0xFCu;
  aLjavaLangStrin[0]^=0x62u;
  aLjavaLangStrin[1]^=0x62u;
  aLjavaLangStrin[2]^=0x62u;
  aLjavaLangStrin[3]^=0x62u;
  aLjavaLangStrin[4]^=0x62u;
  aLjavaLangStrin[5]^=0x62u;
  *(_WORD*)&aLjavaLangStrin[6]=__PAIR__(aLjavaLangStrin[7],
  aLjavaLangStrin[8]^=0x62u;
  aLjavaLangStrin[9]^=0x62u;
  aLjavaLangStrin[10]^=0x62u;
  aLjavaLangStrin[11]^=0x62u;
  aLjavaLangStrin[12]^=0x62u;
  aLjavaLangStrin[13]^=0x62u;
  aLjavaLangStrin[14]^=0x62u;
  aLjavaLangStrin[15]^=0x62u;
  aLjavaLangStrin[16]^=0x62u;
  aLjavaLangStrin[17]^=0x62u;
  aLjavaLangStrin[18]^=0x62u;
  aLjavaLangStrin[19]^=0x62u;
```

0x02 解密后得到如图，于是我们可以定位到对应eq()函数的本地代码函数 A8DEE784 Hi_eq_sub_A8DEE784

（注意：后面的+1，无伤大雅，忽略就行，这是由于ARM的字对齐会忽略最低bit）

```
.8DF2004                ALIGN 0x10
.8DF2010 off_A8DF2010   DCD aAndroidSupport     ; DATA XREF: JNI_OnLoad+4A↑o
.8DF2010                                        ; JNI_OnLoad+4C↑r ...
.8DF2010                                        ; "android/support/v7/app/AppCompiatActivi"...
.8DF2014 off_A8DF2014   DCD aEq                 ; DATA XREF: JNI_OnLoad+78↑o
.8DF2014                                        ; .text:off_A8DEEB48↑o
.8DF2014                                        ; "eq"
.8DF2018                DCD aLjavaLangStrin     ; "(Ljava/lang/String;)Z"
.8DF201C                DCD sub_A8DEE784+1
.8DF2020 ; char gidstr[48]
```

eq()函数伪码相对有点绕，不过我们只关注如何使用key，这时会发现前半部分都是由

unsigned char g_gidstr[] = "650f909c-7217-3647-9331-c82df8b98e98";
char vs16[] = "dbeafc2409715836";

经过多次变换得到ref100，然后才使用key参与运算

```
int __fastcall sub_A8DEE784(int a1)
{
size_t guidlen;// r10
unsigned __int8*gidbuf1_reverse_gid;// r6
_BYTE*gidbuf2_nosplit;// r8
_BYTE*gidbuf3;// r11
signed int gidbuf2_nosplit_len;// r0
size_t v6;// r2
char*v7;// r1
int v8;// r3
int v9;// r1
unsigned int v10;// r2
int v11;// r3
int v12;// r0
int v13;// r4
unsigned __int8 v14;// r0
_BYTE*v15;// r3
char*v16;// r5
char*pbuf2;// r4
int i;// r5
int v19;// r1
int ix;// r0
signed int v21;// r1
int v22;// r2
size_t pwdlen;// r0
unsigned int loc_pwdlen;// r8
unsigned int b64_salt_size;// r5
_BYTE*pwdsaltstr;// r0
int v27;// r3
int txi;// r10
unsigned int pwd_idx;// r2
int txj;// r12
bool b31;// zf
_BYTE*v32;// r1
bool b32;// zf
int v34;// r3
int tx;// r1
unsigned __int8 txijvx;// r11
unsigned int v37;// lr
```

```
char v38;// r1
char*v39;// r2
int v40;// t1
unsigned int pwd_salt_b64_bytelen;// [sp+4h] [bp-234h]
unsigned int pwd_b64_bitlen;// [sp+8h] [bp-230h]
unsigned int v44;// [sp+10h] [bp-228h]
char*pwd;// [sp+14h] [bp-224h]
char loc_buf100[256];// [sp+18h] [bp-220h]
char buf100_2[256];// [sp+118h] [bp-120h]
int v48;// [sp+218h] [bp-20h]

pwd=(char*)(*(int(**)(void))(*(_DWORD*)a1+676))();
guidlen=strlen(gidstr);
gidbuf1_reverse_gid=(unsigned __int8*)malloc(guidlen);
gidbuf2_nosplit=malloc(guidlen);
gidbuf3=malloc(guidlen);
_aeabi_memclr(gidbuf1_reverse_gid,guidlen);
_aeabi_memclr(gidbuf2_nosplit,guidlen);
_aeabi_memclr(gidbuf3,guidlen);
if(guidlen)
{
gidbuf2_nosplit_len=0;
v6=guidlen;
v7=gidstr;
do
{
v8=(unsigned __int8)*v7++;
if(v8!='-' )
gidbuf2_nosplit[gidbuf2_nosplit_len++]=v8;
--v6;
}
while(v6);
if(gidbuf2_nosplit_len>=1)
{
v9=gidbuf2_nosplit_len-1;
v10=-8;
v11=0;
v12=0;
do
{
if((v11|(v10>>2))>3)
{
v13=v12;
}
else
```

