



汇编语言与逆向工程

北京邮电大学
付俊松

2019年10月



第五章 常见加密算法逆向分析

- 5.1 简单加密算法逆向分析
- 5.2 对称加密算法逆向分析
- 5.3 单向散列算法逆向分析



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析

□1 异或加密

- (1) 原理介绍
- (2) 加解密步骤
- (3) 逆向分析

□2 仿射加密

- (1) 原理介绍
- (2) 加解密步骤
- (3) 逆向分析



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ 异或运算符常作为更为复杂加密算法的一个组成部分

- 如果使用不断重复的密钥，利用频率分析就可以破解这种简单的异或密码
- 如果消息的内容被猜出或知道，密钥就会泄露
- 异或密码值得使用的原因主要是其易于实现，而且计算成本小
- 简单重复异或加密有时用于不需要特别安全的情况下隐藏信息



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ (1) 原理介绍

- 异或是一种运算，数学运算符为XOR
- 总结起来就是相同的数（取0或1）异或得到的结果为0，不同则为1

$$A \text{ XOR } 0 = A$$

$$A \text{ XOR } A = 0$$

$$(A \text{ XOR } B) \text{ XOR } B = A$$

- 或，OR，有1即为1（包括两个1和1个1），其他则为0



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ (2) 加解密步骤

- 加密过程只需要将明文和密钥逐字节异或，而解密过程则只需要将密文和密钥逐字节异或



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ (3) 逆向分析

- 运行程序xorencryption.exe，通过逆向分析了解这个逆向CTF程序具有什么功能

```
xor encryption program?  
please input a string:  
-
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

– 使用IDA打开xorencryption.exe，定位到main函数，分析程序的流程

➤ 程序首先获取了用户的输入，并计算输入的长度，如果长度为10，则跳转到0x40107E处继续运行，否则输出wrong，并退出运行。

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main proc near ; CODE XREF: start+AF1p
.text:00401000
.text:00401000 var_C
.text:00401000 argc
.text:00401000 argv
.text:00401000 envp
.text:00401000
.text:00401000
.text:00401003
.text:00401004
.text:00401009
.text:0040100E
.text:00401013
.text:00401018
.text:0040101C
.text:0040101D
.text:00401022
.text:00401027
.text:0040102B
.text:0040102E
.text:00401030
.text:00401033
.text:00401035
.text:00401037
.text:00401038
.text:00401039
.text:0040103C
.text:0040103E
.text:00401043
.text:00401048
.text:0040104D
.text:00401050
.text:00401051
.text:00401056
.text:00401058

= byte ptr -0Ch
= dword ptr 4
= dword ptr 8
= dword ptr 0Ch

sub esp, 0Ch
push edi
push offset aXorEncryptionP ; "xor encryption program!"
call _puts
push offset aPleaseInputAST ; "please input a string:"
call _puts
lea eax, [esp+18h+var_C]
push eax
push offset aS ; "%s"
call _scanf
lea edi, [esp+20h+var_C]
or ecx, 0FFFFFFFFh
xor eax, eax
add esp, 10h
repne scasb
not ecx
dec ecx
pop edi
cmp ecx, 0Ah
jz short loc_40107E
push offset aWrong ; "wrong!"
call _puts
mov eax, stru_4090B0._cnt
add esp, 4
dec eax
mov stru_4090B0._cnt, eax
js short loc_40106A
mov eax, stru_4090B0._ptr
```

获取用户输入

计算用户输入的长度



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 程序首先获取了用户的输入，并计算输入的长度，如果长度为10，则跳转到0x40107E处继续运行，否则输出wrong，并退出运行。

```
.call    _scanf
lea     edi, [esp+20h+var_C]
or      ecx, 0FFFFFFFh
xor     eax, eax
add     esp, 10h
repne scasb
not     ecx
dec     ecx
pop     edi
cmp     ecx, 0Ah
jz      short loc_40107E
```

```
push    offset aWrong ; "wrong!"
call    _puts
mov     eax, stru_4090B0._cnt
add     esp, 4
dec     eax
mov     stru_4090B0._cnt, eax
js      short loc_40106A
```

```
loc_40107E:
xor     eax, eax
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 跟进到0x40107E代码段，程序首先逐字节的将用户输入的内容，与字节数组0x409030处的值（密钥）进行异或操作；

```

pop     ecx
cmp     ecx, 0Ah
jz      short loc_40107E

```

```

push    offset aWrong ; "wrong!"
call    _puts
mov     eax, stru_4090B0._cnt
add     esp, 4
dec     eax
mov     stru_4090B0._cnt, eax
js      short loc_40106A

```

```

loc_40107E:
xor     eax, eax

```

```

mov     eax, stru_4090B0._ptr
inc     eax
mov     stru_4090B0._ptr, eax
or      eax, 0FFFFFFFh
add     esp, 0Ch
retn

```

```

loc_40106A: ; FILE *
push    offset stru_4090B0
call    __filbuf
add     esp, 4
or      eax, 0FFFFFFFh
add     esp, 0Ch
retn

```

