

第五章 常见加密算法逆向分析

④ 5.1 简单加密算法逆向分析

④ 5.2 对称加密算法逆向分析

④ 5.3 单向散列算法逆向分析



5.2 对称加密算法逆向分析

- ④ 加密和解密时使用相同的密钥，或是使用两个简单的可以相互推算的密钥
- ④ 本节将对两种对称加密算法进行介绍
 - ❑ RC4
 - ❑ DES



对称加密算法 RC4

@ 原理介绍

- ❑ RC4本质上是流密码算法，利用生成的密钥流序列和输入明文进行异或完成加密。
- ❑ 密钥流的生成以一个足够大的数组为基础，对其进行非线性变换，把这个大数组称为**S**盒。
- ❑ RC4的处理包括两个过程
 - 一个是密钥调度算法来置乱**S**盒的初始排列
 - 另一个是伪随机生成算法，来输出随机序列并修改**S**盒的当前排列顺序



对称加密算法 RC4

④ 密钥调度算法

□ 密钥调度算法是根据用户选定的密钥**K**:

$$K(0 \leq \text{len}(k) \leq 256)$$

□ 依次对数组中的数据进行换位，进而打乱**S**盒的初始排序

○ 其中，如果**K**的长度小于**256**，则将**K**重复拼接起来，直到长度为**256**为止。



对称加密算法 RC4

④ 伪随机生成算法

- 是利用初始化后的**S**盒，按照一定的规则从中选取数据输出，同时更新**S**盒的排列顺序，达到生成伪随机序列的目的。



对称加密算法 RC4

④ 加解密步骤

- ❑ RC4加解密的关键步骤在于按照密钥生成伪随机序列，得到伪随机序列后直接与明/密文进行异或操作即可。

④ 生成伪随机序列的过程

- ❑ 由密钥初始化S盒
- ❑ 生成密钥流



对称加密算法 RC4

□ 由密钥初始化S盒

- S盒的长度为**256**，程序首先会将**0**到**255**的互不重复的元素装入S盒，使得

$$S[i] = i (0 \leq i \leq 255)$$

- 同时建立一个长度为**256**的临时数组T，如果密钥K的长度等于**256**字节，那么直接将密钥的值赋给T，否则将K的元素依次赋给T，并不断重复的将K的值赋给T，直到T被填满。



对称加密算法 RC4

❑ 伪代码如下

```
for i from 0 to 255
    S[i] := i
end for
j := 0
for( i=0; i<256; i++)
    j := (j + S[i] + key[i mod keylength]) % 256
    swap values of S[i] and S[j]
end for
```



对称加密算法 RC4

□ 生成密钥流

- 利用第一步中得到的S盒便可开始生成用于加密的密钥流了
- 生成密钥流的过程中，根据i、j的值来确定选取S盒的哪个元素，并更新S盒中元素的排列顺序



对称加密算法 RC4

□ 下列伪代码描述了生成密钥流的过程

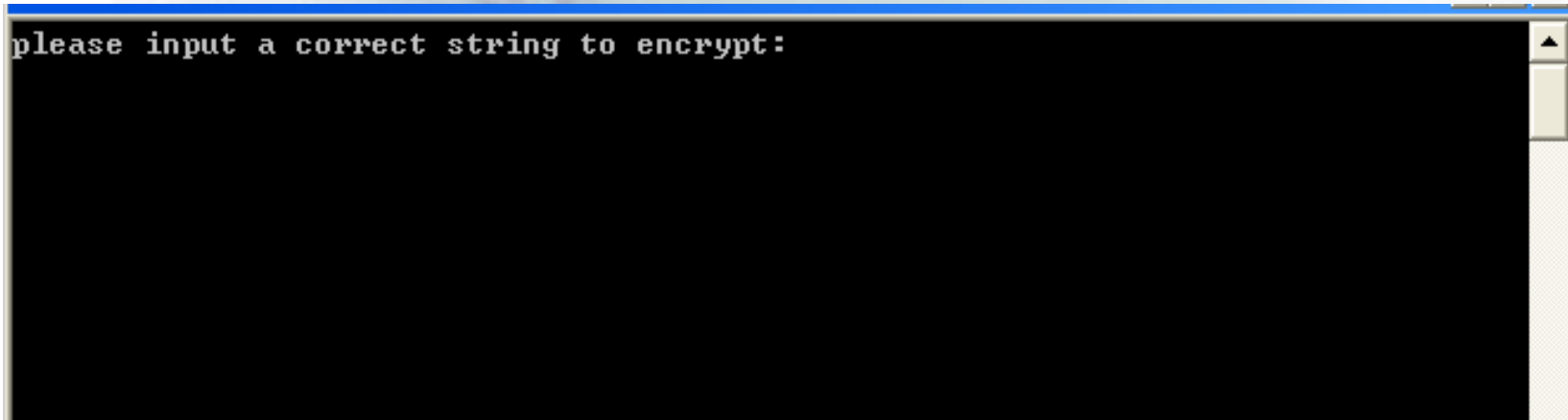
```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    t := (S[i] + S[j]) mod 256
    k := inputByte ^ S[t]
    output k
end while
```