```
{
v13=v12+1;
gidbuf1_reverse_gid[v12]='-';
}
v14=gidbuf2_nosplit[v9--];
v11+=0x40000000;
gidbuf1_reverse_gid[v13]=v14;
++v10;
v12=v13+1;
}
while(v9!=-1);
if(v13>=0)
{
v15=gidbuf3;
while(1)
{
v16=(char*)*gidbuf1_reverse_gid;
if((unsigned __int8)((_BYTE)v16-'a') <= 5u )
break;
if((unsigned __int8)((_BYTE)v16-'0') <= 9u )
{
v16=&aDbeafc24097158[(_DWORD)v16-42];
goto LABEL_18;
}
LABEL_19:
*v15++=(_BYTE)v16;
--v12;
++gidbuf1_reverse_gid;
if(!v12)
goto LABEL_20;
}
v16=&aDbeafc24097158[(_DWORD)v16-'a'];
LABEL_18:
LOBYTE(v16)=*v16;
goto LABEL_19;
}
}
}
LABEL_20:
_aeabi_memcpy8(loc_buf100,Hi_gbuf100,256);
pbuf2=buf100_2;
i=0;
do
{
Hi_idx_mod_len(i,guidlen);
```

```
buf100_2[i++]=gidbuf3[v19];
}
while(i!=256);
ix=(unsigned __int8)(buf100_2[0]-41);
loc_buf100[0]=loc_buf100[ix];
loc_buf100[ix]=-41;
v21=1;
do
{
v22=(unsigned __int8)loc_buf100[v21];
ix=(ix+(unsigned __int8)buf100_2[v21]+v22)%256;
loc_buf100[v21++]=loc_buf100[ix];
loc_buf100[ix]=v22;
}
while(v21!=256);
pwdlen=strlen(pwd);
loc_pwdlen=pwdlen;
b64_salt_size=(unsigned __int8)gidbuf3[3];
pwd_b64_bitlen=8*(3--3*(pwdlen/3));
pwd_salt_b64_bytelen=b64_salt_size+pwd_b64_bitlen/6;
pwdsaltstr=malloc(pwd_salt_b64_bytelen+1);
if(loc_pwdlen)
{
txi=0;
pwd_idx=0;
txj=0;
v44=b64_salt_size;
do
{
txi=(txi+1)%256;
tx=(unsigned __int8)loc_buf100[txi];
txj=(txj+tx)%256;
loc_buf100[txi]=loc_buf100[txj];
loc_buf100[txj]=tx;
pbuf2=(char*)(unsigned __int8)loc_buf100[txi];
txijvx=loc_buf100[(unsigned __int8)(tx+(_BYTE)pbuf2)]^pwd[pwd_idx];
if(pwd_idx&&(v27=0xAAAAAAAB*(unsigned __int64)pwd_idx>>32,v37=3*(pwd_idx/3),v37!=pwd_id
{
b31=pwd_idx==1;
if(pwd_idx!=1)
b31=v37+1==pwd_idx;
if(b31)
{
v32=aAbcdefghijklmn;
pwdsaltstr[v44+pwd_idx]=aAbcdefghijklmn[(unsigned __int8)pwdsaltstr[v44+pwd_idx]|((unsi
```

```
pbuf2=&pwdsaltstr[v44+pwd_idx];
v27=4*txijvx&0x3C;
pbuf2[1]=v27;
if(pwd_idx+1>=loc_pwdlen)
goto LABEL_53;
}
else
{
b32=pwd_idx==2;
if(pwd_idx!=2)
b32=v37+2==pwd_idx;
if(b32)
{
pbuf2=(char*)(txijvx&0xC0);
v34=v44+++pwd_idx;
pwdsaltstr[v34]=aAbcdefghijklmn[(unsigned __int8)pwdsaltstr[v34]|((unsigned int)pbuf2>>
v27=(int)&pwdsaltstr[v34];
*(_BYTE*)(v27+1)=aAbcdefghijklmn[txijvx&0x3F];
}
}
}
else
{
pwdsaltstr[v44+pwd_idx]=aAbcdefghijklmn[(unsigned int)txijvx>>2]^7;
pbuf2=&pwdsaltstr[v44+pwd_idx];
v27=16*txijvx&0x30;
pbuf2[1]=v27;
if(pwd_idx+1>=loc_pwdlen)
{
v38=aAbcdefghijklmn[v27];
*((_WORD*)pbuf2+1)=';;';
goto LABEL_43;
}
}
++pwd_idx;
}
while(pwd_idx<loc_pwdlen);
}
while(1)
{
if(pwd_b64_bitlen)
{
v32=(_BYTE*)1;
pbuf2=(char*)pwd_salt_b64_bytelen;
v39=&Hi_cmp_byte_A8DF04E8;
```