```

loc_401080:
mov     cl, byte_409030[eax]
mov     dl, [esp+eax+0Ch+var_C]
xor     dl, cl
mov     [esp+eax+0Ch+var_C], dl
inc     eax
cmp     eax, 0Ah
jl      short loc_401080

```

```

push    esi
xor     esi, esi

```

```

loc_401099:
movsx   eax, [esp+esi+10h+var_C]
xor     edx, edx
mov     dl, byte_40903C[esi]
cmp     eax, edx
jnz     short loc_4010B2

```

```

inc     esi
cmp     esi, 0Ah
jl      short loc_401099

```

```

jmp     short loc_4010BF

```

```

loc_4010B2: ; "wrong answer!!!!"
push    offset aWrongAnswer

```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 接着将加密得到的结果与字节数组0x40903C处的值（密文）进行比较，不相等则输出“wrong answer!!!!”，相等则输出“goodjob!!!”



(1) ↑

```
mov     eax, stru_4090B0_ptr
inc     eax
mov     stru_4090B0_ptr, eax
or      eax, 0FFFFFFFh
add     esp, 0Ch
retn
```

```
loc_40106A:                ; FILE *
push    offset stru_4090B0
call    __filbuf
add     esp, 4
or      eax, 0FFFFFFFh
add     esp, 0Ch
retn
```

```
loc_401080:
mov     cl, byte_409030[eax]
mov     dl, [esp+eax+0Ch+var_C]
xor     dl, cl
mov     [esp+eax+0Ch+var_C], dl
inc     eax
cmp     eax, 0Ah
j1      short loc_401080
```

```
push    esi
xor     esi, esi
```

```
loc_401099:
movsx   eax, [esp+esi+10h+var_C]
xor     edx, edx
mov     dl, byte_40903C[esi]
cmp     eax, edx
jnz     short loc_4010B2
```

```
inc     esi
cmp     esi, 0Ah
j1      short loc_401099
```

```
jmp     short loc_4010BF
```

```
loc_4010B2:                ; "wrong answer!!!!"
push    offset aWrongAnswer
call    _puts
add     esp, 4
```

```
loc_4010BF:
cmp     esi, 0Ah
pop     esi
jnz     short loc_4010B2
```

```
push    offset aGoodJob ; "good job!!!"
call    _puts
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main          proc near          ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 var_C          = byte ptr -0Ch
.text:00401000 argc          = dword ptr  4
.text:00401000 argv          = dword ptr  8
.text:00401000 envp          = dword ptr 0Ch
.text:00401000
.text:00401000 sub          esp, 0Ch
.text:00401000 push        edi
.text:00401003 push        offset aXorEncryptionP ; "xor encryption program?"
.text:00401004 call        _puts
.text:00401009 push        offset aPleaseInputAS ; "please input a string:"
.text:0040100E call        _puts
.text:00401013 lea          eax, [esp+18h+var_C]
.text:00401018 push        eax
.text:0040101D push        offset aS ; "%s"
.text:00401022 call        _scanf
.text:00401027 lea          edi, [esp+20h+var_C]
.text:0040102B or           ecx, 0FFFFFFFh
.text:0040102E xor          eax, eax
.text:00401030 add          esp, 10h
.text:00401033 repne scasd
.text:00401035 not          ecx
.text:00401037 dec          ecx
.text:00401038 pop          edi
.text:00401039 cmp          ecx, 0Ah
.text:0040103C jz           short loc_40107E
.text:0040103E push        offset aWrong ; "wrong?"
.text:00401043 call        _puts
.text:00401048 mov          eax, stru_4090B0._cnt
.text:0040104D add          esp, 4
.text:00401050 dec          eax
.text:00401051 mov          stru_4090B0._cnt, eax
.text:00401056 js           short loc_40106A
.text:00401058 mov          eax, stru_4090B0._ptr
```

获取用户输入

计算用户输入的长度



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ 每条说明

- 用户输入的内容是保存在地址`esp+20h+var_C`处的，指令`lea edi,[esp+20h+var_C]`功能就是将`esp+20h+var_C`的值放入寄存器`edi`中，所以现在`edi`寄存器是指向用户输入的内容的。
- 指令`or ecx,0FFFFFFFFh`是将寄存器`ecx`的值设置为`0xFFFFFFFF`（也就是所有位设置为1）
- 指令`xor eax, eax`将`eax`的值设置为0（0在C语言中就是字符串的结尾）
- 指令`add esp,10h`是用来平衡栈的（这里不用关心）



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main          proc near          ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 var_C          = byte ptr -0Ch
.text:00401000 argc          = dword ptr  4
.text:00401000 argv         = dword ptr  8
.text:00401000 envp         = dword ptr 0Ch
.text:00401000
.text:00401000 sub          esp, 0Ch
.text:00401000 push        edi
.text:00401003 push        offset aXorEncryptionP ; "xor encryption program?"
.text:00401004 call        _puts
.text:00401009 push        offset aPleaseInputAS ; "please input a string:"
.text:0040100E call        _puts
.text:00401013 lea          eax, [esp+18h+var_C]
.text:00401018 push        eax
.text:0040101D push        offset aS ; "%s"
.text:00401022 call        _scanf
.text:00401027 lea          edi, [esp+20h+var_C]
.text:0040102B or           ecx, 0FFFFFFFh
.text:0040102E xor          eax, eax
.text:00401030 add          esp, 10h
.text:00401033 repne scasd
.text:00401035 not          ecx
.text:00401037 dec          ecx
.text:00401038 pop          edi
.text:00401039 cmp          ecx, 0Ah
.text:0040103C jz           short loc_40107E
.text:0040103E push        offset aWrong ; "wrong?"
.text:00401043 call        _puts
.text:00401048 mov          eax, stru_4090B0._cnt
.text:0040104D add          esp, 4
.text:00401050 dec          eax
.text:00401051 mov          stru_4090B0._cnt, eax
.text:00401056 js           short loc_40106A
.text:00401058 mov          eax, stru_4090B0._ptr
```

