Statically unpacking a simple .NET dropper

malcat.fr/blog/statically-unpacking-a-simple-net-dropper/

Sample:

15180ee9f6a8682b24a0d5cb0491bb4e09d457bfab5a24ec1fcb077dab59773b (Bazaar, VT)

Infection chain:

.NET dropper -> .NET dropper + Reflective DLL -> Loki

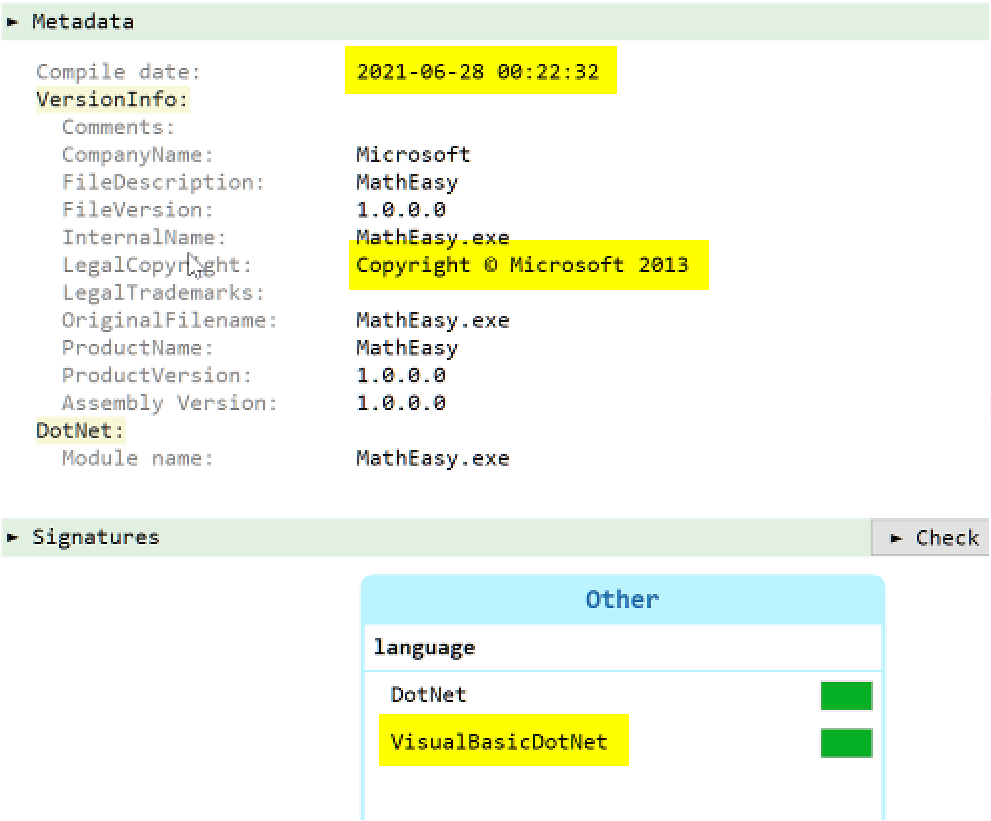
Difficulty:

Easy

Introduction

Today we will try to unpack a simple 2-layers .NET dropper using static analysis only. The goal of most malware packer/obfuscator is not to be hard to crack: it is to circumvent AV detection for a while, and eventually get replaced by a new one afterwards. And at the very end of the packer food chain are packers written in VB, .NET and AutoIT: they are particularly cheap and easy to crack. The sample we are about to analyse is no exception and will make a good introduction to Malcat's decryption algorithms.

A quick glance at the file metadata tells us immediately that the file is suspicious. A VB.NET application from Microsoft with a 2013 copyright but freshly compiled... sure, those version informations are 100% not fake.



```
► Metadata

  Compile date:              2021-06-28 00:22:32
  VersionInfo:
    Comments:
    CompanyName:             Microsoft
    FileDescription:         MathEasy
    FileVersion:             1.0.0.0
    InternalName:            MathEasy.exe
    LegalCopyright:          Copyright © Microsoft 2013
    LegalTrademarks:
    OriginalFilename:        MathEasy.exe
    ProductName:             MathEasy
    ProductVersion:          1.0.0.0
    Assembly Version:        1.0.0.0
  DotNet:
    Module name:             MathEasy.exe

► Signatures                                              ► Check
```

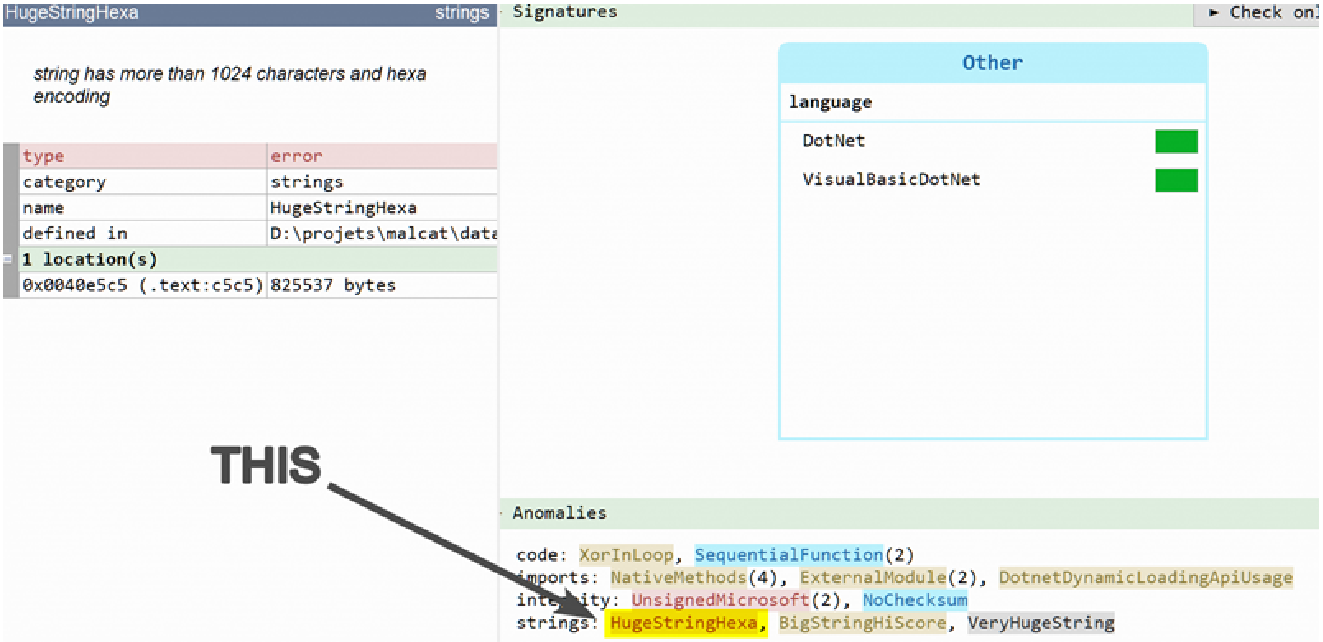
Other	
language	
DotNet	🟩
VisualBasicDotNet	🟩

Figure 1: Fake version information

Let us cut the overview right there as we will directly focus on the packed payload.

Locating the payload

Most .NET packers embed one or more encrypted assemblies. .NET assemblies are not small, they have to be put somewhere. They are usually put inside .NET resources (sometimes insides pictures), .NET static arrays or strings. For this sample, Malcat has already spotted a 800KB+ hexadecimal string inside the program (HugeStringHexa), which is kind of *unusual*.



HugeStringHexa strings

string has more than 1024 characters and hexa encoding

type	error
category	strings
name	HugeStringHexa
defined in	D:\projets\malcat\data
1 location(s)	
0x0040e5c5 (.text:c5c5)	825537 bytes

Signatures

Other

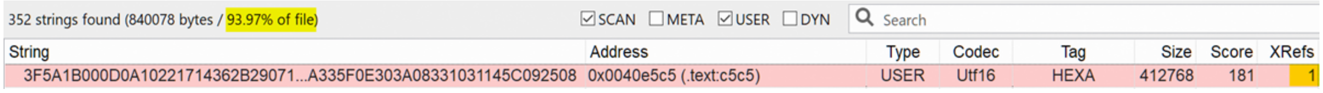
language	
DotNet	🟩
VisualBasicDotNet	🟩

Anomalies

```
code: XorInLoop, SequentialFunction(2)
imports: NativeMethods(4), ExternalModule(2), DotnetDynamicLoadingApiUsage
integrity: UnsignedMicrosoft(2), NoChecksum
strings: HugeStringHexa, BigStringHiScore, VeryHugeString
```

Figure 2: A look at the anomalies

This is confirmed in the Strings view (shortcut: **F6**), which tells us than more than 90% of the file is made of strings, and that our big hexadecimal string is by far the biggest one (the size 412768 is given in *characters*, so actual size for UTF16 is twice as much, about 824KB). Moreover, it has exactly one code reference, which is always a good indicator for packed data:



