# 2021/03/24_x86逆向_第15课_字符串

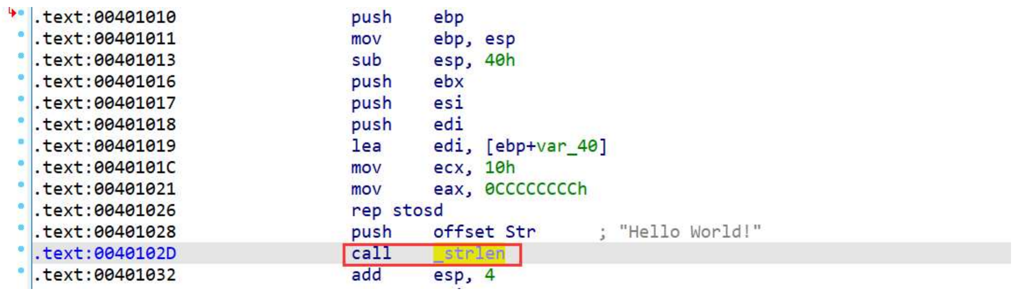
**笔记本:** x86逆向-C
**创建时间:** 2021/3/24 星期三 12:54
**作者:** ileemi

## strlen

### VC++ 6.0

> Debug

没有优化，通过汇编代码可以看到 "strlen" 函数的使用，如下图所示：



```
.text:00401010                 push    ebp
.text:00401011                 mov     ebp, esp
.text:00401013                 sub     esp, 40h
.text:00401016                 push    ebx
.text:00401017                 push    esi
.text:00401018                 push    edi
.text:00401019                 lea     edi, [ebp+var_40]
.text:0040101C                 mov     ecx, 10h
.text:00401021                 mov     eax, 0CCCCCCCCh
.text:00401026                 rep stosd
.text:00401028                 push    offset Str      ; "Hello World!"
.text:0040102D                 call    _strlen
.text:00401032                 add     esp, 4
```

> Release

"strlen" 函数的实现代码被内联到调用的位置，做到了无分支求字符串长度，代码示例：

```
int main(int argc, char* argv[]) {
    return strlen("Hello World\n");
}

// VC++ 6.0 Release strlen 求字符串长度的定式
mov edi, offset aHelloWorld ; "Hello World"
or  ecx, 0FFFFFFFFh  ; ecx 赋值为无符号的最大值，表示循环的次数
```

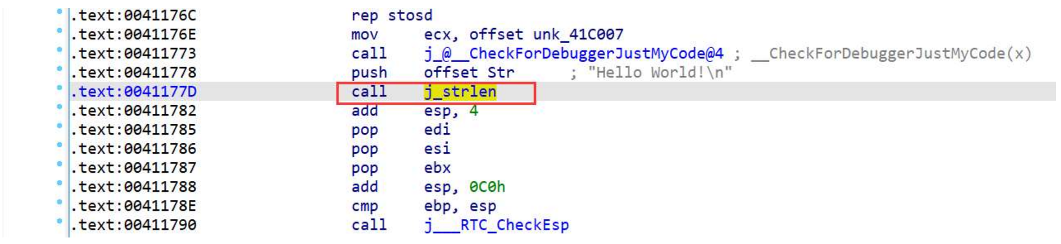
```
xor  eax,  eax  ;  eax  =  0
repne  scasb  ;  scasb  寻找  al,  与al不相等就重复循环 (ecx-1,  edi+1)
not  ecx
dec  ecx

// ecx  和字符串长度的关系(数学推导)
ecx  =  0xFFFFFFFF  =  -1
循环的次数:
  length  +  1
  ecx  =  -1  -  length  -  1
  ecx  =  -length  -  2
  -ecx  =  length  +  2
  -ecx  -  2  =  length
  neg(ecx)  -  2  =  length
  not(ecx)  +  1  -  2  =  length
  not(ecx)  -  1  =  length
```