获取用户输入

计算用户输入的长度



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

– 下面就是最关键的指令 **repne scasb**

– **repne**

- **repne**是一个串操作指令中的条件重复前缀指令，加在串操作指令前，使串操作重复进行。
- **repne**可检查两个字符串是否不同，发现相同立即停止比较。
- **repne**的重复条件是 **CX ≠ 0** 且 **ZF = 0**，每执行一次，**CX**的内容就减1，直到**CX**减为0时，结束串指令操作。若重复条件满足，重复前缀先使 **CX ← CX - 1**，然后执行后面的串指令。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 指令 **repne scasb** 表示如果 **ecx** 的值不为 0 就继续执行后面的内容 --> **scasb**
- **scasb**
 - **scasb** 则是串扫描指令，比较 **edi** 寄存器指向的值与 **eax** 寄存器中的值是否相等，每次将 **edi** 的值增加 1
 - 如果相等就退出循环（执行该指令后面的指令），不相等就继续比较



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- **repne scasb** (**repeat not equal**) 该指令的功能是比较寄存器**edi**指向的值（用户输入值）和寄存器**eax**的值是否相等，如果不相等则将**edi**的值增一，并继续比较，相等则结束循环。
- 这里**eax**寄存器的值为0，将**edi**寄存器指向的值与0比较，是在判断是否到达了用户输入字符串的末尾。
- 每比较一次，寄存器**ecx**的值会减一。
- 因此寄存器**ecx**减少的值即为字符串的长度。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main          proc near          ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 var_C          = byte ptr -0Ch
.text:00401000 argc          = dword ptr  4
.text:00401000 argv         = dword ptr  8
.text:00401000 envp         = dword ptr 0Ch
.text:00401000
.text:00401000 sub          esp, 0Ch
.text:00401000 push         edi
.text:00401000 push         offset aXorEncryptionP ; "xor encryption program!"
.text:00401004 call         _puts
.text:00401009 push         offset aPleaseInputAS ; "please input a string:"
.text:0040100E call         _puts
.text:00401013 lea          eax, [esp+18h+var_C]
.text:00401018 push         eax
.text:0040101D push         offset aS ; "%s"
.text:00401022 call         _scanf
.text:00401027 lea          edi, [esp+20h+var_C]
.text:0040102B or           ecx, 0FFFFFFFh
.text:0040102E xor          eax, eax
.text:00401030 add          esp, 10h
.text:00401033 repne scasd
.text:00401035 not          ecx
.text:00401037 dec          ecx
.text:00401038 pop          edi
.text:00401039 cmp          ecx, 0Ah
.text:0040103C jz           short loc_40107E
.text:0040103E push         offset aWrong ; "wrong!"
.text:00401043 call         _puts
.text:00401048 mov          eax, stru_4090B0._cnt
.text:0040104D add          esp, 4
.text:00401050 dec          eax
.text:00401051 mov          stru_4090B0._cnt, eax
.text:00401056 js           short loc_40106A
.text:00401058 mov          eax, stru_4090B0._ptr
```

获取用户输入

计算用户输入的长度



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□再解释一遍

- 将edi指向了用户的输入，并将eax设置为了字符串结束的标志（0），所以这里就相当于逐字节的将输入的内容与eax（0）进行比较，如果相等了就表示到字符串结束了。
- 注意这里每比较一次ecx的值都会减少1，因此最后只需要看ecx的值减少了多少，用户输入的长度就是多少。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 所以repne scasb指令后面的not ecx以及dec ecx就是在计算ecx减少了多少，经过这两条指令后ecx就是字符串的长度了
- 将ecx与0xAH(即二进制10)比较，看是否相等，不相等就输出wrong



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- **not ecx**以及**dec ecx**是在计算ecx减少了多少
 - 一开始ecx的值为0xFFFFFFFF，假设循环了5次，也就是减了5。
 - 0xFFFFFFFF对应的是-1， $-1-5=-6$ ，-6对应0xFFFFFFFFFA，然后对其取反，得到的值为5
 - 还有一个减1的操作，这样得到的结果就为4



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

➤之所以不是5，这里需要看一下scasb指令具体的流程，

➤scasb :

inc edi

dec ecx

je loopdone

cmp byte [edi-1],al

jne scans

loopdone

➤其中在循环开始处ecx减了1，相当于在扫描到结尾 ‘\0’ 处也将ecx减了1，多减了一个1（也就是将字符串长度多算了一个）



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 讲了这么多汇编代码，其实这就是C语言库函数 `strlen()` 的汇编语言实现方式
- 因为使用的是 `release` 版，这里编译器做了许多优化（可以看到整个程序的汇编代码中都没有使用到 `ebp` 寄存器，全部使用 `esp+xxx` 代替了）



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

IDA - C:\Documents and Settings\Administrator\桌面\课内练习\第5章\第五章\随书代码-2\5-1\程序\xorencryption.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexploded Instruction External symbol

Functions window

Function name

- _main
- _system
- _filbuf
- _scanf
- _puts
- _start
- _amsg_exit
- _fast_error_exit
- _spawnvpe
- _cinit
- _exit
- _doexit
- _initterm
- _spawnve
- _comexecmd
- _access
- _getenv
- _ioinit
- _read
- _getbuf
- sub_401D63
- _fcloseall
- _fflush
- _flush
- sub_401F03
- _flsall
- _input
- _hextodec
- _fgetc
- _un_inc
- _whiteout
- _stbuf
- _ftbuf
- _flsbuf
- _fwrite
- _strlen
- _XcptFilter
- _xcptlookup
- _setenvp
- _setargv
- _parse_cmdline
- _crtGetEnvironmentStringsA
- sub_403357
- sub_403384

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int result; // eax@3
4     signed int u4; // eax@5
5     int u5; // esi@7
6     char u6[12]; // [sp+4h] [bp-Ch]@1
7
8     puts(aXorEncryptionP);
9     scanf(aS, u6);
10    if ( strlen(u6) == 10 )
11    {
12        u4 = 0;
13        while ( u4 < 10 )
14        {
15            u6[u4] ^= byte_409030[u4];
16            ++u4;
17        }
18        while ( u5 < 10 )
19        {
20            u5 = 0;
21            while ( u6[u5] == (unsigned __int8)byte_40903C[u5] )
22            {
23                if ( ++u5 >= 10 )
24                    goto LABEL_12;
25            }
26            puts(aWrongAnswer);
27        LABEL_12:
28            if ( u5 == 10 )
29            {
30                puts(aGoodJob);
31                system(aPause);
32                result = 0;
33            }
34            else
35            {
36                puts(aWrong);
37                if ( --stru_409080._cnt < 0 )
38                {
39                    _filbuf(&stru_409080);
40                    result = -1;
41                }
42                else
43                {
44                    ++stru_409080._ptr;
45                    result = -1;
46                }
47            }
48        }
49        return result;
50    }
```

