"Nice decorating. Let me guess, Satan?" - Dot / MZP Ransomware

dissectingmalwa.re/nice-decorating-let-me-guess-satan-dot-mzp-ransomware.html

Thu 02 January 2020 in Ransomware

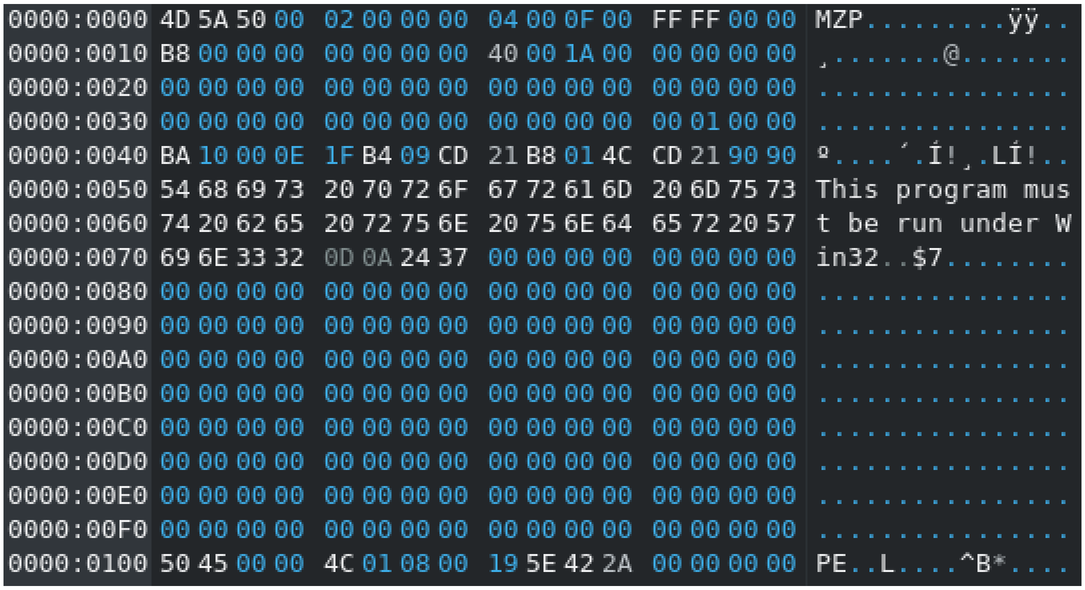
Happy new year y'all. And with it there's new Ransomware to analyze, so come along for the ride :D

Dot "MZP" Ransomware @ AnyRun | VirusTotal | HybridAnalysis --> `sha256 bebf5c12e35029e21c9cca1da53eb43e893f9521435a246ea991bcced2fabe67`

This sample was first discovered by AmigoA and AkhmendTaia on the 31st of December 2019. AV Detections and Ransomnote contents didn't seem to match any previously present strain. The Note is delivered via a *.txt* File with a strange numeric victim ID and only one contact email address. The extension appended to encrypted Files seems to be a random 8 character lowercase string.

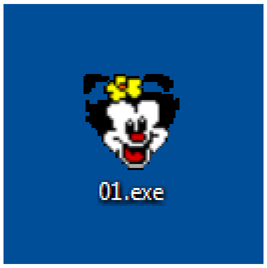
> New #MZP #Ransomwarehttps://t.co/YCY8NXzJZw
> It seems nothing special, but early AV-detections is uninformative.
> Thanks to @AkhmedTaia pic.twitter.com/qS1YapH8jW
>
> — Amigo-A (@Amigo_A_) December 31, 2019

Because of the "MZP" (4D 5A 50) Magic at the beginning of the executable file they dubbed the Malware "MZP" Ransomware. As I explained before with the MZRevenge/MaMo Ransomware the "P" after the MZ Magic String indicates that the binary was built with Borland Delphi and *P* stands for Pascal (the programming language).



```
0000:0000 4D 5A 50 00  02 00 00 00  04 00 0F 00  FF FF 00 00  MZP.........ÿÿ..
0000:0010 B8 00 00 00  00 00 00 00  40 00 1A 00  00 00 00 00  ¸.......@.......
0000:0020 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0030 00 00 00 00  00 00 00 00  00 00 00 00  00 01 00 00  ................
0000:0040 BA 10 00 0E  1F B4 09 CD  21 B8 01 4C  CD 21 90 90  º....´.Í!¸.LÍ!..
0000:0050 54 68 69 73  20 70 72 6F  67 72 61 6D  20 6D 75 73  This program mus
0000:0060 74 20 62 65  20 72 75 6E  20 75 6E 64  65 72 20 57  t be run under W
0000:0070 69 6E 33 32  0D 0A 24 37  00 00 00 00  00 00 00 00  in32..$7........
0000:0080 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0090 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00A0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00B0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00C0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00D0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00F0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0100 50 45 00 00  4C 01 08 00  19 5E 42 2A  00 00 00 00  PE..L....^B*....
```

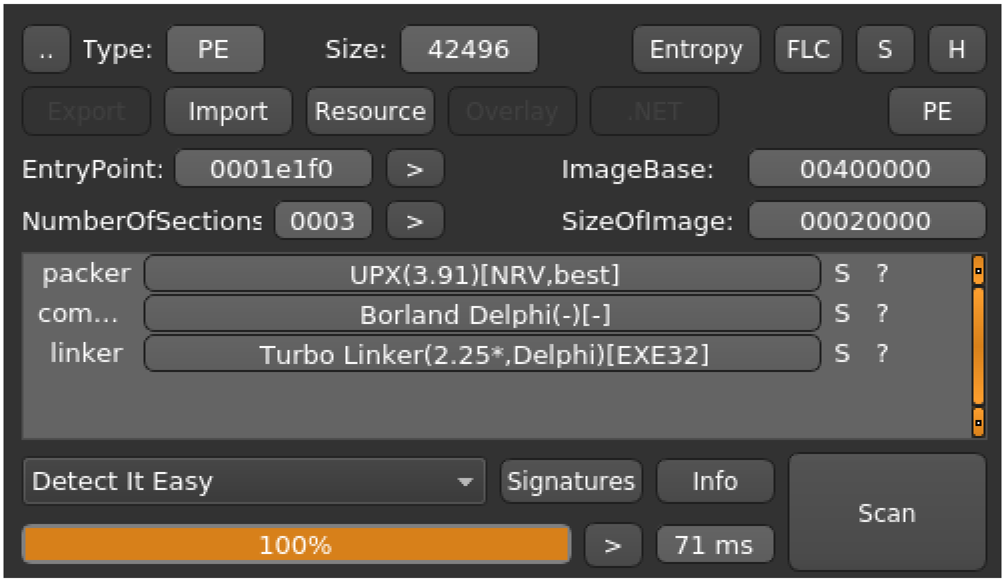
In my Opinion the Name "MZP Ransomware" is too generic to be useful for future reference, so I'd like to propose the name "Dot Ransomware" because of the File Icon found with the Malware Samples. It shows the character "Dot" from the Warner Bros Cartoon Series "Animaniacs" popular in the mid-1990s.