## VS 2019

> Debug

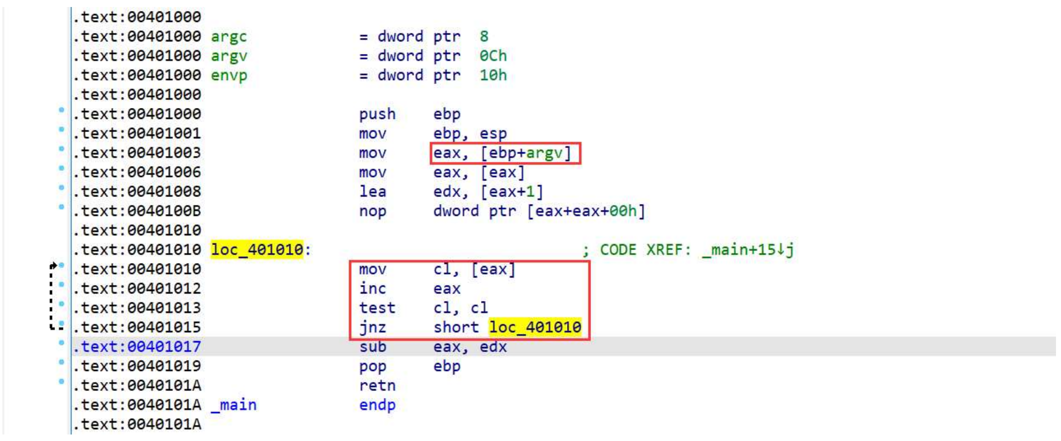
没有优化，通过汇编代码可以看到 "strlen" 函数的使用，如下图所示：



```
.text:0041176C                 rep stosd
.text:0041176E                 mov     ecx, offset unk_41C007
.text:00411773                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00411778                 push    offset Str      ; "Hello World!\n"
.text:0041177D                 call    j_strlen
.text:00411782                 add     esp, 4
.text:00411785                 pop     edi
.text:00411786                 pop     esi
.text:00411787                 pop     ebx
.text:00411788                 add     esp, 0C0h
.text:0041178E                 cmp     ebp, esp
.text:00411790                 call    j___RTC_CheckEsp
```

> Release

使用了循环，每次从字符串中取出一个字符，当寄存器指向字符串的末尾时（等于0），循环结束，如下图所示：



```
.text:00401000
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000 envp            = dword ptr  10h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 mov     eax, [ebp+argv]
.text:00401006                 mov     eax, [eax]
.text:00401008                 lea     edx, [eax+1]
.text:0040100B                 nop     dword ptr [eax+eax+00h]
.text:00401010
.text:00401010 loc_401010:                             ; CODE XREF: _main+15↓j
.text:00401010                 mov     cl, [eax]
.text:00401012                 inc     eax
.text:00401013                 test    cl, cl
.text:00401015                 jnz     short loc_401010
.text:00401017                 sub     eax, edx
.text:00401019                 pop     ebp
.text:0040101A                 retn
.text:0040101A _main           endp
.text:0040101A
```

# strcpy

## VC++ 6.0

> Debug

没有优化，通过汇编代码可以看到 "strcpy" 函数的使用。

> Release

进行了优化，先使用无分支求长度，而后拷贝，代码示例：

```
push 200            ; Size
call ??2@YAPAXI@Z   ; operator new(uint)
mov edx, eax
mov eax, [esp+0Ch+argv]
or ecx, 0FFFFFFFFh
push edx            ; lpMem
mov edi, [eax]
xor eax, eax
repne scasb
not ecx ; 求长度 ecx = strlen(str) + 1, + 1 和 dec  ecx 抵消
sub edi, ecx
mov eax, ecx
mov esi, edi
mov edi, edx
shr ecx, 2 ; ecx / 4, 求出目标字符串有多少个4字节
rep movsd ; 循环拷贝，每次4个字节
mov ecx, eax
and ecx, 3 ; ecx % 4, 求最后的字符串余数
rep movsb ; 拷贝剩余的字节数
```