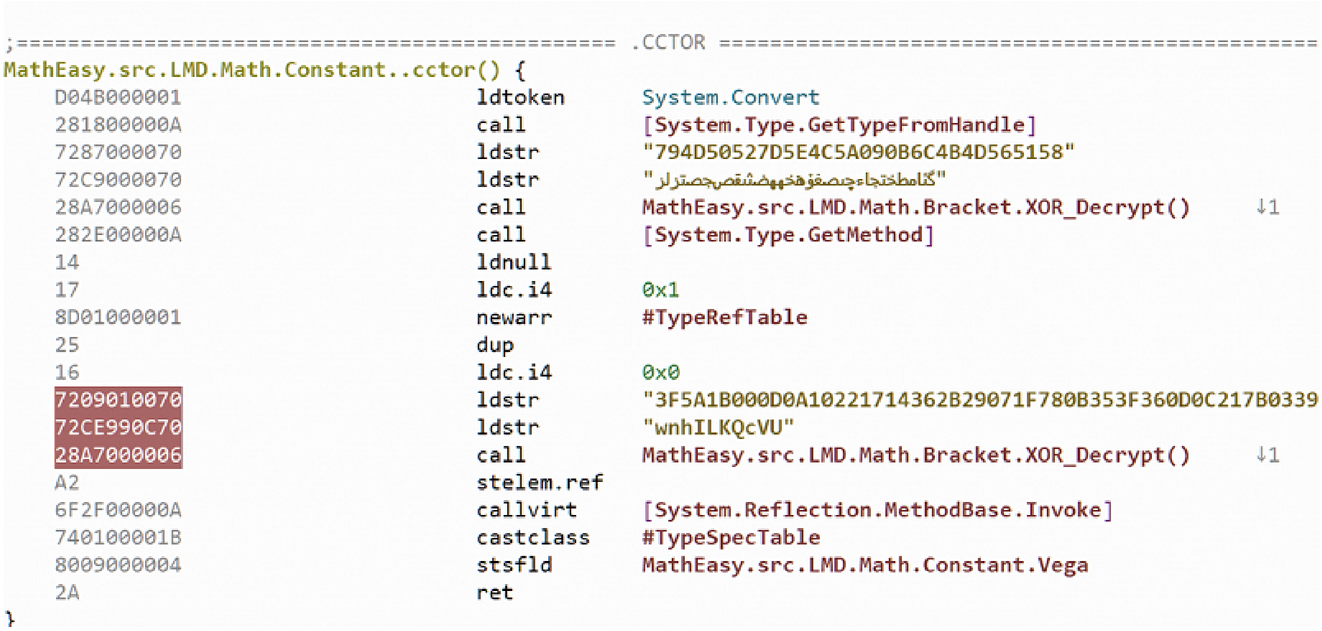
352 strings found (840078 bytes / 93.97% of file)

String	Address	Type	Codec	Tag	Size	Score	XRefs
3F5A1B000D0A10221714362B29071...A335F0E303A08331031145C092508	0x0040e5c5 (.text:c5c5)	USER	Utf16	HEXA	412768	181	1

Figure 3: Big hexa string

If we follow the string reference in the Code view (right-click on the string, and then choose Cross-references sub-menu) we land on the code snippet presented below. By looking at the names of the method and package there, we can infer that the application we are analyzing was most likely a clean .NET software that has been only slightly modified to include a couple of malicious methods. This is a technique commonly used by obfuscators to evade AV heuristics.

The content of the method also tells us that we won't have to start our VM for now. In fact, the hexadecimal string seems to be decrypted using a simple XOR algorithm using the key `"wnhILKQcVU"` :



```
;=============================================== .CCTOR ===============================================
MathEasy.src.LMD.Math.Constant..cctor() {
    D04B000001                        ldtoken      System.Convert
    281800000A                        call         [System.Type.GetTypeFromHandle]
    7287000070                        ldstr        "794D50527D5E4C5A090B6C4B4D565158"
    72C9000070                        ldstr        "گنامطختجاءچبصغفوهخهپضشقصجصترلر"
    28A7000006                        call         MathEasy.src.LMD.Math.Bracket.XOR_Decrypt()     ↓1
    282E00000A                        call         [System.Type.GetMethod]
    14                                ldnull
    17                                ldc.i4       0x1
    8D01000001                        newarr       #TypeRefTable
    25                                dup
    16                                ldc.i4       0x0
    7209010070                        ldstr        "3F5A1B000D0A10221714362B29071F780B353F360D0C217B0339
    72CE990C70                        ldstr        "wnhILKQcVU"
    28A7000006                        call         MathEasy.src.LMD.Math.Bracket.XOR_Decrypt()     ↓1
    A2                                stelem.ref
    6F2F00000A                        callvirt     [System.Reflection.MethodBase.Invoke]
    740100001B                        castclass    #TypeSpecTable
    8009000004                        stsfld       MathEasy.src.LMD.Math.Constant.Vega
    2A                                ret
}
```

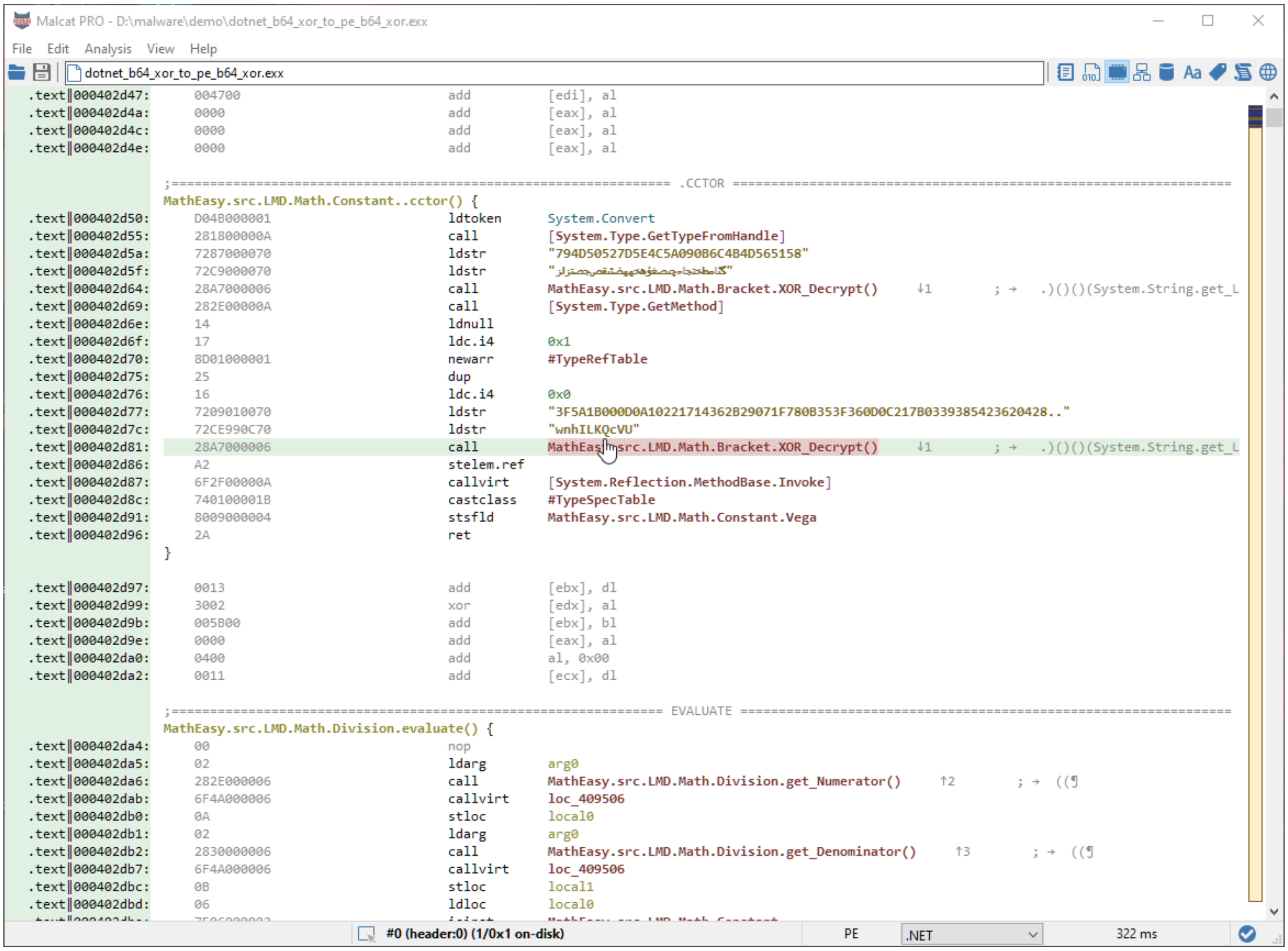
Figure 4: String decryption

Decrypting the first layer

Malcat comes with several decryption algorithms which we will use on the string. First, right-click on the big hexa string and chose the `Transform...` sub-menu. We will apply the following transformations (in order):

- change text encoding from UTF-16le to UTF-8: we get an ascii hexadecimal string
- hex decode the hexadecimal string: we get the raw bytes
- decrypt using the XOR algorithm and the key `"wnhILKQcVU"`

After these three pass, we obtain ... a base64 string, so the job is still not finished. Using Malcat's transformations, we can easily decode the base64 string. The result is identified by Malcat as a ... GZIP archive. Sure, after encoding your payload in hexa and base64, now you start to care about storage efficiency. But ok, Malcat can handle GZIP archives just fine. Just double-click the content stream inside the `files` tab to finally obtain ... a new PE file!


