



Unpacking nTitles Verify Protected Programs

MaDMan_H3rCuL3s of ARTeam

Version 1.0 – May 2006

| | | |
|----|--|----|
| 1. | Abstract | 2 |
| 2. | EP, getting to our mapped executable, and dumping..... | 3 |
| 3. | References..... | 9 |
| 4. | Conclusions..... | 9 |
| 5. | History..... | 10 |
| 6. | Greetings | 10 |

Keywords

Unpacking, nTitles, Verify

Target: [HERE](#)



1. Abstract

Before we begin there's a few things I must clear up. This tutorial will go over the more advanced method of unpacking nTitles Verify. In other cases it is much easier, an example is DJ JAVA Decompiler. With particular application all you need to do is dump the dat file and rename it. While this proves a very weak protection on nTitles part, in this tutorial you will see sometimes it can get more advanced. So now, let us proceed with the learning experience.

The techniques described here are general and not specific to any commercial applications. The whole document must be intended as a document on programming advanced techniques, how you will use these information will be totally up to your responsibility.



2. EP, getting to our mapped executable, and dumping.

Don't know if you even heard of this before, this is a newer software protection, it uses online activation. So the best way to actually get to where I am in the tutorial is to activate it for trial use. Its free, and easy. Once you have done that you can continue on tutorial.

| Address | Hex dump | Disassembly | Comment |
|----------|--------------|--|--------------------------------|
| 004EF644 | 6A 60 | PUSH 60 | |
| 004EF646 | 68 A0F35E00 | PUSH Crypter.005EF3A0 | |
| 004EF648 | E8 50540000 | CALL Crypter.004F4AA0 | |
| 004EF650 | BF 94000000 | MOV EDI,94 | |
| 004EF655 | 8BC7 | MOV EAX,EDI | ntdll.7C910738 |
| 004EF657 | E8 54FEFFFF | CALL Crypter.004EF4B0 | |
| 004EF65C | 8965 E8 | MOV DWORD PTR SS:[EBP-18],ESP | |
| 004EF65F | 8BF4 | MOV ESI,ESP | |
| 004EF661 | 893E | MOV DWORD PTR DS:[ESI],EDI | ntdll.7C910738 |
| 004EF663 | 56 | PUSH ESI | pVersionInformation = FFFFFFFF |
| 004EF664 | FF15 8483570 | CALL DWORD PTR DS:[<&KERNEL32.GetVersion | GetVersionExA |
| 004EF66A | 8B4E 10 | MOV ECX,DWORD PTR DS:[ESI+10] | |
| 004EF66D | 890D EC91650 | MOV DWORD PTR DS:[6591EC].ECX | |

At EP.

Yes we all know this looks like a normal program, trust me it is not normal. If you remember my Exeshield tutorial, how we had a sort of loader for the actual program, well same basic principle here. Except .. yes it does create a executable on disk, called (appname.dat), but I wasn't able to get it running from this. SO I chose another way, dump it from memory. So we go into the MoleBox tutorial on this one, using the same principles there and now use them for this. First thing we want to accomplish is exactly when does it call the real program? We can just use a simple string search and find anything useful. But instead lets run it.



Unpacking nTitles Verify protected programs



We get this nasty nag. So click the “next” button and the program runs.

Well my next solution was to maybe see what API's we got. I discovered a very interesting one here:

```
00578348|.rdata|.Import|.KERNEL32.lstrlenA
0057863C|.rdata|.Import|.imagehlp.MapAndLoad
00578444|.rdata|.Import|.USER32.MapDialogRect
```

MapAndLoad

So it maps the image of the executable, and loads it. I then searched for any reference to it. Only one appears.

| References in Crypter:.text to imagehlp.MapAndLoad | | |
|--|--|---------------------|
| Address | Disassembly | Comment |
| 0041903F | CALL DWORD PTR DS:[&imagehlp.MapAndLoad] | imagehlp.MapAndLoad |

Cool. Double click the line.

```
0041902E|.v|.EB.03|.JMP.SHORT.Crypter.00419033
00419030|.v|.8D41.04|.LEA.EAX.DWORD.PTR.DS:[ECX+4]
00419033|.v|.6A.01|.PUSH.1
00419035|.v|.6A.00|.PUSH.0
00419037|.v|.8D4C24.08|.LEA.ECX.DWORD.PTR.SS:[ESP+8]
00419038|.v|.51|.PUSH.ECX
0041903C|.v|.6A.00|.PUSH.0
0041903E|.v|.50|.PUSH.EAX
0041903F|.v|.FF15.3C865701|.CALL.DWORD.PTR.DS:[&imagehlp.MapAndLoad]
00419045|.v|.85C0|.TEST.EAX.EAX
00419047|.v|.75.08|.JNZ.SHORT.Crypter.00419051
00419049|.v|.32C0|.XOR.AL.AL
0041904B|.v|.00|.NOP
```

So it gets loaded up here.



