Keygen Injection 2.0

Author: Crudd [RET]

Tools:

```
SoftIce (of course)
Hiew
MASM (to code the loader in, or any assembler or compiler)
APIS32
```

Theory:

Ok, maybe this isn't a new idea. But I’ve never heard of it before, so here’s the idea: Code a loader that patches the .exe at runtime (nothing new about this), but the loader will then do some keygen injection. This of course has its advantages and disadvantages. Here are a few differences from keygens/patches:

1. You won’t have to rip/understand the whole serial algorithm. I'm not sure if this is an advantage or disadvantage, but it’s definitely easier. Also, this is nothing new to keygen injection.
2. A keygen only patches the program in memory and not the executable on the file system. This is much better than a patch because you won't have to worry about bugs you may have created by patching.
3. Provides you with a valid serial. Not just a patch that is good for only the current version.
4. Something new and exciting to experiment with. I don't believe I have truly discovered the potential of this new idea. We will see what comes of it.

Now, on to our first target...

Step 1: Scoping our target

Ok, for our target we are going to use AcidBytes’ CFF #4 crackme (get it from http://www.crackmes.de). Taking a quick look at it, we notice it is packed. Let’s run it and see what’s going on. Enter our name and serial. Ill use Crudd and 12345666. Click 'Check if valid'. Pop, we get a message box saying: 'Your name must be at least 6 chars long' or something to that effect. And then another saying our serial is wrong. Ok, let’s put in Crudd [RET] and check again. Pop, 'The serial you entered is wrong in any case'. Ok, so let’s jump into SI and set some breakpoints. We'll go with the usual breakpoints of: GetWindowTextA, GetWindowTextW, GetDlgItemTextA, and GetWindowItemtextW. Out of SI, click the check button again. Pop, same message box and no SoftIce...well shit. Lets see, maybe we can break on MessageBoxA. Let's try.

BOOM

Softice pops up hit F12 and we are at 440213. Let's see if we can find where our bad jump is. Looking around I don't notice anything special. After tracing through a bit we come to 457E67. Looks like our Message Box code is called from 457E62, scrolling up, we notice a JNZ to it at 457E33 and a call right before it at 457E2E. Lets bpx on this call and check it out. Set our bpx and jump out of SI and click check it again. This time we break at our call, so lets F8 to step into it and see what's happening. At 403B87 it would appear that ESI is pointing to our real serial. And right after that EDI is set to point to the serial we entered. Let's test this out. 1082 is what ESI is for me. So I put it in the box. *Boom*, we get our good job message. Now we just need to do our dirty work. We will use 457E33 to jump to our code. We need to remember that offset. Now we'll get on with it...

Step 2: Patching the file in memory

Now, since this targets packed we'll need to let it unpack and then jump to our patch and then to OEP. I'm not going to go over unpacking or patching packed targets in general or the theory behind it. I'm just going to talk about how I did it on this particular target.

First we'll bpx GetProcAddress. Now, keep F5/f12'ing until we are in our programs code and well see this:

```
0046F7FF     Call dword ptr [esi+00070160]     ; GetProcAddress
0046F805     or EAX, EAX
0046F807     jz 46F810
0046F809     mov [EBX], EAX
0046F80B     add EBX, 4
0046F80E     jmp 46F7F1
0046F810     call dword ptr [esi+00070164]     ; ExitProcess
0046F816     popad
0046F817     jmp 00458250
```

46F817 is the jump to OEP. This offset is at 2BC17 in our file, I'm not going to explain how I got this number either, as it's in plenty of tutorials. So we are going to change this offset to jump to our patcher code and then we will jump to OEP when we are finished. So let's find us a cave for our patching work. For those of you who don't know, a cave is an allocated space the program doesn't use itself. It can either be in memory or on disk. First we will do it on disk, then later our loader we will do it in memory. There is one not too far away at 2BC30. So let's fire up Hiew and start assembling our needed patch.

Step 3: The patch

This is going to be the most tedious part of our loader. What we want the program to do is check our serial, if it’s wrong then enter the correct one into the serial edit box. So we will need to a few things to accomplish this. We will need LoadLibraryA, GetProcAddress, and SetWindowTextA. We need the first two (which are used by the packer) to get the third one (which will be used to put our serial in the edit box). If you are confused by the API's and why we need them, just check your API guide, it will also be useful to know which parameters we need to push. We will need the hWnd of the editbox that we are going to put our serial in. Originally, I just had my loader do this for itself, but as it turns out we need the program to do it before the jmp to OEP because after the call to ResumeThread it loads all the resources before my loader gets control again. We'll go ahead and break this step down into smaller steps to make it a bit easier to understand.

