

第八章

本章,我们将来看看前面章节忽略了一些重要的指令。我们学习完了这部分,就可以开始破解一些小玩意儿了。

循环指令

为了实现循环可以使用前面介绍过一些指令。例如,你可以将任意通用寄存器指定为计数器(通常 **ECX** 作为计数器使用),你可以将其初始化为需要循环的次数,然后执行循环体,接着计数器递减 1,判断计数器是否为 0,如果计数器不为 0 继续重复前面的过程,如果计数器为 0,就不继续循环了,而直接执行下面的代码。代码如下:

```
XOR ECX,ECX
```

```
MOV ECX,15h
```

将计数器初始化为循环次数 15h。接下来就是循环体了:

```
Label:
```

```
DEC ECX
```

该计数器每次递减 1。

其实就是循环体了,循环体里面可以是任意指令。

最后,你需要添加一个判断计数器是否为 0 的指令以及条件跳转指令。

```
CMPECX,0
```

```
JNE Label
```

第一次判断,计数器的值为 14h,因为 14h 不为 0,所以将继续执行循环,以此类推,直到计数器为 0 为止。

我们完整的来写一遍上面的循环:

```
XOR ECX,ECX
```

```
ADD ECX,15h
```

```
Label:
```

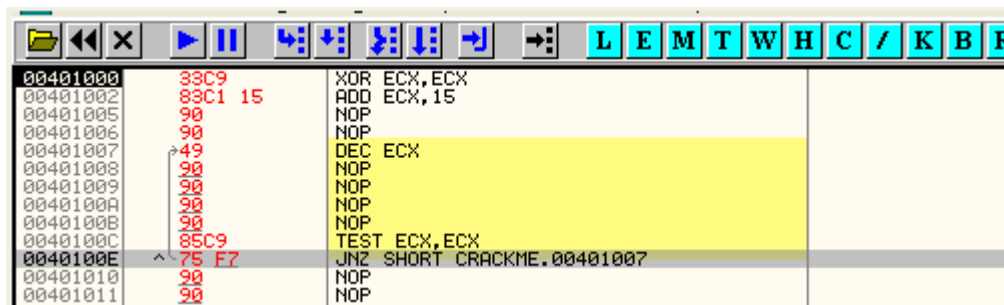
```
DEC ECX
```

```
;循环体
```

```
TEST ECX,ECX
```

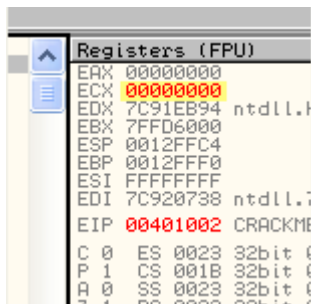
```
JNE Label
```

让我们在 OD 上输入上面的代码:



黄色突出显示的是循环代码,这个部分将重复执行,直到 **ECX** 的值为 0。循环体里面包含了多个 **NOP** 指令。

我们来单步跟踪一下循环代码,亲眼看一下循环的执行过程。按一下 **F7** 键看看 **ECX** 是怎么初始化的。



再次按 F7 键,计数器将变成 15h,保存了需要循环的次数。

Registers (FPU)	
EAX	00000000
ECX	00000015
EDX	7C91EB94 ntd
EBX	7FFD6000
ESP	0012FFC4
EBP	0012FFFF
ESI	FFFFFFFF
EDI	7C920738 ntd
EIP	00401005 CRA
C 0	ES 0023 32b
P 0	CS 001B 32b
D 0	SS 0023 32b

然后继续按 F7 键,直到 DEC ECX,这个时候我们的计数器的值将减少至 14h。

继续 F7 单步直到 TEST ECX,ECX。该指令判断 ECX 是否为 0,为 0 零标志位 Z 就置 1,这样就停止循环,否则将继续循环。

Registers (FPU)	
EAX	00000000
ECX	00000015
EDX	7C91EB94 ntd
EBX	7FFD6000
ESP	0012FFC4
EBP	0012FFFF
ESI	FFFFFFFF
EDI	7C920738 ntd
EIP	00401005 CRA
C 0	ES 0023 32b
P 0	CS 001B 32b
D 0	SS 0023 32b

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	49	DEC ECX
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	85C9	TEST ECX,ECX
0040100E	75 F7	JNZ SHORT CRACKME.00401007
00401010	90	NOP
00401011	90	NOP

