Ghiribizzo's Cracking Tutorial

NuMega's BoundsChecker 5.xx

NuMega's BoundsChecker 5.xx

NuMega's BoundsChecker is a tool for software development but is also useful for crackers. NuMega have been nice enough to distribute their software for free (14 day trial) so go to their site and download their stuff (you can get SoftIce too!).

PGP and Signed Tutorials

My tutorials and programs should be signed electronically using PGP. PGP 5 supports DSS/Diffie-Hellman keys. These keys are not supported by previous versions of PGP.
You should check the signature to make sure that the tutorial and especially its program files have not been tampered with. All cracks, tutorials and zip files I release will be signed. This will prevent tampering and will hopefully reduce the chances of viral infection.
My signature will also be the only way you can identify me as my email address will often change.

My Web Site:	http://www.geocities.com/Athens/3407
My Email:	Ghiribizzo@geocities.com
My Backup Email:	Ghiribizzo@hotmail.com

When I download the program (BoundsChecker 5.01 C++ Builder version), I installed it and went directly to cracking! (a sad affliction of crackers - cracking the software is so much more fun than using it). I ran BoundsChecker and when you load a program into it, you're presented with a screen and an option to purchase. A 16 digit serial number is also displayed. It looks like a standard serial/key combination. It turns out that the key depends entirely upon the 16 digit serial number and not on your name/company.

In my hurry to crack the program, I registered it instantly so I couldn't go back to looking at the key generator routine. The crack itself is trivial - a by the book serial/key generation. I can't remember exactly how I did it but I'll try to give an accurate description. I placed a bpx on hmemcpy and (if I remember correctly) followed the stack backwards. I then scanned the code under SoftIce looking around until I saw the following code:

```
push  eax
push  ecx
call  some function
...
followed by compare routines and a conditional jump to a messagebeep (I think)
```

This couldn't be more suspicious. I dumped the contents of eax and ecx and then found that they contained my guessed key and the correct key. So leaving SoftIce, I typed in the correct serial and that was it. Luckily, I ran C++ and the C++ version of BoundsChecker hadn't been registered yet so I get to go through it again! This time I did a bpx getwindowtext and placed breakpoints on the serial code (search though memory for it). Looking around you can quickly find the following pieces of code (even easier if you bpr on the serial locations) anyway the following code listings are taken from Shadows tutorial on cracking the Visual C++ version of BoundsChecker. The comments are by him the code is pretty self-explanatory.

Processing and storing the serial:

```
014F:014F28C6 PUSH  1501070
              LEA   EAX,[EBP-04]
              PUSH  EAX
              INC   ESI
              CALL  14F4E80
              ADD   ESP,08
              LEA   ECX,[ESP-04]
014F:014F28BD MOVSX EAX, BYTE PTR [ESI:015044DF]
              PUSH  EAX
              PUSH  015014B8
              PUSH  ECX
              CALL  14F4E10
              ADD   ESP,0C
              LEA   ECX,[EBP-04]
              PUSH  ECX
              CALL  14F4E00
              ADD   ESP,04
              MOV   [ESI*4+EBP-48],EAX          ; Store Number
              CMP   ESI,10                      ; Ox10=16 Digits
014F:014F2904 JB    014F28C6
              PUSH  DWORD PTR [EBP-34]
014F:014F2904 JB    014F28C6
```

Notice that there are 16 blocks of code similar to the one below. Five numbers are passed as parameters and a call is made to a function which generates one digit of the key from the 5 numbers.

```
              PUSH  DWORD PTR [EBP-34]  ;P1
              PUSH  DWORD PTR [EBP-08]  ;P2
              PUSH  DWORD PTR [EBP-5C]  ;P3
              PUSH  DWORD PTR [EBP-28]  ;P4
```

```
                PUSH  DWORD PTR [EBP-44]  ;P5
                CALL  014F257B
                ADD   ESP,14
                MOV   [EBP-009C],EAX
```

The following function is the generator itself. It's pretty easy to understand.

```
014F:014F257B PUSH  EBP
              MOV   EAX,[ESP+08]     ;EAX=P5
              MOV   EBP,ESP
              ADD   EAX,[ESP+0C]     ;EAX=P4+P5
              CMP   EAX,9            ;if (EAX>9) EAX-=0A
              JLE   BR1
              SUB   EAX,0A
BR1:          SUB   EAX,[EBP+10]     ;EAX-=P3
              JNS   BR2              ;if (EAX<0) EAX+=0A
              ADD   EAX,0A
BR2:          ADD   EAX,[EBP+14]     ;EAX+=P2
              CMP   EAX,9            ;if (EAX>9) EAX-=0A
              JLE   BR3
              SUB   EAX,0A
BR3:          SUB   EAX,[EBP+18]     ;EAX-=P1
              JNS   BR4              ;if (EAX<0) EAX+=0A
              ADD   EAX,0A
BR4:          POP   EBP
              RET
```

There are 6 numbers in memory before the serial number. This is the version of the program and the build number.