Two things to note about the Output of "Detect it easy" for this sample:

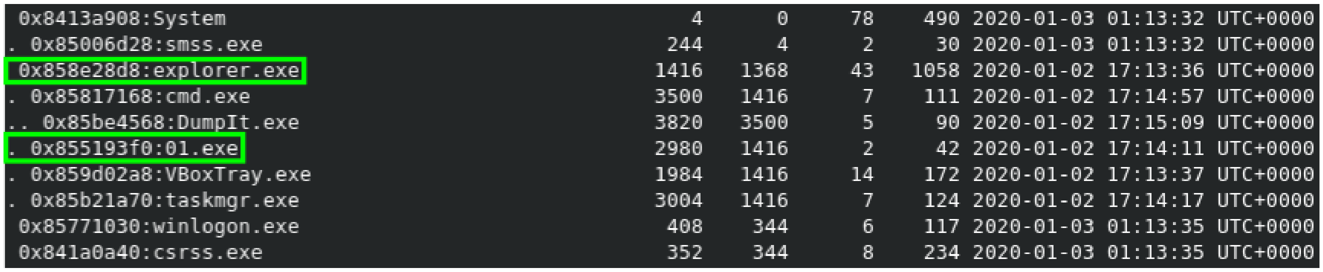
1. It confirms that the Ransomware was built with Borland Delphi (Version 4).

2. This sample seems to be packed with UPX 3.91. Running `upx -d 01.exe` yields us the unpacked Version. The Hashsums can be found in the IOC Section down below



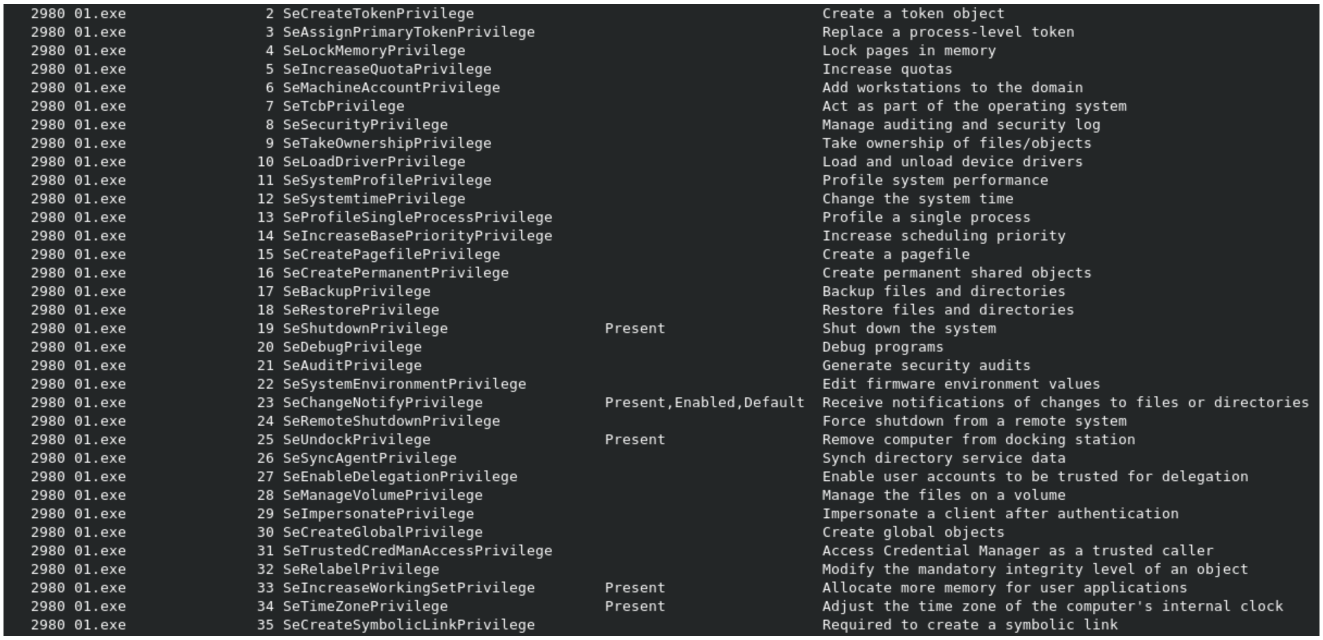
Type: PE
Size: 42496
EntryPoint: 0001e1f0
ImageBase: 00400000
NumberOfSections 0003
SizeOfImage: 00020000
packer UPX(3.91)[NRV,best]
com... Borland Delphi(-)[-]
linker Turbo Linker(2.25*,Delphi)[EXE32]

Let's try something new :D Up until now I pretty much neglected memory dump analysis as a whole, but since I attended the Workshop on Volatility at 36c3 I noticed what I'm missing out on. With `volatility -f IE9WIN7-20200102-171509.dmp --profile=Win7SP1x86_24000 pstree` we can dump the process tree at the time of the capture. We can see that 01.exe is running as a subprocess of explorer.exe.