```
do
{
v27=(unsigned __int8)pwdsaltstr[b64_salt_size++];
v40=(unsigned __int8)*v39++;
if(v40!=v27)
v32=0u;
}
while(b64_salt_size<pwd_salt_b64_bytelen);
}
else
{
v32=(_BYTE*)1;
}
pwdsaltstr=(_BYTE*)(_stack_chk_guard-v48);
if(_stack_chk_guard==v48)
break;
LABEL_53:
v38=v32[v27];
pbuf2[2]='4';
LABEL_43:
pbuf2[1]=v38;
}
return(unsigned __int8)v32;
}
```

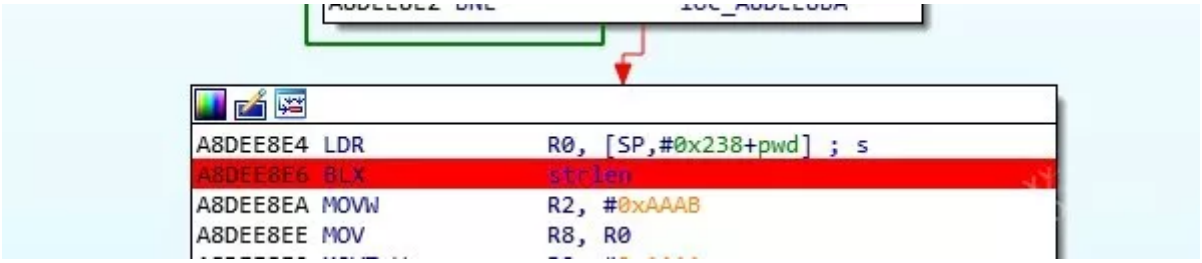
0x03 代码相对比较清晰，由下述横线分为前后两个部分：

（1）上半部分(主要得到loc_buf100表，这个我们可以通过下述断点断下后，通过IDAPython脚本提取buf存储的表；

（2）下半部分就是依次通过(1)表提取因子A与key的字符B异或，得到结果R，而将结果转为类似64进制编码结果。

这里其对64编码结果加了盐，即对特地位异或了不同因子。

最后与Hi_cmp_byte_A8DF04E8 即 xb64_pwd = " {9*8ga*l!Tn?@#fj'j$\\g;;"比较。




```
133 ix=(unsigned __int8)(buf100_2[0]-41);
134 loc_buf100[0]=loc_buf100[ix];
135 loc_buf100[ix]=-41;
136 v21=1;
137 do
138 {
139   v22=(unsigned __int8)loc_buf100[v21];
140   ix=(ix+(unsigned __int8)buf100_2[v21]+v22)%256;
141   loc_buf100[v21++]=loc_buf100[ix];
142   loc_buf100[ix]=v22;
143 }
144 while(v21!=256);
145 pwdlen=strlen(pwd);
146 loc_pwdlen=pwdlen;
147 b64_salt_size=(unsigned __int8)gidbuf3[3];
148 pwd_b64_bitlen=8*(3--3*(pwdlen/3));
149 pwd_salt_b64_bytelen=b64_salt_size+pwd_b64_bitlen/6;
150 pwdsaltstr=malloc(pwd_salt_b64_bytelen+1);
151 if(loc_pwdlen)
152 {
153   txi=0;
154   pwd_idx=0;
155   txj=0;
156   v44=b64_salt_size;
157   do
158   {
159     txi=(txi+1)%256;
160     tx=(unsigned __int8)loc_buf100[txi];
161     txj=(txj+tx)%256;
162     loc_buf100[txi]=loc_buf100[txj];
163     loc_buf100[txj]=tx;
164     pbuf2=(char*)(unsigned __int8)loc_buf100[txi];
165     txijvx=loc_buf100[(unsigned __int8)(tx+(_BYTE)pbuf2)]^pwd[pwd_idx];
166     if(pwd_idx&&(v27=0xAAAAAAAB*(unsigned __int64)pwd_idx>>32,v37=3*(pwd_idx/3),v37!=pwd_idx))
167     {
168       b31=pwd_idx==1;
169       if(pwd_idx!=1)
170         b31=v37+1==pwd_idx;
171       if(b31)
172       {
173         v32=aAbcdefghijklmn;
174         pwdsaltstr[v44+pwd_idx]=aAbcdefghijklmn[(unsigned __int8)pwdsaltstr[v44+pwd_idx]|((unsigned int)txijvx>>4)];
175         pbuf2=&pwdsaltstr[v44+pwd_idx];
176         v27=4*txijvx&0x3C;
```

0x04 因为关键的都是异或，所以可逆，通过断点处的提取，我们得到初使表，此表没选用一个异或因子后，都会发生变动。

memcpy(&loc_buf100[0],&g_buf100[0],0x100);