对称加密算法 RC4

@ 逆向分析

- ❑ 运行程序rc4enc.exe，查看程序的功能
- ❑ 首先需要输入一个正确的字符串来加密



```
please input a correct string to encrypt:
```



对称加密算法 RC4

- ❑ 使用**IDA**打开该程序进行分析，定位到**main**函数
- ❑ 由于**RC4**加密算法较复杂，可使用**IDA**的**F5**插件来反编译程序，在反编译出来的伪代码基础上进行分析



对称加密算法 RC4

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     unsigned int v3; // kr04_4@1
4     int v4; // eax@1
5     int result; // eax@4
6     char v6; // [sp+8h] [bp-C8h]@1
7     char v7; // [sp+9h] [bp-C7h]@1
8     __int16 v8; // [sp+69h] [bp-67h]@1
9     char v9; // [sp+6Bh] [bp-65h]@1
10    char v10; // [sp+6Ch] [bp-64h]@1
11
12    v6 = 0;
13    memset(&v7, 0, 0x60u);
14    v8 = 0;
15    v9 = 0;
16    puts(aPleaseInputACo);
17    scanf(aS, &v10);
18    v3 = strlen(&v10) + 1;
19    sub_4011C0(&v10, &v6);
20    v4 = 0;
21    if ( (signed int)(v3 - 1) <= 0 )
22    {
23    LABEL_4:
24        puts(aGreat);
25        system(aPause);
26        result = 0;
27    }
28    else
29    {
30        while ( *(&v6 + v4) == byte_409130[v4] )
31        {
32            if ( ++v4 >= (signed int)(v3 - 1) )
33                goto LABEL_4;
34        }
35        result = -1;
36    }
37    return result;
```



对称加密算法 RC4

- ❑ 这里可以很清楚的看到main函数主要是
 - 获取用户的输入并将其保存到v10中;
 - 接着将v10作为参数调用了函数0x4011C0;
 - 最后还有一个比较的过程, 将v6指向的内容与字节数组0x409130处的内容进行比较
 - 同时发现v6也作为参数传递给了函数0x4011C0
 - 于是猜测这里v6就是用户的输入加密后得到的密文



对称加密算法 RC4

❑ 字节数组0x409130处的正确密文

```
.data:00409130 ; char byte_409130[]  
.data:00409130 byte_409130      db 1Bh                ; DATA XREF: _main+62↑r  
.data:00409131                db 0CAh ;  
.data:00409132                db 0AEh ;  
.data:00409133                db 0EFh ;  
.data:00409134                db 1Eh  ;  
.data:00409135                db 95h  ;  
.data:00409136                db 4Bh  ; K  
.data:00409137                db 0C2h ;  
.data:00409138                db 0D5h ;  
.data:00409139                db 0E3h ;  
.data:0040913A                db 33h  ; 3  
.data:0040913B                db 76h  ; U  
.data:0040913C                db 4Fh  ; 0  
.data:0040913D                db 0F9h ;  
.data:0040913E                db 4Fh  ; 0  
.data:0040913F                db 0D2h ;  
.data:00409140                db 0FCh ;  
.data:00409141                db 60h  ;  
.data:00409142                db 96h  ;  
.data:00409143                db      0
```



对称加密算法 RC4

- ❑ 下面跟进函数 **0x4011C0**，发现函数中存在着两个函数调用，并且第二个函数调用 **0x401130** 的返回值还与 **v5[v6]** 进行了异或操作
- ❑ 这里即用户输入的内容

$$v5[v6] = v5 + v6 = a2 + a4 - a2 = a4 = a1$$



对称加密算法 RC4

```
1 int __cdecl sub_4011C0(const char *a1, _BYTE *a2)
2 {
3     unsigned int v2; // kr04_4@1
4     int result; // eax@1
5     const char *v4; // edi@2
6     _BYTE *v5; // esi@2
7     int v6; // edi@2
8     int v7; // [sp+10h] [bp+4h]@2
9
10    dword_40BE90 = 0;
11    dword_40BE94 = 0;
12    v2 = strlen(a1) + 1;
13    sub_4010A0();
14    result = 0;
15    if ( (signed int)(v2 - 1) <= 0 )
16    {
17        *a2 = 0;
18    }
19    else
20    {
21        v4 = a1;
22        v7 = v2 - 1;
23        v5 = a2;
24        v6 = v4 - a2;
25        do
26        {
27            *v5 = v5[v6] ^ sub_401130();
28            ++v5;
29            result = v7-- - 1;
30        }
31        while ( v7 );
32        a2[v2 - 1] = 0;
33    }
34    return result;
35 }
```



对称加密算法 RC4

- ❑ 跟进函数**0x4010A0**，该函数主要有两个循环
 - 第一个循环是将地址**0x40BA90**处开始的值赋值为**0,1,2,3.....255**;
 - 第二个循环主要是根据字符串“**RC4key**”的值来对这**256**个值进行交换操作。
- ❑ 看到这里，应该要很快的反应过来这里是**RC4**加密中的初始化**S**盒过程
 - 其中密钥为字符串“**RC4key**”，而地址**0x40BA90**指向的内存便是**S**盒



对称加密算法 RC4