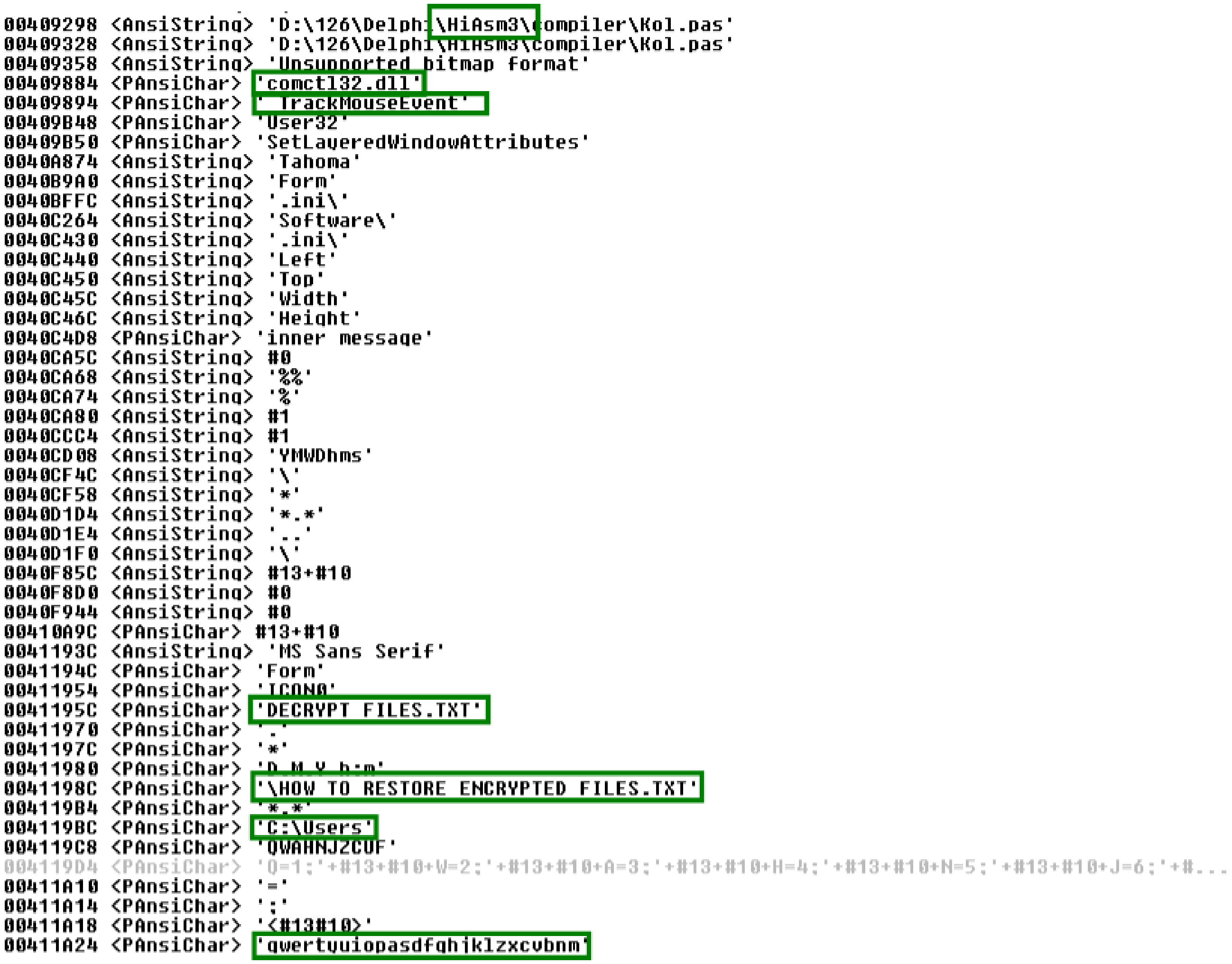
```
 0x8413a908:System                                        4      0     78     490 2020-01-03 01:13:32 UTC+0000
. 0x85006d28:smss.exe                                   244      4      2      30 2020-01-03 01:13:32 UTC+0000
 0x858e28d8:explorer.exe                               1416   1368     43    1058 2020-01-02 17:13:36 UTC+0000
. 0x85817168:cmd.exe                                   3500   1416      7     111 2020-01-02 17:14:57 UTC+0000
.. 0x85be4568:DumpIt.exe                               3820   3500      5      90 2020-01-02 17:15:09 UTC+0000
. 0x855193f0:01.exe                                    2980   1416      2      42 2020-01-02 17:14:11 UTC+0000
. 0x859d02a8:VBoxTray.exe                              1984   1416     14     172 2020-01-02 17:13:37 UTC+0000
. 0x85b21a70:taskmgr.exe                               3004   1416      7     124 2020-01-02 17:14:17 UTC+0000
 0x85771030:winlogon.exe                                408    344      6     117 2020-01-03 01:13:35 UTC+0000
 0x841a0a40:csrss.exe                                   352    344      8     234 2020-01-03 01:13:35 UTC+0000
```