因为此时计数器不为 0,零标志位 Z 没有置 1,所以条件跳转指令 JNZ 将跳转到 401007 地址处。下一步计数器继续递减 1,这一次减少至 13h。继续单步跟踪直到计数器为 0。

当计数器为 0 时,零标志位 Z 将置 1,这个时候 JNZ 将不会跳转至 401007 地址处,而是继续向下执行。

EDI	7C920738 ntdll.7C9
EIP	0040100E CRACKME.0
C 0	ES 0023 32bit 0(F
P 1	CS 001B 32bit 0(F
A 0	SS 0023 32bit 0(F
Z 1	DS 0023 32bit 0(F
S 0	FS 003B 32bit 7FF
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_MO
EFL	00000246 (NO,NB,E
ST0	empty -UNORM BCE0
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 1.0000000000

这里需要注意一下,JNZ 指令与我们前面使用过的 JZ 刚好相反,当零标志位 Z 为 0 的时候跳转,为 1 的时候不跳转。

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	49	DEC ECX
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	85C9	TEST ECX,ECX
0040100E	75 F7	JNZ SHORT CRACKME.00401007
00401010	90	NOP
00401011	90	NOP
00401012	90	NOP
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	90	NOP

见到灰色箭头,意味着什么?这里意味着跳转不会发生。按 F7 键我们将跳出循环继续执行下面的代码。

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	49	DEC ECX	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	85C9	TEST ECX,ECX	
0040100E	75 F7	JNZ SHORT CRACKME.00401007	
00401010	90	NOP	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	FindWindowA
00401018	0000	OR Fox Fox	

我们使用熟悉的指令模拟了一个最简单的循环例子。其实有专门用于循环的指令。我们来看一看。

LOOP

LOOP 指令可以帮助我们完成前面例子中的事情- 将计数器 ECX 的值减 1,判断 ECX 的值是否为 0,如果为 0 就跳转到指定的地址- 将像前面的例子一样。(可惜的是,大多数现代的处理器的效率不如前面模拟的例子。)

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	90	NOP	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	E2 F9	LOOPE SHORT CRACKME.00401007	
0040100E	90	NOP	
0040100F	90	NOP	
00401010	90	NOP	

在 DEC ECX 指令上单击鼠标右键选择-Binary-Fill with NOPS。对 TEST ECX,ECX 和 JNZ 401007 两条指令也进行同样的操作。这三条指令用一条 LOOP 401007 指令替代。

第一行突出显示(401000)-单击鼠标右键选择-New origin here。现在我们来执行新的 LOOP 指令吧。

按 F7 键,再次看到计数器 ECX 首先被初始化为 0 了,然后又被设置为 15h 了。我们继续单步跟踪,直到 LOOP 指令。

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	90	NOP	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	E2 F9	LOOPE SHORT CRACKME.00401007	
0040100E	90	NOP	
0040100F	90	NOP	
00401010	90	NOP	

这里还是像之前一样,跳转至 401007 地址处,因为计数器不为 0。这里,计数器递减 1,现在为 14h。

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	90	NOP	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	E2 F9	LOOPE SHORT CRACKME.00401007	
0040100E	90	NOP	
0040100F	90	NOP	

继续单步跟踪。当计数器为 0 时,循环将结束。

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	E2 F9	LOOPD SHORT CRACKME.00401007
0040100E	90	NOP
0040100F	9A	NOP

这里还有一些与 LOOP 指令相关的指令:

LOOPZ, LOOPE 重复循环,直到零标志位 Z 置 1

LOOPNZ, LOOPNE 重复循环,直到零标志位 Z 清 0

这几条指令同样是循环指令,即重复循环体,直到计数器为 0,每次循环将计数器的值递减 1。此外,LOOPZ,LOOPNZ 指令还将检查零标志位 Z 是否为 0。只有计数器的值和零标志 Z 同时满足条件时才循环。

接着,我们来介绍一下字符串操作类指令。

串操作

下面,介绍一下串操作类的基本指令

MOVS

该指令是从一个地址向另一个地址复制数据。源地址保存在 ESI 寄存器中,目的地址保存在 EDI 寄存器中。我们不需要显示地指定参数。现在我们在 OD 中写入 MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]指令,样子如下:

00401000	BE 6C364000	MOV ESI,CRACKME.0040366C
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C
0040100A	A5	MOV DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0040100B	90	NOP
0040100C	6A 9A	PUSH -9A

