第八章

本章,我们将来看看前面章节忽略了的一些重要的指令。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。

CMP ECX,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:

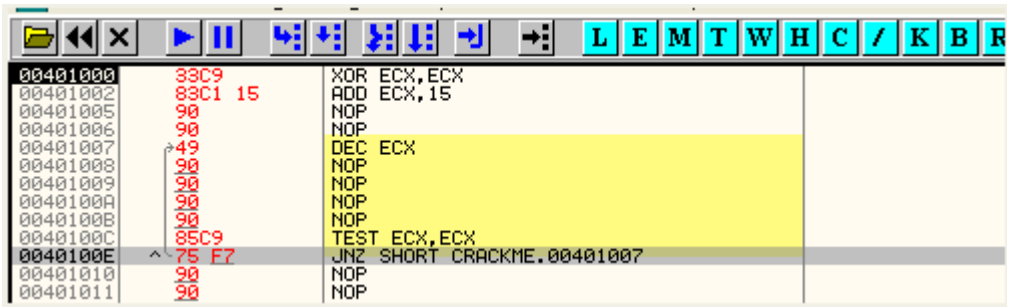
DEC ECX

;循环体

TEST ECX,ECX

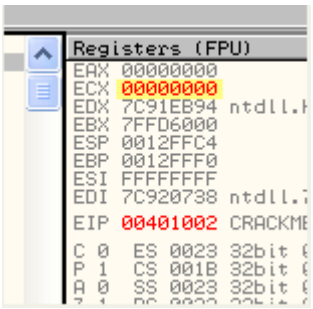
JNE Label

让我们在 OD 上输入上面的代码:

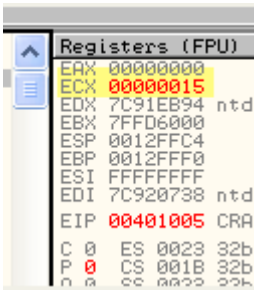


黄色突出显示的是循环代码,这个部分将重复执行,直到 ECX 的值为 0。循环体里面包含了多个 NOP 指令。

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。

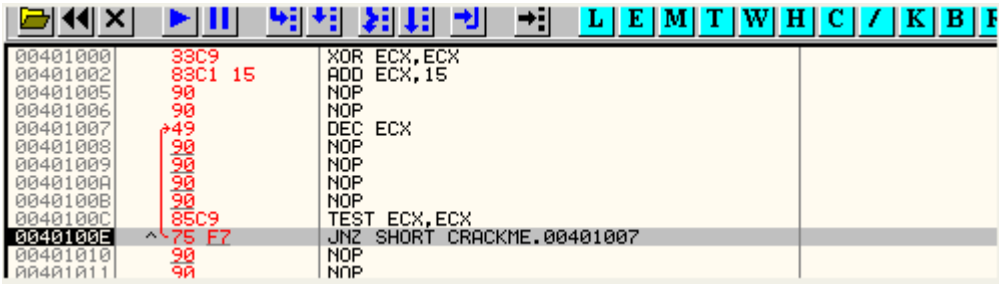


再次按 F7 键,计数器将变成 15h,保存了需要循环的次数。



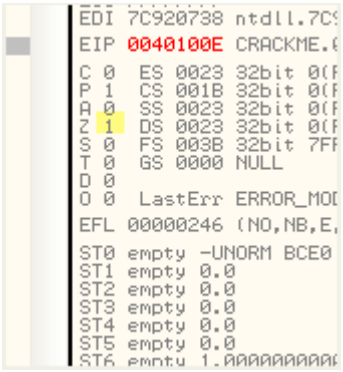
然后继续按 F7 键,直到 DEC ECX,这个时候我们的计数器的值将减少至 14h。

