

Reverse01 分析报告

1 推理过程

进行完方法二，即通过硬件断点寻找用户代码区域的常规操作：即在输入框中输入内容，在“memory”中查找并设置硬件断点，点击运行，在点击“register”后再次点击运行，找到rep，按alt+F9回到用户代码空间（00开头）并删除硬件断点后。经过多个retn，没有遇到非系统空间函数，在跟进4E05C0和有多个push非常可疑的4D5FCC没有发现有用信息后，选择先不跟进函数内。

0040179A	. 83C4 0C	add esp, 0C	
0040179D	. 8D5D C4	lea ebx, [ebp-3C]	
004017A0	. 6A 01	push 1	Arg2 = 00000001
004017A2	. 8A03	mov al, [ebx]	
004017A4	. 50	push eax	Arg1
004017A5	. E8 9EFEFFFF	call 00401648	Reverse2.00401648
004017AA	. 83C4 08	add esp, 8	
004017AD	. 8B03	mov [ebx], al	
004017AF	. 6A 02	push 2	
004017B1	. 8A53 01	mov dl, [ebx+1]	
004017B4	. 52	push edx	
004017B5	. E8 B6FEFFFF	call 00401670	
004017BA	. 8B43 01	mov [ebx+1], al	
004017BD	. 33C0	xor eax, eax	
004017BF	. 8A43 02	mov al, [ebx+2]	
004017C2	. 83C4 08	add esp, 8	
004017C5	. 8063 02 0F	and byte ptr [ebx+2], 0F	
004017C9	. 8A53 03	mov dl, [ebx+3]	
004017CC	. 8063 03 F0	and byte ptr [ebx+3], 0F0	
004017D0	. 80E2 0F	and dl, 0F	
004017D3	. C1E2 04	shl edx, 4	
004017D6	. 0853 02	or [ebx+2], dl	
004017D9	. C1F8 04	sar eax, 4	
004017DC	. 0843 03	or [ebx+3], al	
004017DF	. 8B4D C4	mov ecx, [ebp-3C]	
004017E2	. 81F1 68245710	xor ecx, 13572468	
004017E8	. 3B0D 40254E00	cmp ecx, [4E2540]	
004017EE	. 75 0C	jnz short 004017FC	
004017F0	. C705 AC7D4E00	mov dword ptr [4E7DAC], 004E28B8	ASCII "秋璞"
004017FA	. EB 0A	jmp short 00401806	
004017FC	. C705 AC7D4E00	mov dword ptr [4E7DAC], 004E28BD	ASCII "第?"
00401806	. A1 D8784E00	mov eax, [4E78D8]	
00401808	. 8B08	mov ecx, [eax]	
0040180D	. B2 01	mov dl, 1	
0040180F	. A1 0C294E00	mov eax, [4E290C]	
00401814	. E8 37080000	call 00402050	
00401819	. 8BD8	mov ebx, eax	
0040181B	. A1 A0784E00	mov eax, [4E78A0]	
00401820	. 8918	mov [eax], ebx	
00401822	. 8BC3	mov eax, ebx	
00401824	. 8B10	mov edx, [eax]	Reverse2.004E2958
00401826	. FF92 D8000000	call [edx+D8]	
0040182C	. 8B0D A0784E00	mov ecx, [4E78A0]	Reverse2._frmMsg

结合此段代码，分别push了输入的注册码的56和78位，处理了前四位的操作以及比较跳转语句，推测这里便是真正的注册码函数计算和比较区域。

00401814	. E8 37080000	call 00402050	
00401819	. 8BD8	mov ebx, eax	
0040181B	. A1 A0784E00	mov eax, [4E78A0]	
00401820	. 8918	mov [eax], ebx	
00401822	. 8BC3	mov eax, ebx	
00401824	. 8B10	mov edx, [eax]	Reverse2.004E2958
00401826	. FF92 D8000000	call [edx+D8]	
0040182C	. 8B0D A0784E00	mov ecx, [4E78A0]	Reverse2._frmMsg