With the *privs* plugin Volatility can show which process privileges are present, enabled, and/or enabled by default. Below you can see a screencapture of the output for the Ransomware. The Plugins *cmdscan* and *consoles* sadly did not return any output for 01.exe.



```
  2980 01.exe                 2 SeCreateTokenPrivilege                                      Create a token object
  2980 01.exe                 3 SeAssignPrimaryTokenPrivilege                               Replace a process-level token
  2980 01.exe                 4 SeLockMemoryPrivilege                                       Lock pages in memory
  2980 01.exe                 5 SeIncreaseQuotaPrivilege                                    Increase quotas
  2980 01.exe                 6 SeMachineAccountPrivilege                                   Add workstations to the domain
  2980 01.exe                 7 SeTcbPrivilege                                              Act as part of the operating system
  2980 01.exe                 8 SeSecurityPrivilege                                         Manage auditing and security log
  2980 01.exe                 9 SeTakeOwnershipPrivilege                                    Take ownership of files/objects
  2980 01.exe                10 SeLoadDriverPrivilege                                       Load and unload device drivers
  2980 01.exe                11 SeSystemProfilePrivilege                                    Profile system performance
  2980 01.exe                12 SeSystemtimePrivilege                                       Change the system time
  2980 01.exe                13 SeProfileSingleProcessPrivilege                             Profile a single process
  2980 01.exe                14 SeIncreaseBasePriorityPrivilege                             Increase scheduling priority
  2980 01.exe                15 SeCreatePagefilePrivilege                                   Create a pagefile
  2980 01.exe                16 SeCreatePermanentPrivilege                                  Create permanent shared objects
  2980 01.exe                17 SeBackupPrivilege                                           Backup files and directories
  2980 01.exe                18 SeRestorePrivilege                                          Restore files and directories
  2980 01.exe                19 SeShutdownPrivilege              Present                    Shut down the system
  2980 01.exe                20 SeDebugPrivilege                                            Debug programs
  2980 01.exe                21 SeAuditPrivilege                                            Generate security audits
  2980 01.exe                22 SeSystemEnvironmentPrivilege                                Edit firmware environment values
  2980 01.exe                23 SeChangeNotifyPrivilege          Present,Enabled,Default    Receive notifications of changes to files or directories
  2980 01.exe                24 SeRemoteShutdownPrivilege                                   Force shutdown from a remote system
  2980 01.exe                25 SeUndockPrivilege                Present                    Remove computer from docking station
  2980 01.exe                26 SeSyncAgentPrivilege                                        Synch directory service data
  2980 01.exe                27 SeEnableDelegationPrivilege                                 Enable user accounts to be trusted for delegation
  2980 01.exe                28 SeManageVolumePrivilege                                     Manage the files on a volume
  2980 01.exe                29 SeImpersonatePrivilege                                      Impersonate a client after authentication
  2980 01.exe                30 SeCreateGlobalPrivilege                                     Create global objects
  2980 01.exe                31 SeTrustedCredManAccessPrivilege                             Access Credential Manager as a trusted caller
  2980 01.exe                32 SeRelabelPrivilege                                          Modify the mandatory integrity level of an object
  2980 01.exe                33 SeIncreaseWorkingSetPrivilege    Present                    Allocate more memory for user applications
  2980 01.exe                34 SeTimeZonePrivilege              Present                    Adjust the time zone of the computer's internal clock
  2980 01.exe                35 SeCreateSymbolicLinkPrivilege                               Required to create a symbolic link
```

Let's check out what IDR (Interactive Delphi Reconstructor) can tell us about the binary. First off: Strings.



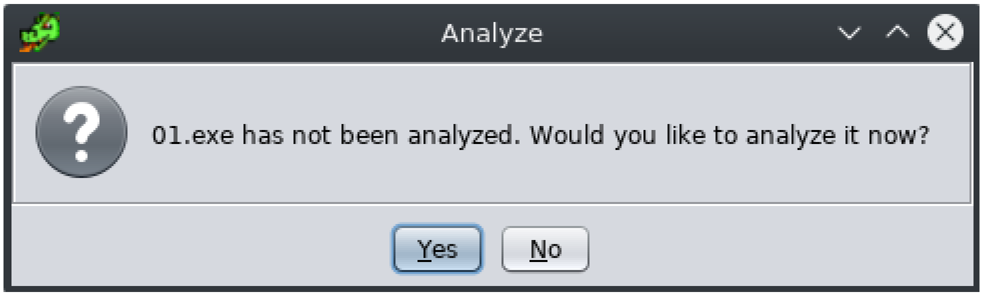
```
00409298 <AnsiString> 'D:\126\Delphi\HiAsm3\compiler\Kol.pas'
00409328 <AnsiString> 'D:\126\Delphi\HiAsm3\compiler\Kol.pas'
00409358 <AnsiString> 'Unsupported bitmap format'
00409884 <PAnsiChar> 'comctl32.dll'
00409894 <PAnsiChar> '_TrackMouseEvent'
00409B48 <PAnsiChar> 'User32'
00409B50 <PAnsiChar> 'SetLayeredWindowAttributes'
0040A874 <AnsiString> 'Tahoma'
0040B9A0 <AnsiString> 'Form'
0040BFFC <AnsiString> '.ini\'
0040C264 <AnsiString> 'Software\'
0040C430 <AnsiString> '.ini\'
0040C440 <AnsiString> 'Left'
0040C450 <AnsiString> 'Top'
0040C45C <AnsiString> 'Width'
0040C46C <AnsiString> 'Height'
0040C4D8 <PAnsiChar> 'inner message'
0040CA5C <AnsiString> #0
0040CA68 <AnsiString> '%%'
0040CA74 <AnsiString> '%'
0040CA80 <AnsiString> #1
0040CCC4 <AnsiString> #1
0040CD08 <AnsiString> 'YMWDhms'
0040CF4C <AnsiString> '\'
0040CF58 <AnsiString> '*'
0040D1D4 <AnsiString> '*.*'
0040D1E4 <AnsiString> '..'
0040D1F0 <AnsiString> '\'
0040F85C <AnsiString> #13+#10
0040F8D0 <AnsiString> #0
0040F944 <AnsiString> #0
00410A9C <PAnsiChar> #13+#10
0041193C <AnsiString> 'MS Sans Serif'
0041194C <PAnsiChar> 'Form'
00411954 <PAnsiChar> 'ICON0'
0041195C <PAnsiChar> 'DECRYPT FILES.TXT'
00411970 <PAnsiChar> '.'
0041197C <PAnsiChar> '*'
00411980 <PAnsiChar> 'D.M.Y h:m'
0041198C <PAnsiChar> '\HOW TO RESTORE ENCRYPTED FILES.TXT'
004119B4 <PAnsiChar> '*.*'
004119BC <PAnsiChar> 'C:\Users'
004119C8 <PAnsiChar> 'QWAHNJZCUF'
004119D4 <PAnsiChar> 'Q=1;'+#13+#10+W=2;'+#13+#10+A=3;'+#13+#10+H=4;'+#13+#10+N=5;'+#13+#10+J=6;'+#...
00411A10 <PAnsiChar> '='
00411A14 <PAnsiChar> ';'
00411A18 <PAnsiChar> '<#13#10>'
00411A24 <PAnsiChar> 'qwertyuiopasdfghjklzxcvbnm'
```

The first String related to the Compiler tells us that the criminals likely used *HiASM* (an old russian IDE for Delphi Development) to build the Malware. The DLL mentioned below *comctl32.dll* is often targeted for UAC Bypasses. It also seems to track Mouse events to some extent this could either be used as an evasion mechanism or entropy collection (the first option is a lot more plausible). **"HOW TO RESTORE ENCRYPTED FILES.txt"** is the filename of the dropped ransomnote, although I'm not sure about the use of **"DECRYPT FILES.txt"** since this file was not present on any infected system (Speculation: Does is select one out of multiple Filenames to make tracking more difficult?). Lastly we have a filepath and a string that looks like the criminal dragged his face across the keyboard once.