Unpacking nTitles Verify protected programs

The next idea I had came easily. If you scroll below this you see the following:

```

004190A7 . 72 00      JNB SHORT Crypter.00419073
004190A8 . 5D         POP EBP
004190AC . 5B         POP EBX
004190AD > 8D4424 08   LEA EAX,DWORD PTR SS:[ESP+8]
004190B1 . 50         PUSH EAX
004190B2 . FF15 40865701 CALL DWORD PTR DS:[&imageh lp.UnMapAndLoad]
004190B8 . 5F         POP EDI
004190B9 . B0 01      MOV AL,1
004190BB . 5E         POP ESI
004190BC . 83C4 30    ADD ESP,30
004190BD . 72 00      JNB SHORT Crypter.00419073

```

So above this it maps the image, then it unloads it.

Hmmm. So I set a BP on the code below, restart it, then I broke at the place:

```

0041901E . CC        INT3
0041901F . CC        INT3
00419020 > 8B41 18    MOV EAX,DWORD PTR DS:[ECX+18]
00419023 . 83EC 30    SUB ESP,30
00419026 . 83F8 10    CMP EAX,10
00419029 > 72 05     JB SHORT Crypter.00419030
0041902B . 8B41 04    MOV EAX,DWORD PTR DS:[ECX+4]
0041902E > EB 03     JMP SHORT Crypter.00419033
00419030 > 8D41 04    LEA EAX,DWORD PTR DS:[ECX+4]
00419033 > 6A 01     PUSH 1
00419035 . 6A 00     PUSH 0
00419037 . 8D4C24 08  LEA ECX,DWORD PTR SS:[ESP+8]
0041903B . 51        PUSH ECX
0041903C . 6A 00     PUSH 0
0041903E . 50        PUSH EAX
0041903F . FF15 3C865701 CALL DWORD PTR DS:[&imageh lp.MapAndLoad]
00419041 . 85C0      TEST EAX,EAX

```

Now we trace a bit...

```

0041903C . 6A 00     PUSH 0
0041903E . 50        PUSH EAX
0041903F . FF15 3C865701 CALL DWORD PTR DS:[&imageh lp.MapAndLoad]
00419041 . 85C0      TEST EAX,EAX

```

| | PUSH NAME |
|----------------------|-----------|
| imageh lp.MapAndLoad | |

In EAX we see the FileName.

```

EAX 0133FD6C ASCII "Crypter.dat"
ECX 0133FC00
EDX 0133FC64

```

So we know the name (replace the .dat with .exe, but not really here, just for reference)

```

0041903E . 50        PUSH EAX
0041903F . FF15 3C865701 CALL DWORD PTR DS:[&imageh lp.MapAndLoad]
00419041 . 85C0      TEST EAX,EAX
00419043 > 75 08     JNZ SHORT Crypter.00419051
00419045 . 32C0      XOR AL,AL

```

| | PUSH NAME |
|----------------------|-----------|
| imageh lp.MapAndLoad | |

We enter the Call and exit it, we see that EAX = 1, meaning it mapped the image just fine, so we JMP.

```

0041904E . C2 0C00   RET 0C
00419051 > 8B4C24 0C  MOV ECX,DWORD PTR SS:[ESP+C]
00419055 . 0FB751 14  MOVZX EDX,WORD PTR DS:[ECX+14]
00419059 . 56        PUSH ESI
0041905A . 33F6      XOR ESI,ESI

```

We look closely at the pointer.

```

Stack SS:[0134FC0C]=01350080, (ASCII "PE")
ECX=00001082
Jump from 00419047

```

YUP!... this is the mapped image.