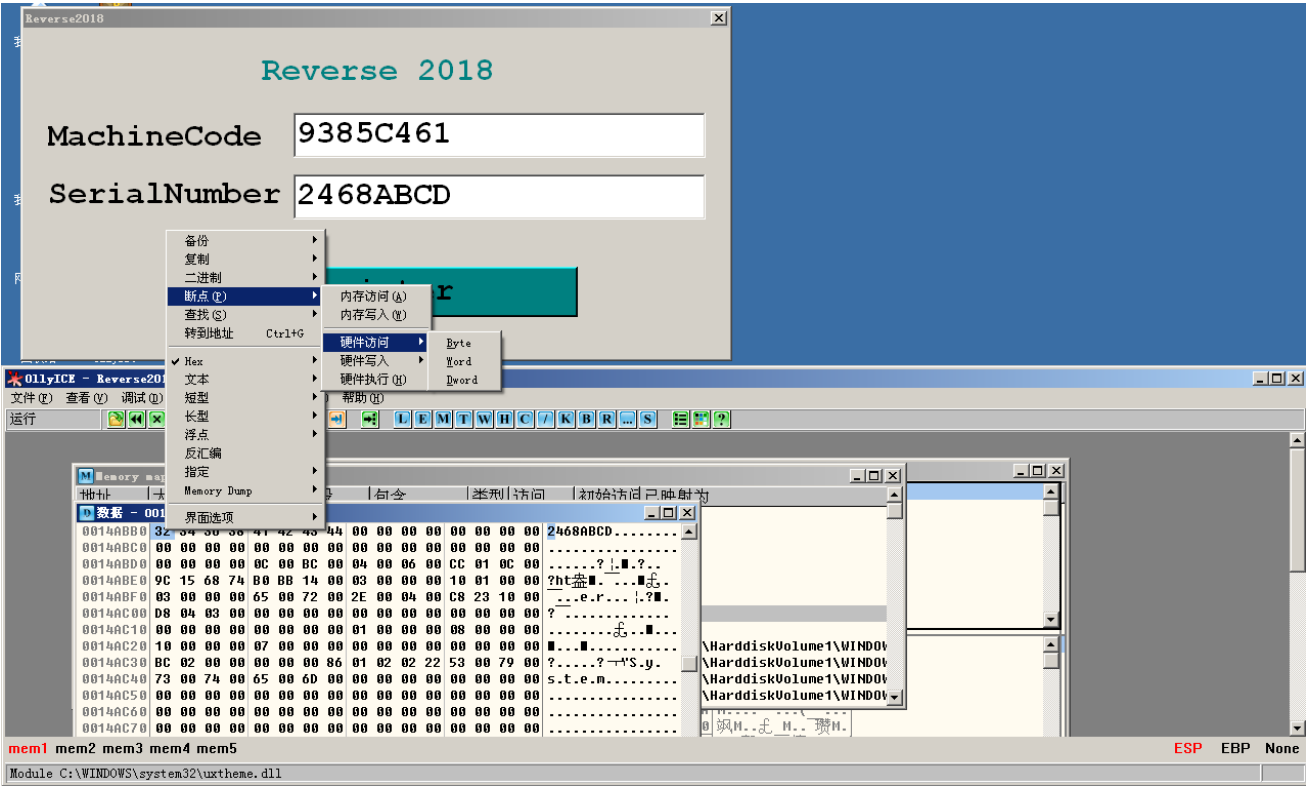
随后为了验证猜想跟入报错即输出“BAD”前的两个函数，均没有发现，仅仅为调用了窗体，故开始对上述可以代码段进行跟踪。

并分析得到对应的注册码算法。（详细的跟踪过程见跟踪步骤）

2 跟踪步骤（内含核心代码注释）

2.1 准备过程（找到处理函数所在的用户代码段）

在这里我使用了白老师的第二种方法，即在输入框中输入内容（需要有辨识度，如ABCDEFGH、12345678....之类本来就可能在内存中的不可取）后，在“memory”中查找并设置硬件断点。