```
unsigned char ref100[] = {
0xF0, 0x37, 0xE1, 0x9B, 0x2A, 0x15, 0x17, 0x9F, 0xD7, 0x58, 0x4D, 0x6E, 0x33, 0xA0, 0x3
0x04, 0xD0, 0xBE, 0xED, 0xF8, 0x66, 0x5E, 0x00, 0xD6, 0x91, 0x2F, 0xC3, 0x10, 0x4C, 0xF
0xC1, 0xEC, 0x6D, 0x0B, 0x50, 0x65, 0xBB, 0x34, 0xFA, 0xA4, 0x2D, 0x3B, 0x23, 0xA1, 0x9
0x1D, 0x38, 0x56, 0x0A, 0x5D, 0x4F, 0xE4, 0xCC, 0x24, 0x0D, 0x12, 0x87, 0x35, 0x85, 0x8
0xC6, 0x13, 0x9A, 0xD3, 0xFC, 0xE7, 0x08, 0xAC, 0xB7, 0xE9, 0xB0, 0xE8, 0x41, 0xAA, 0x5
0xC2, 0x42, 0xBC, 0xE6, 0x0F, 0x8A, 0x86, 0xA8, 0xCF, 0x84, 0xC5, 0x48, 0x74, 0x36, 0x0
0x88, 0x51, 0xF6, 0x7F, 0x57, 0x05, 0x63, 0x3E, 0xFE, 0xB8, 0xC9, 0xF5, 0xAF, 0xDF, 0xE
0x44, 0xF9, 0xCD, 0x06, 0xBA, 0x30, 0x47, 0x40, 0xDE, 0xFD, 0x1C, 0x7C, 0x11, 0x5C, 0x0
0x2C, 0x9C, 0x5F, 0x46, 0x27, 0xC4, 0x83, 0x73, 0x16, 0x90, 0x20, 0x76, 0x7B, 0xF2, 0xE
```

```
0x77, 0x52, 0x80, 0x25, 0x09, 0x26, 0x3F, 0xC7, 0x18, 0x1B, 0xA3, 0xFF, 0xFB, 0xCB, 0xA
0x54, 0x7A, 0x68, 0xB4, 0x70, 0x4B, 0xE2, 0x49, 0x22, 0x7E, 0xA5, 0xB6, 0x81, 0x9D, 0x4
0xF1, 0xA7, 0x3C, 0xD9, 0x94, 0xEF, 0x32, 0x6B, 0x1F, 0xB1, 0x60, 0xB9, 0x64, 0x59, 0x0
0x7D, 0xE0, 0x6C, 0xAD, 0x97, 0x19, 0xB5, 0x3A, 0xF4, 0xD8, 0x8D, 0x98, 0x03, 0x93, 0x1
0x1E, 0x4A, 0xC0, 0x5A, 0xE5, 0xD1, 0x3D, 0x14, 0xC8, 0x79, 0xBD, 0x43, 0xDB, 0x69, 0xD
0x95, 0x9E, 0x21, 0x45, 0x89, 0x2B, 0xAB, 0x29, 0xA2, 0x8B, 0x2E, 0xD4, 0x0E, 0x62, 0xC
0xDA, 0x5B, 0x72, 0x8F, 0x99, 0x75, 0xEE, 0x78, 0x0C, 0x71, 0xBF, 0xDD, 0xCE, 0x92, 0x6
};
```

有了表，通过简单运算，我们就可以得到key

基本逻辑是我们先值得得到由表的异或因子与key异或的b64编码加密前的结果b64bin

通过b64bin的字节依次与表产生的异或因子异或，即可得到key

```
txi=0;
i=0;
txj=0;
//v44=b64_salt_size;
do
{
txi=(txi+1)%256;
tx=(unsigned char)loc_buf100[txi];
txj=(txj+tx)%256;
loc_buf100[txi]=loc_buf100[txj];
loc_buf100[txj]=tx;
pwdch=loc_buf100[(unsigned char)(tx+loc_buf100[txi])]^b64bin[i];
gpwd[i] = pwdch;
++i;
}
while(i<16);
printf("----------------------------------------\n");
txi=0;
i=0;
txj=0;
//v44=b64_salt_size;
do
{
txi=(txi+1)%256;
tx=(unsigned char)ref100[txi];
txj=(txj+tx)%256;
ref100[txi]=ref100[txj];
ref100[txj]=tx;
```

```
printf("0x%02X, ",ref100[(unsigned char)(tx+ref100[txi])]);
pwdch=ref100[(unsigned char)(tx+ref100[txi])]^b64bin[i];
gpwd[i] = pwdch;
++i;
}
while(i<20);
printf("gidbuf3 len: %d\tbytes:[%s]\n",16,gpwd);
```

0x05 如何得到b64bin

这里先直接给出

