How to reverse Neopaint 4b.

Written by hogboblin.

Greetings all crackers and newbies.

Here is a tutorial on how the program Neopaint 4b can be reversed. This tutorial is primarely written for newbies. In this tutorial I will concentrate on how to patch a Shrinker packed file. I will show you how a progam can be cracked by adding a couple of new instructions in the program code. Hopefully it will also shed some light on the "shareware flag" type of protection frequently used by software developers.

Target is: Neopaint 4b, which can be found at www.softseek.com. Just do a search and you'll find it.

This program is powerful image editor with a lot of useful features, and it is well worth its money. So if you're merely planning on using this tutorial in ripping of the programmer, stop reading and go find a warez site or some other place were lamers hang out.

Tools: Softice, W32dasm, Procdump, Hexworkshop, Deshrinker , Gettype, Hiew.

Let us take a look at the program. This program is packed with Shrinker 3.4. Looking the program up in Gettype didn't reveal this in clear language, but if you try it, you'll see that the header information displays that the file contains 4 sections named shrinker 0, shrinker 1, shrinker 2 and shrinker 3. That indicates that shrinker 3.4 has been used to pack the file.
Now, I tried to unpack it using Deshrinker 1.5 and Procdump, but without success. That is, the unpacking process went well, but the program will not run after unpacking it. It displays a run time kernel error. The same thing happened after trying to unpack it manually using Softice and Procdump. The only result was a file that could be disassembled in W32dasm. But that's nice. We need that.
Most of the time, unpacked files, whether they run properly or not, can be disassembled in W32dasm. (And if that don't work, use IDA. That gem of a program disassembles almost anything). But every now and then you will encounter unpacked program that W32dasm can't handle. Tip: If this happens, run the program, then minimize the program window. But don't shut it down. Then make a full dump of it using Procdump. Very often this file can be disassembled properly, even if you had problems disassembling it after a regular unpacking process.

Now, let us get back to business.
I first tried to sniff out the serial using Softice. It was quite easy to locate the calculation routines and where the comparison routine is. But getting the real serial displayed proved to be much more complicated. So I skipped that. While I was going through the deadlisting from W32dasm, I got a pretty good idea on how to crack this program anyway. And since I kind of like the challenge of patching packed/compressed files, that's what I did.

When we run the program, the first thing we see is the opening nag displaying the text "Unregistered version". After some seconds the main window appears, but at the same time we see a shareware/evaluation nag screen with some shareware information. This is the same

dialogbox you will see if you click on help/about in the menu bar. In the registered version you will see some other text though. The caption text says “Unregistered version”, and at the bottom there is some text stating how many days you have been evaluating this program. When you shut down the program, you will see a little guy strolling along the bottom of your computer with a sign saying “Don’t forget to ordrer Neopaint” or something like that.
I don’t actually know whether this program stops to work after the 30 day evaluation. I didn’t try.

Well, after tracing through the deadlisting for a while, I saw something interesting. While I had the string reference box open I doubleclicked on the textstring “days…|Now might be a good time to purchase Neopaint!” I ended up here:

```
..
..
:004814F1 648920                mov dword ptr fs:[eax], esp
:004814F4 833D68874D0000        cmp dword ptr [004D8768], 00000000
:004814FB 7556                  jne 00481553
:004814FD E8E266F8FF            call 00407BE4
:00481502 E83115F8FF            call 00402A38
:00481507 2B0570874D00          sub eax, dword ptr [004D8770]
:0048150D 40                    inc eax
:0048150E 83F81E                cmp eax, 0000001E
:00481511 7E40                  jle 00481553
:00481513 E8CC66F8FF            call 00407BE4
:00481518 E81B15F8FF            call 00402A38
:0048151D 2B0570874D00          sub eax, dword ptr [004D8770]
:00481523 40                    inc eax
:00481524 8D55FC                lea edx, dword ptr [ebp-04]
:00481527 E88454F8FF            call 004069B0
:0048152C 8D45FC                lea eax, dword ptr [ebp-04]
:0048152F 50                    push eax
:00481530 8D55F8                lea edx, dword ptr [ebp-08]

* Possible Reference to String Resource ID=00200: " days...|Now might be a
good time to purchase NeoPaint!"
                                |
:00481533 B8C8000000            mov eax, 000000C8
:00481538 E83355F8FF            call 00406A70
:0048153D 8B55F8                mov edx, dword ptr [ebp-08]
:00481540 58                    pop eax
:00481541 E8A622F8FF            call 004037EC
:00481546 8B55FC                mov edx, dword ptr [ebp-04]
..
..
```