Line 1 of 130

Output window

401000: using guessed type char var_C[12];

401000: using guessed type char var_C[12];

Python

AU: idle Up Disk: 35GB

开始 程序 IDA_Pro_v6.8 第5章 IDA - C:\Do...



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ 异或加密过程

- 看一下0x40107E代码段
- 程序首先逐字节的将用户输入的内容与字节数组0x409030处的值（密钥）进行异或操作；



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:0040107E loc_40107E:                                ; CODE XREF: _main+3C↑j
.text:0040107E                                     xor     eax, eax
.text:00401080                                     ; CODE XREF: _main+94↓j
.text:00401080 loc_401080:
.text:00401080                                     mov     cl, byte_409030[eax]
.text:00401086                                     mov     dl, [esp+eax+0Ch+var_C]
.text:0040108A                                     xor     dl, cl
.text:0040108C                                     mov     [esp+eax+0Ch+var_C], dl
.text:00401090                                     inc     eax
.text:00401091                                     cmp     eax, 0Ah
.text:00401094                                     jl      short loc_401080
.text:00401096                                     push    esi
.text:00401097                                     xor     esi, esi
.text:00401099                                     ; CODE XREF: _main+AE↓j
.text:00401099 loc_401099:
.text:00401099                                     movsx   eax, [esp+esi+10h+var_C]
.text:0040109E                                     xor     edx, edx
.text:004010A0                                     mov     dl, byte_40903C[esi]
.text:004010A6                                     cmp     eax, edx
.text:004010A8                                     jnz     short loc_4010B2
.text:004010AA                                     inc     esi
.text:004010AB                                     cmp     esi, 0Ah
.text:004010AE                                     jl      short loc_401099
.text:004010B0                                     jmp     short loc_4010BF
.text:004010B2 ; -----
.text:004010B2 loc_4010B2:                                ; CODE XREF: _main+A8↑j
.text:004010B2                                     push    offset aWrongAnswer ; "wrong answer!!!"
.text:004010B7                                     call     _puts
.text:004010BC                                     add     esp, 4
.text:004010BF loc_4010BF:                                ; CODE XREF: _main+B0↑j
.text:004010BF                                     cmp     esi, 0Ah
.text:004010C2                                     pop     esi
```

逐字节异或加密部分

判断是否是期望的加密结果



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 第一条指令mov指令从指定的字节数组0x409030处取出一个字节，放到寄存器ecx中
- 看第一条指令mov cl,byte_409030[eax]，其中地址0x409030指向的是一片内存，byte_409030[eax]的意思就是取地址0x409030+eax处的一个字节的意思
- 这样不断将eax的值增加1，实现逐字节取数据



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:0040107E loc_40107E:                ; CODE XREF: _main+3C↑j
.text:0040107E                xor     eax, eax
.text:00401080                ; CODE XREF: _main+94↓j
.text:00401080 loc_401080:                ; CODE XREF: _main+94↓j
.text:00401080                mov     cl, byte_409030[eax]
.text:00401086                mov     dl, [esp+eax+0Ch+var_C]
.text:0040108A                xor     dl, cl
.text:0040108C                mov     [esp+eax+0Ch+var_C], dl
.text:00401090                inc     eax
.text:00401091                cmp     eax, 0Ah
.text:00401094                jl      short loc_401080
.text:00401096                push    esi
.text:00401097                xor     esi, esi
.text:00401099                ; CODE XREF: _main+AE↓j
.text:00401099 loc_401099:                ; CODE XREF: _main+AE↓j