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。

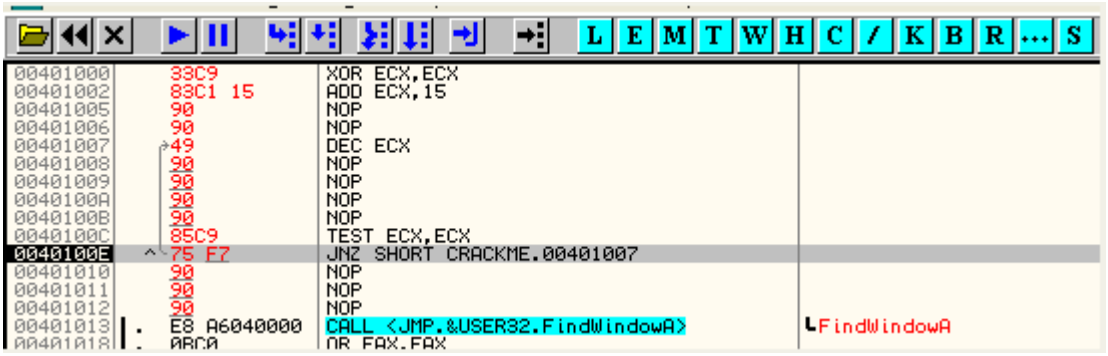


因为此时计数器不为 0,零标志位 Z 没有置 1,所以条件跳转指令 JNZ 将跳转到 401007 地址处。下一步计数器继续递减 1,这一次减少至 13h。继续单步跟踪直到计数器为 0。

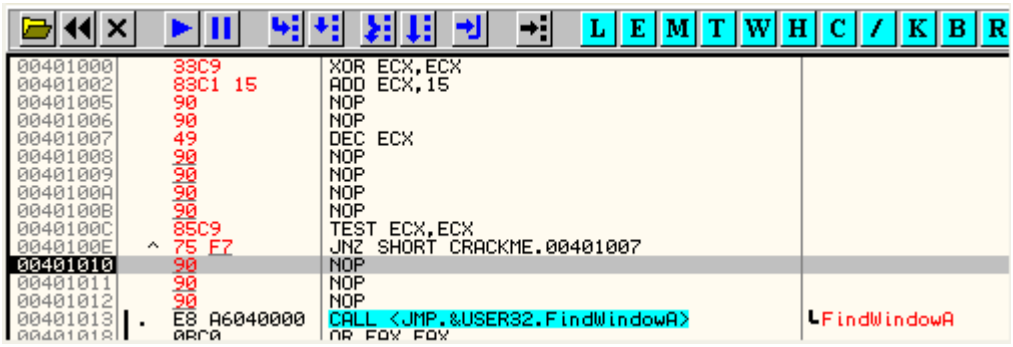
当计数器为 0 时,零标志位 Z 将置 1,这个时候 JNZ 将不会跳转至 401007 地址处,而是继续向下执行。



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。



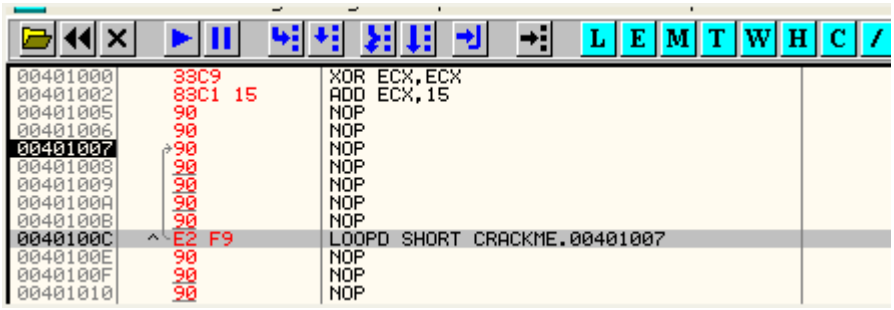
见到灰色箭头,意味着什么?-这里意味着跳转不会发生。按 F7 键我们将跳出循环继续执行下面的代码。



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

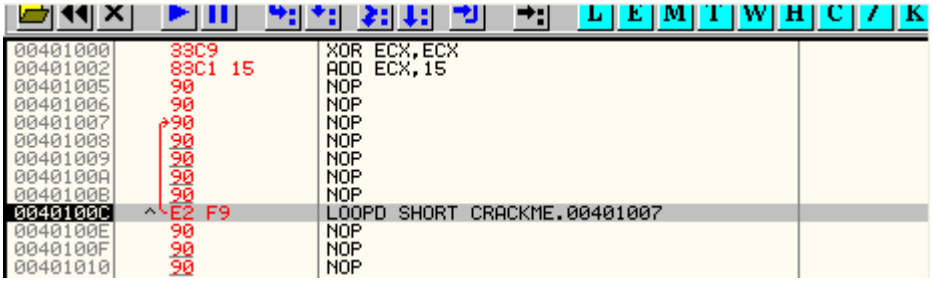
LOOP 指令可以帮我们完成前面例子中的事情- 将计数器 ECX 的值减 1,判断 ECX 的值是否为 0,如果为 0 就跳转到指定的地址-将像前面的例子一样。(可惜的是,大多数现代的处理器中该指令的效率不如前面模拟的例子。)