```
1 int sub_4010A0()
2 {
3     unsigned int v0; // kr04_4@1
4     int v1; // edx@1
5     int *v2; // eax@1
6     int v3; // edi@3
7     signed int v4; // ebx@3
8     int *v5; // esi@3
9     int result; // eax@4
10    unsigned __int8 v7; // ST0C_1@4
11
12    v0 = strlen(aRc4key) + 1;
13    v1 = 0;
14    v2 = dword_40BA90;
15    do
16    {
17        *v2 = v1;
18        ++v2;
19        ++v1;
20    }
21    while ( (signed int)v2 < (signed int)&dword_40BE90 );
22    v3 = 0;
23    v4 = 0;
24    v5 = dword_40BA90;
25    do
26    {
27        v3 = (*v5 + (unsigned __int8)aRc4key[v4 % (signed int)(v0 - 1)] + v3) % 256;
28        result = dword_40BA90[v3];
29        v7 = *(_BYTE *)v5;
30        *v5 = result;
31        ++v5;
32        ++v4;
33        dword_40BA90[v3] = v7;
34    }
35    while ( (signed int)v5 < (signed int)&dword_40BE90 );
36    return result;
37 }
```



对称加密算法 RC4

- ❑ 现在知道了初始化S盒的函数，且知道函数 **0x401130** 的返回值与用户的输入进行了异或操作
- ❑ 则很容易的猜到函数 **0x401130** 就是生成密钥流的函数了
- ❑ 利用 **IDA** 的快捷键 ‘N’ 来对变量重新命名，以便于更好的分析



对称加密算法 RC4

```
1 char Generate_Key()
2 {
3     int v0; // eax@1
4     signed int v1; // ecx@1
5     unsigned __int8 v2; // d1@1
6
7     v0 = (pos_i + 1) % 256;
8     v1 = pos_j + *(_DWORD *)&Sbox[4 * v0];
9     pos_i = (pos_i + 1) % 256;
10    v1 %= 256;
11    v2 = Sbox[4 * v0];
12    pos_j = v1;
13    *(_DWORD *)&Sbox[4 * v0] = *(_DWORD *)&Sbox[4 * v1];
14    *(_DWORD *)&Sbox[4 * v1] = v2;
15    return Sbox[4 * ((v2 + *(_DWORD *)&Sbox[4 * v0]) % 256)];
16 }
```



对称加密算法 RC4

- ❑ 分析生成密钥流的函数，可以看到：利用两个变量 **pos_i**, **pos_j** 来选择一个 **S** 盒中的值作为函数的返回值，并打乱 **S** 盒的数据（交换）。
- ❑ 因此，知道了加密算法为 **RC4**，加密的密钥为字符串 “**RC4key**”，最终的密文为字节数组 **0x409130** 处的值，便可以编写解密函数来解密了。
- ❑ 对于 **RC4** 加密算法来说，解密只需要生成和加密过程一样的密钥流即可。



对称加密算法 RC4

❑ 解密函数如下（因算法较长，仅列出关键函数，其中初始化S盒以及生成密钥流的部分与加密算法相同）：

```
int main()  
{  
    char result[MAX_STR]={0};  
    int len;  
    unsigned char  
cipher[MAX_STR]={0x1b,0xca,0xae,0xef,0x1e,0x95,0x4b,0xc2,0xd5,0x  
e3,0x33,0x76,0x4f,0xf9,0x4f,0xd2,0xfc,0x60,0x96,0x0};  
    decryption((unsigned char*)cipher,(unsigned char*)result);  
    printf("%s",result);  
    system("pause");  
    return 0;  
}
```



对称加密算法 RC4

```
void decryption(unsignedchar* ciphertext,unsignedchar*result){
    pos_i =0;
    pos_j =0;
    int len = strlen((constchar*)ciphertext);
    int i=0;
    init_sbox();
    for(i=0;i<len;++i)
        result[i]= ciphertext[i]^generate_key();
    result[i]='\0';
}
```



对称加密算法 RC4

❑ 运行解密函数，得到正确的输入

```
Is_Th13_Simple_Rc4?请按任意键继续. . .
```

❑ 进行验证

```
please input a correct string to encrypt:  
Is_Th13_Simple_Rc4?  
Great!!!  
请按任意键继续. . .
```