.text:00401099                movsx   eax, [esp+esi+10h+var_C]
.text:0040109E                xor     edx, edx
.text:004010A0                mov     dl, byte_40903C[esi]
.text:004010A6                cmp     eax, edx
.text:004010A8                jnz     short loc_4010B2
.text:004010AA                inc     esi
.text:004010AB                cmp     esi, 0Ah
.text:004010AE                jl      short loc_401099
.text:004010B0                jmp     short loc_4010BF
.text:004010B2 ; -----
.text:004010B2 loc_4010B2:                ; CODE XREF: _main+A8↑j
.text:004010B2                push    offset aWrongAnswer ; "wrong answer!!!!"
.text:004010B7                call    _puts
.text:004010BC                add     esp, 4
.text:004010BF loc_4010BF:                ; CODE XREF: _main+B0↑j
.text:004010BF                cmp     esi, 0Ah
.text:004010C2                pop     esi
```

逐字节异或加密部分

判断是否是期望的加密结果



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 第二条mov指令则是从用户输入的内容中取一个字节放入edx寄存器中
- 第三条指令xor dl,cl就是异或操作指令
 - 异或操作xor dl,cl，会将寄存器edx和ecx的异或后的结果放入到第一个操作数edx中
- 第四条指令，该指令将异或操作后得到的结果又放入了用户输入的地址处。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 这里为什么会是逐字节进行异或的，就是通过在最开始的`xor eax,eax`指令将`eax`寄存器置为0，然后用`eax`当做数组的下标，每次增加1来实现循环取数据并进行异或操作
- 可以看到第四条`mov`下面的`inc eax`就是将`eax`的值增加1的指令，然后在下面一条指令`cmp eax, 0xA`中将`eax`的值与10进行比较，判断是否要退出循环了
- 最后在将`eax`的值与10比较，小于10就跳转到地址`0x401080`处（也就是循环的开始）继续执行--->实现循环



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

```
.text:0040107E loc_40107E:                                ; CODE XREF: _main+3C↑j
.text:0040107E                                xor     eax, eax
.text:00401080                                ; CODE XREF: _main+94↓j
.text:00401080 loc_401080:                                ; CODE XREF: _main+94↓j
.text:00401080                                mov     cl, byte_409030[eax]
.text:00401086                                mov     dl, [esp+eax+0Ch+var_C]
.text:0040108A                                xor     dl, cl
.text:0040108C                                mov     [esp+eax+0Ch+var_C], dl
.text:00401090                                inc     eax
.text:00401091                                cmp     eax, 0Ah
.text:00401094                                jl      short loc_401080
.text:00401096                                push    esi
.text:00401097                                xor     esi, esi
.text:00401099                                ; CODE XREF: _main+AE↓j
.text:00401099 loc_401099:                                ; CODE XREF: _main+AE↓j
.text:00401099                                movsx   eax, [esp+esi+10h+var_C]
.text:0040109E                                xor     edx, edx
.text:004010A0                                mov     dl, byte_40903C[esi]
.text:004010A6                                cmp     eax, edx
.text:004010A8                                jnz     short loc_4010B2
.text:004010AA                                inc     esi
.text:004010AB                                cmp     esi, 0Ah
.text:004010AE                                jl      short loc_401099
.text:004010B0                                jmp     short loc_4010BF
.text:004010B2 ; -----
.text:004010B2 loc_4010B2:                                ; CODE XREF: _main+A8↑j
.text:004010B2                                push    offset aWrongAnswer ; "wrong answer!!!!"
.text:004010B7                                call    _puts
.text:004010BC                                add     esp, 4
.text:004010BF loc_4010BF:                                ; CODE XREF: _main+B0↑j
.text:004010BF                                cmp     esi, 0Ah
.text:004010C2                                pop     esi
```