Alright, let's move along. Because Delphi is notoriously weird and difficult to disassemble/decompile it is time to try a new tool again. Today I will be using Ghidra with *Dhrake* developed by Jesko Hüttenhain. You can find the Git repository below and if you would like to know more about the inner workings of the two scripts you should definitely read his article about them here.

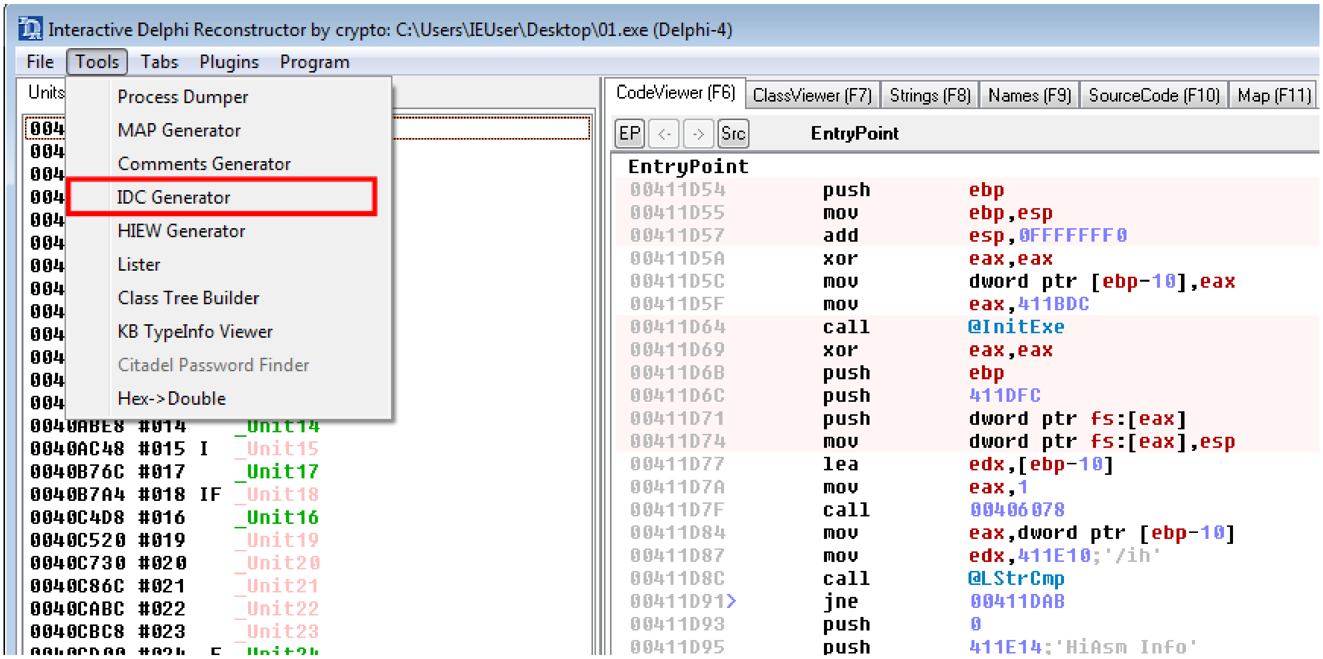
A short tl;dr: Dhrake is short for "Delphi hand rake" and tries to fix missing symbols and borked function signatures by matching to the symbols extracted through IDR beforehand. This will not only clean up the decompilation results in Ghidra but also automatically create structs and virtual method tables for you instead of doing it by hand (as if reversing Delphi wasn't already painfull enough). It's pretty cool, give it a try!

The first step to success (lol who am I kidding) is firing up Ghdira and loading the sample. Tell it to auto-analyze the file.



01.exe has not been analyzed. Would you like to analyze it now?