Malcat PRO - D:\malware\demo\dotnet_b64_xor_to_pe_b64_xor.exx
File Edit Analysis View Help
dotnet_b64_xor_to_pe_b64_xor.exx

```
 .text║000402d47:    004700                           add        [edi], al
 .text║000402d4a:    0000                             add        [eax], al
 .text║000402d4c:    0000                             add        [eax], al
 .text║000402d4e:    0000                             add        [eax], al

                 ;============================================================= .CCTOR =============================================================
                 MathEasy.src.LMD.Math.Constant..cctor() {
 .text║000402d50:    D04B000001                       ldtoken    System.Convert
 .text║000402d55:    281800000A                       call       [System.Type.GetTypeFromHandle]
 .text║000402d5a:    7287000070                       ldstr      "794D50527D5E4C5A090B6C4B4D565158"
 .text║000402d5f:    72C9000070                       ldstr      "گاطختجاءچمغفوهخيهضشقصجمتزلز"
 .text║000402d64:    28A7000006                       call       MathEasy.src.LMD.Math.Bracket.XOR_Decrypt()     ↓1        ; →   .)()()(System.String.get_L
 .text║000402d69:    282E00000A                       call       [System.Type.GetMethod]
 .text║000402d6e:    14                               ldnull
 .text║000402d6f:    17                               ldc.i4     0x1
 .text║000402d70:    8D01000001                       newarr     #TypeRefTable
 .text║000402d75:    25                               dup
 .text║000402d76:    16                               ldc.i4     0x0
 .text║000402d77:    7209010070                       ldstr      "3F5A1B000D0A10221714362B29071F780B353F360D0C217B0339385423620428.."
 .text║000402d7c:    72CE990C70                       ldstr      "wnhILKQcVU"
 .text║000402d81:    28A7000006                       call       MathEasy.src.LMD.Math.Bracket.XOR_Decrypt()     ↓1        ; →   .)()()(System.String.get_L
 .text║000402d86:    A2                               stelem.ref
 .text║000402d87:    6F2F00000A                       callvirt   [System.Reflection.MethodBase.Invoke]
 .text║000402d8c:    740100001B                       castclass  #TypeSpecTable
 .text║000402d91:    8009000004                       stsfld     MathEasy.src.LMD.Math.Constant.Vega
 .text║000402d96:    2A                               ret
                 }

 .text║000402d97:    0013                             add        [ebx], dl
 .text║000402d99:    3002                             xor        [edx], al
 .text║000402d9b:    005B00                           add        [ebx], bl
 .text║000402d9e:    0000                             add        [eax], al
 .text║000402da0:    0400                             add        al, 0x00
 .text║000402da2:    0011                             add        [ecx], dl

                 ;============================================================ EVALUATE ============================================================
                 MathEasy.src.LMD.Math.Division.evaluate() {
 .text║000402da4:    00                               nop
 .text║000402da5:    02                               ldarg      arg0
 .text║000402da6:    282E000006                       call       MathEasy.src.LMD.Math.Division.get_Numerator()     ↑2        ; →  ((¶
 .text║000402dab:    6F4A000006                       callvirt   loc_409506
 .text║000402db0:    0A                               stloc      local0
 .text║000402db1:    02                               ldarg      arg0
 .text║000402db2:    2830000006                       call       MathEasy.src.LMD.Math.Division.get_Denominator()     ↑3        ; →  ((¶
 .text║000402db7:    6F4A000006                       callvirt   loc_409506
 .text║000402dbc:    0B                               stloc      local1
 .text║000402dbd:    06                               ldloc      local0
```

Figure 5: Unpacking the first layer

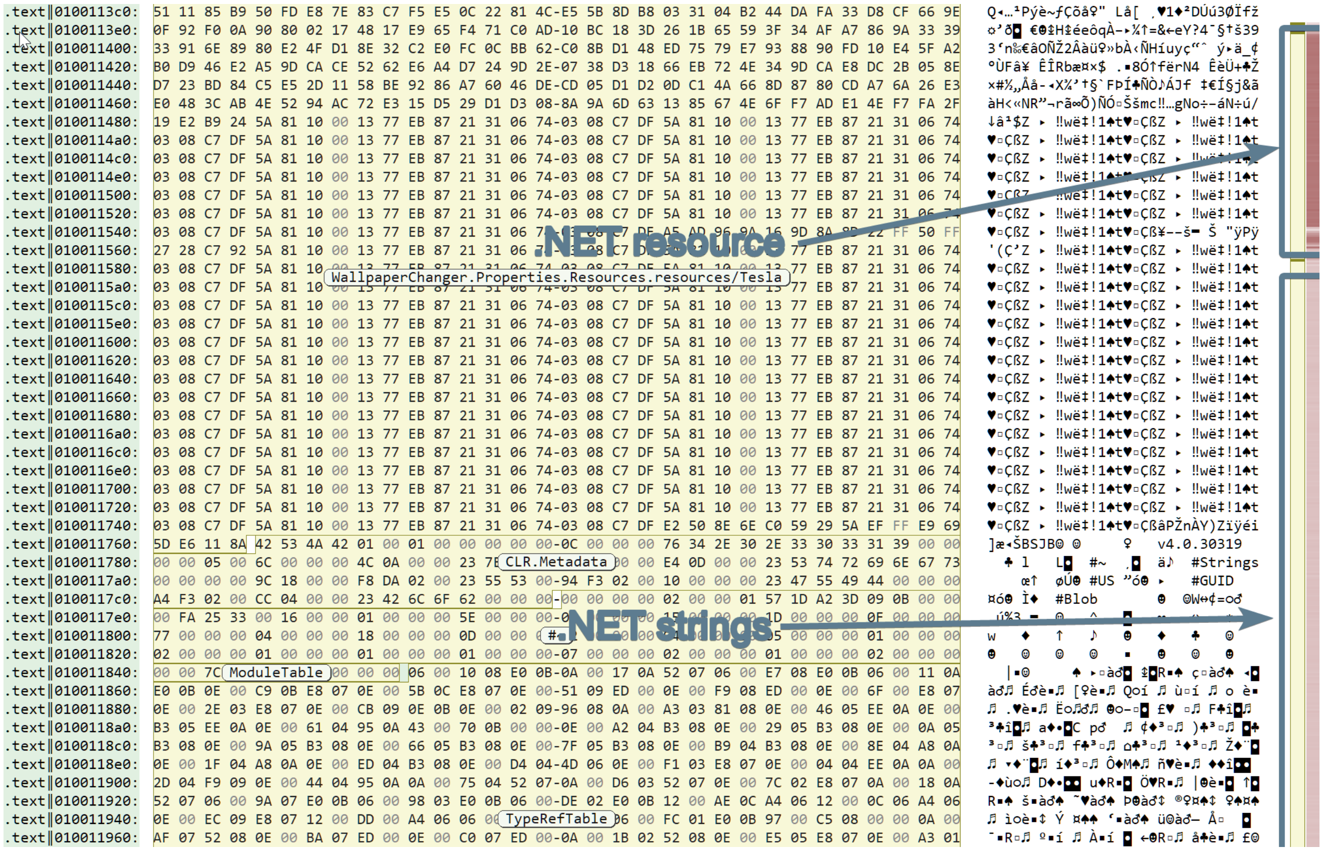
At this point we can discard the rest of the application: the payload we just decrypted made for more than 90% of the file and the packer authors cared enough to pack it several time. So it's pretty safe to assume that we got everything there was to see there.

Decrypting the second layer

The second layer is also a .NET executable which also contains stolen VersionInformations (claims to be WallpaperChanger.dll). This time, there seem to be more than one packed content:

- we see a high-entropy .net resource named `Tesla` of about 60Kb
- one big base64 string of about 185Kb at offset `0x100131da`
- two small hexadecimal strings of ~100 bytes