Follow in DUMP



Unpacking nTitles Verify protected programs

| Address | Hex dump | ASCII |
|----------|---|------------------|
| 01350000 | 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 | MZE.♦...♦... .. |
| 01350010 | B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 | 7.....0..... |
| 01350020 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |C... |
| 01350030 | 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 |L=+q0L=+Th |
| 01350040 | 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 | is program canno |
| 01350050 | 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F | t be run in DOS |
| 01350060 | 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 | mode...\$. |
| 01350070 | 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 | PE..L0♦.S♦dD.. |
| 01350080 | 50 45 00 00 4C 01 04 00 53 DC 64 44 00 00 00 00 | ...α.0000...4♦ |
| 01350090 | 00 00 00 00 E0 00 0E 01 0B 01 08 00 00 C0 08 00 | ...E...♦...♦ |
| 013500A0 | 00 90 00 00 00 00 00 00 FE D4 08 00 00 20 00 00 | ♦...♦...♦...♦ |
| 013500B0 | 00 E0 08 00 00 00 40 00 20 00 00 00 10 00 00 | ♦...♦...♦...♦ |
| 013500C0 | 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 013500D0 | 00 A0 09 00 00 10 00 00 00 00 00 00 02 00 00 | ♦...♦...♦...♦ |
| 013500E0 | 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 | ♦...♦...♦...♦ |
| 013500F0 | 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350100 | A4 D4 08 00 57 00 00 00 00 00 00 00 09 00 58 | ♦...♦...♦...♦ |
| 01350110 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350120 | 00 80 09 00 0C 00 00 00 E0 08 00 1C 00 00 00 | ♦...♦...♦...♦ |
| 01350130 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350140 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350150 | 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00 | ♦...♦...♦...♦ |
| 01350160 | 00 00 00 00 00 00 00 00 08 20 00 48 00 00 00 | ♦...♦...♦...♦ |
| 01350170 | 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 | ♦...♦...♦...♦ |
| 01350180 | 04 B5 08 00 00 20 00 00 C0 08 00 00 10 00 00 | ♦...♦...♦...♦ |
| 01350190 | 00 00 00 00 00 00 00 00 00 00 00 00 20 00 60 | ♦...♦...♦...♦ |
| 013501A0 | 2E 73 64 61 74 61 00 00 AD 00 00 00 E0 08 00 | ♦...♦...♦...♦ |
| 013501B0 | 00 10 00 00 00 00 08 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 013501C0 | 00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 | ♦...♦...♦...♦ |
| 013501D0 | 58 6D 00 00 00 00 09 00 70 00 00 00 E0 08 00 | ♦...♦...♦...♦ |
| 013501E0 | 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 | ♦...♦...♦...♦ |
| 013501F0 | 2E 72 65 6C 6F 63 00 00 0C 00 00 00 80 09 00 | ♦...♦...♦...♦ |
| 01350200 | 00 10 00 00 00 50 09 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350210 | 00 00 00 00 40 00 00 42 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350220 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |
| 01350230 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ♦...♦...♦...♦ |

There's our EXE. But not really ☺

The problem you will face is that the Import table is not complete here, nor is it for a bit longer. The way I worked it is this, first we can hit CTRL+F9 and exit this function.

| | | |
|----------|---------|------------|
| 004190B8 | 5F | POP EDI |
| 004190B9 | B0 01 | MOV AL,1 |
| 004190BB | 5E | POP ESI |
| 004190BC | 83C4 30 | ADD ESP,30 |
| 004190BE | C2 0C00 | RET 0C |
| 004190C2 | CC | INT3 |

Hit F7 look below:

| Address | Hex dump | ASCII |
|----------|----------|-------|
| 01350000 | | |
| 01350010 | | |
| 01350020 | | |
| 01350030 | | |
| 01350040 | | |
| 01350050 | | |
| 01350060 | | |
| 01350070 | | |
| 01350080 | | |
| 01350090 | | |
| 013500A0 | | |
| 013500B0 | | |
| 013500C0 | | |
| 013500D0 | | |
| 013500E0 | | |
| 013500F0 | | |
| 01350100 | | |
| 01350110 | | |
| 01350120 | | |
| 01350130 | | |
| 01350140 | | |
| 01350150 | | |
| 01350160 | | |
| 01350170 | | |
| 01350180 | | |

Our image is gone ☹



Unpacking nTitles Verify protected programs

| Address | Hex dump | Disassembly |
|----------|----------------|--------------------------------|
| 0044A4E0 | . 84C0 | TEST AL,AL |
| 0044A4E2 | . 75 1E | JNZ SHORT Crypter.0044A502 |
| 0044A4E4 | . 8D8C24 28010 | LEA ECX,DWORD PTR SS:[ESP+128] |
| 0044A4E8 | . C68424 50010 | MOV BYTE PTR SS:[ESP+150],2 |

You should now be here.