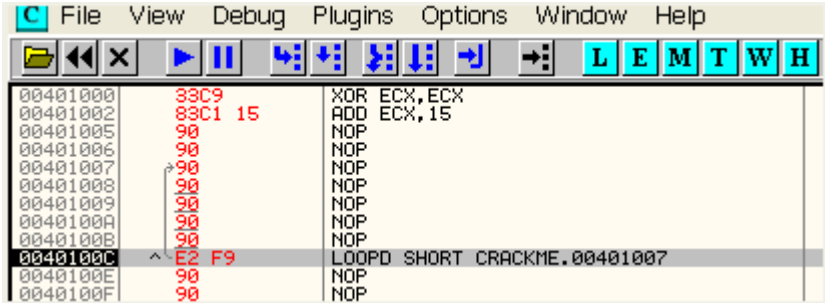
在 DEC ECX 指令上单击鼠标右键选择-Binary-Fill with NOPS。对 TEST ECX,ECX 和 JNZ 401007 两条指令也进行同样的操作。这三条指令用一条 LOOP 401007 指令替代。

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。

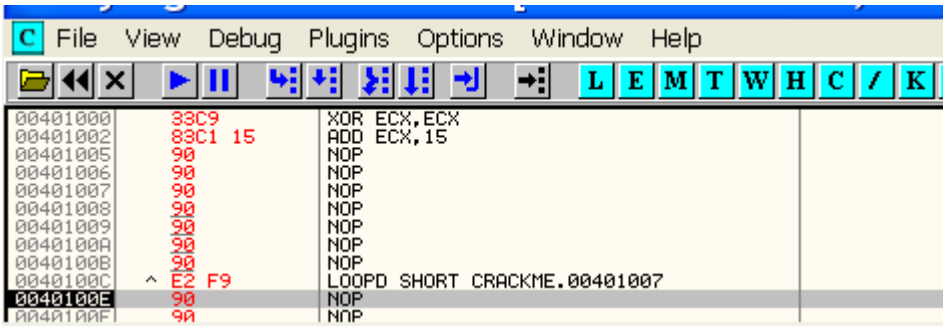
按 F7 键,再次看到计数器 ECX 首先被初始化为 0 了,然后又被设置为 15h 了。我们继续单步跟踪,直到 LOOP 指令。



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。



继续单步跟踪。当计数器为 0 时,循环将结束。



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。

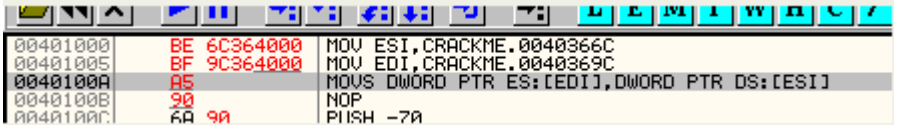
接着,我们来介绍一下字符串操作类指令。

串操作

下面,介绍一下串操作类的基本指令

MOVS

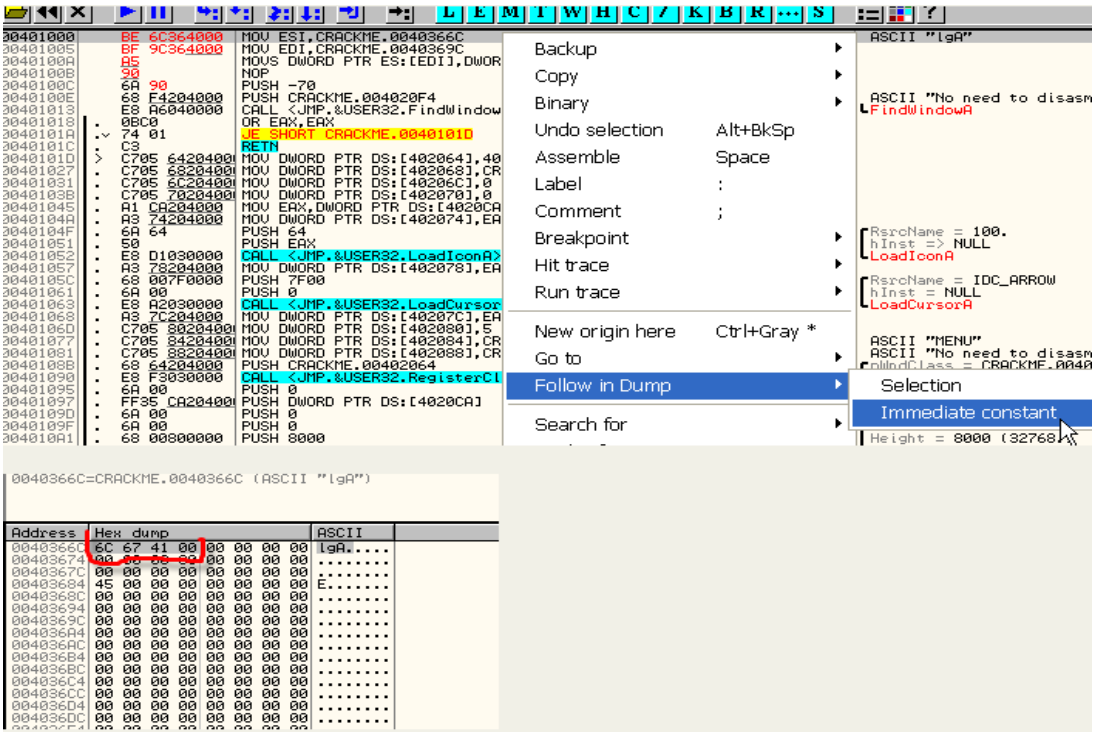
该指令是从一个地址向另一个地址复制数据。源地址保存在 ESI 寄存器中,目的地址保存在 EDI 寄存器中。我们不需要显示地指定参数。现在我们在 OD 中写入 MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]指令,样子如下:



这里我们用 ESI 来保存源地址,EDI 来保存目的地址。

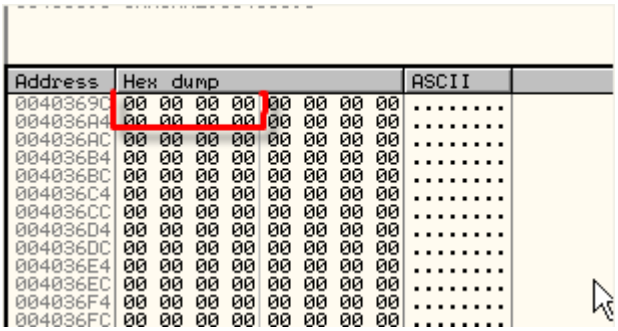
在拷贝内容之前,让我们在数据窗口中来看一看源地址和目的地址的情况。

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:



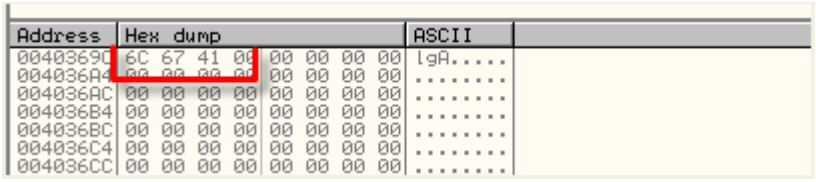
源地址(ESI)中指定的地址单元中的待拷贝的内容。

目的地址(EDI)指向的地址单元



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 MOVS 指令,我们可以看到 4 个字节被拷贝了。



这里 MOVS 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。

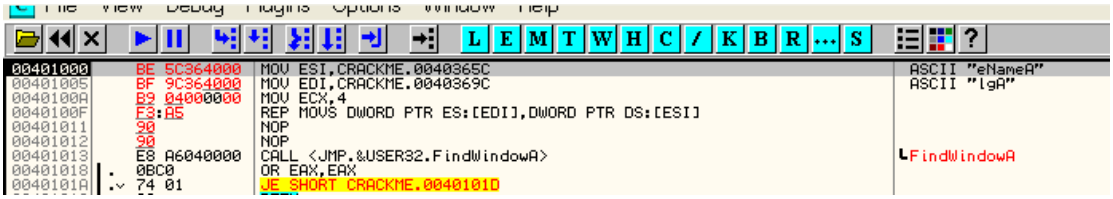
REP

该指令可做为前面介绍的一些指令的前缀,尤其是 MOVS 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。

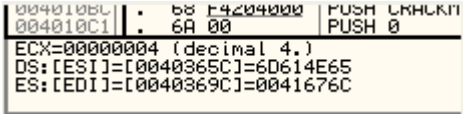
因此,REP MOVS 不一定拷贝是 4 个字节,它拷贝的大小为 每次拷贝的大小 * ECX, 源指针 ESI 和目的指针 EDI 每次递增 4 或者递减 4(递增或递减取决于方向标志位 D)。该指令看起来很实用,是不是?