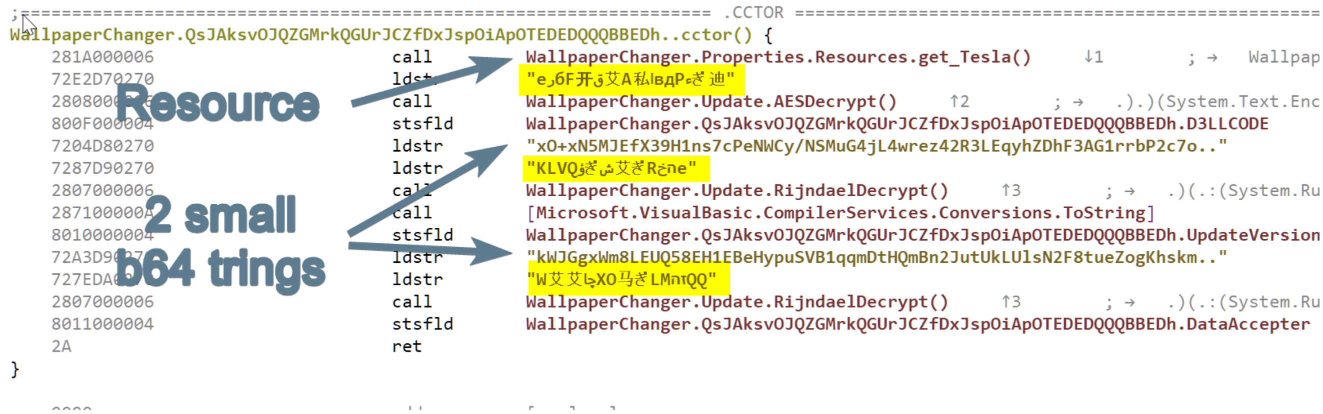
The rest of the application seem to be a clean app, with a few added malicious methods inside the class `WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh`. So we will save us some time and not analyze the code, and instead focus on the packed data: the big resource and the big base64 string. Let us start with the resource.



```
.text‖010113c0: 51 11 85 B9 50 FD E8 7E 83 C7 F5 E5 0C 22 81 4C-E5 5B 8D B8 03 31 04 B2 44 DA FA 33 D8 CF 66 9E   Q◂…¹Pýè~ƒÇõå♀" Lå[ ,♥1♦²DÚú3ØÏfž
.text‖010113e0: 0F 92 F0 0A 90 80 02 17 48 17 E9 65 F4 71 C0 AD-10 BC 18 3D 26 1B 65 59 3F 34 AF A7 86 9A 33 39   ☼’ð◙ €☻↨H↨éeôqÀ-▸¼↑=&←eY?4¯§†š39
.text‖01011400: 33 91 6E 89 80 E2 4F D1 8E 32 C2 E0 FC 0C BB 62-C0 8B D1 48 ED 75 79 E7 93 88 90 FD 10 E4 5F A2   3‘n‰€âOÑŽ2Âàü♀»bÀ‹ÑHíuyç“ˆ ý▸ä_¢
.text‖01011420: B0 D9 46 E2 A5 9D CA CE 52 62 E6 A4 D7 24 9D 2E-07 38 D3 18 66 EB 72 4E 34 9D CA E8 DC 2B 05 8E   °ÙFâ¥ ÊÎRbæ¤×$ .•8Ó↑fërN4 ÊèÜ+♣Ž
.text‖01011440: D7 23 BD 84 C5 E5 2D 11 58 BE 92 86 A7 60 46 DE-CD 05 D1 D2 0D C1 4A 66 8D 87 80 CD A7 6A 26 E3   ×#½„Åå-◂X¾’†§`FÞÍ♣ÑÒ♪ÁJf ‡€Í§j&ã
.text‖01011460: E0 48 3C AB 4E 52 94 AC 72 E3 15 D5 29 D1 D3 08-8A 9A 6D 63 13 85 67 4E 6F F7 AD E1 4E F7 FA 2F   àH<«NR”¬rã§Õ)ÑÓ▫Ššmc‼…gNo÷-áN÷ú/
.text‖01011480: 19 E2 B9 24 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ↓â¹$Z ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010114a0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010114c0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010114e0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011500: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011520: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011540: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF A5 AD 96 9A 16 9D 8A 8D 22 FF 50 FF   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫Çß¥--š▬ Š "ÿPÿ
.text‖01011560: 27 28 C7 92 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   '(Ç’Z ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011580: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010115a0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010115c0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010115e0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011600: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011620: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011640: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011660: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011680: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010116a0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010116c0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖010116e0: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011700: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011720: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßZ ▸ ‼wë‡!1♠t
.text‖01011740: 03 08 C7 DF 5A 81 10 00 13 77 EB 87 21 31 06 74-03 08 C7 DF E2 50 8E 6E C0 59 29 5A EF FF E9 69   ♥▫ÇßZ ▸ ‼wë‡!1♠t♥▫ÇßâPŽnÀY)Zïÿéi
.text‖01011760: 5D E6 11 8A 42 53 4A 42 01 00 01 00 00 00 00 00-0C 00 00 00 76 34 2E 30 2E 33 30 33 31 39 00 00   ]æ◂ŠBSJB☺ ☺      ♀   v4.0.30319
.text‖01011780: 00 00 05 00 6C 00 00 00 4C 0A 00 00 23 7E 00 00-00 00 E4 0D 00 00 23 53 74 72 69 6E 67 73     ♣ l   L◙  #~  ,◙  ä♪  #Strings
.text‖010117a0: 00 00 00 00 9C 18 00 00 F8 DA 02 00 23 55 53 00-94 F3 02 00 10 00 00 00 23 47 55 49 44 00 00 00       œ↑  øÚ☻ #US ”ó☻ ▸   #GUID
.text‖010117c0: A4 F3 02 00 CC 04 00 00 23 42 6C 6F 62 00 00 00-00 00 00 00 02 00 00 01 57 1D A2 3D 09 0B 00 00   ¤ó☻ Ì♦  #Blob       ☻  ☺W↔¢=○♂
.text‖010117e0: 00 FA 25 33 00 16 00 00 01 00 00 00 5E 00 00 00-08 00 00 00 15 00 00 00 1D 00 00 00 0F 00 00 00    ú%3 ▬  ☺   ^   ◘
.text‖01011800: 77 00 00 00 04 00 00 00 18 00 00 00 0D 00 00 00-02 00 00 00 04 00 00 00 05 00 00 00 01 00 00 00   w   ♦   ↑   ♪   ☻   ♦   ♣   ☺
.text‖01011820: 02 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00-07 00 00 00 02 00 00 00 01 00 00 00 02 00 00 00   ☻   ☺   ☺   ☺   •   ☻   ☺   ☻
.text‖01011840: 00 00 7C 00 00 00 00 00 00 00 00 00 06 00 10 08 E0 0B-0A 00 17 0A 52 07 06 00 E7 08 E0 0B 06 00 11 0A     |•☺      ♠ ▸▫à♂◙ ‡◙R•♠ ç▫à♂♠ ◂◙
.text‖01011860: E0 0B 0E 00 C9 0B E8 07 0E 00 5B 0C E8 07 0E 00-51 09 ED 00 0E 00 F9 08 ED 00 0E 00 6F 00 E8 07   à♂♬ É♂è•♬ [♀è•♬ Q○í ♬ ù▫í ♬ o è•
.text‖01011880: 0E 00 2E 03 E8 07 0E 00 CB 09 0E 0B 0E 00 02 09-96 08 0A 00 A3 03 81 08 0E 00 46 05 EE 0A 0E 00   ♬ .♥è•♬ Ëo♬♂♬ ☻o–▫◙ £♥ ▫♬ F♣î◙♬
.text‖010118a0: B3 05 EE 0A 0E 00 61 04 95 0A 43 00 70 0B 00 00-0E 00 A2 04 B3 08 0E 00 29 05 B3 08 0E 00 0A 05   ³♣î◙♬ a♦•◙C p♂  ♬ ¢♦³▫♬ )♣³▫♬ ◙♣
.text‖010118c0: B3 08 0E 00 9A 05 B3 08 0E 00 66 05 B3 08 0E 00-7F 05 B3 08 0E 00 B9 04 B3 08 0E 00 8E 04 A8 0A   ³▫♬ š♣³▫♬ f♣³▫♬ ⌂♣³▫♬ ¹♦³▫♬ Ž♦¨◙
.text‖010118e0: 0E 00 1F 04 A8 0A 0E 00 ED 04 B3 08 0E 00 D4 04-4D 06 0E 00 F1 03 E8 07 0E 00 04 04 EE 0A 0A 00   ♬ ▾♦¨◙♬ í♦³▫♬ Ô♦M♠♬ ñ♥è•♬ ♦♦î◙◙
.text‖01011900: 2D 04 F9 09 0E 00 44 04 95 0A 0A 00 75 04 52 07-0A 00 D6 03 52 07 0E 00 7C 02 E8 07 0A 00 18 0A   -♦ùo♬ D♦•◙◙ u♦R•◙ Ö♥R•♬ |☻è•◙ ↑◙
.text‖01011920: 52 07 06 00 9A 07 E0 0B 06 00 98 03 E0 0B 06 00-DE 02 E0 0B 12 00 AE 0C A4 06 12 00 0C 06 A4 06   R•♠ š•à♂♠ ˜♥à♂♠ Þ☻à♂↕ ®♀¤♠↕ ♀♠¤♠
.text‖01011940: 0E 00 EC 09 E8 07 12 00 DD 00 A4 06 06 00 00 00 06 00 FC 01 E0 0B 97 00 C5 08 00 00 0A 00   ♬ ìoè•↕ Ý ¤♠♠ ‘•à♂♠ ü☺à♂— Å▫  ◙
.text‖01011960: AF 07 52 08 0E 00 BA 07 ED 00 0E 00 C0 07 ED 00-0A 00 1B 02 52 08 0E 00 E5 05 E8 07 0E 00 A3 01   ¯•R▫♬ º•í ♬ À•í ◙ ←☻R▫♬ å♣è•♬ £☺
```

