Target: ..... BackupMagic 1.4.0
Company: ..... Moonsoftware
Url: ..... http://www.moonsoftware.com
Protection type: ..... time limit, serial protection
What we want to achive: ... Here we are going to code a keygen ;-) as the serial algo is a very simple one
Tools required:
SoftIce
W32Dasm
Tasm 5.0 (assembler compiler)

Hi guys,
Again welcome to my tutorial.

Today I'll show you how easy it can sometimes be to code a keygen. Of course it depends on the algorithmic implementation done by the author. However, there are lot of programmer who concern merely in implementing the software itself and do not care much in security stuff as the serial algo will be discovered anyway by smart people ;-)

Let's begin: First of all launch the program and go to Order!/Enter Key Code. You'll see the usual Dialog Window requesting the Name and the Key for the registration. Enter some data and confirm.

A **MessageBox** will be displayed "*invalid Key .....*" or something like that. Ok, now we know the breakpoint to break at, namely **MessgageBoxA**. Go again to the **regwindow** and insert some data. Before confirming enter Softice and set a breakpoint on **MessageBoxA**.

bpx MessageBoxA

Press OK and you'll be thrown in softice. Exit the API function (F11). Again a dialog. Press OK once ... in Softice again. You'll break at the next instruction after the API call, which should be 0046817B -- sign up this suspicious instruction. A few instructions later notice the RET. So trance the RET (F10) and you 'll be thrown at the beginning of a loop, where you should see a call at this address 00477796.

Well, the next step is to laungh W32Dasm and disassemble BMagic.exe. Than go to the first address found above. Here you won't notice something sospicious. However I reccomend to check as many calls or jump references as possible! Be aware that most programmer hide such rutines in very elegant stile.

OK, than go to the second address found (00477796). A few lines above notice references to registration like "LicensedTo" ecc. As an experienced cracker the first thing to do is to check the conditional jump references. There is one at 0047766F. Well, point to that address. Here is what you get:

```
:00477621 A1089F4700        mov eax, dword ptr [00479F08]
:00477626 8B00              mov eax, dword ptr [eax]
:00477628 8B80D0020000      mov eax, dword ptr [eax+000002D0]
:0047762E E8B94AFBFF        call 0042C0EC
:00477633 8B45F0            mov eax, dword ptr [ebp-10]
:00477636 8D55FC            lea edx, dword ptr [ebp-04]
:00477639 E81A0AF9FF        call 00408058
:0047763E 8D55EC            lea edx, dword ptr [ebp-14]
:00477641 A1089F4700        mov eax, dword ptr [00479F08]
:00477646 8B00              mov eax, dword ptr [eax]
:00477648 8B80E8020000      mov eax, dword ptr [eax+000002E8]
:0047764E E8994AFBFF        call 0042C0EC
:00477653 8B45EC            mov eax, dword ptr [ebp-14]
:00477656 8D55F8            lea edx, dword ptr [ebp-08]
```

```
:00477659 E8FA09F9FF        call 00408058
:0047765E 66B99502          mov cx, 0295
:00477662 8B55F8            mov edx, dword ptr [ebp-08]
:00477665 8B45FC            mov eax, dword ptr [ebp-04]
:00477668 E8DF13FFFF        call 00468A4C
:0047766D 84C0              test al, al
:0047766F 0F84E9000000      je 0047775E
:00477675 A12CA04700        mov eax, dword ptr [0047A02C]
:0047767A C60001            mov byte ptr [eax], 01
:0047767D A110A04700        mov eax, dword ptr [0047A010]
:00477682 C60000            mov byte ptr [eax], 00
:00477685 8B8384040000      mov eax, dword ptr [ebx+00000484]
:0047768B 33D2              xor edx, edx
```