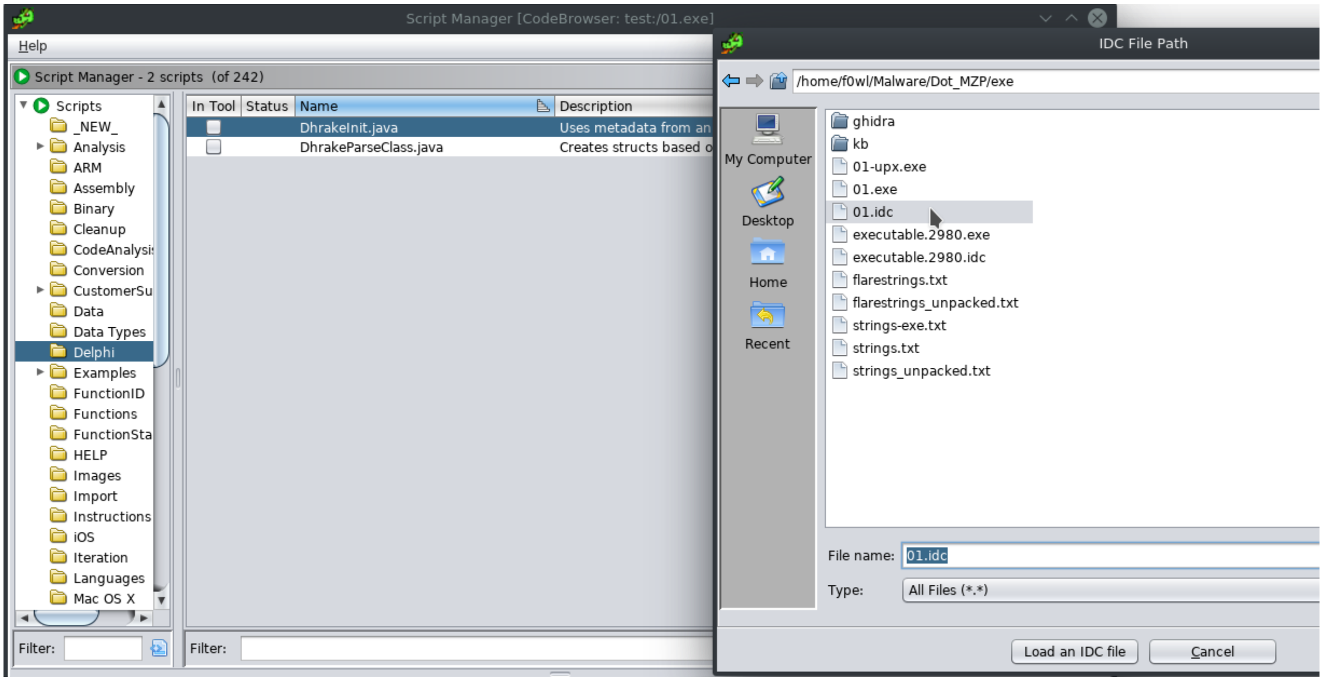
Next we need to extract the *.IDC* Symbol file with the Help of IDR. For this it is sufficient to clone the Git Repo and paste the Knowledge Base files from the Dropbox linked at the end of the Readme into it. After that is done just run IDR.exe, import the binary and choose *IDC Generator* under **Tools**.



```
0040ABE8 #014      _Unit14
0040AC48 #015 I    _Unit15
0040B76C #017      _Unit17
0040B7A4 #018 IF   _Unit18
0040C4D8 #016      _Unit16
0040C520 #019      _Unit19
0040C730 #020      _Unit20
0040C86C #021      _Unit21
0040CABC #022      _Unit22
0040CBC8 #023      _Unit23
```

```
EntryPoint
00411D54        push     ebp
00411D55        mov      ebp,esp
00411D57        add      esp,0FFFFFFF0
00411D5A        xor      eax,eax
00411D5C        mov      dword ptr [ebp-10],eax
00411D5F        mov      eax,411BDC
00411D64        call     @InitExe
00411D69        xor      eax,eax
00411D6B        push     ebp
00411D6C        push     411DFC
00411D71        push     dword ptr fs:[eax]
00411D74        mov      dword ptr fs:[eax],esp
00411D77        lea      edx,[ebp-10]
00411D7A        mov      eax,1
00411D7F        call     00406078
00411D84        mov      eax,dword ptr [ebp-10]
00411D87        mov      edx,411E10;'/ih'
00411D8C        call     @LStrCmp
00411D91>       jne      00411DAB
00411D93        push     0
00411D95        push     411E14;'HiAsm Info'
```

After copying the two Dhrake scripts into your ghidra_scripts folder (e.g. ~/ghidra_scripts) you can refresh the list in the Script Manager once and switch to the Delphi Category. Run DhrakeInit and select the IDC file you created earlier.

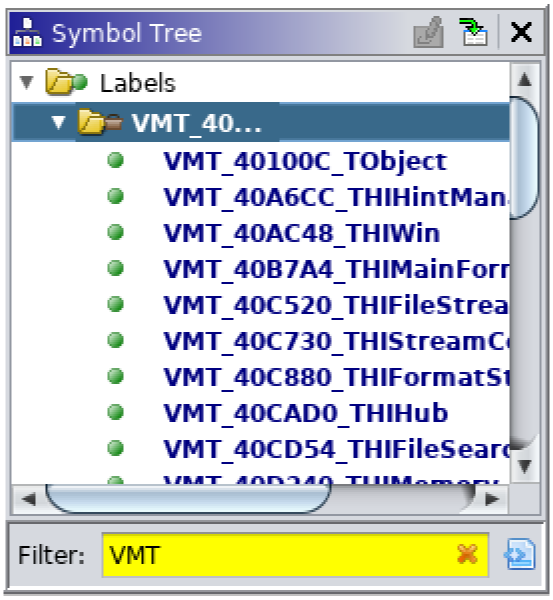


In Tool	Status	Name	Description
□		DhrakeInit.java	Uses metadata from an
□		DhrakeParseClass.java	Creates structs based o

File name: 01.idc

Type: All Files (*.*)

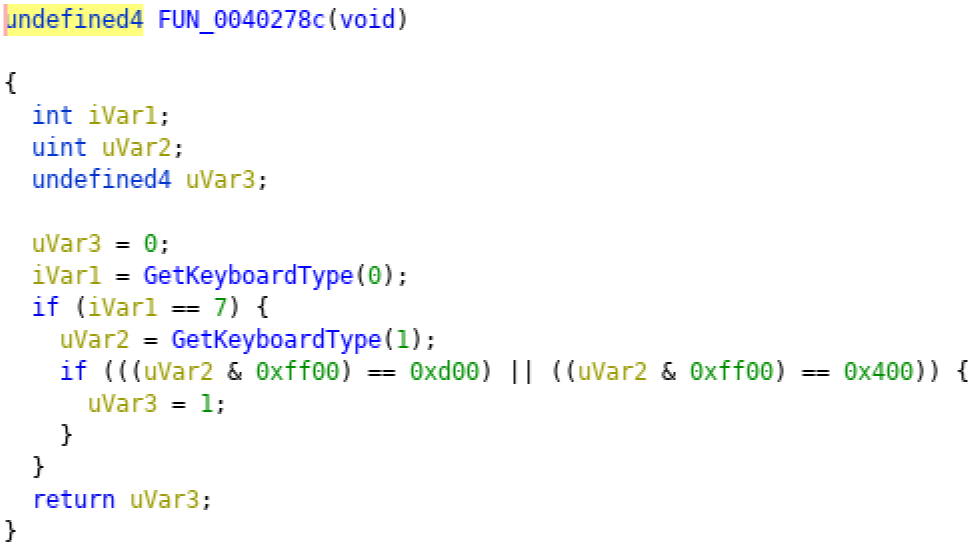
Filtering for "VMT" in the Symbol Tree gives you all the Symbols relevant to Dhrake. Just click the Name in the Listing view once and run DhrakeParseClass (set the checkbox "In Tool" and press F8 to run). The Script will now automatically create the corresponding class and vtable struct.