Figure 6: Second layer overview

When adding a resource to a .NET program under VisualStudio, a standard resource getter name `get_<resource_name>` is often created. So we will go into the symbols list (shortcut: **F5**), hit **Ctrl+F** and look for `Tesla`. There is exactly one method named `WallpaperChanger.Properties.Resources.get_Tesla` at offset `0x1000278c`. The getter has only one code reference at address `0x100026dc` which looks promising:



```
;================================================================== .CCTOR ==================================================
WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh..cctor() {
    281A000006          call      WallpaperChanger.Properties.Resources.get_Tesla()    ↓1          ; →   Wallpap
    72E2D70270          ldstr     "e ̧6F开ق艾A私lвдPءざ 迪"
    2808000006          call      WallpaperChanger.Update.AESDecrypt()     ↑2          ; →   .).)(System.Text.Enc
    800F000004          stsfld    WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.D3LLCODE
    7204D80270          ldstr     "xO+xN5MJEfX39H1ns7cPeNWCy/NSMuG4jL4wrez42R3LEqyhZDhF3AG1rrbP2c7o.."
    7287D90270          ldstr     "KLVQوざش艾ざRخne"
    2807000006          call      WallpaperChanger.Update.RijndaelDecrypt()     ↑3          ; →   .)(.:(System.Ru
    287100000A          call      [Microsoft.VisualBasic.CompilerServices.Conversions.ToString]
    8010000004          stsfld    WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.UpdateVersion
    72A3D90270          ldstr     "kWJGgxWm8LEUQ58EH1EBeHypuSVB1qqmDtHQmBn2JutUkLUlsN2F8tueZogKhskm.."
    727EDA0270          ldstr     "W艾 艾چXO马ざLMnτQQ"
    2807000006          call      WallpaperChanger.Update.RijndaelDecrypt()     ↑3          ; →   .)(.:(System.Ru
    8011000004          stsfld    WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.DataAccepter
    2A                  ret
}
```

Figure 7: the method decrypting resource + strings

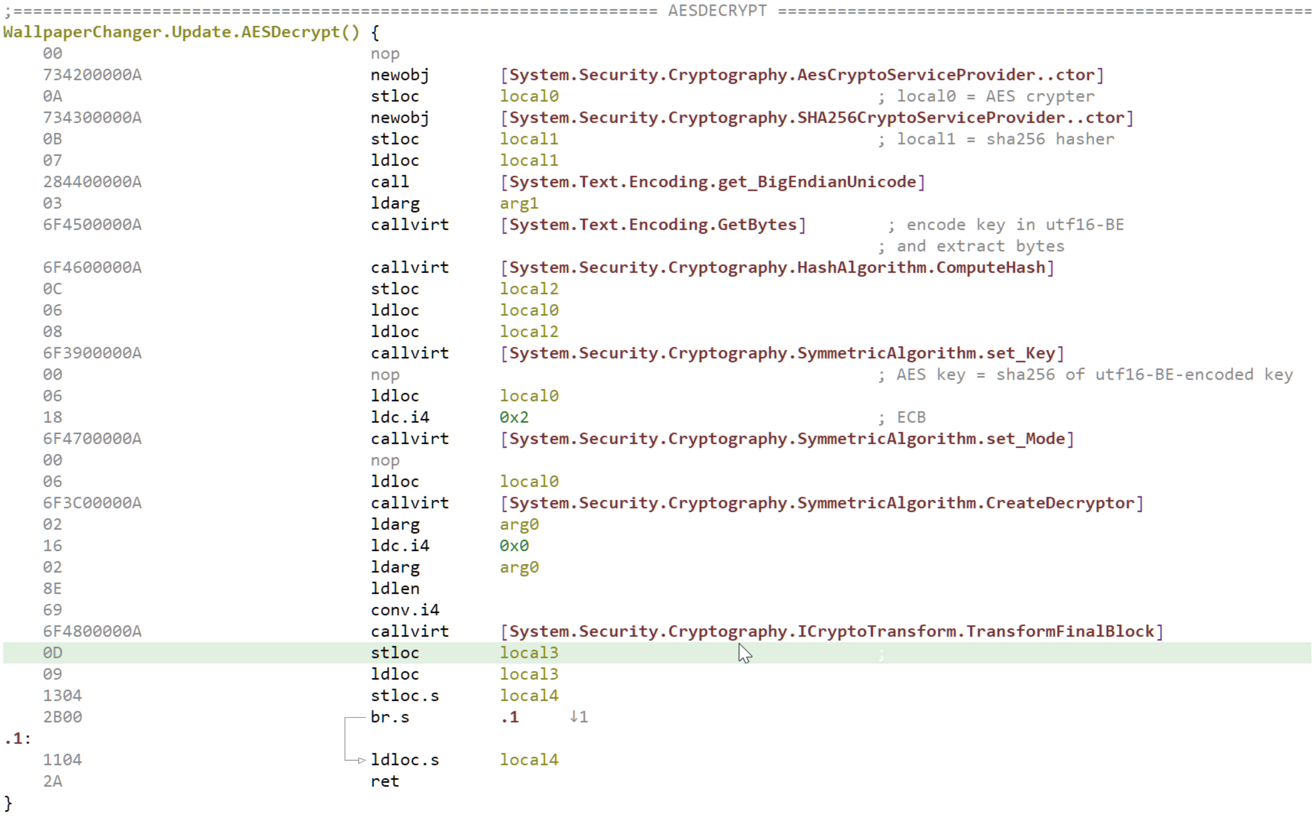
We see two different decryption methods called there:

- the method `AESDecrypt` used to decrypt the .NET resource `Tesla`
- the method `RijndaelDecrypt` used to decrypt the two small base64 strings we spotted earlier.

The big base64 string does not seem to be decrypted there. Since the small strings seem to be of little interest, let us focus on the method `AESDecrypt` first.

Decrypting the Tesla resource

It looks like the authors of the packer were not satisfied with the security offered by XOR encryption and chose to step up their game:



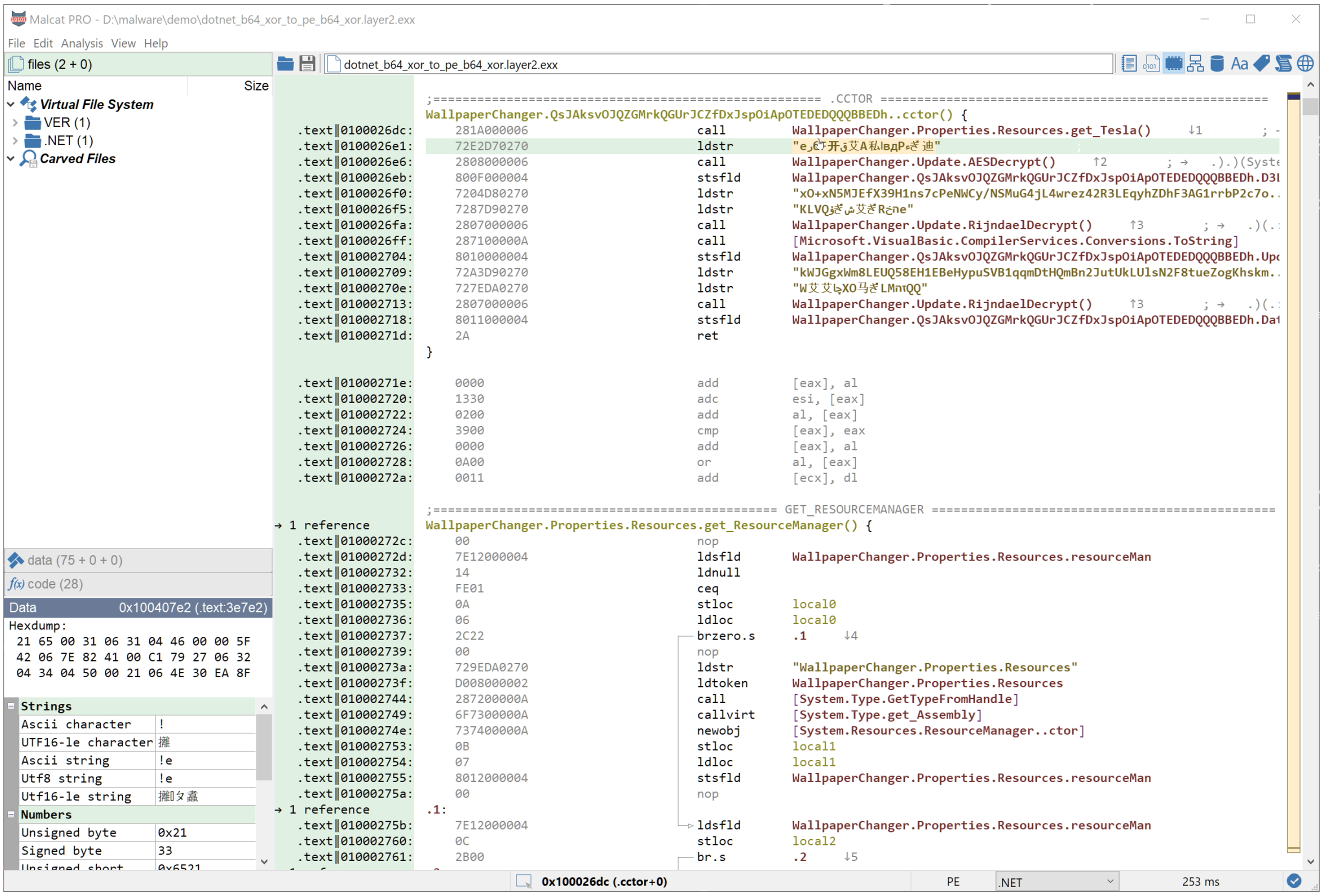
```
;================================================================= AESDECRYPT ===================================================
WallpaperChanger.Update.AESDecrypt() {
    00                              nop
    734200000A                      newobj      [System.Security.Cryptography.AesCryptoServiceProvider..ctor]
    0A                              stloc       local0                          ; local0 = AES crypter
    734300000A                      newobj      [System.Security.Cryptography.SHA256CryptoServiceProvider..ctor]
    0B                              stloc       local1                          ; local1 = sha256 hasher
    07                              ldloc       local1
    284400000A                      call        [System.Text.Encoding.get_BigEndianUnicode]
    03                              ldarg       arg1
    6F4500000A                      callvirt    [System.Text.Encoding.GetBytes]          ; encode key in utf16-BE
                                                                                ; and extract bytes
    6F4600000A                      callvirt    [System.Security.Cryptography.HashAlgorithm.ComputeHash]
    0C                              stloc       local2
    06                              ldloc       local0
    08                              ldloc       local2
    6F3900000A                      callvirt    [System.Security.Cryptography.SymmetricAlgorithm.set_Key]
    00                              nop                                         ; AES key = sha256 of utf16-BE-encoded key
    06                              ldloc       local0
    18                              ldc.i4      0x2                             ; ECB
    6F4700000A                      callvirt    [System.Security.Cryptography.SymmetricAlgorithm.set_Mode]
    00                              nop
    06                              ldloc       local0
    6F3C00000A                      callvirt    [System.Security.Cryptography.SymmetricAlgorithm.CreateDecryptor]
    02                              ldarg       arg0
    16                              ldc.i4      0x0
    02                              ldarg       arg0
    8E                              ldlen
    69                              conv.i4
    6F4800000A                      callvirt    [System.Security.Cryptography.ICryptoTransform.TransformFinalBlock]
    0D                              stloc       local3                                      ;
    09                              ldloc       local3
    1304                            stloc.s     local4
    2B00                            br.s        .1    ↓1
.1:
    1104                            ldloc.s     local4
    2A                              ret
}
```