Notice the colored code. There is a move of some dword value in eax, a call, a test and a je. This stuff seems incredibly be involved in serial generation and checking. So go to call 00468A4C

```
:00468A4C 55                push ebp
:00468A4D 8BEC              mov ebp, esp
:00468A4F 6A00              push 00000000
:00468A51 53                push ebx
:00468A52 56                push esi
:00468A53 57                push edi
:00468A54 8BF9              mov edi, ecx
:00468A56 8BF2              mov esi, edx
:00468A58 8BD8              mov ebx, eax
:00468A5A 33C0              xor eax, eax
:00468A5C 55                push ebp
:00468A5D 68A18A4600        push 00468AA1
:00468A62 64FF30            push dword ptr fs:[eax]
:00468A65 648920            mov dword ptr fs:[eax], esp
:00468A68 85DB              test ebx, ebx
:00468A6A 7504              jne 00468A70
:00468A6C 33DB              xor ebx, ebx
:00468A6E EB1B              jmp 00468A8B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00468A6A(C)
|
:00468A70 8D4DFC            lea ecx, dword ptr [ebp-04]
:00468A73 8BD7              mov edx, edi
:00468A75 8BC3              mov eax, ebx
:00468A77 E81CFFFFFF        call 00468998
:00468A7C 8B45FC            mov eax, dword ptr [ebp-04]
:00468A7F 8BD6              mov edx, esi
:00468A81 E88EB4F9FF        call 00403F14
:00468A86 0F94C0            sete al
:00468A89 8BD8              mov ebx, eax
```

Our cracking experience tells us that we're very closed!!! I'm not able to explain you the plain functionality as I cannot program in asm, but I think it's sufficient to gain of a certain feeling for those instructions. Try to step into the two calls preceding the sete al instruction. The first is just a checking routine, in our case nothing interesting. The second call is exactly what we were searching for ... the serial generation routine!!! Mega storia!!!! Move to that location. A bit below you'll see this -- a very short serial algo!!!!

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004689EA(C)
|
:004689D1 33C9              xor ecx, ecx
:004689D3 8A4C06FF          mov cl, byte ptr [esi+eax-01]
:004689D7 03C8              add ecx, eax
```

```
:004689D9 0FB77DFE          movzx edi, word ptr [ebp-02]
:004689DD 0FAFCF            imul ecx, edi
:004689E0 69C9B2000000      imul ecx, 000000B2
:004689E6 03D9              add ebx, ecx
:004689E8 40                inc eax
:004689E9 4A                dec edx
:004689EA 75E5              jne 004689D1
```

Ok, now lets code the keygen (in asm obviously). Personally I've found a suitable template searching the internet. So all I've to do every time I code a keygen (rarely) is to change some things.

Here is the asm code which will compile with TASM 5.0 (.com file) --- Just a bit commented.

```
Ideal
Model Tiny
p386
CodeSeg
Org 100h

Inizio:
mov ah, 9
lea dx, [Intro]
int 21h ; introduction

inc ah
lea dx, [L_Max]
int 21h ; get name

mov bl, [L_Eff]
mov [byte ptr Buff+bx], 0 ; remove last character (0Dh)

lea esi, [Buff] ; name offest in esi
xor ebx, ebx ; prepare the other registers
xor eax, eax
inc eax

xor edx, edx
mov dl, [L_Eff]

; calculation algo
Loop1:
xor ecx, ecx
mov cl, [byte ptr esi+eax-01]
add ecx, eax
movzx edi, [word ptr Const1]
imul ecx, edi
imul ecx, 0B2h
add ebx, ecx
inc eax
dec edx
jne Loop1

mov eax, ebx ; in ebx there is the calculated code
xor ebx, ebx
mov bl, 8 ; repeat 8 times
xor ecx, ecx
mov cl, 10 ; division factor

; hex -> dec conversion routine
Dump:
xor edx, edx
cdq
idiv ecx
add dl, 30h
mov [Codice+bx-1], dl
dec bx
jne Dump

mov ah, 9
lea dx, [Cod] ; display calculated serial
int 21h
int 20h
```

```
Intro db 'BY RUSTY - The Hacker',13,10,'Key Generator for BackupMagic 1.4.0',13,10
db 'a very easy one!!!!!',13,10, 'Enter your name : $'
L_Max db 16
L_Eff db 0
Buff db 16 dup (0) ; Buffer where name is saved
Const1 dw 661
Cod db 13,10,'serial : '
Codice db '00000000$' ; here is where serial is saved
End Inizio
```

Here is a Makefile-example for the pure newbies.

```
# makefile for RUSTY.COM
# Copyright (c) 1988, 1996 by Borland International, Inc.
#
# make -B To build RUSTY.com

rusty.com:
tasm rusty.asm
tlink /t rusty.obj
```

Well, I hope you have learned something. Ah, I forgot: Buy those software products from Moonsoftware as they are really not expensive and very useful sometimes !!!!! Support those fair programmers!!!

See you in the next time
Bye ;-)
Rusty