ESET Research white papers // June 2018 // Author: Kaspars Osis

QUASAR, SOBAKEN AND VERMIN:

A deeper look into an ongoing espionage campaign

CONTENTS

INTRODUCTION

Using remote access tools Quasar, Sobaken and Vermin, cybercriminals have been systematically spying on Ukrainian government institutions and exfiltrating data from their systems. The threat actors, first mentioned in a report from January 2018 and tracked by ESET since mid-2017, continue to develop new versions of their stealthy malware.

In this white paper, we take a closer look at this ongoing campaign. We provide further details on the malware used to compromise victims' systems and on the payloads installed on compromised systems, and describe the various methods the attackers use to distribute and target their malware while avoiding detection.

ATTACKER PROFILE

Even though these threat actors don't seem to possess advanced skills or access to 0-day vulnerabilities, they have been quite successful in using social engineering to both distribute their malware and fly under the radar for extended periods of time.

We were able to trace attacker activity back to October 2015; however, it is possible that the attackers have been active even longer.

These attackers use three different .NET malware strains in their attacks – Quasar RAT, Sobaken (a RAT derived from Quasar) and a custom-made RAT called Vermin. All three malware strains have been in active use against different targets at the same time, they share some infrastructure and connect to the same C&C servers. A possible explanation for using three parallel malware strains is that each strain is developed independently.

VICTIMS

This malware cluster has been used to target Ukrainian government institutions as seen in Figure 1. ESET's telemetry shows a few hundred victims in different organizations and several hundred executable files related to the campaign.

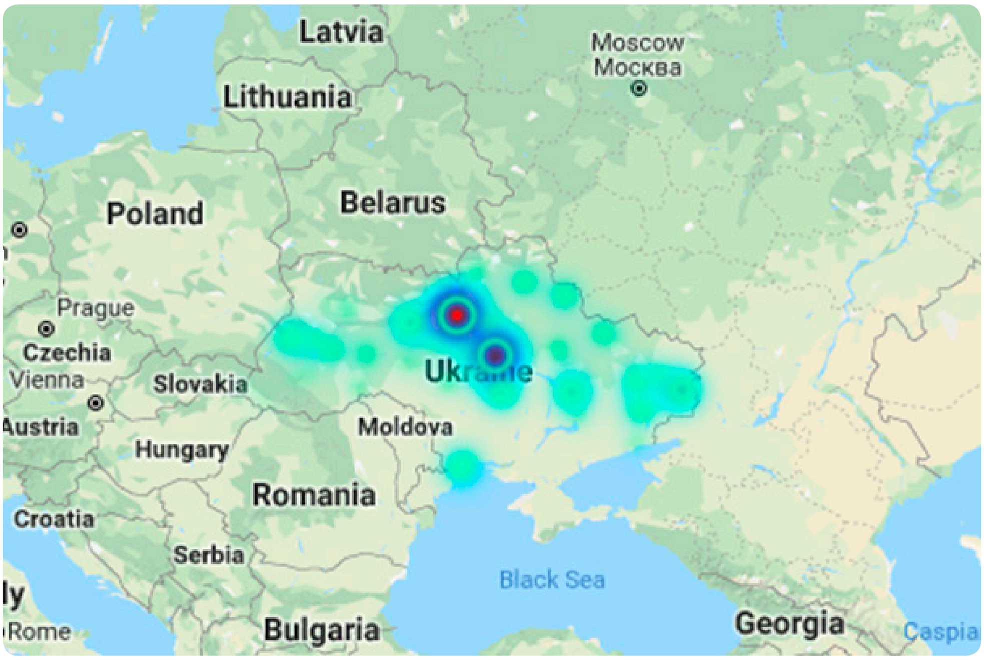


Figure 1 // ESET's detections of Quasar, Sobaken, Vermin, and their malicious components (Map data ©2018 Google, ORION-ME).

TIMELINE

Another way to look at telemetry data is in the form of a timeline, as in Figure 2.