Figure 8: the method AESDecrypt

The code is pretty straightforward: the string `"eرбF开ق艾A私اвдPءぎ迪"` is first encoded in utf16-BE and then hashed using the SHA256 algorithm. The result will be used as KEY for the AES algorithm. No IV is defined, since the encryption mode is set to ECB. At the end, the resource content is decrypted using AES. We could easily recover the decrypted content using a debugger there, but since the code is pretty straightforward, we can also do everything statically inside Malcat. First, we need to compute the AES key. We can simulate what the code is doing using the following script:

```
import hashlib
raw_bytes = "eرбF开ق艾A私اвдPءぎ迪".encode("utf-16-be")
print(hashlib.sha256(raw_bytes).hexdigest())
# ->
"ab6edf45e299a7b2968a9d7cd013c1164efc6165508d691f085b7d9462ee945b"
```

Hit `F8` to enter the script editor, remove the example script, paste this content and you will see the result in the output window. Copy the key in the clipboard and you are ready to decrypt the resource using Malcat's AES transform:


Malcat PRO - D:\malware\demo\dotnet_b64_xor_to_pe_b64_xor.layer2.exx
File Edit Analysis View Help
files (2 + 0)
Name
Size
Virtual File System
VER (1)
.NET (1)
Carved Files
data (75 + 0 + 0)
code (28)
dotnet_b64_xor_to_pe_b64_xor.layer2.exx
0x100026dc (.cctor+0)
PE
.NET
253 ms

```
Data                0x100407e2 (.text:3e7e2)
Hexdump:
 21 65 00 31 06 31 04 46 00 00 5F
 42 06 7E 82 41 00 C1 79 27 06 32
 04 34 04 50 00 21 06 4E 30 EA 8F
```

Strings	
Ascii character	!
UTF16-le character	攡
Ascii string	!e
Utf8 string	!e
Utf16-le string	攡㄀夆
Numbers	
Unsigned byte	0x21
Signed byte	33
Unsigned short	0x6521

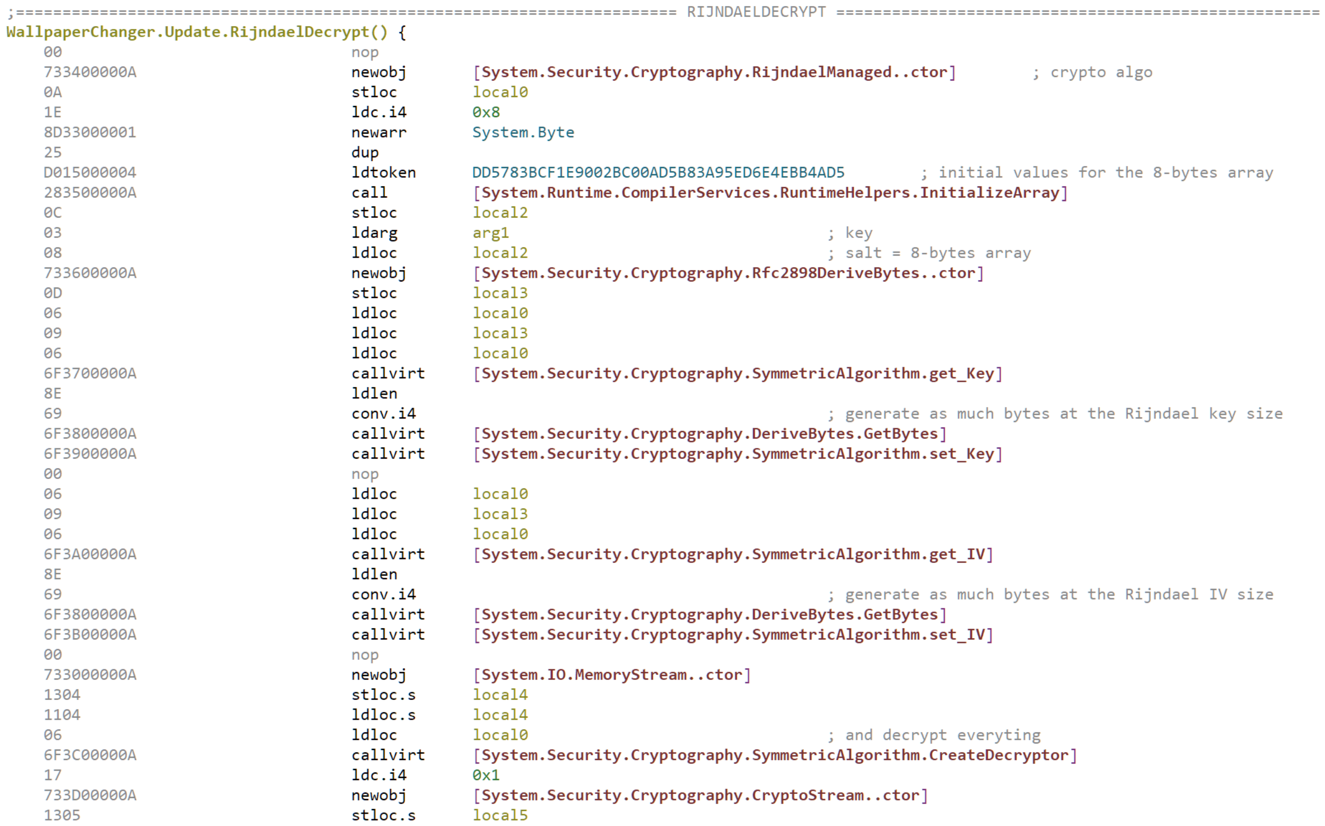
```
                          ;===================================================== .CCTOR =====================================================
                          WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh..cctor() {
  .text‖0100026dc:           281A000006                      call         WallpaperChanger.Properties.Resources.get_Tesla()     ↓1          ; -
  .text‖0100026e1:           72E2D70270                      ldstr        "e‚6开ق艾A私lвдPءざ 迪"
  .text‖0100026e6:           2808000006                      call         WallpaperChanger.Update.AESDecrypt()      ↑2         ; →    .).)(Syste
  .text‖0100026eb:           800F000004                      stsfld       WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.D3l
  .text‖0100026f0:           7204D80270                      ldstr        "xO+xN5MJEfX39H1ns7cPeNWCy/NSMuG4jL4wrez42R3LEqyhZDhF3AG1rrbP2c7o..
  .text‖0100026f5:           7287D90270                      ldstr        "KLVQوざش艾ざRخne"
  .text‖0100026fa:           2807000006                      call         WallpaperChanger.Update.RijndaelDecrypt()     ↑3         ; →    .)(.:
  .text‖0100026ff:           287100000A                      call         [Microsoft.VisualBasic.CompilerServices.Conversions.ToString]
  .text‖010002704:           8010000004                      stsfld       WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.Upc
  .text‖010002709:           72A3D90270                      ldstr        "kWJGgxWm8LEUQ58EH1EBeHypuSVB1qqmDtHQmBn2JutUkLUlsN2F8tueZogKhskm..
  .text‖01000270e:           727EDA0270                      ldstr        "W艾 艾چاXO马ざLMnıQQ"
  .text‖010002713:           2807000006                      call         WallpaperChanger.Update.RijndaelDecrypt()     ↑3         ; →    .)(.:
  .text‖010002718:           8011000004                      stsfld       WallpaperChanger.QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.Dat
  .text‖01000271d:           2A                              ret
                          }

  .text‖01000271e:           0000                            add          [eax], al
  .text‖010002720:           1330                            adc          esi, [eax]
  .text‖010002722:           0200                            add          al, [eax]
  .text‖010002724:           3900                            cmp          [eax], eax
  .text‖010002726:           0000                            add          [eax], al
  .text‖010002728:           0A00                            or           al, [eax]
  .text‖01000272a:           0011                            add          [ecx], dl

                          ;============================================== GET_RESOURCEMANAGER ==============================================
→ 1 reference             WallpaperChanger.Properties.Resources.get_ResourceManager() {
  .text‖01000272c:           00                              nop
  .text‖01000272d:           7E12000004                      ldsfld       WallpaperChanger.Properties.Resources.resourceMan
  .text‖010002732:           14                              ldnull
  .text‖010002733:           FE01                            ceq
  .text‖010002735:           0A                              stloc        local0
  .text‖010002736:           06                              ldloc        local0
  .text‖010002737:           2C22                          ┌ brzero.s     .1      ↓4
  .text‖010002739:           00                            │ nop
  .text‖01000273a:           729EDA0270                    │ ldstr        "WallpaperChanger.Properties.Resources"
  .text‖01000273f:           D008000002                    │ ldtoken      WallpaperChanger.Properties.Resources
  .text‖010002744:           287200000A                    │ call         [System.Type.GetTypeFromHandle]
  .text‖010002749:           6F7300000A                    │ callvirt     [System.Type.get_Assembly]
  .text‖01000274e:           737400000A                    │ newobj       [System.Resources.ResourceManager..ctor]
  .text‖010002753:           0B                            │ stloc        local1
  .text‖010002754:           07                            │ ldloc        local1
  .text‖010002755:           8012000004                    │ stsfld       WallpaperChanger.Properties.Resources.resourceMan
  .text‖01000275a:           00                            │ nop
→ 1 reference             .1:                              │
  .text‖01000275b:           7E12000004                    └▷ldsfld       WallpaperChanger.Properties.Resources.resourceMan
  .text‖010002760:           0C                              stloc        local2
  .text‖010002761:           2B00                          ┌ br.s         .2      ↓5
```