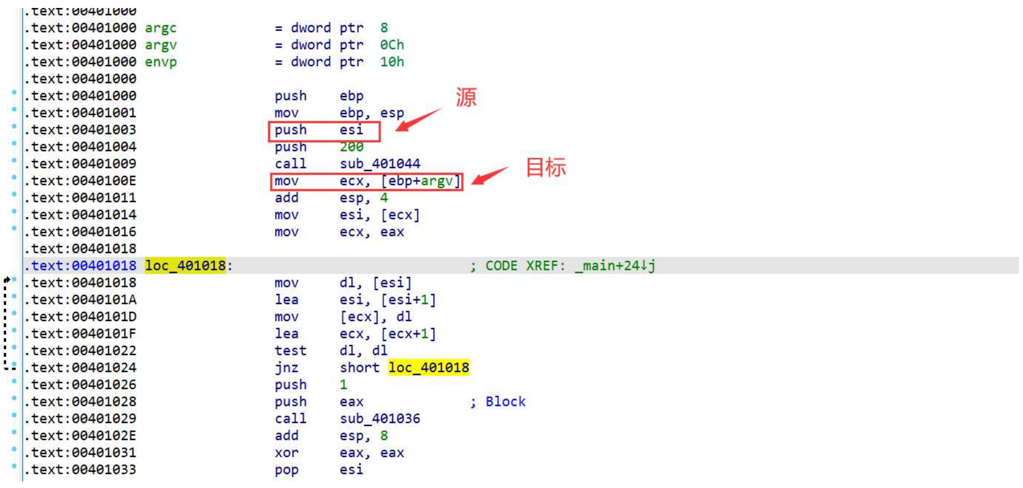
## VS 2019

> Debug

没有优化，通过汇编代码可以看到 "strcpy" 的使用

> Release

使用了循环结构，每次重源地址中拷贝一个字节数据到目标地址上，如下图所示：



```
.text:00401000
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000 envp            = dword ptr  10h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    esi
.text:00401004                 push    200
.text:00401009                 call    sub_401044
.text:0040100E                 mov     ecx, [ebp+argv]
.text:00401011                 add     esp, 4
.text:00401014                 mov     esi, [ecx]
.text:00401016                 mov     ecx, eax
.text:00401018
.text:00401018 loc_401018:                             ; CODE XREF: _main+24↓j
.text:00401018                 mov     dl, [esi]
.text:0040101A                 lea     esi, [esi+1]
.text:0040101D                 mov     [ecx], dl
.text:0040101F                 lea     ecx, [ecx+1]
.text:00401022                 test    dl, dl
.text:00401024                 jnz     short loc_401018
.text:00401026                 push    1
.text:00401028                 push    eax             ; Block
.text:00401029                 call    sub_401036
.text:0040102E                 add     esp, 8
.text:00401031                 xor     eax, eax
.text:00401033                 pop     esi
```