VMT_40100C_TObject
VMT_40A6CC_THIHintMan
VMT_40AC48_THIWin
VMT_40B7A4_THIMainForr
VMT_40C520_THIFileStrea
VMT_40C730_THIStreamC
VMT_40C880_THIFormatSt
VMT_40CAD0_THIHub
VMT_40CD54_THIFileSearc
VMT_40D240_THIMemory

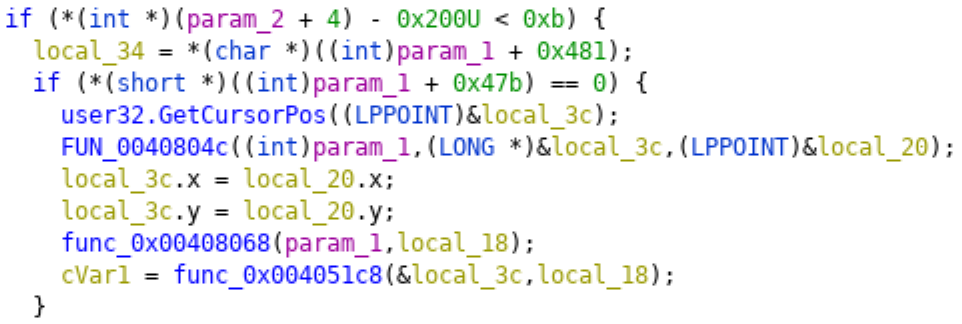
Filter: VMT

So I guess we should continue with the analysis now :D As 90% of ransomware strains do "Dot" will read the Keyboard Layout as well. *GetKeyboardLayout(0)* returning 7 would be equivalent to a Japanese Keyboard Layout (wtf?). Passing 1 to GetKeyboardType will return the Subtype which is OEM specific, but will tell you how many function keys there are. Weird. Here's the Documentation.



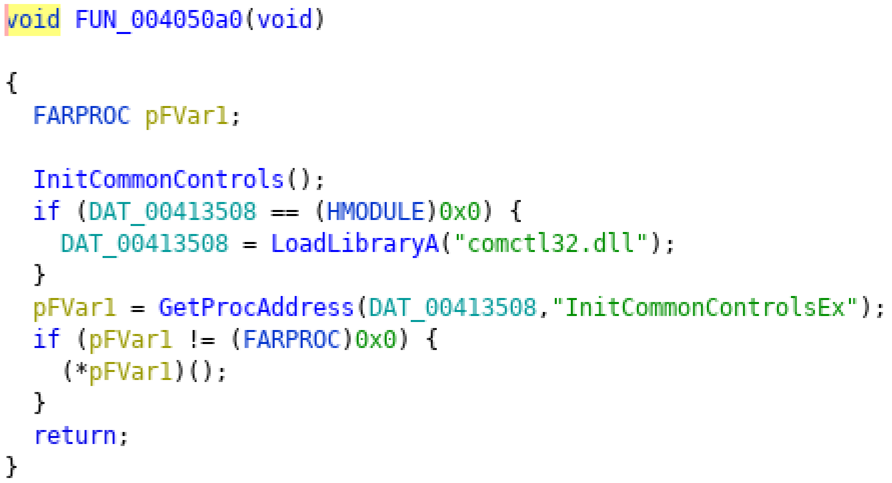
```
undefined4 FUN_0040278c(void)

{
  int iVar1;
  uint uVar2;
  undefined4 uVar3;

  uVar3 = 0;
  iVar1 = GetKeyboardType(0);
  if (iVar1 == 7) {
    uVar2 = GetKeyboardType(1);
    if (((uVar2 & 0xff00) == 0xd00) || ((uVar2 & 0xff00) == 0x400)) {
      uVar3 = 1;
    }
  }
  return uVar3;
}
```

Dot also queries the current cursor position on the screen and passes it on to another function. Haven't investigated further yet.