点击运行，在点击“register”后再次点击运行，找到rep。

77D33566	F3:A5	rep	movs dword ptr es:[edi], dword ptr
77D33568	8BC8	mov	ecx, eax
77D3356A	83E1 03	and	ecx, 3
77D3356D	F3:A4	rep	movs byte ptr es:[edi], byte ptr
77D3356F	E8 E3FBFFFF	call	77D33157

按alt+F9回到用户代码空间（00开头）并删除硬件断点。

004A488B	. 8943 0C	mov	[ebx+C], eax
004A488E	> 8B03	mov	eax, [ebx]
004A4890	. 83F8 0C	cmp	eax, 0C
004A4893	~ 75 1B	jnz	short 004A48B0

2.2 寻找可疑函数过程

F8单步执行下去，执行了多个retn，遇到了第一个函数，查找到对应位置看一下。

004016C0	. 894D FC	mov	[ebp-4], ecx	
004016C3	. FF45 E4	inc	dword ptr [ebp-1C]	
004016C6	. 8B83 E02000	mov	eax, [ebx+2E0]	
004016CC	. E8 67F40900	call	004A0B38	
004016D1	. 837D FC 00	cmp	dword ptr [ebp-4], 0	
004016D5	. 0F94C1	sete	cl	
004016D8	. 83E1 01	and	ecx, 1	
004016DB	. 8D45 FC	lea	eax, [ebp-4]	
004016DE	. 51	push	ecx	Arg1
004016DF	. BA 02000000	mov	edx, 2	
004016E4	. FF4D E4	dec	dword ptr [ebp-1C]	
004016E7	. E8 D4EE0D00	call	004E05C0	Reverse2.004E05C0
004016EC	. 59	pop	ecx	
004016ED	. 84C9	test	cl, cl	
004016EF	. 74 0E	je	short 004016FF	
004016F1	. 8B45 C8	mov	eax, [ebp-38]	
004016F4	. 64:A3 000000	mov	fs:[0], eax	
004016FA	. E9 64010000	jmp	00401863	
004016FF	> 66:C745 D8 14	mov	word ptr [ebp-28], 14	
00401705	. 33D2	xor	edx, edx	
00401707	. 8955 F8	mov	[ebp-8], edx	
0040170A	. 8D55 F8	lea	edx, [ebp-8]	

输入要跟随的表达式
 4E05C0
☒ VA/API ☐ RVA ☐ Offset

和0相比较，应该是观察有没有输入的，与注册码的计算无关回到原先位置，继续跟，并在之后的跟踪中直接执行4E05C0这个函数。

004E05CB	. 837D FC 00	cmp	dword ptr [ebp-4], 0
004E05CF	. 74 19	je	short 004E05EA
004E05D1	. 8B45 FC	mov	eax, [ebp-4]
004E05D4	. E8 CB09FFFF	call	004D0FA4
004E05D9	. F7C6 01000000	test	esi, 1
004E05DF	. 74 09	je	short 004E05EA
004E05E1	. FF75 FC	push	dword ptr [ebp-4]
004E05E4	. E8 132AFFFF	call	004D2FFC
004E05E9	. 59	pop	ecx
004E05EA	> 5E	pop	esi
004E05EB	. 5B	pop	ebx
004E05EC	. 59	pop	ecx
004E05ED	. 5D	pop	ebp
004E05EE	. C3	retn	

这里有多個push作为输入，比较可疑，F7跟进去看一下。

0040178E	. 51	push	ecx	Arg3
0040178F	. 68 4E254E00	push	004E254E	Arg2 = 004E254E ASCII "%X"
00401794	. 50	push	eax	Arg1
00401795	. E8 32480D00	call	004D5FCC	Reverse2.004D5FCC
0040179A	. 83C4 0C	add	esp, 0C	

并没有发现重要的处理语句，仅仅有将数字和0A，8等相比较的语句，推测为判断输入的函数。（在跟踪完成后发现"%X"就是读入16进制数，ecx为数字个数，eax为字符串的地址）自此对4D5FCC也直接执行。

