

MD5算法逆向分析

主逻辑

```
loc_413D92:
mov     esi, esp
push    4 ; Count
lea     eax, [ebp+input]
push    eax ; Source
lea     ecx, [ebp+tbuf]
push    ecx ; Destination
call    ds:__imp__strncpy
add     esp, 0Ch
cmp     esi, esp
call    j__RTC_CheckEsp
nop
push    4 ; inLen
lea     eax, [ebp+tbuf]
push    eax ; input
lea     ecx, [ebp+resultBuf]
push    ecx ; out
call    j_?md5_calc@@YAXPADPBDI@Z ; md5_calc(char *,char const *,uint)
add     esp, 0Ch
mov     esi, esp
push    20h ; ' ' ; MaxCount
push    offset Str2 ; "fae8a9257e154175da4193dbf6552ef6"
lea     eax, [ebp+resultBuf]
push    eax ; Str1
call    ds:__imp__strncmp
add     esp, 0Ch
cmp     esi, esp
call    j__RTC_CheckEsp
test    eax, eax
jnz     short loc_413DFA
```

```
1"
push    offset aCorrect ; "Correct!\n"
call    j__printf
add     esp, 4
jmp     short loc_413E07
```

```
loc_413DFA:
push    offset aWrongTryAgain ; "Wrong,try again!\n"
call    j__printf
```

以 call 为分界，分为以下几个步骤：

1. 从 input 取前4个字节存入 tbuf ；
2. 调用 md5_calc() 函数，参数为输出 resultBuf、输入 tbuf、输入长度 4 ；
3. 比较 resultBuf 和 Str2 ，相同则成功。
4. 破解：只处理前四个字符，可爆破。

md5_calc() 分析

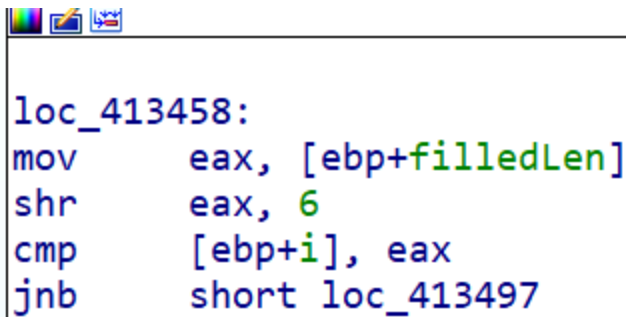
1. 调用 md5_update(), 参数为 inLen、input、filledData(out), 返回值放入 filledLen;

```
mov     eax, [ebp+inLen]
push    eax                ; inLen
mov     ecx, [ebp+input]
push    ecx                ; input
lea     edx, [ebp+filledData]
push    edx                ; out
call    j_?md5_update@@YAIPAPADPBDI@Z
add     esp, 0Ch
mov     [ebp+filledLen], eax
```

2. 初始化, 调 md5_init();

```
lea     eax, [ebp+D]
push    eax                ; D
lea     ecx, [ebp+C]
push    ecx                ; C
lea     edx, [ebp+B]
push    edx                ; B
lea     eax, [ebp+A]
push    eax                ; A
call    j_?md5_init@@YAXPAI000@Z ; md5_init(uint *,uint *,uint *,uint *)
```

3. 循环 filledLen / 64 次 (右移6位实现) 处理;



```
loc_413458:
mov     eax, [ebp+filledLen]
shr     eax, 6
cmp     [ebp+i], eax
jnb     short loc_413497
```

4. 处理过程: 先调用 md5_transform(), 参数为 filledData[i*64](in) (左移6位实现, 一次传64字节) 和 M(out), 再调用 data_round(), 参数为 M 和四个幻数;

```

mov     eax, [ebp+i]
shl     eax, 6
add     eax, [ebp+filledData]
push    eax                ; input
mov     ecx, [ebp+M]
push    ecx                ; out
call    j_?md5_transform@@YAXPAIPAD@Z ; r
add     esp, 8
mov     eax, [ebp+M]
push    eax                ; m
lea     ecx, [ebp+D]
push    ecx                ; D
lea     edx, [ebp+C]
push    edx                ; C
lea     eax, [ebp+B]
push    eax                ; B
lea     ecx, [ebp+A]
push    ecx                ; A
call    j_?data_round@@YAXPAI000PBI@Z ; (
add     esp, 14h
jmp     short loc_41344F

```

5. 对四个幻数调用 `shift()` 处理，然后依次写入 `out`。

```

mov     eax, [ebp+D]
push    eax                ; a
call    j_?shift@@YAIIZ ; shift(uint)
add     esp, 4
push    eax
mov     ecx, [ebp+C]
push    ecx                ; a
call    j_?shift@@YAIIZ ; shift(uint)
add     esp, 4
push    eax
mov     edx, [ebp+B]
push    edx                ; a
call    j_?shift@@YAIIZ ; shift(uint)
add     esp, 4
push    eax
mov     eax, [ebp+A]
push    eax                ; a
call    j_?shift@@YAIIZ ; shift(uint)
add     esp, 4
push    eax
push    offset _Format     ; "%08x%08x%08x%08x"
mov     ecx, [ebp+out]
push    ecx                ; _Buffer
call    j__sprintf