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。

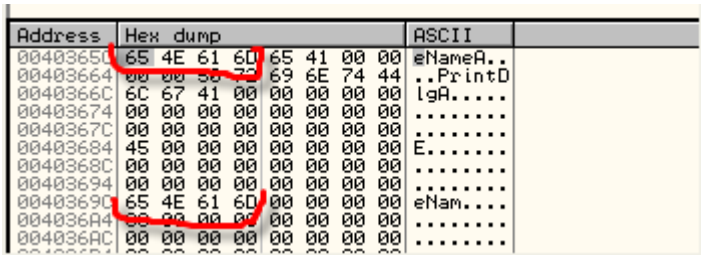
我们来修改前面的例子:添加 REP 前缀以及初始化计数器 ECX。



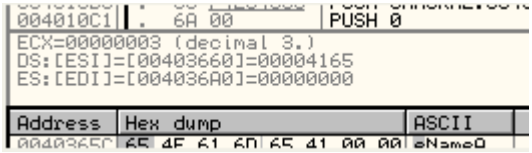
源地址现在是 40365C(仅供说明),目的地址和之前的一样,是 40369C。我们来到第一条指令处,单步跟踪到 REP MOVS 指令处。



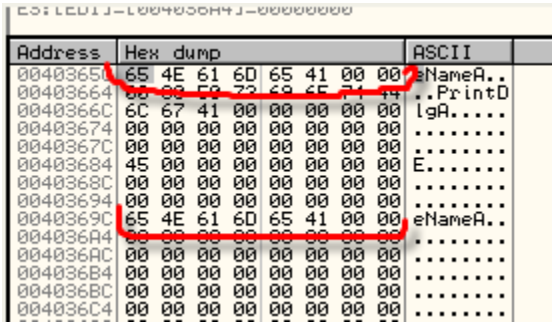
我们注意到 OD 的解释窗口中-出现了源地址和目的地址已经里面的包含的内容以及其他一些有用的信息。按 F7 键。



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。

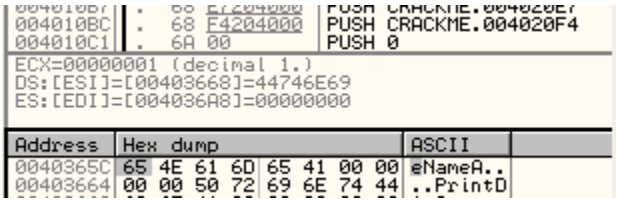


按 F7 键。

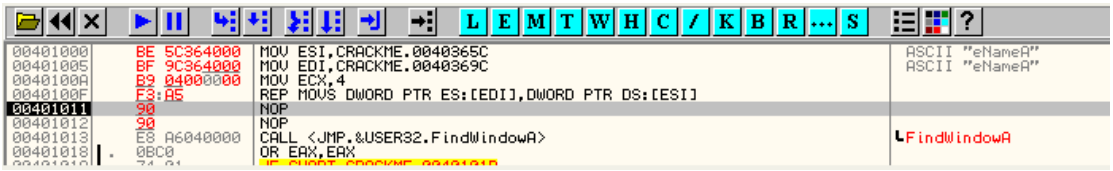


又有 4 个字节被成功拷贝,现在 ECX 减少至 2。

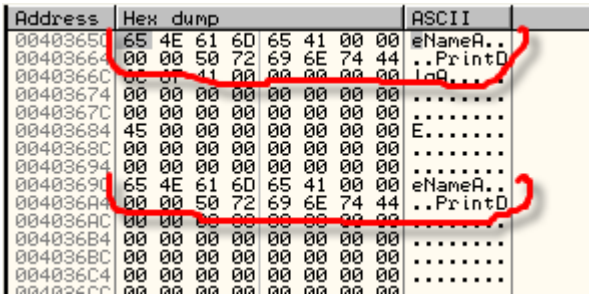
再次 F7,ECX 变成 1。



继续 F7。



拷贝结束,因为此时计数器为 0。



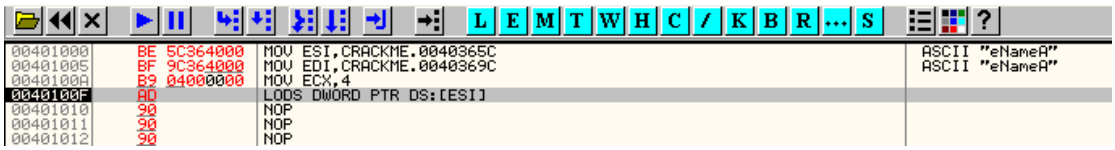
总结:加上 REP 前缀的 MOVS 指令拷贝的字节数为单独使用 MOVS 拷贝字节数的 4 倍,这要归功与 REP 前缀指令,从而从源地址向目的地址成功拷贝 16 个字节的内容。