逐字节异或加密部分

判断是否是期望的加密结果



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 理解了第一个方框的内容后第二个方框的内容就很好理解了，也是一个循环，只不过这里是利用esi寄存器来实现循环的（通过xor esi,esi将esi置0）
- 将用户输入的数据取出一字节，放入寄存器eax，将字节数组0x40903C处的数据（正确的密文）取出一字节放入edx中，接着比较eax与edx的值是否相等，不相等（jnz short loc_4010B2）就跳到0x4010B2处执行（这里是输出wrong answer），否则继续循环，直到循环了10次为止



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 从第一个方框是将输入的内容取出一个字节，加密后放回原处，所以在第二个方框里面用户输入的内容其实就是密文了，通过比较该密文是否和标准的密文（0x40903C处的数据）相等来判断用户输入的内容是否正确
- 因为在经过第一个方框的处理后用户输入的内容已经经过加密了（变成了密文），而在第二个方框又将该数据与字节数组0x40903C处的数据比较，看是否相等，所以0x40903C处的数据应该就是正确的密文了



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

– 0x409030处的密钥和0x40903C处的正确密文值为

```
.data:00409030 aAbcdefg123      db 'abcdefg123',0      ; DATA XREF: _main:loc_401080↑r
.data:0040903B                align 4
.data:0040903C byte_40903C db 28h                ; DATA XREF: _main+A0↑r
.data:0040903D                db 3Dh ; =
.data:0040903E                db 24h ; $
.data:0040903F                db 54h ; T
.data:00409040                db 0Ah
.data:00409041                db 12h
.data:00409042                db 38h ; 8
.data:00409043                db 7Ah ; z
.data:00409044                db 57h ; W
.data:00409045                db 4Ah ; J
.data:00409046                db 0
.data:00409047                db 0
.data:00409048 char aPause[1]
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 至此，整个程序的流程就很清楚了
 - 首先获取用户的输入
 - 如果输入的长度符合要求，便对其进行异或加密，加密的密钥为0x409030处的字节数组
 - 最后将得到的密文与0x40903C处的字节数组进行比较，判断是否是期望的密文。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 在知道了程序使用的加密算法、加密密钥以及密文后，便可以对其进行解密
- 根据异或加密的原理，使用加密时的密钥来与密文进行异或即可完成解密



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

□ 解题程序

```
#include<stdio.h>
int main(){
    char token[11]="abcdefg123";
    unsignedchar
ciphertext[]={0x28,0x3d,0x24,0x54,0x0a,0x12,0x38,0x7a,0x57,0x4a};
    char result[11];
    int i;

    for(i=0;i<10;++i){
        result[i]= token[i]^ciphertext[i];
    }
    result[i]='\0';
    printf("result is:%s\n",result);
    getchar();
    return0;
}
```




第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 异或加密

- 运行该解密函数，得到正确的输入，并对结果进行验证

```
result is:I_G00t_Key
```

```
xor encryption program!  
please input a string:  
I_G00t_Key  
good job!!!  
请按任意键继续. . .
```




第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析

□1 异或加密

□2 仿射加密

- (1) 原理介绍
- (2) 加解密步骤
- (3) 逆向分析



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

□ 古典密码

– 置换密码

- 根据一定的规则重新排列明文

– 代换密码

- 将明文中的字符串替换为其他字符
- 仿射加密便是代换密码中的一种



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

□ (1) 原理介绍

- 仿射加密的加密算法是一个线性变换，即对任意的明文字符 x ，对应的密文字符 y 为

$$y \equiv ax + b \pmod{26}$$

- 其中 a 、 b 为整数，且 $\gcd(a, 26) = 1$
- 上述变换是一一对应的

➤ 对于任意一个明文 x ，有且仅有一个密文 y 与之对应。对于任意一个密文 y ，有且仅有一个明文 x 与之对应

$x \equiv (y - b) * (a^{-1} \pmod{26})$ 与之对应 (a 与 26 互素，故 a^{-1} 存在)



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

□ (2) 加解密步骤

根据仿射密码的原理，加密过程为使用已选定的符合条件的密钥(a,b)，逐字符的对明文 x 进行 $y \equiv ax + b \pmod{26}$ 的运算即可完成加密。解密过程则逐字符的每一个密文 y 进行 $x \equiv (y - b) * a^{-1} \pmod{26}$ 的运算即可。

(注：如果 $a = 1$ ， $b = 3$ 时，这种加密就是著名的凯撒密码)



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

□ (3) 逆向分析

- 用一个仿射加密的实例来进行分析，逆向分析程序 `fangsheenc.exe`，查看程序的功能

```
please input a string:  
aaaaaadddddddd  
sorry  
请按任意键继续. . .
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