Scroll down a bit till you see this:

| | | |
|----------|----------------|--|
| 0044A6C5 | . C1E9 02 | SHR ECX,2 |
| 0044A6C8 | . F3:A5 | REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI] |
| 0044A6CA | . 8BC8 | MOV ECX,EAX |
| 0044A6CC | . 83E1 03 | AND ECX,3 |
| 0044A6CF | . F3:A4 | REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI] |
| 0044A6D1 | . 8D4C24 2C | LEA ECX,DWORD PTR SS:[ESP+2C] |
| 0044A6D3 | . C68424 50010 | MOV BYTE PTR SS:[ESP+150],2 |

Here we are moving some data, line in MoleBox Pro tutorial.

Check whats being moved:

```
ECX=00023000 (decimal 143360.)
DS:[ESI]=[01351020]=58E6DC25
ES:[EDI]=[00710020]=00000000
```

What the heck?

| Address | Hex dump | ASCII |
|----------|---|------------------|
| 01350000 | 00 00 71 00 50 00 E4 00 00 00 00 00 00 00 00 00 | ..q.P.2..... |
| 01350010 | 00 70 09 00 00 70 09 00 00 10 00 00 00 0B 00 00 | .p...p...d... |
| 01350020 | 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 | MZ...♦..... |
| 01350030 | B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 | ?.....@..... |
| 01350040 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |C..... |
| 01350050 | 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 |C..... |
| 01350060 | 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 | ...L...L...Th |
| 01350070 | 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F | is program canno |
| 01350080 | 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 | t be run in DOS |
| 01350090 | 6D 6F 64 65 2E 0D 0A 24 00 00 00 00 00 00 00 00 | mode...\$. |
| 013500A0 | 50 45 00 00 4C 01 04 00 53 DC 64 44 00 00 00 00 | PE..L0...s.d0... |
| 013500B0 | 00 00 00 00 0E 00 00 00 01 0B 01 08 00 00 C0 08 | ...α.000...4... |
| 013500C0 | 00 90 00 00 00 00 00 00 FE D4 08 00 00 20 00 00 | .E...♦...4... |
| 013500D0 | 00 E0 08 00 00 40 00 00 20 00 00 00 10 00 00 | ...♦...@...1... |
| 013500E0 | 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 | ♦...♦...@...4... |
| 013500F0 | 00 A0 09 00 00 10 00 00 00 00 00 00 02 00 00 | .á...p...@...♦ |
| 01350100 | 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 | ...p...p...1... |
| 01350110 | 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 | ...p...p...@... |
| 01350120 | A4 D4 08 00 57 00 00 00 00 00 00 00 58 6D 00 00 | ñ...W...Xm... |
| 01350130 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |L..... |
| 01350140 | 00 80 09 00 0C 00 00 00 E0 08 00 1C 00 00 00 | .C...α...L... |
| 01350150 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |@...L... |
| 01350160 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |@...L... |
| 01350170 | 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00 |@...L... |
| 01350180 | 00 00 00 00 00 00 00 00 08 20 00 48 00 00 00 |@...L... |
| 01350190 | 00 00 00 00 00 00 00 00 2F 74 6E 78 74 00 00 |@...L... |

Our image is back, but a bit offset from original mapping, please take note why. This seems to me like automatic dumping protection, you would dump the image before, (NO IAT) this time image is mapped a bit differently (by 20 bytes) SO what we do now is on the image header (offset 01350020) we set a bp Memory on access., then run it and see what happens.

| | |
|------------|-------------------|
| Breakpoint | Memory, on access |
| Search for | Memory, on write |



Unpacking nTitles Verify protected programs

```
00449F74 . 8BC1 H0 MOV EDI,DWORD PTR SS:[EBP-00]
00449F76 . C1E9 02 SHR ECX,2
00449F79 . F3:A5 REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00449F7B . 8BC8 MOV ECX,EAX
00449F7D . 83E1 03 AND ECX,3
00449F80 . F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00449F82 . 8B4D 98 MOV ECX,DWORD PTR SS:[EBP-68]
00449F85 . 51 PUSH ECX
00449F86 . FF15 CC835701 CALL DWORD PTR DS:[<&OLEAUT32.#24>]
```

We break here.