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。

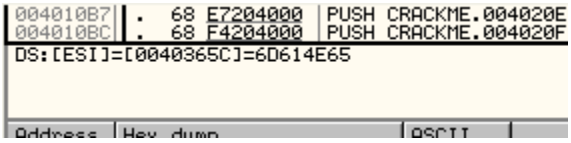
此外,REP 前缀还可以衍生出 REPE/REPZ 和 REPNE/REPNZ 指令,这个时候我们就需要考虑零标志位 Z 了。但是 REPE/REPZ 或者 REPNZ/REPNE 连同 MOVS 指令一起使用就没有太大意义了。此外还有其他指令支持前缀,我们以后再讨论。

LODS

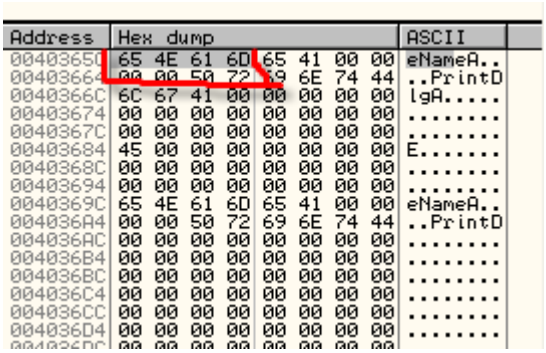
该指令从源地址(像之前一样,ESI)拷贝数据到 EAX 中。



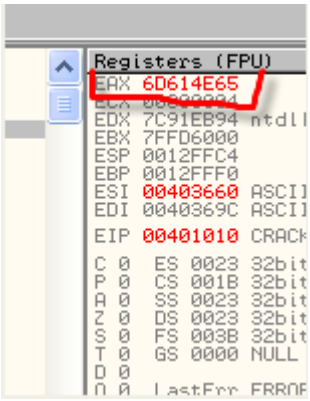
我们在 OD 里面来看一个 LODS 指令例子,这里 OD 写成了 LODS DWORD PTR DS:[ESI],如果你懒得去寄存器窗口中看 ESI 的值,可以看 OD 解释窗口中的提示信息。