Step 3a: Gaining control

Load the file into Hiew and go to our offset (2BC16). Then hit F3 to edit the file, and then NOP the 'popad' instruction. We are doing this because right now ESI+xxxxx points to our needed API's, we will put the 'popad' right before we jump to OEP. Then move down to 2BC17 hit F2 to change into ASM mode. Then all we need to do right here is change it to 'jmp 2BC50' to make it jump to where our patch will be. We now have control of the program.

Step 3b: Getting what we need

First we will need to load User32.dll so we can get the address of our needed SetWindowTextA. To do this we need to find where LoadLibrary is called. So we'll just put a bpx on it and see what happens. Run the program, F12 into the crackme and we end up at 46F7F0. Right above that we see 'Call [ESI+7015C]', this is our call to LoadLibraryA. We will remember that call dword ptr [esi+00070160] is our call to GetProcAddress. We now have what we need to obtain SetWindowTextA. We will need to put the Strings of the .dll and API we want to load into our cave somewhere. So ill just go down to the bottom (so we still have plenty of room for our patch) and put these two values in Hiew. I started a 2BDC1, just use any hexeditor to type these in. User32.DLL(null)SetWindowTextA(null). The two nulls are just zero-bytes to terminate our stings, they need to be there. Make sure you are at the correct offset. Now we have our calls and the parameters we need so let’s do a little coding.

Step 3c: Coding: Part 1

Ok, we have our jmp to our cave so let’s go ahead and get our SetWindowTextA API. So let’s go into Hiew, go to our cave and at 2BC50. We will push the .dll we need loaded (remember where we put User32.dll). Then we will call LoadLibrary. Next we'll push

SetWindowsTextA, EAX (the handle of User32.dll) and then call GetProcAddress. Here's what you should type into Hiew:

```
push 0046F9C1                    ;User32.dll
call dword ptr [esi+0007015C]    ;LoadLibraryA
push 0046F9CC                    ;SetWindowTextA
push eax                         ;The handle to User32.dll that
LoadLibrary returns
call dword ptr [esi+00070160]    ;Call GetProcAddress
```

Now, we will need the handle that GetProcAddress returns to call SetWindowText later, so we will store this somewhere. Searching through the data window in SI I found a decent sized cave at 45C700. This is where we will store our needed info. Now back to Hiew, go to ASM mode and type: 'mov ebx, 45C700' then enter and then 'mov d,[ebx],eax'. Like I said earlier, we need to get the handle of our serial window now or else we will miss all the CreateWindowExA calls. So first let's find where it stores our edit box's hWnd. To do this we will use APIS32 (get it from protools). So fire it up, open the crackme and click 'run'. It'll give us a bunch of information, but the only things that concern us are the calls to CreateWindowExA. Specifically the second TEdit (we will assume this is our serials edit box). We notice it's the 4th call to CreateWindowExA so let's set our breakpoint and F5 3 times to get to our call. We end up at 427200. Here eax gets moved into ESI+140. This is where our handle to our serial edit box is stored. Originally I assumed this to be constant, but after writing this essay and then the loader, it turns out its not. So now we need to place a jump here to our cave, save ESI+140 and then jump back where we left off. We are going to put our patch at the bottom of our other cave because I already coded the entire cave before I ran into this problem. This doesn't affect the rest of the essay; it's just a note about why the first patch is at the bottom. So set a bpx at the beginning of our patch (2BC50) and then run the program. Step down a few lines to the end of our patch to 46F86E. Now we are going to assemble our patch in SI because we can't do it in Hiew without knowing the bytes of the patch. So for now, just asm popad and jmp 458250 (OEP). And then bpx on 427200 (our call to CreateWindowExA). Hit F5 and we will pop up at 427200. Here we want to jump to our cave at 458305 (the bottom of our soon to be patch). So type 'a' and hit enter. This will assemble on the current line. And then just type 'jmp 458305' [ENTER] then 'NOP' [ENTER]. The NOP is to clean it up a bit because our new op code is one byte shorter than the old one. Now write down your new bytes. 'D 427200' and we will see 'E9 00 11 03 00 90'. The 90 is our NOP and the first 5 are our jmp. Finished there, so lets F8 or F10 to jmp to the cave. Here we will just need to move EAX to ESI+140 and to our data cave and then jump to the next op code of the program. Once again type 'a', then:

```
mov dword ptr [ESI+140],EAX     ;Save our handle for the crackme's use
mov dword ptr [45C708], EAX     ;Save it for our use :)
jmp 427206                      ;jump back to the crackme
```