这里看到了往函数"401648"，push了两个数，'1'和输入的注册码最后两位'CD'，非常可疑F7跟进去。

0040178F	. 68 4E254E00	push	004E254E	Arg2 = 004E254E ASCII "%X"
00401794	. 50	push	eax	Arg1
00401795	. E8 32480D00	call	004D5FCC	Reverse2.004D5FCC
0040179A	. 83C4 0C	add	esp, 0C	
0040179D	. 8D5D C4	lea	ebx, [ebp-3C]	
004017A0	. 6A 01	push	1	Arg2 = 00000001
004017A2	. 8A03	mov	al, [ebx]	
004017A4	. 50	push	eax	Arg1 = 000000CD
004017A5	. E8 9EFEFFFF	call	00401648	Reverse2.00401648

寄存器 (FPU)
 EAX 000000CD
 ECX 0012FB00
 EDI 0012FB18
 EBX 0012FB88
 ESP 0012FB14
 EBP 0012FBC4
 ESI 00A42F15
 EDI 0012FB2D

2.2.1 分析函数401648

看到了它的代码，分析一下其功能。

00401646	90	nop			寄存器 (FPU)
00401647	90	nop			EAX 000000CD
00401648	55	push ebp			ECX 0012FB00
00401649	8BEC	mov ebp, esp			EDX 0012FB18
0040164B	53	push ebx			EBX 0012FB88
0040164C	8B55 0C	mov edx, [ebp+C]			ESP 0012FB0C
0040164F	8B45 08	mov eax, [ebp+8]			EBP 0012FBC4
00401652	8BCA	mov ecx, edx			ESI 00042F15
00401654	8BD8	mov ebx, eax			EDI 0012FB2D
00401656	D2E3	shl bl, cl			EIP 00401648 Reverse2.00401648
00401658	89 08000000	mov ecx, 8			C 0 ES 0023 32位 0(FFFFFFFF)
0040165D	2BCA	sub ecx, edx			P 1 CS 001B 32位 0(FFFFFFFF)
0040165F	25 FF000000	and eax, 0FF			A 1 SS 0023 32位 0(FFFFFFFF)
00401664	D3F8	sar eax, cl			Z 0 DS 0023 32位 0(FFFFFFFF)
00401666	0AD8	or bl, al			S 0 FS 003B 32位 7FFDF000(FFF)
00401668	8BC3	mov eax, ebx			T 0 GS 0000 NULL
0040166A	58	pop ebx			D 0
0040166B	5D	pop ebp			O 0 LastErr ERROR_SUCCESS (00000000)
0040166C	C3	ret			EFL 00000216 (NO,NB,NE,A,NS,PE,GE,G)
0040166D	90	nop			ST0 empty -1.6259183529734161960e+4755
0040166E	90	nop			
0040166F	90	nop			

```
push ebp;前三行为一般函数的开头
mov ebp, esp
push ebx
mov edx, [ebp+C];这里将1传给edx
mov eax, [ebp+8];这里将CD(后两位)传给eax
mov ecx, edx;ecx得到1
mov ebx, eax;ebx得到CD
shl bl, cl;将bl(ebx)后8位的内容即'CD'逻辑左移1位
mov ecx, 8
sub ecx, edx;计算得到8-1=7
and eax, 0FF
sar eax, cl;算数右移7位，但由于输入的都正值，符号位为0，与逻辑右移没有区别
or bl, al;按位或操作
mov eax, ebx;把按位或后的结果赋值给eax
pop ebx
pop ebp;清空栈
ret
```

2.2.2 分析函数401670

把对第七八位的计算的结果'9B'存到[ebx]（地址为0012FB88）内存单元中，将2和第五六位的AB push后进入函数401670。

004017AA	83C4 08	add esp, 8
004017AD	8803	mov [ebx], al
004017AF	6A 02	push 2
004017B1	8A53 01	mov dl, [ebx+1]
004017B4	52	push edx
004017B5	E8 B6FEFFFF	call 00401670