这里我们用 ESI 来保存源地址,EDI 来保存目的地址。

在拷贝内容之前,让我们在数据窗口中来看一看源地址和目的地址的情况。

你可以在数据窗口中,单击鼠标右键选择-Go to Expression,然后输入地址 40366C。还有一个更加简单的方法:在第一条指令上面单击鼠标右键选择-Follow in Dump-Immediate constant,如下图所示:

00401000 BE 6C364000 MOV ESI,CRACKME.0040366C

00401005 BF 9C364000 MOV EDI,CRACKME.0040369C

0040100A A5 MOV DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]

0040100B 90 NOP

0040100C 6A 9A PUSH -9A

0040100D 68 F4204000 PUSH CRACKME.004020F4

00401010 E8 A0400000 CALL <JMP.&USER32.FindWindow

00401011 90 NOP

00401012 74 01 JE SHORT CRACKME.0040101D

00401013 C3 RETN

00401014 C705 64204000 MOV DWORD PTR DS:[402064],40

00401015 C705 6C204000 MOV DWORD PTR DS:[40206C],0

00401016 C705 70204000 MOV DWORD PTR DS:[402070],0

00401017 A1 CA204000 MOV EAX,DWORD PTR DS:[4020CA]

00401018 74 204000 OR EAX,0

00401019 6A 64 PUSH 64

0040101A 50 PUSH EAX

0040101B E8 D1030000 CALL <JMP.&USER32.LoadIconA>

0040101C A3 78204000 MOV DWORD PTR DS:[402078],EAX

0040101D 68 007F0000 PUSH 7F00

0040101E 6A 00 PUSH 0

0040101F E8 A2030000 CALL <JMP.&USER32.LoadCursor

00401020 A3 7C204000 MOV DWORD PTR DS:[40207C],EAX

00401021 C705 80204000 MOV DWORD PTR DS:[402080],5

00401022 C705 84204000 MOV DWORD PTR DS:[402084],0

00401023 C705 88204000 MOV DWORD PTR DS:[402088],0

00401024 68 64204000 PUSH CRACKME.00402064

00401025 E8 F0030000 CALL <JMP.&USER32.RegisterClassExA>

00401026 6A 00 PUSH 0

00401027 FF35 CA204000 PUSH DWORD PTR DS:[4020CA]

00401028 6A 00 PUSH 0

00401029 6A 00 PUSH 0

0040102A 68 00000000 PUSH 0000

0040366C=CRACKME.0040366C (ASCII "lgR")

Address	Hex dump	ASCII
0040366C	6C 67 41 00 00 00 00 00	lgR.....
00403670	00 00 00 00 00 00 00 00
00403674	45 00 00 00 00 00 00 00	E.....
00403678	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403680	00 00 00 00 00 00 00 00
00403684	00 00 00 00 00 00 00 00
00403688	00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00
00403690	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
00403698	00 00 00 00 00 00 00 00
0040369C	00 00 00 00 00 00 00 00
004036A0	00 00 00 00 00 00 00 00
004036A4	00 00 00 00 00 00 00 00
004036A8	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B0	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036B8	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C0	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036C8	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D0	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00
004036D8	00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00

源地址(ESI)中指定的地址单元中的待拷贝的内容。

目的地址(EDI)指向的地址单元

Address	Hex dump	ASCII
0040369C	00 00 00 00 00 00 00 00	
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00
004036E4	00 00 00 00 00 00 00 00
004036EC	00 00 00 00 00 00 00 00
004036F4	00 00 00 00 00 00 00 00
004036FC	00 00 00 00 00 00 00 00

目的地址指向的地址单元将保存拷贝过来的内容。

按 F7 键,执行 MOVSB 指令,我们可以看到 4 个字节被拷贝了。

Address	Hex dump	ASCII
0040369C	6C 67 41 00 00 00 00 00	lgA.....
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00

这里 MOVSB 拷贝 4 个字节的内容,即 DWORD,另外一种书写形式为:MOVSD。与之对应的还有拷贝两个字节的 MOVSW 指令和拷贝一个字节的 MOVSB 指令。

请注意 ESI,EDI 拷贝的方向,拷贝的方向取决于方向标志位 D。

REP

该指令可做为前面介绍的一些指令的前缀,尤其是 MOVSB 指令。该前缀表示当前指令需要执行的次数 ECX。每次循环计数器 ECX 的值递减 1,和前面介绍的循环一样。

因此,REP MOVSB 不一定拷贝是 4 个字节,它拷贝的大小为 每次拷贝的大小 * ECX, 源指针 ESI 和目的指针 EDI 每次递增 4 或者递减 4(递增或递减取决于方向标志位 D)。该指令看起来很实用,是不是?