```
unsigned char b64bin[16] = {0xFD, 0x1E, 0x8A, 0x4E, 0x09, 0xCA, 0x90, 0x03, 0xE7, 0xF1,
unsigned char gpwd[20]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

因为b64bin经过64编码后再局部加密得到 xb64_pwd = " {9*8ga*l!Tn?@#fj'j$\\g;;"

```
xb64_pwd = " {9*8ga*l!Tn?@#fj'j$"#||g;;"
b64str = ''
for i in range(0,xb64_pwd.__len__()):
if i%4 == 0:
b64_ch = chr(ord(xb64_pwd[i])^0x07)
elif i%4 == 2:
b64_ch = chr(ord(xb64_pwd[i])^0x0F)
else:
b64_ch = xb64_pwd[i]
b64str+=b64_ch

print b64str.__repr__()
"'{6*?gn*k![n8@,fm'e$^"
```

我们得到b64局部加密后的结果，再b64解密得到

```
b64 = r"!:#$%&()+-*/`~_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\'"
b64str = "'{6*?gn*k![n8@,fm'e$"
b64bin = ''
for i in range(0,b64str.__len__(),4):
b3 = b64str[i:i+4]
b0 = ((b64.index(b3[0])<<2)|(b64.index(b3[1])>>4))&0xFF
b1 = ((b64.index(b3[1])<<4)|(b64.index(b3[2])>>2))&0xFF
b2 = (((b64.index(b3[2])&0x3)<<6)|b64.index(b3[3]))&0xFF
```

```
b64bin += ''.join([chr(c) for c in [b0,b1,b2]])

print b64bin.__repr__()
'\xfd\x1e\x8aN\t\xca\x90\x03\xe7\xf1\x85\x9f\x9b\xf7\x83'
```