Also, if you enter the wrong code and step through the code afterwards you'll see that a conditional jump which makes a call to a function ***which overwrites the correct key in memory with a false one!*** This is quite interesting. I assume that the code has been put in to thwart attempts to 'hear the echo'. The false key is placed in memory ***directly before*** the key you entered. Anyone who runs the program once through and then tries to do a memory dump will fall for this trap. I don't know what entering this key will do, and I can't test it as I've already registered the C++ builder version. But it could do anything from falsely 'registering' you or something more sinister...

That was the only special thing in the protection scheme. It is otherwise quite boring and very easy to break. Let me finish by giving the build numbers of some of the versions you can download from NuMega's website.

BoundsChecker 5.02 Visual C++ Edition: 502597
BoundsChecker 5.01 C++ Builder Edition: 501697
BoundsChecker 5.01 Delphi Edition: 510597

Well a cracker's work is never finished. Having registered both the standard version and the Builder edition, I couldn't look at the protection further (never be too quick to register!). So I downloaded the Delphi edition. There was an initial fiasco of pressing 'cancel' on the trial screen - this altered things so that BoundsChecker no longer loaded when starting Delphi. I tried to sift through the registry and put in 'IntegratedDebugging=1' as it appeared in the Builder version, but this didn't help. In the end I had to uninstall BoundsChecker, restore an earlier copy of the registry (these are VERY useful!) and then reinstall BoundsChecker. When I loaded Delphi, BoundsChecker popped up, but it said that my 14 day trial had ended. Though it kindly offered me the chance to purchase it so I could look at the protection scheme again.

Looking through the code again, I noticed some things that I missed first time round. In the earlier WinIce sessions, I had jotted down some search strings to locate some sections of the protection code.

Searching the main executable for this yielded nothing. Probably a .dll then. There are many ways you could find this. I chose to use WinIce because there were other things I wanted to check (e.g. the build number for Delphi version etc.). Once I landed in the protection code, I looked at the stack and was amazed by what I saw: TL32V20! NuMega is using TimeLock to protect it's code! Now I haven't looked at TimeLock protections before (not knowingly anyway!) but I vaguely remember seeing some tutorials written about it before. Strange that no one has spotted this yet as the key generator is quite distinguishable! Anyway it happens that version A of TimeLock is used TL32V20.DLL is 91,648 bytes long.

Let's use Hiew and dump some of the code:

This is where the Version/Build number is fetched and stored and straight after that the serial number is fetched and stored.

```
.000025CB: 6870100110                  push   010011070
..
..    snip
..
*.00002602: 8944B5A0                    mov    [ebp][esi]*4[-0060],eax
*.00002606: 83FE06                      cmp    esi,006
*.00002609: 72C0                        jb    .0000025CB   ---------- (4)
.0000260B: 33F6                        xor    esi,esi
.0000260D: 6870100110                  push   010011070
..
..    snip
..
*.00002644: 8944B5B8                    mov    [ebp][esi]*4[-0048],eax
*.00002648: 83FE10                      cmp    esi,010
*.0000264B: 72C0                        jb    .00000260D   ---------- (8)
```

This is where I got my serial for the standard version of BoundsChecker. It is an obvious compare, test, jump sequence.

```
.00003ED0: FF15DC630110                call   GetWindowTextA ;USER32.dll
.00003ED6: 8D45D8                      lea    eax,[ebp][-0028]
.00003ED9: 50                          push   eax
.00003EDA: E885E9FFFF                  call  .000002864   ---------- (3)
.00003EDF: 83C404                      add    esp,004
.00003EE2: 8D45EC                      lea    eax,[ebp][-0014]
.00003EE5: 8D4DD8                      lea    ecx,[ebp][-0028]
*.00003EE8: 50                          push   eax <-key you entered
*.00003EE9: 51                          push   ecx <-real key
*.00003EEA: E891170000                  call  .000005680   ---------- (4)
*.00003EEF: 83C408                      add    esp,008
*.00003EF2: 85C0                        test   eax,eax
*.00003EF4: 7553                        jne   .000003F49   ---------- (5)
```

Now let's take a look at what happens when we enter the wrong code:

```
.00003F49: 8D45D8                      lea    eax,[ebp][-0028]
.00003F4C: 50                          push   eax
*1.00003F4D: E859E6FFFF                  call  .0000025AB   ---------- (1)
```

*1 Routine to overwrite key with hardwired key

```
.00003F52: 83C404                      add    esp,004
.00003F55: 8D45EC                      lea    eax,[ebp][-0014]
.00003F58: 8D4DD8                      lea    ecx,[ebp][-0028]
*2.00003F5B: 50                          push   eax
*.00003F5C: 51                          push   ecx
```

```
*.00003F5D: E81E170000                    call  .000005680   ---------- (2)
*.00003F62: 83C408                        add    esp,008
*.00003F65: 85C0                          test   eax,eax
*.00003F67: 753E                          jne   .000003FA7   ---------- (3)
```