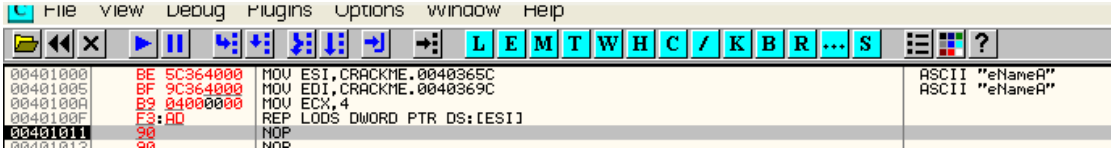
我们再到数据窗口中来看看 ESI 指向内存单元中内容。



这 4 个字节将保存到 EAX 中。按 F7 键。

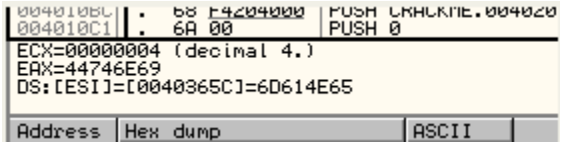


我们可以看到,EAX 保存了这 4 个字节的内容。

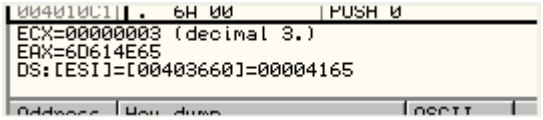


REP 前缀也可以与 LODS 指令配合使用,他们会重复执行直到计数器 ECX 的值为 0。

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。

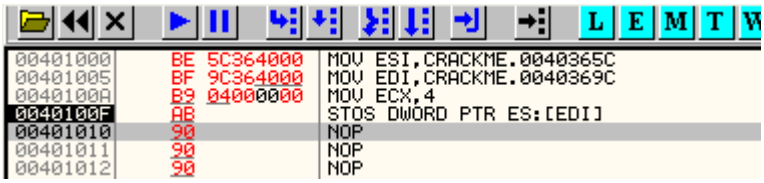


现在 ECX 为 3,ESI 的值增加了 4-它指向下一个将被拷贝到 EAX 中的 4 个字节的内容。

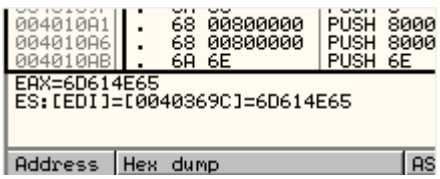
也有一次拷贝两个字节和一个字节的 LODSW 和 LODSB 指令。

STOS

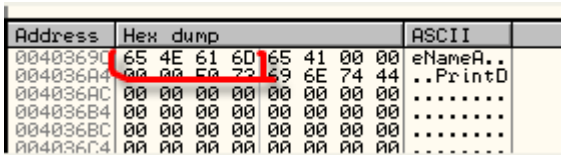
该指令是将 EAX 的值拷贝到 EDI 指向的内存单元中。



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。



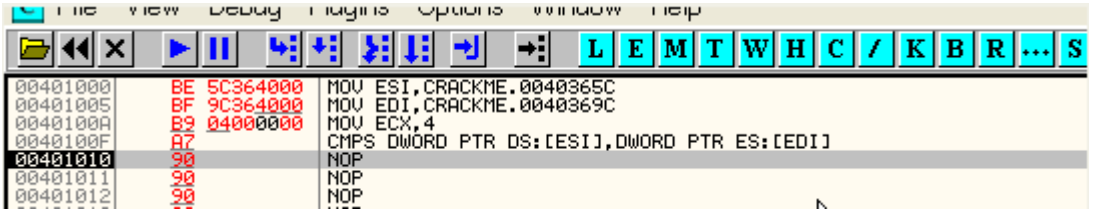
在数据窗口中查看一下 EDI 指向内存单元中的情况。



和之前介绍的一样,我们可以使用 REP 前缀,还有每次操作两个字节的 STOSW 指令和每次操作一个字节的 STOSB 指令。

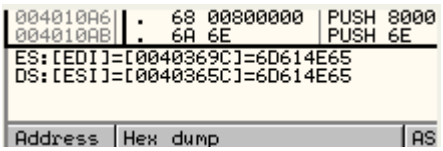
CMPS

该指令比较 ESI 和 EDI 指向内存单元的内容。

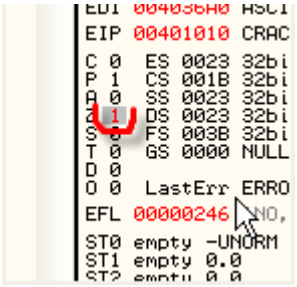


我们在 OD 中输入 CMPS DWORD PTR [ESI],DWORD PTR [EDI],OD 中将显示为 CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]。

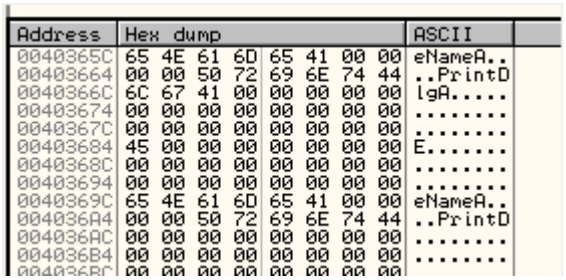
解释窗口中会提示要比较的值:



正如你所猜想的一样,该指令执行的是算数减法运算,这里,差值为 0,所以零标志位 Z 将置 1。

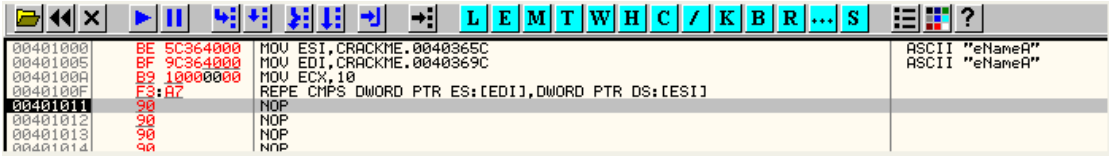


因为该指令影响的零标志位 Z,所以你可以配合 REPE/REPZ 前缀指令使用,直到计数器 ECX 的值为 0 或者零标志位清 0。

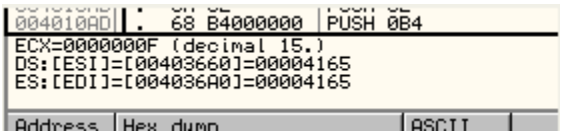


在数据窗口中的 40365C 和 40369C 地址处,我们看到之前操作过的内容。

这里将 ECX 初始化为 10h。使用 REPE 前缀指令,当 ECX 为 0 或者零标志位 Z 清 0 时将停止比较。

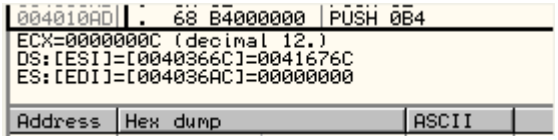


单步跟踪直到 REPE 指令。然后按 F7 键,ESI 和 EDI 指向内存单元的内容相等,所以零标志位 Z 置 1。

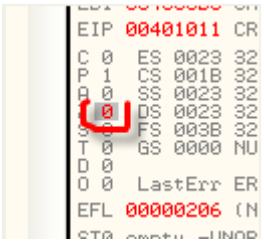


零标志位 Z 置 1 表示继续循环比较。按 F7 键。

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:



随后按下 F7 键,零标志位 Z 清 0,随即循环终止。



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

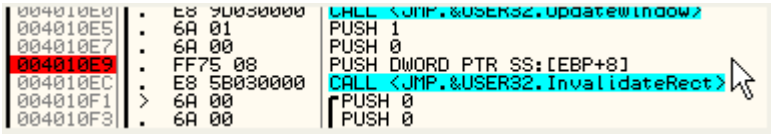
CALL EAX

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。

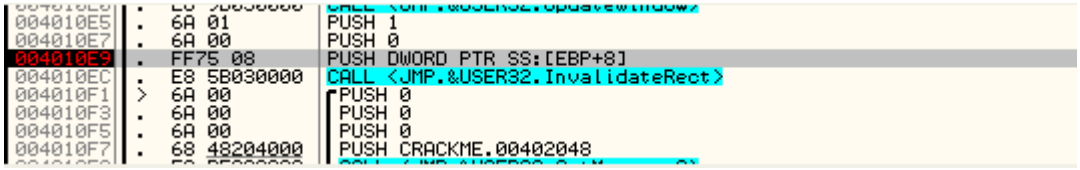
现在我们再次用 OD 加载 Cruehead’a 的 CrackMe。



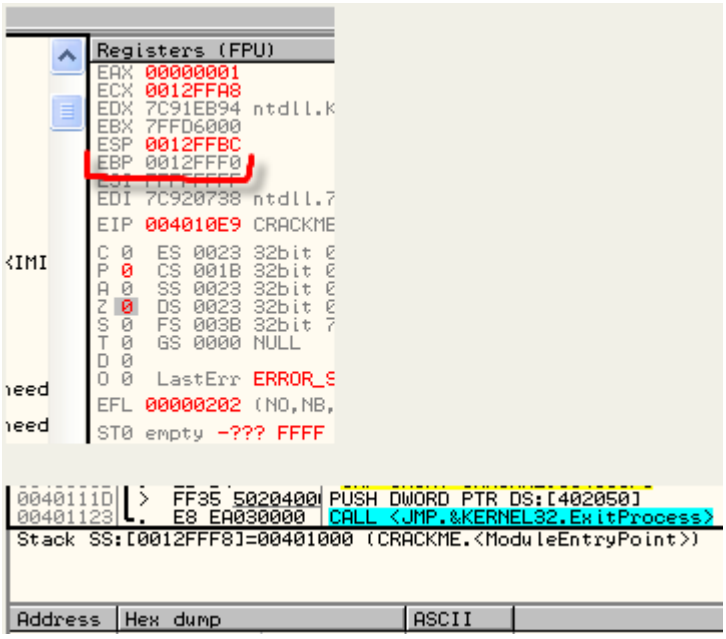
这里我们来看一看间接寻址的例子:PUSH [EBP + 8]

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直到 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。

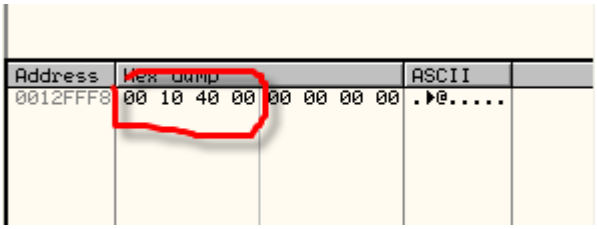


解释窗口中提示 EBP+8=12FFF8。在我的机器上面此时 EBP 为 12FFF0,所以 EBP+8=12FFF8。



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

内容是



按 F7 键,这个值将被压入到堆栈中。

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。