– 使用IDA来分析该程序，定位到main函数的位置

```
.text:00401000
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main          proc near          ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 var_64          = byte ptr -64h
.text:00401000 var_63          = byte ptr -63h
.text:00401000 argc           = dword ptr  4
.text:00401000 argv           = dword ptr  8
.text:00401000 envp          = dword ptr 0Ch
.text:00401000
.text:00401000 sub          esp, 64h
.text:00401003 push         edi
.text:00401004 mov          ecx, 18h
.text:00401009 xor          eax, eax
.text:0040100B lea          edi, [esp+68h+var_63]
.text:0040100F mov          [esp+68h+var_64], 0
.text:00401014 push        offset aPleaseInputASt ; "please input a string:"
.text:00401019 rep stosd
.text:0040101B stosw
.text:0040101D stosb
.text:0040101E call         _puts
.text:00401023 lea          eax, [esp+6Ch+var_64]
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 程序首先获取用户的输入，并计算输入内容的长度，如果长度等于0，则直接跳转到0x40105A处执行，否则先执行完0x401047处的循环后再执行0x40105A处的代码

```
.text:00401027
.text:00401028
.text:0040102D
.text:00401032
.text:00401036
.text:00401039
.text:0040103B
.text:0040103E
.text:00401040
.text:00401042
.text:00401043
.text:00401045
.text:00401047
loc_401047:
.text:00401047
.text:00401048
.text:0040104E
.text:00401050
.text:00401053
.text:00401055
.text:00401056
.text:00401058
.text:0040105A
.text:0040105A loc_40105A:
.text:0040105A
.text:0040105B
.text:0040105C
.text:0040105E
.text:00401060
.text:00401062

push    eax
push    offset aS      ; "%5"
call    _scanf
lea     edi, [esp+74h+var_64]
or      ecx, 0FFFFFFFh
xor     eax, eax
add     esp, 0Ch
repne   scasb
not     ecx
dec     ecx
test    ecx, ecx
jle     short loc_40105A

mov     dl, [esp+eax+68h+var_64] ; CODE XREF: _main+58j
cmp     dl, 61h
jl      short loc_4010B3
cmp     dl, 7Ah
jg      short loc_4010B3
inc     eax
cmp     eax, ecx
jl      short loc_401047

push    ebx
push    esi
xor     esi, esi
test    ecx, ecx
jle     short loc_4010B2 ; CODE XREF: _main+45j
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 分析上述汇编代码，可以知道0x401047处的循环为判断用户输入的内容是否有小于0x61（对应字符为‘a’）或大于0x7A（对应字符为‘z’）的情况，有则跳转到0x4010B3处

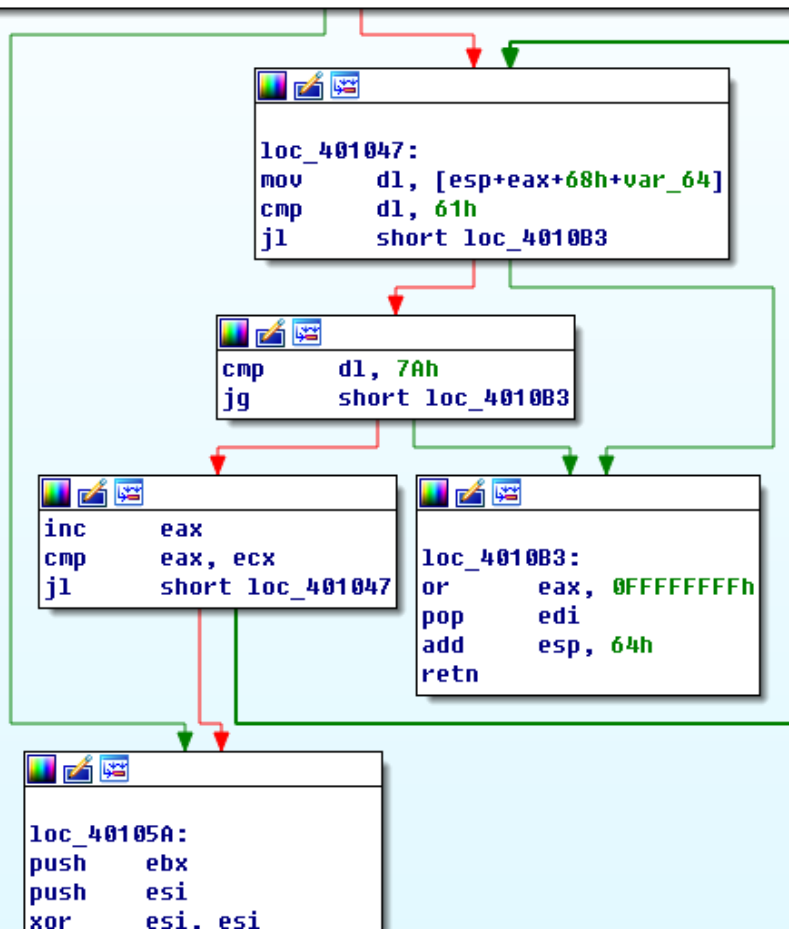
```
.text:004010B3 loc_4010B3:                                ; CODE XREF: _main+4E1j  
.text:004010B3                                ; _main+531j  
.text:004010B3      or      eax, 0FFFFFFFFh  
.text:004010B6      pop     edi  
.text:004010B7      add     esp, 64h  
.text:004010BA      retn  
+out+004010DD +-----
```