由于其编码的特殊情形，其最后的"\\g;;"直接解码得到

```
0x3E = b64.index('\\'))|b64.index('g')
```

于是有了b64bin,完整代码如下，通过 cl.exe 直接编译，执行得到key，其中前半部分许多位测试过程中国使用的中间拟合逻辑。

由于前半部分拟合出了纰漏，最终直接选中提取其中间运算过程中现成的表来使用，大大简化分析。

```
//test_main.cpp
//to compile: cl.exe test_main.cpp
//to run: test_main.exe

#include <windows.h>
#include <stdio.h>

unsigned char g_gidstr[] = "650f909c-7217-3647-9331-c82df8b98e98";
char vs16[] = "dbeafc2409715836";
unsigned char g_buf100[] = {
0xD7, 0xDF, 0x02, 0xD4, 0xFE, 0x6F, 0x53, 0x3C, 0x25, 0x6C, 0x99, 0x97, 0x06, 0x56, 0x8
0x40, 0x11, 0x64, 0x07, 0x36, 0x15, 0x70, 0xCA, 0x18, 0x17, 0x7D, 0x6A, 0xDB, 0x13, 0x3
0x29, 0x60, 0xE1, 0x23, 0x28, 0x8A, 0x50, 0x8C, 0xAC, 0x2F, 0x88, 0x20, 0x27, 0x0F, 0x7
0xA2, 0xAB, 0xFC, 0xA1, 0xCC, 0x21, 0x14, 0x1F, 0xC2, 0xB2, 0x8B, 0x2C, 0xB0, 0x3A, 0x6
0x3D, 0xBB, 0x42, 0xA5, 0x0C, 0x75, 0x22, 0xD8, 0xC3, 0x76, 0x1E, 0x83, 0x74, 0xF0, 0xF
0x26, 0xD1, 0x4F, 0x0B, 0xFF, 0x4C, 0x4D, 0xC1, 0x87, 0x03, 0x5A, 0xEE, 0xA4, 0x5D, 0x9
0xC8, 0x0D, 0x62, 0x63, 0x3E, 0x44, 0x7B, 0xA3, 0x68, 0x32, 0x1B, 0xAA, 0x2D, 0x05, 0xF
0x16, 0x61, 0x94, 0xE0, 0xD0, 0xD3, 0x98, 0x69, 0x78, 0xE9, 0x0A, 0x65, 0x91, 0x8E, 0x3
0x7A, 0x51, 0x86, 0x10, 0x3F, 0x7F, 0x82, 0xDD, 0xB5, 0x1A, 0x95, 0xE7, 0x43, 0xFD, 0x9
0x45, 0xEF, 0x92, 0x5C, 0xE4, 0x96, 0xA9, 0x9C, 0x55, 0x89, 0x9A, 0xEA, 0xF9, 0x90, 0x5
0x04, 0x84, 0xCF, 0x67, 0x93, 0x00, 0xA6, 0x39, 0xA8, 0x4E, 0x59, 0x31, 0x6B, 0xAD, 0x5
0x77, 0xB1, 0x54, 0xDC, 0x38, 0x41, 0xB6, 0x47, 0x9F, 0x73, 0xBA, 0xF8, 0xAE, 0xC4, 0xB
0x01, 0x4B, 0x2A, 0x8D, 0xBD, 0xC5, 0xC6, 0xE8, 0xAF, 0xC9, 0xF5, 0xCB, 0xFB, 0xCD, 0x7
0x12, 0x71, 0xD2, 0xFA, 0x09, 0xD5, 0xBC, 0x58, 0x19, 0x80, 0xDA, 0x49, 0x1D, 0xE6, 0x2
0x7E, 0xB7, 0x3B, 0xB3, 0xA0, 0xB9, 0xE5, 0x57, 0x6E, 0xD9, 0x08, 0xEB, 0xC7, 0xED, 0x8
0xF2, 0xBF, 0xC0, 0xA7, 0x4A, 0xD6, 0x2B, 0xB4, 0x72, 0x9D, 0x0E, 0x6D, 0xEC, 0x48, 0xE
};
unsigned char ref100[] = {
```

```
0xF0, 0x37, 0xE1, 0x9B, 0x2A, 0x15, 0x17, 0x9F, 0xD7, 0x58, 0x4D, 0x6E, 0x33, 0xA0, 0x3
0x04, 0xD0, 0xBE, 0xED, 0xF8, 0x66, 0x5E, 0x00, 0xD6, 0x91, 0x2F, 0xC3, 0x10, 0x4C, 0xF
0xC1, 0xEC, 0x6D, 0x0B, 0x50, 0x65, 0xBB, 0x34, 0xFA, 0xA4, 0x2D, 0x3B, 0x23, 0xA1, 0x9
0x1D, 0x38, 0x56, 0x0A, 0x5D, 0x4F, 0xE4, 0xCC, 0x24, 0x0D, 0x12, 0x87, 0x35, 0x85, 0x8
0xC6, 0x13, 0x9A, 0xD3, 0xFC, 0xE7, 0x08, 0xAC, 0xB7, 0xE9, 0xB0, 0xE8, 0x41, 0xAA, 0x5
0xC2, 0x42, 0xBC, 0xE6, 0x0F, 0x8A, 0x86, 0xA8, 0xCF, 0x84, 0xC5, 0x48, 0x74, 0x36, 0x0
0x88, 0x51, 0xF6, 0x7F, 0x57, 0x05, 0x63, 0x3E, 0xFE, 0xB8, 0xC9, 0xF5, 0xAF, 0xDF, 0xE
0x44, 0xF9, 0xCD, 0x06, 0xBA, 0x30, 0x47, 0x40, 0xDE, 0xFD, 0x1C, 0x7C, 0x11, 0x5C, 0x0
0x2C, 0x9C, 0x5F, 0x46, 0x27, 0xC4, 0x83, 0x73, 0x16, 0x90, 0x20, 0x76, 0x7B, 0xF2, 0xE
0x77, 0x52, 0x80, 0x25, 0x09, 0x26, 0x3F, 0xC7, 0x18, 0x1B, 0xA3, 0xFF, 0xFB, 0xCB, 0xA
0x54, 0x7A, 0x68, 0xB4, 0x70, 0x4B, 0xE2, 0x49, 0x22, 0x7E, 0xA5, 0xB6, 0x81, 0x9D, 0x4