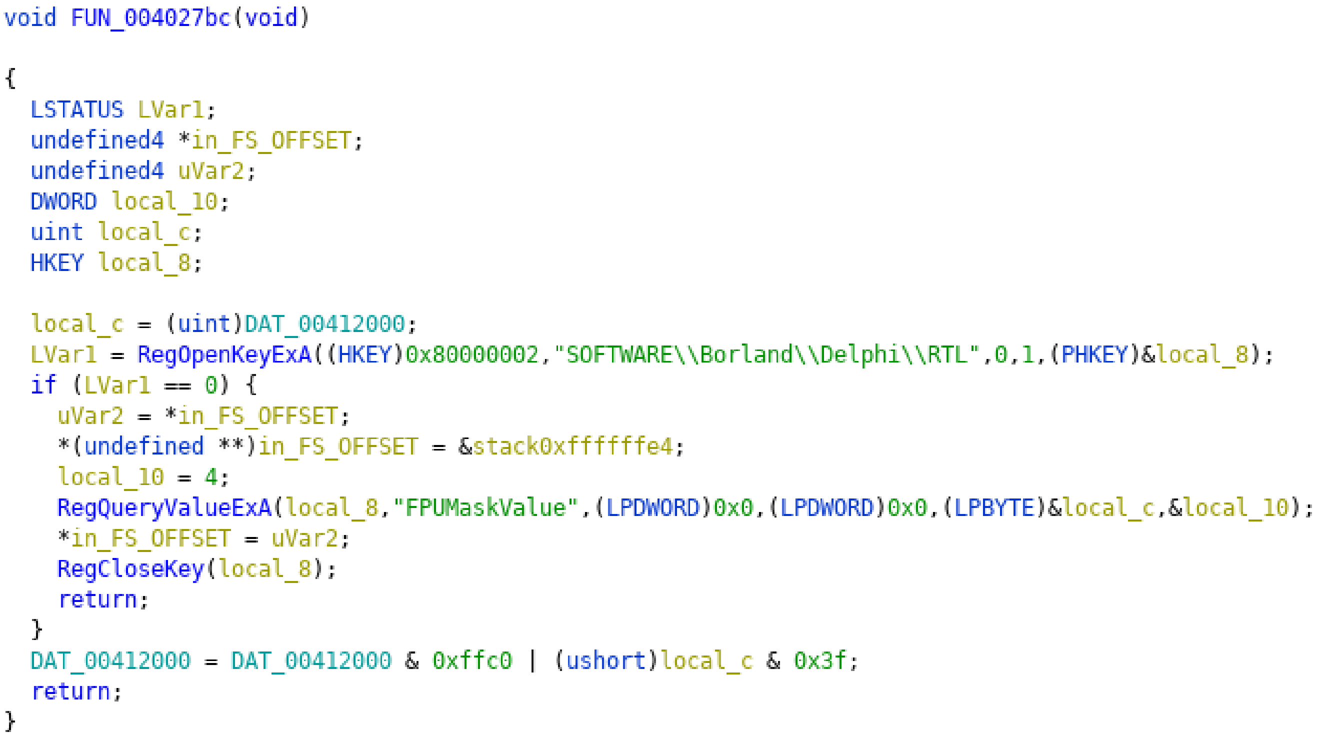
```
if (*(int *)(param_2 + 4) - 0x200U < 0xb) {
  local_34 = *(char *)((int)param_1 + 0x481);
  if (*(short *)((int)param_1 + 0x47b) == 0) {
    user32.GetCursorPos((LPPOINT)&local_3c);
    FUN_0040804c((int)param_1,(LONG *)&local_3c,(LPPOINT)&local_20);
    local_3c.x = local_20.x;
    local_3c.y = local_20.y;
    func_0x00408068(param_1,local_18);
    cVar1 = func_0x004051c8(&local_3c,local_18);
  }
```

Here we are again: weird DLLs that may or may not be a UAC Bypass. UACme mentions two Methods (#21 and #22) employing comctl32.dll. Unsure what to make of this at the moment.



```
void FUN_004050a0(void)

{
  FARPROC pFVar1;

  InitCommonControls();
  if (DAT_00413508 == (HMODULE)0x0) {
    DAT_00413508 = LoadLibraryA("comctl32.dll");
  }
  pFVar1 = GetProcAddress(DAT_00413508,"InitCommonControlsEx");
  if (pFVar1 != (FARPROC)0x0) {
    (*pFVar1)();
  }
  return;
}
```

In one of the Szenarios I ran Regshot to see whether the Ransomware adds/modifies/deletes Registry Keys, but there weren't any changes that I can attribute to it. Dot tries to read *SOFTWARE\Borland\Delphi\RTL* FPUMaskValue.



```
void FUN_004027bc(void)

{
  LSTATUS LVar1;
  undefined4 *in_FS_OFFSET;
  undefined4 uVar2;
  DWORD local_10;
  uint local_c;
  HKEY local_8;

  local_c = (uint)DAT_00412000;
  LVar1 = RegOpenKeyExA((HKEY)0x80000002,"SOFTWARE\\Borland\\Delphi\\RTL",0,1,(PHKEY)&local_8);
  if (LVar1 == 0) {
    uVar2 = *in_FS_OFFSET;
    *(undefined **)in_FS_OFFSET = &stack0xffffffe4;
    local_10 = 4;
    RegQueryValueExA(local_8,"FPUMaskValue",(LPDWORD)0x0,(LPDWORD)0x0,(LPBYTE)&local_c,&local_10);
    *in_FS_OFFSET = uVar2;
    RegCloseKey(local_8);
    return;
  }
  DAT_00412000 = DAT_00412000 & 0xffc0 | (ushort)local_c & 0x3f;
  return;
}
```

This is another work in progress article as I've come down with the "Congress Flu", so check back in a few days for an update. Probably the most important thing this "report" is still missing is a look at the crypto implementation. A look at the Imports reveals that it is not using the Windows Crypto API but rather a weird Delphi one. We'll see.

MITRE ATT&CK

T1107 --> File Deletion --> Defense Evasion

T1045 --> Software Packing --> Defense Evasion

T1012 --> Query Registry --> Discovery

T1076 --> Remote Desktop Protocol --> Lateral Movement

IOCs

Dot Samples

```
01.exe --> SHA256: bebf5c12e35029e21c9cca1da53eb43e893f9521435a246ea991bcced2fabe67
           SSDEEP:
768:Qa8bmv7hNAMbgYT6hQdPLC7TasOKS/3U7fzd4tA9yenQ779Zo2lPnoCLnS9QtRbY:Ebmvs71+DKoKS/kjz


01.exe --> SHA256: aa85b2ec79bc646671d7280ba27f4ce97e8fabe93ab7c97d0fd18d05bab6df29
           SSDEEP:
98304:mt+HWV4nwA+8PgzCRfjMlFBiZhfcrQSav//dH768QyO4YXoftvFUmgaJml9iUybR:NddPgzC+lFkZhER


unpacked:
01.exe --> SHA256: 814e061d2e58720a43bcb3fe0478a8088053f0a407e25ff84fb98850d128f81c
           SSDEEP: 1536:CCq2EikJZdZ529nEaqQOyergddb6apjAwzHx4D:7IZYxEHJrIdFjAwzHx4
```

Registry Changes

Inconclusive as Regshot didn't show anything suspicious, only Delphi related Keys at most

E-Mail Addresses

```
recover_24_7@protonmail[.]com
```

Ransomnote

```
If you want to return your .[REDACTED: random 8-letter lowercase extension] files,
contact us and we will send you a decryptor and a unique decryption key.
recover_24_7@protonmail[.]com


All your files have been encrypted!
Your personal identifier:

======================================================================================


-------------------------------------------------------------------------------------
------
[REDACTED: 606-digit numeric ID]
-------------------------------------------------------------------------------------
------

======================================================================================
```