```

md5_update() 分析

1. 计算输入（原始消息）对应的bit数（长度*8，右移3位实现）：

```

mov     eax, [ebp+inLen]
shl     eax, 3
mov     [ebp+bitLen], eax

```

2. 计算 $56 - \text{inLen} \% 64$ 存入 `fillLen`，即需要填充的长度，并计算出最终消息长度
`filledLen = inLen + fillLen + 8`；（单位都为字节）

```

mov     eax, [ebp+inLen]
xor     edx, edx
mov     ecx, 40h ; '@'
div     ecx
mov     eax, 38h ; '8'
sub     eax, edx
mov     [ebp+fillLen], eax
mov     eax, [ebp+fillLen]
mov     ecx, [ebp+inLen]
lea     edx, [ecx+eax+8]
mov     [ebp+filledLen], edx

```

3. 请求 filledLen 大小的空间 outBuf，并将原消息写入，同时将 0x80 作为填充的第一个字节的内容；

```

push    1 ; Size
mov     eax, [ebp+filledLen]
push    eax ; Count
call    ds:__imp__calloc
add     esp, 8
cmp     esi, esp
call    j__RTC_CheckEsp
mov     [ebp+outBuf], eax
mov     eax, [ebp+inLen]
push    eax ; Size
mov     ecx, [ebp+input]
push    ecx ; Src
mov     edx, [ebp+outBuf]
push    edx ; void *
call    j__memcpy
add     esp, 0Ch
mov     eax, [ebp+outBuf]
add     eax, [ebp+inLen]
mov     byte ptr [eax], 80h ; '€'

```

4. 添加最后的数据长度部分，将 bitLen 值的每个字节（bitLen[i]）依次复制到 outBuf 的对应位置（outLen[filledLen+i-8]，filledLen 为最后一位，减去8是这八个字节对应填写内容的地方，最后 +i 用于指定当前位置）上；

```

mov     [ebp+j], 4
mov     [ebp+i], 0
jmp     short loc_41387D

```

```

loc_41387D:
mov     eax, [ebp+i]
cmp     eax, [ebp+j]
jge     short loc_4138A7

```

```

push    1 ; Size
mov     eax, [ebp+i]
lea     ecx, [ebp+eax+bitLen]
push    ecx ; Src
mov     edx, [ebp+outBuf]
add     edx, [ebp+filledLen]
mov     eax, [ebp+i]
lea     ecx, [edx+eax-8]
push    ecx ; void *
call    j__memcpy
add     esp, 0Ch
jmp     short loc_413874

```

```

loc_4138A7:
mov     eax, [ebp+out]
mov     ecx, [ebp+outBuf]
mov     [eax], ecx
mov     eax, [ebp+filledLen]
push    edx
mov     ecx, ebp
push    eax
lea     edx, stru_4138E4
call    j_@_RTC_CheckSta
pop     eax
pop     edx

```

注：这里源代码使用的是 `sizeof(size_t)`，用的 `x86` 编译的所以显示的 `j` 为4，正确的应该是8（64bit存储原消息长度）

5. 最后将 `outBuf` 的首地址作为 `out` 的值（输出过程），并返回 `filledLen`。

```

mov     eax, [ebp+out]
mov     ecx, [ebp+outBuf]
mov     [eax], ecx
mov     eax, [ebp+filledLen]
push    edx

```

md5_init() 分析

初始化四个幻数：A、B、C、D。

```
//int A=0x67452301  
//int B=0xEFCDAB89  
//int C=0x98BADCFE  
//int D=0x10325476
```

```

mov     [ebp+data], 1
mov     [ebp+data+1], 23h ; '#'
mov     [ebp+data+2], 45h ; 'E'
mov     [ebp+data+3], 67h ; 'g'
mov     [ebp+data+4], 89h
mov     [ebp+data+5], 0ABh
mov     [ebp+data+6], 0CDh
mov     [ebp+data+7], 0EFh
mov     [ebp+data+8], 0FEh
mov     [ebp+data+9], 0DCh
mov     [ebp+data+0Ah], 0BAh
mov     [ebp+data+0Bh], 98h
mov     [ebp+data+0Ch], 76h ; 'v'
mov     [ebp+data+0Dh], 54h ; 'T'
mov     [ebp+data+0Eh], 32h ; '2'
mov     [ebp+data+0Fh], 10h
push    4 ; Size
lea     eax, [ebp+data]
push    eax ; Src
mov     ecx, [ebp+A]
push    ecx ; void *
call    j__memcpy
add     esp, 0Ch
push    4 ; Size
lea     eax, [ebp+data+4]
push    eax ; Src
mov     ecx, [ebp+B]
push    ecx ; void *
call    j__memcpy
add     esp, 0Ch
push    4 ; Size
lea     eax, [ebp+data+8]
push    eax ; Src
mov     ecx, [ebp+C]
push    ecx ; void *
call    j__memcpy
add     esp, 0Ch
push    4 ; Size
lea     eax, [ebp+data+0Ch]
push    eax ; Src
mov     ecx, [ebp+D]
push    ecx ; void *
call    j__memcpy