Seeing whats being moved around:

```
ECX=00025800 (decimal 153600.)
DS:[ESI]=[01350020]=00905A40
ES:[EDI]=[03890048]=00000000
```

Okay to let you know this is the last time this app is shuffled around.

Also note EAX value. This is the image size. So using lord PE lets dump it, at the offset 03890048, with size as 96000 (reason for the 03890048 is because our image is shifted again slightly upon remapping it). **Another note to make is that in order for you to have a full image when you dump, you must execute to the PUSH ECX at offset 00449F85**, the reason is that the image is not properly created if you dump before this. So hit F7 one time on the first REP command, and then continue hitting F8 until you land on the PUSH ECX instruction.

Now you may dump.

```
EAX 00096000
ECX 00000000
```

EAX value.

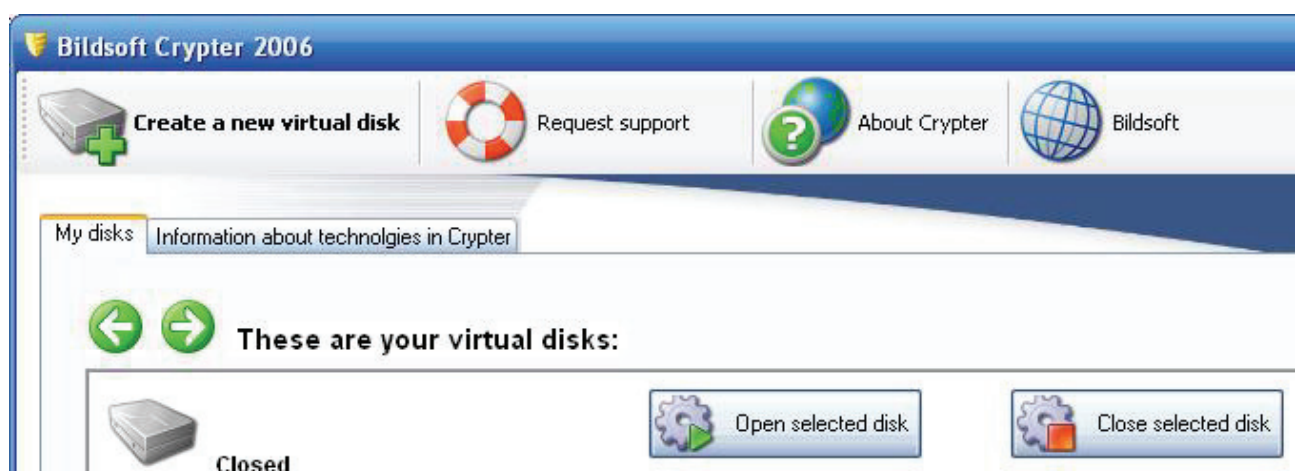
Like this:



Then we check the IAT

| ImportTable] | | | | | |
|---------------|--------------------|---------------|----------------|-------------|------------|
| DllName | OriginalFirstThunk | TimeDateStamp | ForwarderChain | Name | FirstThunk |
| mscoree.dll | 0008D4CC | 00000000 | 00000000 | 0008D4EE | 00002000 |
| | | | | | |
| ThunkRVA | ThunkOffset | ThunkValue | Hint | ApiName | |
| 0008D4CC | 0008C4CC | 0008D4E0 | 0000 | _CorExeMain | |

We see that this app is .NET ☺



And it RUNS!!!!

3. References

- [1] "Unpacking MoleBox Pro v2.5", MaDMAN_H3rCuL3s , <http://tutorials.accessroot.com>
- [2] "Unpacking Exeshield", MaDMAN_H3rCuL3s, <http://tutorials.accessroot.com>

4. Conclusions

You learned another thing. Place this inside your brain and marinate on it a bit. Read my next tutorials, only from your Number 1 source of Reverse Engineering Tutorials, none other than ARTeam.



Unpacking nTitles Verify protected programs

Don't use these concepts for making illegal operation, all the info here reported are only meant for studying and to help having a better knowledge of application code security techniques.

5. History

- Version 1.0 – First public release!

6. Greetings

I wish to tank all the ARTeam members of course and who read the beta versions of this tutorial and contributed,.. and of course you, who are still alive at the end of this quite long and complex document! Much respect to all the unpack.cn forum members, and PEdiy members.



<http://cracking.accessroot.com>