0xF1, 0xA7, 0x3C, 0xD9, 0x94, 0xEF, 0x32, 0x6B, 0x1F, 0xB1, 0x60, 0xB9, 0x64, 0x59, 0x0
0x7D, 0xE0, 0x6C, 0xAD, 0x97, 0x19, 0xB5, 0x3A, 0xF4, 0xD8, 0x8D, 0x98, 0x03, 0x93, 0x1
0x1E, 0x4A, 0xC0, 0x5A, 0xE5, 0xD1, 0x3D, 0x14, 0xC8, 0x79, 0xBD, 0x43, 0xDB, 0x69, 0xD
0x95, 0x9E, 0x21, 0x45, 0x89, 0x2B, 0xAB, 0x29, 0xA2, 0x8B, 0x2E, 0xD4, 0x0E, 0x62, 0xC
0xDA, 0x5B, 0x72, 0x8F, 0x99, 0x75, 0xEE, 0x78, 0x0C, 0x71, 0xBF, 0xDD, 0xCE, 0x92, 0x6
};

//unsigned char b64bin[18] = {0xF9, 0x1E, 0x8A, 0x4E, 0x09, 0xCA, 0x90, 0x03, 0xE7, 0xF
//unsigned char b64bin[16] = {0xF9, 0x1E, 0x8A, 0x4E, 0x09, 0xCA, 0x90, 0x03, 0xE7, 0xF
unsigned char b64bin[16] = {0xFD, 0x1E, 0x8A, 0x4E, 0x09, 0xCA, 0x90, 0x03, 0xE7, 0xF1,

unsigned char gpwd[20]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void test1(){
unsigned char *gidstr = &g_gidstr[0];
unsigned char *gidbuf1 = NULL;
unsigned char *gidbuf2_nosplit = NULL;
unsigned char *gidbuf3 = NULL;
unsigned char *v7,v14,*v15,pwdch,tx;
char * v16,*pbuf2;
int guidlen,gidbuf2_nosplit_len,v6,v8,v9,v11,v12,v13,ix,v21,v22,txi,txj;
unsigned int v1,i;
char loc_buf100[0x100];
char buf100_2[0x100];
guidlen = strlen((const char*)gidstr);
gidbuf1 = (unsigned char *)malloc(guidlen);
gidbuf2_nosplit = (unsigned char *)malloc(guidlen);
gidbuf3 = (unsigned char *)malloc(guidlen);
memset(gidbuf1,0,guidlen);
memset(gidbuf2_nosplit,0,guidlen);
memset(gidbuf3,0,guidlen);

if(guidlen){
gidbuf2_nosplit_len=0;
```

```
v6=guidlen;
v7=gidstr;
do
{
v8=(unsigned char)*v7++;
if(v8!='-' )
gidbuf2_nosplit[gidbuf2_nosplit_len++]=v8;
--v6;
}
while(v6);
}
printf("gidbuf2 len: %d\tbytes:[%s]\n",gidbuf2_nosplit_len,gidbuf2_nosplit);
//gidbuf2 len: 32 bytes:[650f909c721736479331c82df8b98e98]
if(gidbuf2_nosplit_len > 1){
v9=gidbuf2_nosplit_len-1;
v1=-8;
v11=0;
v12=0;
do{
if((v11|(v1>>2))>3){
v13=v12;
}
else{
v13=v12+1;
gidbuf1[v12]='-';
}
v14=gidbuf2_nosplit[v9--];
v11+=0x40000000;
gidbuf1[v13]=v14;
++v1;
v12=v13+1;
}while(v9!=-1);
}
printf("gidbuf1 len: %d %d\tbytes:[%s]\n",v13,v12,gidbuf1);
//gidbuf1 len: 35 bytes:[89e89b8f-d28c-1339-7463-7127c909f056]
if(v13>=0){
v15=gidbuf3;
do{
v16=(char*)gidbuf1;
if((unsigned char)(*v16-'a') <= 5u ){
v16 = &vs16[*v16-'a'];
}else if((unsigned char)(*v16-'0') <= 9u ){
v16=&vs16[*v16-'0'];
}
*v15=*v16;
```

```
v15++;
--v12;
++gidbuf1;
}while(v12!=0);
}
printf("gidbuf3 len: %d\tbytes:[%s]\n",v15-gidbuf3,gidbuf3);
//gidbuf3 len: 36 bytes:[09f09b0c-ae0e-baa9-4f2a-4be4e9d9cdc2]
memcpy(&loc_buf100[0],&g_buf100[0],0x100);
pbuf2=&buf100_2[0];
i=0;
do{
buf100_2[i++]=gidbuf3[i%guidlen];
}
while(i!=256);
//
ix=(unsigned char)(buf100_2[0]-41);
loc_buf100[0]=loc_buf100[ix];
loc_buf100[ix]=0xd7;//-41;
v21=1;
do
{
v22=(unsigned char)loc_buf100[v21];
ix=(ix+(unsigned char)buf100_2[v21]+v22)%256;
loc_buf100[v21++]=loc_buf100[ix];
loc_buf100[ix]=v22;
}
while(v21!=256);
printf("----------------------------------------\n");