该指令可以配合前面介绍的指令实现从一个地址单元拷贝任意数目的字节内容到另一个地址单元,但是很多现代处理器中实现的该指令效率并不是很高。

我们来修改前面的例子:添加 REP 前缀以及初始化计数器 ECX。

Address	Hex dump	Disassembly	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	B3 04000000	MOV ECX, 4	
0040100F	ES: A5	REP MOVSD PTR ES:[EDI], DWORD PTR DS:[ESI]	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	
00401018	0BC0	OR EAX, EAX	
0040101A	74 01	JE SHORT CRACKME.0040101D	

源地址现在是 40365C(仅供说明),目的地址和之前的一样,是 40369C。我们来到第一条指令处,单步跟踪到 REP MOVSD 指令处。

Address	Hex dump	Disassembly
0040100C	58 F4040000	PUSH CRACKME
0040100E	6A 00	PUSH 0

ECX=00000004 (decimal 4)
DS:[ESI]=0040365C=6D614E65
ES:[EDI]=0040369C=0041676C

我们注意到 OD 的解释窗口中-出现了源地址和目的地址已经里面的包含的内容以及其他一些有用的信息。按 F7 键。

Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 00 00 00 00	eNam.....
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00

正如你所看到的,前面 4 个字节已经成功拷贝,由于 ECX 不等于 0,所以将继续指令 REP 指令。这里 ECX 为 3,指针 ESI 和 EDI 递增 4,只要我们按 F7 键,后面的 4 个字节将继续被拷贝。

004010C1	6A 00	PUSH 0
ECX=00000003 (decimal 3.)		
DS:[ESI]=[00403660]=00004165		
ES:[EDI]=[004036A0]=00000000		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..

按 F7 键。

E3:[EDI]=004036A4=00000000		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 65 41 00 00	eNameA..
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00

又有 4 个字节被成功拷贝,现在 ECX 减少至 2。

004010C1	6A 00	PUSH 0
ECX=00000002 (decimal 2.)		
DS:[ESI]=[00403664]=72500000		
ES:[EDI]=[004036A4]=00000000		
Address	Hex dump	

再次 F7,ECX 变成 1。

004010B7	68 F4204000	PUSH CRACKME.004020E7
004010BC	68 F4204000	PUSH CRACKME.004020F4
004010C1	6A 00	PUSH 0
ECX=00000001 (decimal 1.)		
DS:[ESI]=[00403668]=44746E69		
ES:[EDI]=[004036A8]=00000000		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD

继续 F7。

00401000	BE 5C364000	MOV ESI,CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C	ASCII "eNameA"
0040100A	B9 04000000	MOV ECX,4	
0040100F	F3:AS	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP,&USER32.FindWindowA>	FindWindowA
00401018	0BC0	OR EAX,EAX	

拷贝结束,因为此时计数器为 0。

Address	Hex dump	ASCII
00403650	65 4E 61 60 65 41 00 00	eNameA..
00403660	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 60 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00

总结:加上 REP 前缀的 MOVS 指令拷贝的字节数为单独使用 MOVS 拷贝字节数的 4 倍,这要归功于 REP 前缀指令,从而从源地址向目的地址成功拷贝 16 个字节的内容。

请记住,MOVS 指令不能将数据拷贝到没有写入权限的内存单元中,强制写入的话会引发异常。

此外,REP 前缀还可以衍生出 REPE/REPZ 和 REPNE/REPZ 指令,这个时候我们就需要考虑零标志位 Z 了。但是 REPE/REPZ 或者 REPZ/REPNE 连同 MOVS 指令一起使用就没有太大意义了。此外还有其他指令支持前缀,我们以后再讨论。

LODS

该指令从源地址(像之前一样,ESI)拷贝数据到 EAX 中。

Address	Hex dump	Instruction	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	B9 04000000	MOV ECX, 4	
0040100F	8D	LODS DWORD PTR DS:[ESI]	
00401010	90	NOP	
00401011	90	NOP	
00401012	90	NOP	