看到代码如下（与401648相近）：

00401670	55	push	ebp		寄存器 (FPU)
00401671	8BEC	mov	ebp, esp		EAX 00000098
00401673	53	push	ebx		ECX 00000007
00401674	B9 00000000	mov	ecx, 8		EDX 000000A8
00401679	8B55 0C	mov	edx, [ebp+C]		EBX 0012FB88
0040167C	8B45 08	mov	eax, [ebp+8]		ESP 0012FB0C
0040167F	2BCA	sub	ecx, edx		EBP 0012FBC4
00401681	8BD8	mov	ebx, eax		ESI 00A42F15
00401683	D2E3	shl	bl, cl		EDI 0012FB20
00401685	8BCA	mov	ecx, edx		EIP 00401670 Reverse2.00401670
00401687	25 FF000000	and	eax, 0FF		C 0 ES 0023 32位 0(FFFFFFFF)
0040168C	D3F8	sar	eax, cl		P 1 CS 001B 32位 0(FFFFFFFF)
0040168E	0A08	or	bl, al		A 0 SS 0023 32位 0(FFFFFFFF)
00401690	8BC3	mov	eax, ebx		Z 0 DS 0023 32位 0(FFFFFFFF)
00401692	5B	pop	ebx		S 0 FS 003B 32位 7FFDF000(FFF)
00401693	5D	pop	ebp		T 0 GS 0000 NULL
00401694	C3	ret			D 0
00401695	90	nop			0 0 LastErrr ERROR_SUCCESS (00000000)
00401696	90	nop			EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
00401697	90	nop			ST0 empty -1.6259183529734161960e+4755
00401698	55	push	ebp		ST1 empty -UNORM B27C 00000004 00000000
00401699	8BEC	mov	ebp, esp		ST2 empty -UNORM B564 00000000 950404BF
					ST3 empty -8.4211152663148318880e-853

```

push ebp;前三行为一般函数的开头
mov ebp, esp
push ebx
mov ecx, 8
mov edx, [ebp+C];这里将2传给edx
mov eax, [ebp+8];这里将AB(第5第6位)传给eax
sub ecx, edx;计算得到8-2=6
mov ebx, eax;ebx得到CD
shl bl, cl;将bl(ebx)后8位的内容即'CD'逻辑左移6位
mov ecx, edx;ecx得到2
and eax, 0FF
sar eax, cl;算数右移2位,但由于输入的都是正值,符号位为0,与逻辑右移没有区别
or bl, al;按位或操作
mov eax, ebx;把按位或后的结果赋值给eax
pop ebx
pop ebp;清空栈
ret

```

把对第五六位的结果EA存到[ebx+1]内存单元（地址为0012FB89）中。

2.2.3分析代码段4017BF-4017FC

在处理完后四位数后，直到跳转这一步没有函数了，那么自然地推测对前四位的处理直接由下方的指令完成。

004017B4	52	push	edx		寄存器 (FPU)
004017B5	E8 B6FFFFFF	call	00401670		EAX 000000EA
004017B8	8B43 01	mov	[ebx+1], al		ECX 00000002
004017BD	33C0	xor	eax, eax		EDX 00000002
004017BF	8A43 02	mov	al, [ebx+2]		EBX 0012FB88
004017C2	83C4 08	add	esp, 8		ESP 0012FB10
004017C5	8063 02 0F	and	byte ptr [ebx+2], 0F		EBP 0012FBC4
004017C9	8A53 03	mov	d1, [ebx+3]		ESI 00A42F15
004017CC	8063 03 F0	and	byte ptr [ebx+3], 0F0		EDI 0012FB20
004017D0	80E2 0F	and	d1, 0F		EIP 004017BD Reverse2.004017BD
004017D3	C1E2 04	shl	edx, 4		C 0 ES 0023 32位 0(FFFFFFFF)
004017D6	0853 02	or	[ebx+2], d1		P 0 CS 001B 32位 0(FFFFFFFF)
004017D9	C1F8 04	sar	eax, 4		A 0 SS 0023 32位 0(FFFFFFFF)
004017DC	0843 03	or	[ebx+3], al		Z 0 DS 0023 32位 0(FFFFFFFF)
004017DF	8B4D C4	mov	ecx, [ebp-3C]		S 1 FS 003B 32位 7FFDF000(FFF)
004017E2	81F1 68245711	xor	ecx, 13572468		T 0 GS 0000 NULL
004017E8	3B0D 40254E0	cmp	ecx, [4E2540]		D 0
004017EE	75 0C	jnz	short 004017FC		0 0 LastErrr ERROR_SUCCESS (00000000)
004017F0	C705 AC7D4E0	mov	dword ptr [4E7DAC], 004E28B8	ASCII "秋棋"	EFL 00000202 (NO,NB,NE,A,S,P0,L,LE)
004017FA	E8 0A	jmp	short 00401806		ST0 empty -1.6259183529734161960e+4755
004017FC	C705 AC7D4E0	mov	dword ptr [4E7DAC], 004E28BD	ASCII "梦?"	ST1 empty -UNORM B27C 00000004 00000000
00401806	A1 D8784E0	mov	eax, [4E78D8]		ST2 empty -UNORM B564 00000000 950404BF
					ST3 empty -8.4211152663148318880e-853