for(i=0;i<0x100;i++){
printf("%02X ",(unsigned char)loc_buf100[i]);
if((i+1)%16==0){
printf("\n");
}
}
printf("----------------------------------------\n");
/*
0xF0, 0x37, 0xE1, 0x9B, 0x2A, 0x15, 0x17, 0x9F, 0xD7, 0x58, 0x4D, 0x6E, 0x33, 0xA0, 0x3
0x04, 0xD0, 0xBE, 0xED, 0xF8, 0x66, 0x5E, 0x00, 0xD6, 0x91, 0x2F, 0xC3, 0x10, 0x4C, 0xF
0xC1, 0xEC, 0x6D, 0x0B, 0x50, 0x65, 0xBB, 0x34, 0xFA, 0xA4, 0x2D, 0x3B, 0x23, 0xA1, 0x9
0x1D, 0x38, 0x56, 0x0A, 0x5D, 0x4F, 0xE4, 0xCC, 0x24, 0x0D, 0x12, 0x87, 0x35, 0x85, 0x8
0xC6, 0x13, 0x9A, 0xD3, 0xFC, 0xE7, 0x08, 0xAC, 0xB7, 0xE9, 0xB0, 0xE8, 0x41, 0xAA, 0x5
0xC2, 0x42, 0xBC, 0xE6, 0x0F, 0x8A, 0x86, 0xA8, 0xCF, 0x84, 0xC5, 0x48, 0x74, 0x36, 0x0
0x88, 0x51, 0xF6, 0x7F, 0x57, 0x05, 0x63, 0x3E, 0xFE, 0xB8, 0xC9, 0xF5, 0xAF, 0xDF, 0xE
0x44, 0xF9, 0xCD, 0x06, 0xBA, 0x30, 0x47, 0x40, 0xDE, 0xFD, 0x1C, 0x7C, 0x11, 0x5C, 0x0
0x2C, 0x9C, 0x5F, 0x46, 0x27, 0xC4, 0x83, 0x73, 0x16, 0x90, 0x20, 0x76, 0x7B, 0xF2, 0xE
```

```
0x77, 0x52, 0x80, 0x25, 0x09, 0x26, 0x3F, 0xC7, 0x18, 0x1B, 0xA3, 0xFF, 0xFB, 0xCB, 0xA
0x54, 0x7A, 0x68, 0xB4, 0x70, 0x4B, 0xE2, 0x49, 0x22, 0x7E, 0xA5, 0xB6, 0x81, 0x9D, 0x4
0xF1, 0xA7, 0x3C, 0xD9, 0x94, 0xEF, 0x32, 0x6B, 0x1F, 0xB1, 0x60, 0xB9, 0x64, 0x59, 0x0
0x7D, 0xE0, 0x6C, 0xAD, 0x97, 0x19, 0xB5, 0x3A, 0xF4, 0xD8, 0x8D, 0x98, 0x03, 0x93, 0x1
0x1E, 0x4A, 0xC0, 0x5A, 0xE5, 0xD1, 0x3D, 0x14, 0xC8, 0x79, 0xBD, 0x43, 0xDB, 0x69, 0xL
0x95, 0x9E, 0x21, 0x45, 0x89, 0x2B, 0xAB, 0x29, 0xA2, 0x8B, 0x2E, 0xD4, 0x0E, 0x62, 0x0
0xDA, 0x5B, 0x72, 0x8F, 0x99, 0x75, 0xEE, 0x78, 0x0C, 0x71, 0xBF, 0xDD, 0xCE, 0x92, 0x6
*/
txi=0;
i=0;
txj=0;
//v44=b64_salt_size;
do
{
txi=(txi+1)%256;
tx=(unsigned char)loc_buf100[txi];
txj=(txj+tx)%256;
loc_buf100[txi]=loc_buf100[txj];
loc_buf100[txj]=tx;
pwdch=loc_buf100[(unsigned char)(tx+loc_buf100[txi])]^b64bin[i];
gpwd[i] = pwdch;
++i;
}
while(i<16);
printf("----------------------------------------\n");
txi=0;
i=0;
txj=0;
//v44=b64_salt_size;
do
{
txi=(txi+1)%256;
tx=(unsigned char)ref100[txi];
txj=(txj+tx)%256;
ref100[txi]=ref100[txj];
ref100[txj]=tx;
printf("0x%02X, ",ref100[(unsigned char)(tx+ref100[txi])]);
pwdch=ref100[(unsigned char)(tx+ref100[txi])]^b64bin[i];
gpwd[i] = pwdch;
++i;
}
while(i<16);
printf("\ngidbuf3 len: %d\tbytes:[%s]\n",16,gpwd);
for(i=0;i<20;i++){
printf("%02X ",(unsigned char)gpwd[i]);
```

```
}
printf("\n");
}


int main(int argc,char* argv[]){
test1();
return 0;
}
```

```
0x9B, 0x6B, 0xBA, 0x25, 0x73, 0x82, 0xE0, 0x31, 0x86, 0x80, 0xF1, 0xC5, 0xDA, 0x82, 0xDA, 0x08,
gidbuf3 len: 16 bytes:[fu0kzHp2aqtZAuY6]
66 75 30 6B 7A 48 70 32 61 71 74 5A 41 75 59 36 00 00 00 00
```

今天的题目解析就到这里啦，预告一下，我们将在未来的8天里对比赛的剩下的八道题目挨个进行分析哦～
明天是题目《影分身之术》的解析，我们相约明天同一时间，不见不散～

- End -

往期文章一览

1、2019KCTF 晋级赛Q1 | 第一题点评及解题思路

2、【已更新】看雪课程 | LLVM 编译框架详解

3、赠书 | Android软件安全权威指南【签名版】

公众号ID：ikanxue

官方微博：看雪安全

商务合作：wsc@kanxue.com

戳原文，查看更多精彩writeup！

阅读原文