```


md5_transform() 分析

数组赋值，一次读64字节内容。

```
push    40h ; '@' ; Size
mov     eax, [ebp+input]
push    eax ; Src
mov     ecx, [ebp+out]
push    ecx ; void *
call    j__memcpy
```

data_round() 分析

说实话写的挺抽象，直接手敲了4大轮、64小轮的迭代（所有的索引都手敲好了不用算），逻辑非常好分析，就是套用加密给出的公式，放一段：

```
mov     eax, 4
imul    ecx, eax, 0
mov     edx, ds:S[ecx]
push    edx ; s
mov     eax, 4
imul    ecx, eax, 0
mov     edx, ds:T[ecx]
push    edx ; t
mov     eax, 4
imul    ecx, eax, 0
mov     edx, [ebp+m]
mov     eax, [edx+ecx]
push    eax ; m
mov     ecx, [ebp+d]
push    ecx ; d
mov     edx, [ebp+c]
push    edx ; c
mov     eax, [ebp+b]
push    eax ; b
lea     ecx, [ebp+a]
push    ecx ; a
call    j_?FF@@YAXPAIIIIII@Z ; FF(uin
```

其中 FF() 如下图，首先将 B、C、D 作为参数传入 F 函数，结果再与 m（明文）、t（T盒）、A 相加，然后把 s（S盒当前轮次值）和结果传入 ROTATE_LEFT（看起来就是循环左移），最后再加上 B 作为输出。

```

mov     eax, [ebp+d]
push    eax                ; Z
mov     ecx, [ebp+c]
push    ecx                ; Y
mov     edx, [ebp+b]
push    edx                ; X
call    j_?F@@YAIIII@Z    ; F(uint,uint,uint)
add     esp, 0Ch
add     eax, [ebp+m]
add     eax, [ebp+t]
mov     ecx, [ebp+a]
add     eax, [ecx]
mov     edx, [ebp+a]
mov     [edx], eax
mov     eax, [ebp+s]
push    eax                ; n
mov     ecx, [ebp+a]
mov     edx, [ecx]
push    edx                ; x
call    j_?ROTATE_LEFT@@YAIIII@Z ; ROTATE_LEFT(uint,uint)
add     esp, 8
mov     ecx, [ebp+a]
mov     [ecx], eax
mov     eax, [ebp+a]
mov     ecx, [eax]
add     ecx, [ebp+b]
mov     edx, [ebp+a]
mov     [edx], ecx

```

F() 函数如下图，即 $(X \& Y) \mid ((\sim X) \& Z)$ 。

```

mov     eax, [ebp+X]
and     eax, [ebp+Y]
mov     ecx, [ebp+X]
not     ecx
and     ecx, [ebp+Z]
or      eax, ecx

```

ROTATE_LEFT() 如下图，即先将 x 左移 n 存入 eax，再将 x 右移 32 - x，存入 edx，eax 再和 edx 做或操作，实现循环左移 n 位。

```

mov     eax, [ebp+x]
mov     ecx, [ebp+n]
shl     eax, cl
mov     ecx, 20h ; ' '
sub     ecx, [ebp+n]
mov     edx, [ebp+x]
shr     edx, cl
or      eax, edx

```

G、H、I 系列同上，就是函数和一些取值不一样。

shift() 分析

```
mov     eax, [ebp+a]
and     eax, 0FFh
shl     eax, 18h
mov     [ebp+t1], eax
mov     eax, [ebp+a]
and     eax, 0FF00h
shl     eax, 8
mov     [ebp+t2], eax
mov     eax, [ebp+a]
shr     eax, 8
and     eax, 0FF00h
mov     [ebp+t3], eax
mov     eax, [ebp+a]
shr     eax, 18h
and     eax, 0FFh
mov     [ebp+t4], eax
mov     eax, [ebp+t1]
add     eax, [ebp+t2]
add     eax, [ebp+t3]
add     eax, [ebp+t4]
```

1. 取 a（8字节）和 FFh 做与，取低8位，然后左移 18h(24) 位，存到 t1 里，再做 a & FF00h 即取16位到9位（从低位开始数），左移8位存到 t2，同理将 a 的高8位放到 t3，24到17位（从低位开始数）放到 t4；
2. 将四个 t? 相加，完成运算，实际上就是完成了小端序向大端序的转换。