We will need to write down these bytes for our loader as well. So 'd 458305' and we see '89 86 40 01 00 00 A3 08 C7 45 00 E9 F1 EE FC FF'. Back to Hiew, we need to write these bytes before we jump to OEP. And remember each dword will be in reverse order. In Hiew go to 2BC6E (the current bottom of our cave). Here's where we will code the last part of this patch. In Hiew hit F3 (edit) F2 (ASM) then:

```
mov eax, 00427200                ;Set EAX to our patch location
mov dword ptr [eax], 031100E9    ;First dword of patch
add eax, 00000004                ;move past our patch
mov word ptr [eax], 9000         ;Last two bytes of patch
mov eax, 00458305                ;Set EAX to our cave
mov dword ptr [eax], 01408689    ;First dword patch
add eax, 00000004                ;Move EAX to our second dword
mov dword ptr [eax], 08A30000    ;Patch it
add eax, 00000004                ;3rd dword
mov dword ptr [eax], E90045C7    ;Patch
add eax, 00000004                ;Forth
mov dword ptr [eax], FFFCEEF1    ;Patch
popad                            ;Restore the registers
jmp 00458250                     ;Jump to OEP
```

Step 3d: Coding: Part 2

Well, now that we have made the unpacking routine fetch and store our API and our handle we just need to get control where it checks if our serial is good or not. Then jump to our code, and put the right serial in the edit box if it's wrong, otherwise show the 'good boy' message. To do this, there are a few things we need. The hWnd of our edit box (which we have), our API (which we have) and the location of our good serial (which we got earlier). It stores our serial at a different place in memory each time the 'Check it' button is pushed, so we'll have to store our good serial somewhere. At 403B87 we will jump to a cave at 4582D0 and save this value right next to the address of SetWindowText. So in SI, type 'asm 403B87'. Note that the jump is far and it'll write over some other instructions. We will just put this in our cave too and jump to the next full instruction. Just type 'jmp 4582D0'. Go down to 4582D0 and asm there. We will need to store the address of our real serial, move eax to esi and edx to edi and compare eax and edx (these are the 3 instructions our jmp destroyed) then jump back to 403B8D. So in SI type 'a 4582D0'. Then 'mov dword ptr [45C704], eax', this stores the address to our serial in our .data cave. Then, just restore our destroyed instructions:

```
mov esi,eax                      ;Crackme does this, we are just fixing
whut our jump fucked up
mov edi,edx                      ;ditto
cmp eax,edx                      ;ditto
jmp 403B8D                       ;Back to the program
```

Ok, finished there. Now we just have to make it display the real serial in the edit box. Now at 457E33 we need to jump to our new routine. Type 'a 457E33' then 'jmp 4582E0'. This is in the location right after our last patch. Make sure we note once again which instruction we write over with our far jump. This time we will handle the overwritten instructions first. In SI type:

```
a 4582E0
jne 4582EE                       ;jmp to our patch otherwise its the
right serial
```

```
push 0                          ;params for the good_boy msgbox: MB_OK
mov ecx, 457F38                 ;msgbox title: 'Congratz'
jmp 457E3C                      ;jmp to the rest of the msgbox params
and call
```

Now our patch to display the correct serial (finally, I know...I had no idea it would be this long of a paper.) So next we type in SI:

```
push dword ptr [45C704]         ;our good serial
push dword ptr [45C708]         ;our serial boxes handle
call dword ptr [45C700]         ;SetWindowTextA
jmp 457E67                      ;back to our program
```

Wow, all patched and working in memory; now to code our loader.

Step 4: The loader

Get the loader here. We will need to write down all the bytes we patched from SI or dump it. I wrote them all down so you don't have to :) The source for the loader is pretty self explanatory so I won't go over it. I've commented the important or hard to understand parts. If you have any questions, as always you can email me at: Crudd@DrunkenBastards.com or catch me on #efnet ...

Final Words

I hope that explains everything and makes sense. This is the longest essay I've ever written and I hope I didn't go off track and out of order too much on it. We could've made this do other neat things but it's just a theory essay. It wouldn't have been much more work to make our patch 'push' the 'Check it' button or put the write serial in a messagebox for us. I think there is plenty more to be discovered here. Hope everyone enjoyed this. The full source of my loader is included in the .zip file. Any Question's; feel free to mail me.

Greets: [RET], Devine9, Muad'Dib, SantMat, noptical, kw, thesna, NeO'X'QuiCk, anyone I forgot (sorry) and anyone who reads this.

Thanks to: Beer, Acid Bytes for this crackme, drunken chicks, and of course you.

Mail me at: Crudd@DrunkenBastards.com
or check out: http://crudd.cjb.net and http://www.reteam.org