It’s quite obvious that this message never will show up in a registered version. It looks like a time check at the adress 0048150E. If the value in eax is less than 1E (which is 30 in decimal), the program jumps. If not the message is displayed. But look at the address 004814F4. If the value stored in [004D8768) is anything else than 0, the program will jump over this message. It also jumps over the timecheck. This looks like a “shareware flag” to me. Let us check this out. When I did a search for [004D8768] in w32dasm, I found 14 occurrences where the program compares the stored value in this adress with 0. I also found 1 occurrence where value in [004D8768] was pushed into the edx reister. Check it out for yourself.
All of the 14 occurrences dealt with shareware text and so on. The interesting thing is that the last occurrence was found within the section

of code that deals with the writing/storing of a reg number and name. And this routine is only called once in the program, and that only happens if the program accepts the entered serial. (You can check this out by going to the address 0049980C and the check out calls and code from there].
An educated guess would be this: The value stored in [004D8768] is by default 0. And it will be changed to something else after a successful registration.
Now, why don't we try to write in some new code that's executed at startup that makes sure that the value in [004D8768] is for instance 1?

Put a bpx GetModulehandleA in Softice and run the program. Softice will break, and go on push F11 and then F5. After doing this a few times, you'll end up at the adress 006755A8. (Putting a breakpoint on the api call GetModuleHandleA ensures that Softice breaks very early in the program execution). When we're dealing with shrinker compressed programs, and we're about to patch them, we have to make sure that Softice breaks in the loader section. You will not see this name using Procdump or a hexeditor, but you will see it displayed in the text line in the Softice window.We have to make our redirectional jump in this section of the program.
Now, trace down the code until you reach this code:

```
..
..
:0067565C  FF7510      push dword ptr [ebp+10]
:0067565F  FF750C      push dword ptr [ebp+0C]
:00675662  FF7508      push dword ptr [ebp+08]
:00675665  FF55DC      call [ebp-24]
..
..
```

When the program executes the last of the listed instructions in this push table, the program has finished the unpacking process, and continues as normal. If you're a newbie to shrinker packed programs, make a note of this. This is a basic characteristic for shrinker compressed programs. This is also the best place to redirect the program to a place where we can write in some new code.
Now, open up a your hexeditor. (I use Hexworkshop). Do a search for the code I just listed. When you find that, scroll up/own until you find some empty space. I scrolled down to the offset adress 100638. Lots of space there. I wrote down the offset, and opened up Neopaint in Hiew. I did a search for the offset there. The purpose of this is to find out what the adress was. We need this back in Softice later on. I found the adress to be 0066F038.
Back in Softice, after a bpx GetModuleHandleA and tracing down to the adress 0067565C, I typed a eip, enter and then changed the instructions to this:

```
:0067565C  E9D799FFFF        jmp 0066F038
:00675661  90                nop
..
```

(This code has later on to be written into the program at the offset location you found in the hexeditor).

When the program comes to the adress 0066F038, take a look at the register values displayed in Softice. Look at the esi value. At this time of teh program execution it's 00000001. Perfect for our use here. Type in mov [004D8768], esi and enter. Then type in the code we overwrote at 0067565C. The new code should look like this after writing it in (at offset 100638):

```
:0066F038  66893568874000          mov [004D8768], si
:0066F03F  FF75C10                 push dword ptr [ebp+10]
:0066F042  FF750C                  push dword ptr [ebp+0C]
:0066F045  FF7508                  push dword ptr [ebp+08]
:0066F048  FF55DC                  call [ebp-24]
..
```

We need to write in the instructions we overwrote. The instruction call [ebp-24] tells the program to go to the unpacked programs entry point.

Now, as the final thing open Neopaint in your favourite hexeditor and write in the new code. Run the program. Did it work? Yes, the patching was successful.