*2 Your key is compared with the hardwired key

```
*3.00003F69: 6A6B                          push   06B
*.00003F6B: E8CAD1FFFF                    call  .00000113A   ---------- (4)
*.00003F70: 83C404                        add    esp,004
*.00003F73: 85C0                          test   eax,eax
.00003F75: 7420                          je    .000003F97   ---------- (5)
```

*3 What does this call do? Does it really restore you trial period?

```
*4.00003F77: 6800200000                    push   000002000
*.00003F7C: 68193B0110                    push   010013B19
*.00003F81: 6855410110                    push   010014155
*.00003F86: 56                            push   esi
*.00003F87: FF15A8630110                  call   MessageBoxA ;USER32.dll
```

*4 This message box says that your trial period has been restored!

```
.00003F8D: C7053410011001000000          mov    d,[010011034],000000001
.00003F97: 6A01                          push   001
.00003F99: 56                            push   esi
.00003F9A: FF15B8630110                  call   EndDialog ;USER32.dll
.00003FA0: B801000000                    mov    eax,000000001
.00003FA5: EB5D                          jmps  .000004004   ---------- (6)

*5.00003FA7: 6800200000                    push   000002000
*.00003FAC: 68193B0110                    push   010013B19
*.00003FB1: 68EB410110                    push   0100141EB
*.00003FB6: 56                            push   esi
*.00003FB7: FF15A8630110                  call   MessageBoxA ;USER32.dll
.00003FBD: EB2D                          jmps  .000003FEC   ---------- (7)
```

This is just the ‘incorrect key’ message box.

Now I didn’t execute the code listed in 3. Call it a healthy dose of paranoia! (Have you read my tutorial - ‘How to Protect Better: A Strategy’?). I’ll leave you to investigate for yourself. Placing the protection in the TimeLock dll opens a world of opportunity. There is more than enough space within the dll to place your own routines. Keeping a ‘cracked’ copy of this dll could mean automatic registering of all TimeLock protected programs... but where’s the fun in that? If you decide to patch the dll itself, I’d recommend putting in code to display the correct key in a message box rather than have it perform an automatic registration. Then you can continue to crack the program and also have a check for later. See my tutorial on cracking HexWorkshop to find out the half-baked way to creating this sort of message box. Also for those who can’t be bothered, get my keymaker. This will register all the BoundsChecker programs currently available for trial by NuMega and may also work for TimeLock protections in general - though I have yet to test it on such a program.

OK. I just can’t help myself. Still curious about TL32V20.DLL, I loaded it into W32DASM and had a look around. Try it, there’s some quite interesting stuff in there. Remember the key generation scheme? The block of 16 of these:

```
PUSH  DWORD PTR [EBP-34]  ;P1
PUSH  DWORD PTR [EBP-08]  ;P2
PUSH  DWORD PTR [EBP-5C]  ;P3
```

```
PUSH  DWORD PTR [EBP-28]  ;P4
PUSH  DWORD PTR [EBP-44]  ;P5
CALL  014F257B
ADD   ESP,14
MOV   [EBP-009C],EAX
```

You can't fail to miss it really. And you certainly can't fail to miss 2 blocks of 16. Two blocks? Taking a closer look, I discover that the same key digit function is called and the same version/build/serial numbers are called, only the parameters passed into the key digit function are different. I already am thinking that this is the generator for the 'fake' key and that the key isn't hardcoded into TimeLock nor in BoundsChecker as I had first thought (I already took a quick look at a few of the files and had found nothing). I quickly rewrite a new keygenerator for these new parameters and out pops the 'fake' key.

I've still not tried entering in this key yet, but what I imagine happens is that the trial period is reset and then the serial must be changed to prevent someone reusing this key. This would give NuMega the option of restoring a customers installation if it went wrong. I'll be phoning NuMega soon to ask if my trial can be reset.

Well, I couldn't be bothered contacting NuMega. Instead, I went ahead and entered the 'fake' key. My prediction was right. The trial period is reset and then the serial number is changed. I did this while filemon and regmon were running. Having a quick peek around, I noticed that certain registry entries were changed and also a file (bcdlphi5.tsf - 2602 bytes) was accessed. Having a quick look through some tutorials regarding TimeLock protections I found that decrypting the tsf file has already been covered so I wont go into it here. Also I noticed that Riz la+ had commented in his tutorial: "What about testing your knowledge on [tlock32.dll], which seems to be TimeLock v1.0? It is used by NuMega's BoundsChecker". So it was already known that early versions of BoundsChecker used TimeLock.

I think that I had better end the tutorial here as it is dragging on. I am astounded that such a poor protection scheme is used by so many programs. I cracked BoundsChecker standard edition in less than 5 minutes.

If you have further comments on this or want to send me your tutorials, then contact me. Don't send me any tutorials without first asking!