## strcmp

### VC++ 6.0

> Debug

没有优化，通过汇编代码可以看到 "strcmp" 函数的使用。

> Release

有两个 "sub" 指令的结尾特征，使用分支结构逐字节对比，代码形式如下：
sbb eax, eax
sbb eax, 0FFFFFFFFh

```
.text:00401010          mov          eax,  [esp+14h+argv]
...
.text:0040101A          mov          edi,  [eax]
...
.text:00401038          mov          esi,  offset  aHelloWorld  ;  "Hello

.text:0040103D
.text:0040103D  loc_40103D:
.text:0040103D          mov          dl,  [eax]
.text:0040103F          mov          bl,  [esi]
.text:00401041          mov          cl,  dl
//  逐字节比较
.text:00401043          cmp          dl,  bl
.text:00401045          jnz          short  loc_401065
.text:00401047          test        cl,  cl
.text:00401049          jz            short  loc_401061
.text:0040104B          mov          dl,  [eax+1]
.text:0040104E          mov          bl,  [esi+1]
.text:00401051          mov          cl,  dl
.text:00401053          cmp          dl,  bl
```

```
.text:00401055          jnz         short  loc_401065
.text:00401057          add          eax,   2
.text:0040105A          add          esi,   2
//  判断是否到字符串末尾的  '\0',  不是继续比较
.text:0040105D          test        cl,   cl
.text:0040105F          jnz          short  loc_40103D
.text:00401061
.text:00401061  loc_401061:
//  eax  =  0  返回0，两个字符串相等
.text:00401061          xor          eax,   eax
.text:00401063          jmp          short  loc_40106A
.text:00401065  ;  --------------------------------
.text:00401065  loc_401065:
.text:00401065
//  eax  =  eax  -  eax;  //  eax  =  0;
.text:00401065          sbb          eax,   eax
//  eax  =  eax  -  (-1);  //  eax  =  0  +  1  =  1
.text:00401067          sbb          eax,   0FFFFFFFFh
.text:0040106A
.text:0040106A  loc_40106A:
.text:0040106A          test        eax,   eax
.text:0040106C          jz           short  loc_40107B
.text:0040106E          push        offset  Format      ;  "ok\n"
.text:00401073          call        _printf
.text:00401078          add          esp,   4
.text:0040107B
.text:0040107B  loc_40107B:
.text:0040107B          push        ebp                          ;  lpMem

.text:0040107C          call        sub_401090
.text:00401081          add          esp,   4
.text:00401084          xor          eax,   eax
.text:00401086          pop          edi
.text:00401087          pop          esi
.text:00401088          pop          ebp
.text:00401089          pop          ebx
.text:0040108A          retn
.text:0040108A  _main  endp
```

## VS 2019

> Debug

没有优化，通过汇编代码可以看到 "strcmp" 函数的使用。

> Release

有一个 "sub" 和一个 "or" 指令的结尾特征，使用分支结构逐字节对比，代码形式如下：
sbb eax, eax
or eax, 1

```
.text:0040101F          mov         ecx,  [ebp+argv]
.text:00401022          mov         edi,  eax
.text:00401024          add         esp,  4
.text:00401027          mov         esi,  edi
.text:00401029          mov         edx,  [ecx]
.text:0040102B          nop         dword  ptr  [eax+eax+00h]
.text:00401030  loc_401030:
.text:00401030          mov         cl,  [edx]
.text:00401032          lea         edx,  [edx+1]
.text:00401035          mov         [esi],  cl
.text:00401037          lea         esi,  [esi+1]
.text:0040103A          test       cl,  cl  //  判断是否到达末尾
.text:0040103C          jnz         short  loc_401030
.text:0040103E          mov         ecx,  offset  aHelloWorld  ;  "Hello

.text:00401043
.text:00401043  loc_401043:
.text:00401043          mov         dl,  [eax]
.text:00401045          cmp         dl,  [ecx]  //  逐字节比较
.text:00401047          jnz         short  loc_401063
.text:00401049          test       dl,  dl
.text:0040104B          jz           short  loc_40105F
.text:0040104D          mov         dl,  [eax+1]
.text:00401050          cmp         dl,  [ecx+1]
.text:00401053          jnz         short  loc_401063
.text:00401055          add         eax,  2
.text:00401058          add         ecx,  2
.text:0040105B          test       dl,  dl
.text:0040105D          jnz         short  loc_401043
.text:0040105F
.text:0040105F  loc_40105F:
.text:0040105F          xor         eax,  eax  //  两个字符相等
.text:00401061          jmp         short  loc_401068
.text:00401063  ;  -----------------------------------
.text:00401063  loc_401063:
.text:00401063          sbb         eax,  eax  //  eax  =  0
.text:00401065          or           eax,  1  //  eax  =  1
.text:00401068  loc_401068:
.text:00401068          test       eax,  eax
.text:0040106A          jz           short  loc_401079
```

```
.text:0040106C          push      offset  Format      ;  "ok\n"
.text:00401071          call      _printf
.text:00401076          add        esp,   4
.text:00401079
.text:00401079  loc_401079:
.text:00401079          push      1
.text:0040107B          push      edi                         ;  Block

.text:0040107C          call      _delete
.text:00401081          add        esp,   8
.text:00401084          xor        eax,   eax
.text:00401086          pop        edi
.text:00401087          pop        esi
.text:00401088          pop        ebp
.text:00401089          retn
.text:00401089  _main   endp
```