Figure 9: decrypting the Tesla resource

What we get is a reflexive PE injector .NET DLL rightly named `RunPE.dll`. This is the kind of utility assembly which is used by dropper to inject their payload into a running process. Interesting, but it's definitely not our payload.

Decrypting the base64 string

Our next payload candidate is the big 185kb base64-encoded string located at address `0x100131da`. There is again only on code location referencing this string at address `0x1000208c`. We can see that the string is decrypted using the method `RijndaelDecrypt` this time using the key `"wnhILKQcVU"`. This is the same key which was used in the first layer for the XOR encryption.



```
;================================================================= RIJNDAELDECRYPT ================================================
WallpaperChanger.Update.RijndaelDecrypt() {
    00                            nop
    733400000A                    newobj      [System.Security.Cryptography.RijndaelManaged..ctor]       ; crypto algo
    0A                            stloc       local0
    1E                            ldc.i4      0x8
    8D33000001                    newarr      System.Byte
    25                            dup
    D015000004                    ldtoken     DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5        ; initial values for the 8-bytes array
    283500000A                    call        [System.Runtime.CompilerServices.RuntimeHelpers.InitializeArray]
    0C                            stloc       local2
    03                            ldarg       arg1                                ; key
    08                            ldloc       local2                              ; salt = 8-bytes array
    733600000A                    newobj      [System.Security.Cryptography.Rfc2898DeriveBytes..ctor]
    0D                            stloc       local3
    06                            ldloc       local0
    09                            ldloc       local3
    06                            ldloc       local0
    6F3700000A                    callvirt    [System.Security.Cryptography.SymmetricAlgorithm.get_Key]
    8E                            ldlen
    69                            conv.i4                                         ; generate as much bytes at the Rijndael key size
    6F3800000A                    callvirt    [System.Security.Cryptography.DeriveBytes.GetBytes]
    6F3900000A                    callvirt    [System.Security.Cryptography.SymmetricAlgorithm.set_Key]
    00                            nop
    06                            ldloc       local0
    09                            ldloc       local3
    06                            ldloc       local0
    6F3A00000A                    callvirt    [System.Security.Cryptography.SymmetricAlgorithm.get_IV]
    8E                            ldlen
    69                            conv.i4                                         ; generate as much bytes at the Rijndael IV size
    6F3800000A                    callvirt    [System.Security.Cryptography.DeriveBytes.GetBytes]
    6F3B00000A                    callvirt    [System.Security.Cryptography.SymmetricAlgorithm.set_IV]
    00                            nop
    733000000A                    newobj      [System.IO.MemoryStream..ctor]
    1304                          stloc.s     local4
    1104                          ldloc.s     local4
    06                            ldloc       local0                              ; and decrypt everyting
    6F3C00000A                    callvirt    [System.Security.Cryptography.SymmetricAlgorithm.CreateDecryptor]
    17                            ldc.i4      0x1
    733D00000A                    newobj      [System.Security.Cryptography.CryptoStream..ctor]
    1305                          stloc.s     local5
```

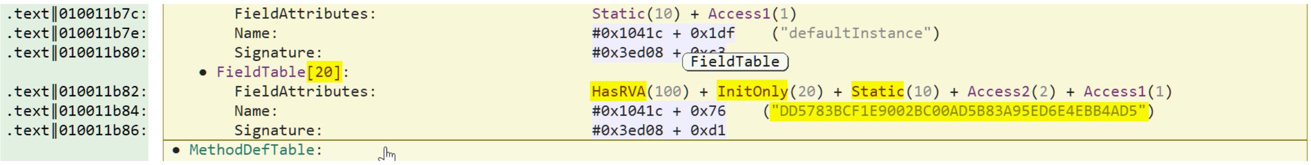
Figure 10: the RijndaelDecrypt method

This time the block cipher is used in CBC mode (the default in .NET) and the key generation is based on the Rfc2898 (aka PBKDF2) algorithm. If we have a look at the offical documentation, we can see that the constructor of the class `Rfc2898DeriveBytes` takes two inputs:

- a key, which in our case would be the string `"wnhILKQcVU"` (encoded in UTF-8 by default, since no encoding is specified)
- a salt, which looks like a 8 bytes array initialized with the value of the field `DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5`

The class `Rfc2898DeriveBytes` is then used to generate a given number of bytes (32 and then 16 in this case) which are used as key and IV for the cipher. Regarding the Rijndael algorithm, we can see that in the .NET core implementation, it defaults to AES256. This is good news for us, this means that the only thing we have to figure out is how to generate the key and IV. Again, we could debug the sample, but where is the fun in that? We will rewrite it in python instead.

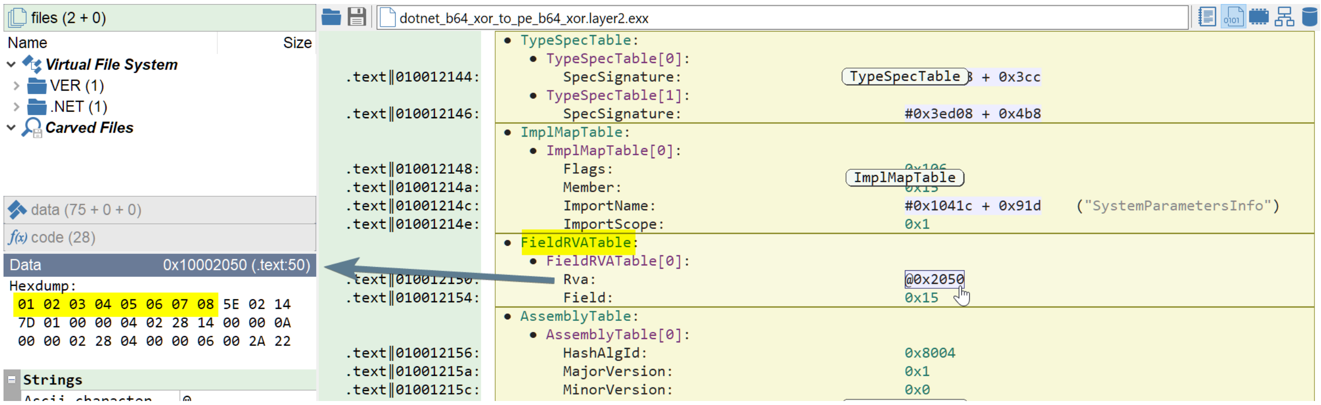
First thing first, we have to retrieve the salt value (an 8 bytes array) which is located in the field `DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5`. By clicking on the field in the Code view, we can see its definition in the `FieldTable` structure. This field has three important flags set: `HasRVA`, `Static` and `InitOnly` which indicates that this is a static initialized variable. Also the `HasRVA` flag tells us that the field has an entry inside the .NET `FieldRVA` table.



```
.text‖010011b7c:        FieldAttributes:              Static(10) + Access1(1)
.text‖010011b7e:        Name:                         #0x1041c + 0x1df    ("defaultInstance")
.text‖010011b80:        Signature:                    #0x3ed08 + 0xc3
                  • FieldTable[20]:
.text‖010011b82:        FieldAttributes:              HasRVA(100) + InitOnly(20) + Static(10) + Access2(2) + Access1(1)
.text‖010011b84:        Name:                         #0x1041c + 0x76    ("DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5")
.text‖010011b86:        Signature:                    #0x3ed08 + 0xd1
                • MethodDefTable:
```