But there is one more thing: When we run the program now everything seems to be okey, except from the opening splash screen. All references to a shareware/evaluation program is removed from the main program. It also works even if you push your systemclock a year ahead. But when we open the program, you will see the the opening splash screen with the text «Unregistered Version» written out at the bottom of it.
So, I guess we have to remove that.
I simply traced through the code after starting the program in the Softice loader, checking and putting bpx'es on calls until I reached this code:

```
..
..
:004994D2 8B45F8                   mov eax, dword ptr [ebp-08]
:004994D5 E822FFFFFF               call 004993FC
:004994DA 84C0                     test al, al
:004994DC 7529                     jne 00499507
:004994DE 833DB4A94D0000           cmp dword ptr [004DA9B4], 00000000
:004994E5 7420                     je 00499507
:004994E7 A1B4A94D00               mov eax, dword ptr [004DA9B4]
:004994EC 8B80B4010000             mov eax, dword ptr [eax+000001B4]
:004994F2 E8DDC5F7FF               call 00415AD4
:004994F7 A1B4A94D00               mov eax, dword ptr [004DA9B4]
:004994FC 8B80B4010000             mov eax, dword ptr [eax+000001B4]
:00499502 8B10                     mov edx, dword ptr [eax]
:00499504 FF5250                   call [edx+50]
..
..
```

The call made at 004994F2 paints the background for the text, and the call made from 00499504 writes out the text on the splash screen. If the call made at 004994D5 returns the value of 1 in al, the program jumps and you will not see the text. But you will see the splash scren. I checked out this call to see whether I could patch it to always return the value of 1 in al. This routine are called from two separate places in the program. But patching this routine is not really necessary. If you want to remove the text, you can just change the jne instruction at 004994DC to EB, and you'll never see the text again.
Even though it is a nice splash, you might want to remove it alltogether. Then check out the instruction at address 004994DE. If [004DA9B4] contains the value 0, the program will jump over the paint/write out calls beneath. When you do a search for [004DA9B4), you will find it, besides the location above, at two other places. One of them is a comparison like the one you see above. The other one is the important one, and looks like this:

```
..
```

```
..
:0047130E B201                  mov dl, 01
:00471310 B8F8114700            mov eax, 004711F8
:00471315 E81222FBFF            call 0042352C
:0047131A A3B4A94D00            mov dword ptr [004DA9B4], eax
:0047131F A1B4A94D00            mov eax, dword ptr [004DA9B4]
:00471324 E85B4EFBFF            call 00426184
:00471329 A128A64D00            mov eax, dword ptr [004DA628]
:0047132E E85965FBFF            call 0042788C
:00471333 33C0                  xor eax, eax
..
```

At the address 0047131A the value stored in eax is pushed into [004DA628]. I put a bpx on the call at 00471315 and ran the program. When Softice broke, I pushed F10 once, and then changed the value in eax to zero, and then pushed F5. No splash screen.:-)

When I checked the call made at 471315, I saw that it can be called from a lot of places within the program. It definitly didn't look like a safe place to make any patches, so I just changed the call instruction at 00471315 to

```
:00471315  B800000000            mov eax, 00000000
```

By doing this I ensure that the value in eax always is 0 when the next instruction is executed.
Well, to make this happen, we have to rewrite our patch a little bit. We must tell the program to execute something else than the existing instruction at 00471315. The commands we can use to make this happen are:

```
mov byte ptr  [00xxxxxx], xx            (changing one byte)
mov word ptr  [00xxxxxx], xxxx          (changing two bytes)
mov dword ptr [00xxxxxx], xxxxxxxx      (changing four bytes)
```

We have to replace an instruction that takes up 5 bytes. To do that I chose a combination of a mov byte and mov dword instruction. The good place to implement this new code would be after our patch instruction at 0066F038. So for me the new patch looked like this:

```
..
..
:0066F038  66893568874D00          mov [004D8768], si
:0066F03F  C60515134700B8          mov byte ptr  [00471315], B8
:0066F046  C7051613470000000000    mov dword ptr [00471316], 00000000
:0066F050  FF7510                  push dword ptr [ebp+10]
:0066F053  FF750C                  push dword ptr [ebp+0C]
:0066F056  FF7508                  push dword ptr [ebp+08]
:0066F059  FF55DC                  call [ebp-24]
..
..
```

Remember one thing when you write your mov instructions (you don't see it clearly here): The instructions that supposed to be written (moved) into a spesific address must be typed in reversed order.

Well, that should be all. When you run Neopaint by now you should see no opening splash screen, and no hints to the program being an evaluation copy.

Any comments or questions can be addressed to me at:
the.hobgoblin@moss.online.no
Or you can simply reach me by posting some words at +Sandmans messageboard.

Greetings goes out to all the regulars at +Sandmans messageboard. You know who you are:-)

And remember, the best way to keep the knowledge alive is to pass it on..

hobgoblin.