注意到如下代码：

```

xor eax, eax;把eax清空

```

```

mov al, [ebx+2];把3, 4位放入al
add esp, 8;把栈的指针移动两个字节
and byte ptr [ebx+2], 0F;在内存中把第四位留下第三位清空
mov dl, [ebx+3];把1, 2位放入dl
and byte ptr [ebx+3], 0F0;在内存中把第一位留下第二位清空
and dl, 0F;在dl中把第二位留下第一位清空
shl edx, 4;把第二位对应的四位二进制数从后四位移动到前四位
or [ebx+2], dl;原先存三四位的空间变成了存二四位
sar eax, 4;把第三位对应的四位二进制数从前四位移动到后四位, 第四位的值被清空
or [ebx+3], al;原先存一二位的空间变成了存一三位
mov ecx, [ebp-3C];把最后得到的经处理的八位数传到ecx中
xor ecx, 13572468;与key值异或
cmp ecx, [4E2540];与machine code相比较
jnz short 004017FC;如果与machine code不等跳转
mov dword ptr [4E7DAC], 004E288B8
jmp short 00401806
mov dword ptr [4E7DAC], 004E288BD

```

继续跟踪，直至出现“BAD”此后没有再调用machine code和注册码，故推测此处处理已完成，

3 注册码算法

3.1 分析汇总处理和判定方法：

- 1.将输入的注册码XXXXXX的第7和第8位（视作十六进制数，每个代表4位二进制数）所组成的一个字节（8位二进制数）的内容，左移1位，存下，再将左移前的数右移7位，存下，将两个得到的数进行按位或操作，其效果等同于将两位十六进制数字所代表的八位二进制数的最前一位挪到最后端。
- 2.然后处理第5和第6位（视作十六进制数，每个代表4位二进制数）所组成的一个字节（8位二进制数）的内容，左移6位，存下，再将左移前的数右移2位，存下，将两个得到的数进行按位或操作，其效果等同于将两位十六进制数字所代表的八位二进制数的最后端两位挪到最前端。
- 3.而后对第1、2、3、4位进行操作，将第三和第四位视作十六进制数，每个代表4位二进制数）所组成的一个字节（8位二进制数）的内容和F0相与，保留第三位的内容。
- 4.最后与Key值13572468相异或。
- 5.得到的八位数据与machine code相比较，若相等输出“GOOD”，反之“BAD”。