Figure 11: the field holding the salt value

The `FieldRVA` table has only one entry for field number 0x15 (aka 21) which is our field (the field `DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5` is at index 20 aka 0x14 in the `FieldTable`, but Field references start at 1 because 0 is reserved).



```
Data                    0x10002050 (.text:50)
Hexdump:
 01 02 03 04 05 06 07 08 5E 02 14
 7D 01 00 00 04 02 28 14 00 00 0A
 00 00 02 28 04 00 00 06 00 2A 22
```

```
                     • TypeSpecTable:
                         • TypeSpecTable[0]:
.text‖010012144:             SpecSignature:            #0x3ed08 + 0x3cc
                         • TypeSpecTable[1]:
.text‖010012146:             SpecSignature:            #0x3ed08 + 0x4b8
                     • ImplMapTable:
                         • ImplMapTable[0]:
.text‖010012148:             Flags:                    0x106
.text‖01001214a:             Member:                   0x15
.text‖01001214c:             ImportName:               #0x1041c + 0x91d    ("SystemParametersInfo")
.text‖01001214e:             ImportScope:              0x1
                     • FieldRVATable:
                         • FieldRVATable[0]:
.text‖010012150:             Rva:                      @0x2050
.text‖010012154:             Field:                    0x15
                     • AssemblyTable:
                         • AssemblyTable[0]:
.text‖010012156:             HashAlgId:                0x8004
.text‖01001215a:             MajorVersion:             0x1
.text‖01001215c:             MinorVersion:             0x0
```

Figure 12: the corresponding FieldRVA entry

The format of the data stored depends on the field type (and whether or not a `ClassLayout` exists). But we are dealing with a very simple 8 bytes array here, so reading the initial value is very simple: it is `{ 1, 2, 3, 4, 5, 6, 7, 8 }`, our salt.

Next, we need to emulate the behavior of the class `Rfc2898DeriveBytes`. We will use the `Cryptodome` python package which comes bundled with Malcat and its PBKDF2 algorithm. Go into the script editor (shortcut: **F8**) and paste the following code:

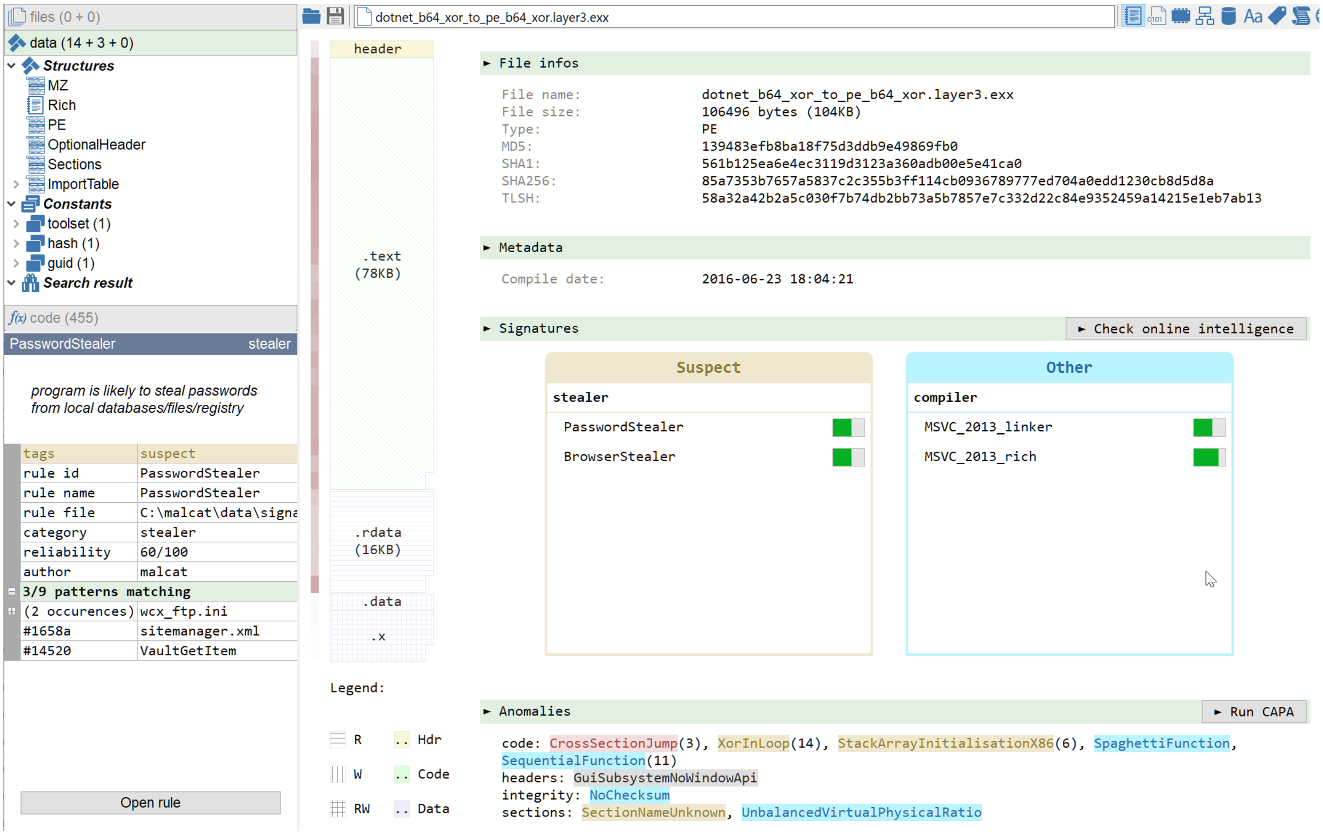
```
from Cryptodome.Protocol.KDF import PBKDF2
pwd = "wnhILKQcVU".encode("utf8")
salt = bytes(range(1, 9))           # content of
DD5783BCF1E9002BC00AD5B83A95ED6E4EBB4AD5
data = PBKDF2(pwd, salt, 32 + 16)
print(data[:32].hex())              # the first 32 bytes are used for the key
# -> "34ca280dd207ea1e1915f7ccdc5d59344c55c6863947e507e982a337bdc57742"
print(data[32:].hex())              # the next 16 bytes are used for the IV
# -> "be77bdd5564bbc0c4da984f89d88213d"
```

Now that we know both the key and the IV, we can decrypt the string at offset `0x100131da` using the usual steps:

- Right-click on the string from the code view or the strings view and chose **Transform..**

- Change encoding from utf16 to utf8
- Base64 decode the result
- AES decrypt the result in CBC mode using the key and IV found above
- Base64 decode the result ...
- Extract the GZipped content ......
- We get a new PE file!

The PE file looks like a native infostealer and is detected as *Loki* on VirusTotal. While a lot of its content is in plain text, some strings and configurations are still encrypted. The decryption process may be the subject of another blog post.



PasswordStealer — stealer

program is likely to steal passwords from local databases/files/registry

tags	suspect
rule id	PasswordStealer
rule name	PasswordStealer
rule file	C:\malcat\data\signa
category	stealer
reliability	60/100
author	malcat
3/9 patterns matching	
(2 occurences)	wcx_ftp.ini
#1658a	sitemanager.xml
#14520	VaultGetItem

```
► File infos

  File name:     dotnet_b64_xor_to_pe_b64_xor.layer3.exx
  File size:     106496 bytes (104KB)
  Type:          PE
  MD5:           139483efb8ba18f75d3ddb9e49869fb0
  SHA1:          561b125ea6e4ec3119d3123a360adb00e5e41ca0
  SHA256:        85a7353b7657a5837c2c355b3ff114cb0936789777ed704a0edd1230cb8d5d8a
  TLSH:          58a32a42b2a5c030f7b74db2bb73a5b7857e7c332d22c84e9352459a14215e1eb7ab13

► Metadata

  Compile date:  2016-06-23 18:04:21

► Signatures
```

Suspect	
stealer	
PasswordStealer	🟩
BrowserStealer	🟩

Other	
compiler	
MSVC_2013_linker	🟩
MSVC_2013_rich	🟩

```
► Anomalies

  code: CrossSectionJump(3), XorInLoop(14), StackArrayInitialisationX86(6), SpaghettiFunction,
  SequentialFunction(11)
  headers: GuiSubsystemNoWindowApi
  integrity: NoChecksum
  sections: SectionNameUnknown, UnbalancedVirtualPhysicalRatio
```

Figure 13: the final payload: Loki infostealer

Conclusion

We have seen how to navigate inside a .NET program, look for possible payload locations and how to use the different decryption algorithms inside Malcat to extract the stages of the malware. We also introduced the python script engine of Malcat, even if we just scratched the surface there (a scripting example which makes use of the bindings will be the subject of a future blog post).

Statically unpacking a sample, while more complicated than debugging, offer many advantages:

- we get better quality dumps

- we don't care about anti-debugging and anti-sandboxing tricks
- the scripts which were developed can be reused on other samples in the future
- it forces us to better understand the packing logic, and makes us less likely to miss something

I hope you enjoyed this first tutorial, feel free to share with us your remarks or suggestions!