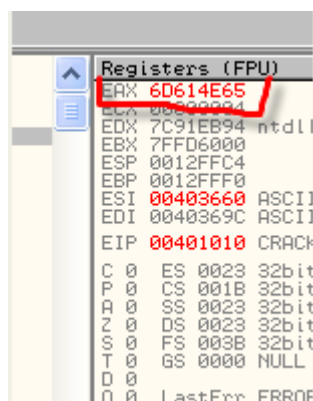
我们在 OD 里面来看一个 LODS 指令例子,这里 OD 写成了 LODS DWORD PTR DS:[ESI],如果你懒得去寄存器窗口中看 ESI 的值,可以看 OD 解释窗口中的提示信息。

004010B7	68 E7204000	PUSH CRACKME.004020E
004010BC	68 F4204000	PUSH CRACKME.004020F
DS:[ESI]=0040365C=60614E65		
Address	Hex dump	ASCII

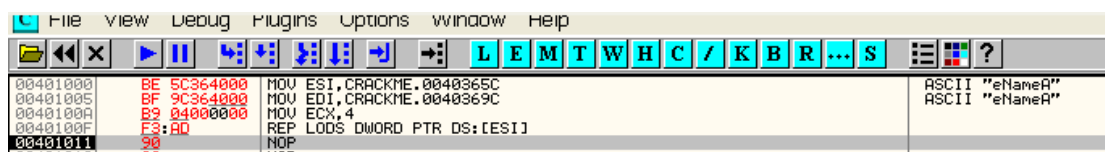
我们再到数据窗口中来看看 ESI 指向内存单元中内容。

Address	Hex dump	ASCII
00403650	65 4E 61 60 65 41 00 00	eNameA..
00403660	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 60 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00

这 4 个字节将保存到 EAX 中。按 F7 键。

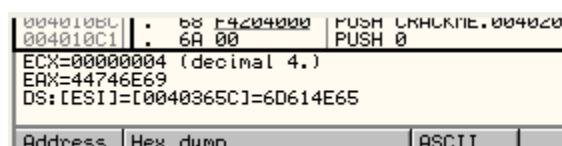


我们可以看到,EAX 保存了这 4 个字节的内容。

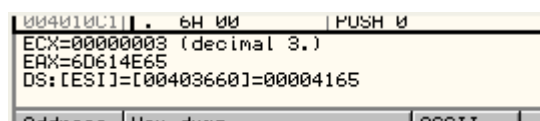


REP 前缀也可以与 LODS 指令配合使用,他们会重复执行直到计数器 ECX 的值为 0。

当循环执行 REP LODS 指令时,解释窗口中会提示 ECX 的值,以及 ESI 指向的内存单元中下一次将被拷贝到 EAX 寄存器中的内容。



按 F7 键。



现在 ECX 为 3,ESI 的值增加了 4-它指向下一个将被拷贝到 EAX 中的 4 个字节的内容。

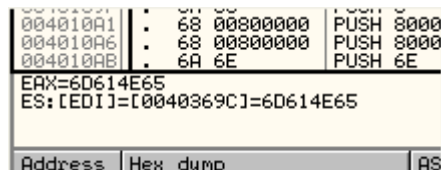
也有一次拷贝两个字节和一个字节的 LODSW 和 LODSB 指令。

STOS

该指令是将 EAX 的值拷贝到 EDI 指向的内存单元中。



我们单步跟踪直到执行完 STOS 指令,在解释窗口我们可以看到当前 EAX 的值以及当前 EDI 指向内存单元中的值。



在数据窗口中查看一下 EDI 指向内存单元中的情况。

004010AD	68 B4000000	PUSH 0B4
ECX=0000000F (decimal 15.)		
DS:[ESI]=[00403660]=00004165		
ES:[EDI]=[004036A0]=00004165		
Address	Hex dump	ASCII

零标志位 Z 置 1 表示继续循环比较。按 F7 键。

继续按 F7 键,直到比较的内容不同为止。ECX 没有递减至 0 之前就会有不同的比较内容。如下:

004010AD	68 B4000000	PUSH 0B4
ECX=0000000C (decimal 12.)		
DS:[ESI]=[0040366C]=0041676C		
ES:[EDI]=[004036AC]=00000000		
Address	Hex dump	ASCII

随后按下 F7 键,零标志位 Z 清 0,随即循环终止。

EIP	00401011	CR
C 0	ES 0023	32
P 1	CS 001B	32
D 0	SS 0023	32
S 0	DS 0023	32
F 0	FS 003B	32
T 0	GS 0000	NU
O 0		
O 0	LastErr	ER
EFL	00000206	(N
CTA	00000000	LNDR