第五讲 常见加密算法逆向分析

(1)

```
call    _scanf
lea     edi, [esp+74h+var_64]
or      ecx, 0FFFFFFFh
xor     eax, eax
add     esp, 0Ch
repne   scasb
not     ecx
dec     ecx
test    ecx, ecx
jle     short loc_40105A
```





第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 在对输入进行判断之后便来到仿射加密的关键部分了，这里可以看到几个特殊的值0x1A、0x61等。首先将用户输入的内容逐字节取出，放入寄存器eax中，并将 $eax+eax*2-0x11C=3*eax-0x11C$ 的结果放入eax中；用eax当做被除数，edi当做除数（1A即26），得到的余数再加上0x61（a的ASCII码）后写入内存

```
.text:00401062
.text:00401062 loc_401062:                                ; CODE XREF: _main+80↓j
.text:00401062      movsx    eax, [esp+esi+70h+var_64]
.text:00401067      mov     edi, 1Ah
.text:0040106C      lea     eax, [eax+eax*2-11Ch]
.text:00401073      cdq
.text:00401074      idiv    edi
.text:00401076      add     dl, 61h
.text:00401079      mov     [esp+esi+70h+var_64], dl
.text:0040107D      inc     esi
.text:0040107E      cmp     esi, ecx
.text:00401080      jl      short loc_401062
.text:00401082
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

那么这里如何根据 $3 * \text{eax} - 0x11C$ 来推算出加密密钥 a 、 b 的值呢？结合加密算法： $a * (\text{eax} - 0x61) + b$ ，将其进行因式分解得到： $a * \text{eax} - a * 0x61 + b$ 。这样就能很容易的得到 a 的值为 3，再将 $a = 3$ 带入即可得到 b 的值为 7。

$$- 3 * \text{eax} - 0x11C = a * \text{eax} - a * 0x61 + b$$

$$- a = 3, b = 7$$



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 程序在完成加密后，便将加密得到的结果与 0x401082 处的字符串（密文）进行比较，如果相等，则输出 “ok, you really know”

```
.text:00401082
.text:00401082 loc_401082:                                ; CODE XREF: _main+60↑j
.text:00401082                                     mov     esi, offset aQxbxpluxvwhuzj ; "qxbxpluxvwhuzjct"
.text:00401087                                     lea     eax, [esp+70h+var_64]
.text:0040108B
.text:0040108B loc_40108B:                                ; CODE XREF: _main+AD↓j
.text:0040108B                                     mov     dl, [eax]
.text:0040108D                                     mov     bl, [esi]
.text:0040108F                                     mov     cl, dl
.text:00401091                                     cmp     dl, bl
.text:00401093                                     jnz     short loc_4010BB
.text:00401095                                     test    cl, cl
.text:00401097                                     jz      short loc_4010AF
.text:00401099                                     mov     dl, [eax+1]
.text:0040109C                                     mov     bl, [esi+1]
.text:0040109F                                     mov     cl, dl
.text:004010A1                                     cmp     dl, bl
.text:004010A3                                     jnz     short loc_4010BB
.text:004010A5                                     add     eax, 2
.text:004010A8                                     add     esi, 2
.text:004010AB                                     test    cl, cl
.text:004010AD                                     jnz     short loc_40108B
.text:004010AF
.text:004010AF loc_4010AF:                                ; CODE XREF: _main+97↑j
.text:004010AF                                     xor     eax, eax
.text:004010B1                                     jmp     short loc_4010C0
.text:004010B2
```



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 因此，我们知道了程序加密采用的密钥为 $a=3$ ， $b=7$ ；密文为0x401082处的字符串“qxbxpluxvw huzjct”。



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

对于仿射密码的解密，根据解密公式 $x \equiv (y - b) * (a^{-1} \bmod 26)$ ，需要计算出来 a^{-1} ，这里根据数论相关知识计算出来 $a^{-1} = 9$ 。|



第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

□ 编写

```
#include<stdio.h>
#include<string.h>
int main()
{
    int key_a =3;
    int key_b =7;
    int re_key_a =9;
    char ciphertext[]="qxbxpluxvwhuzjct";
    char result[20]={0};
    int i;
    int len;
    int temp;

    len = strlen(ciphertext);
    for(i=0;i<len;++i){
        temp=ciphertext[i]-'a';
        temp=((temp-key_b+26)*re_key_a)%26;
        result[i]=temp+'a';
    }
    printf("plaintext is:\n%s\n",result);
    system("pause");
}
```




第五章 常见加密算法逆向分析

(1) 简单加密算法逆向分析 — 仿射加密

- 运行该解密函数，得到解密结果

```
plaintext is:  
doyouknowfangshe  
请按任意键继续. . .
```

- 验证其正确性

```
please input a string:  
doyouknowfangshe  
ok, you really know  
请按任意键继续. . .
```