3.2 分析逆向计算方法：

$$MachineCode \oplus 13572468 = XXXXXXXX$$

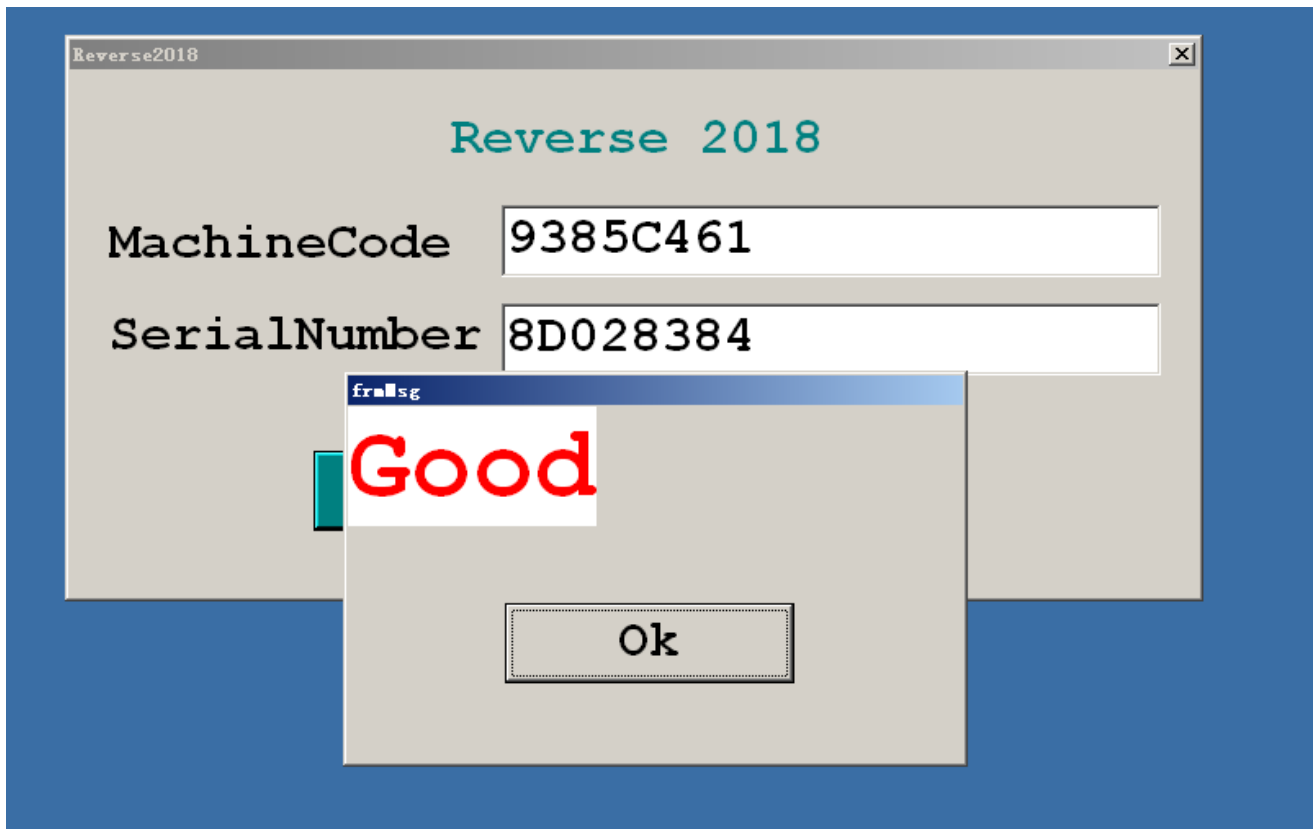
这个计算得到的值（8位16进制数）是需要和13572468这个Key值异或最后和MachineCode相比对的。

在此之前这个值的78位，被进行了左移一位，右移七位的操作，第56位则被进行了左移六位右移两位的操作，最后将23位相互交换了，所以需要逆向这个过程。

可以分为以下几个步骤：

1. 将二三位数字交换；
2. 将第五第六两位十六进制数字所代表的八位二进制数的最前端两位挪到最后端；
3. 将第七第八两位十六进制数字所代表的八位二进制数的最后一位挪到最前端；

```
E:\vc\VC6\MyProjects\reverse01\Debug\reverse01.exe
please enter your MachineCode:9385C461
That is your register code: 8D028384
```



附：汇编指令说明

SHL

SHL是一个汇编指令，作用是逻辑左移指令，将目的操作数顺序左移1位或CL寄存器中指定的位数。左移一位时，操作数的最高位移入进位标志位CF，最低位补零。

SAR

SAR是一个汇编指令，作用是算术右移指令，将目的操作数顺序右移1位或CL寄存器中指定的位数。右移一位时，保留操作数的符号，即用符号位来补足。