如果全部 16 个双字都是相等的话,那么循环终止将是由计数器 ECX 为 0 造成的。

前面提到的 REPNZ 也同样可以与 CMPS 配合使用。

我们已经介绍了很多有用的指令,但是浮点运算指令我们还没有介绍,我们将在后面讨论。大家需要重复练习前面介绍的内容,直到大家能够熟练运用为止。

寻址方式

直接寻址

这是最简单的一种寻址方式-该指令的操作数中包含一个具体的地址。

例如:

MOV DWORD PTR [00513450], ECX

MOV AX, WORD PTR [00510A25]

MOV AL, BYTE PTR [00402811]

CALL 452200

JMP 421000

不需要进行任何有关地址解析的计算,地址的值是纯数字。

间接寻址

MOV DWORD PTR[EAX], ECX

CALL EAX

JMP [EBX + 4]

要想在指令执行之前看到真实地址,需要在该指令上下断点,断下来以后查看寄存器的值或者查看解释窗口中的提示信息。

许多程序使用间接寻址来完成一些复杂的操作,因此刚开始分析调试的时候真实地址并不会显示出来。直到我们执行到这条指令的时候,查看相应寄存器的值才能够直到真实的地址。

现在我们再次用 OD 加载 Cruehead'a 的 CrackMe。

004010E0	E8 90030000	CALL <JMP.&USER32.UpdateWindow>
004010E5	6A 01	PUSH 1
004010E7	6A 00	PUSH 0
004010E9	FF75 08	PUSH DWORD PTR SS:[EBP+8]
004010EC	E8 5B030000	CALL <JMP.&USER32.InvalidateRect>
004010F1	6A 00	PUSH 0
004010F3	6A 00	PUSH 0

这里我们来看一看间接寻址的例子: PUSH [EBP + 8]

因为当前我们处于入口点,在执行 PUSH [EBP + 8]指令之前我们并不能预先得知 EBP 寄存器的值,我们在该指令处(4010E9)按下 F2 键,设置一个断点。

然后按 F9 键(运行)-CrackMe 运行一会儿就会断在我们设置的断点处。

004010E9	. 6A 01	PUSH 1
004010E5	. 6A 00	PUSH 0
004010E7	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
004010E9	. E8 5B030000	CALL <JMP.&USER32.InvalidRect>
004010EC	> 6A 00	PUSH 0
004010F1	. 6A 00	PUSH 0
004010F3	. 6A 00	PUSH 0
004010F5	. 6A 00	PUSH 0
004010F7	. 68 48204000	PUSH CRACKME.00402048
004010F9	. 68 48204000	PUSH CRACKME.00402048

解释窗口中提示 EBP+8=12FFF8。在我的机器上面此时 EBP 为 12FFF0,所以 EBP+8=12FFF8。

Registers (FPU)	
EAX	00000001
ECX	0012FFA8
EDX	7C91EB94 ntdll.k
EBX	7FFD6000
ESP	0012FFBC
EBP	0012FFF0
ESI	0012FFF0
EDI	7C920738 ntdll.7
EIP	004010E9 CRACKME
C 0 ES 0023 32bit 0	
P 0 CS 001B 32bit 0	
A 0 SS 0023 32bit 0	
Z 0 DS 0023 32bit 0	
S 0 FS 003B 32bit 7	
T 0 GS 0000 NULL	
D 0	
O 0 LastErr ERROR_S	
EFL 00000202 (NO,NB,	
ST0 empty -??? FFFF	

0040111D	> FF35 50204000	PUSH DWORD PTR DS:[402050]
00401123	. E8 EA030000	CALL <JMP.&KERNEL32.ExitProcess>

Stack SS:[0012FFF8]=00401000 (CRACKME.<ModuleEntryPoint>)

Address	Hex dump	ASCII
---------	----------	-------

现在让我们在数据窗口中看看 EBP+8 指向内存单元的值,在数据窗口中单击鼠标右键选择-Go to-Expression, 输入 EBP+8

Enter expression to follow in Dump

ebp+8

OKCancel

内容是

Address	Hex dump	ASCII
0012FFF8	00 10 40 00 00 00 00 00	..@.....

按 F7 键,这个值将被压入到堆栈中。

0012FFB8	00401000	hWnd = 00401000
0012FFBC	00000000	pRect = NULL
0012FFC0	00000001	Erase = TRUE
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD6000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	843BAC70	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	CRACKME.<ModuleEntryPoint>
0012FFFC	00000000	

使用间接寻址的指令,只能在执行这条指令的时候获取地址当前的值。

第九章我们会将基础知识运用到破解中去。