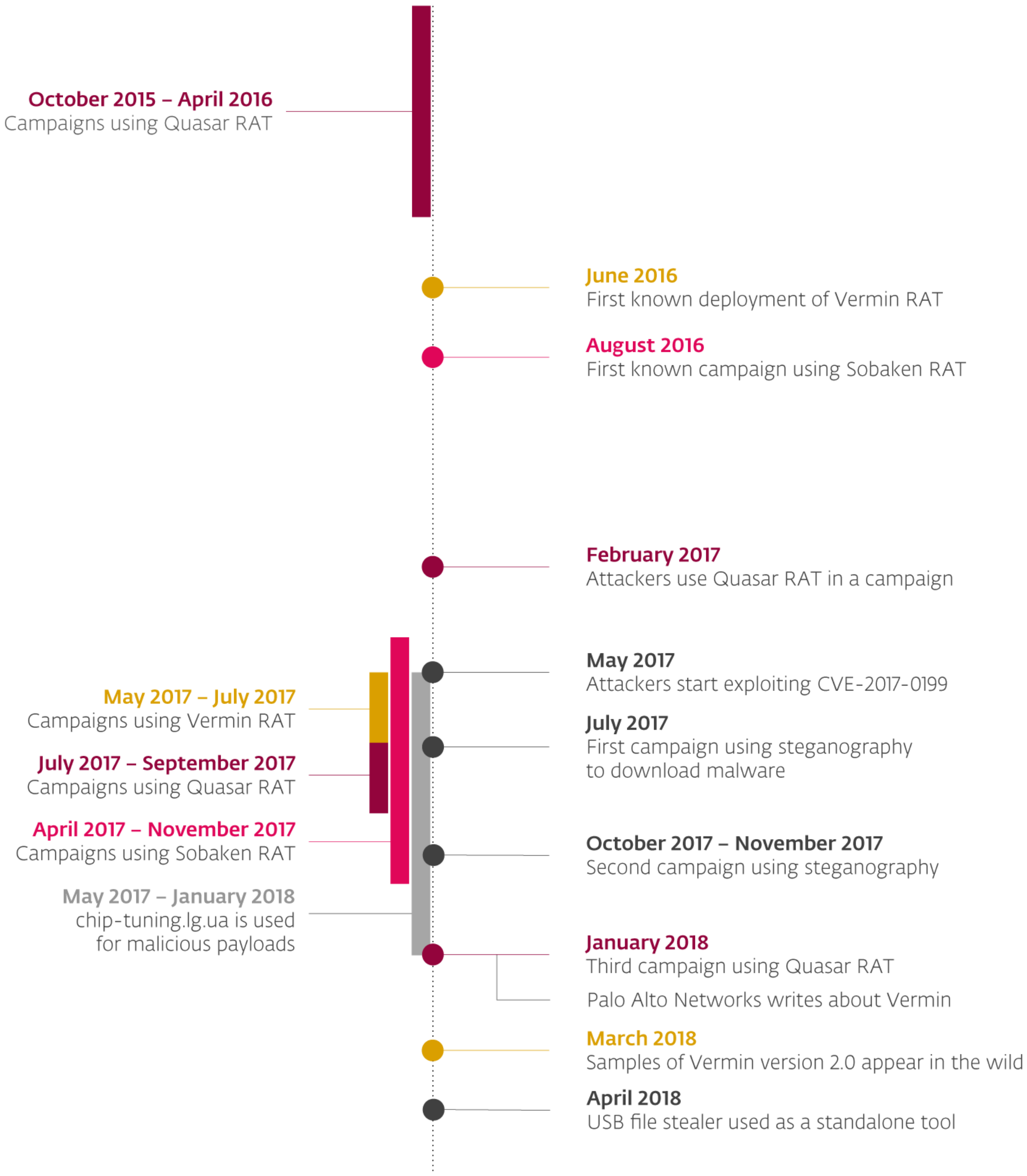

October 2015 – April 2016
Campaigns using Quasar RAT
June 2016
First known deployment of Vermin RAT
August 2016
First known campaign using Sobaken RAT
February 2017
Attackers use Quasar RAT in a campaign
May 2017
Attackers start exploiting CVE-2017-0199
May 2017 – July 2017
Campaigns using Vermin RAT
July 2017
First campaign using steganography to download malware
July 2017 – September 2017
Campaigns using Quasar RAT
April 2017 – November 2017
Campaigns using Sobaken RAT
October 2017 – November 2017
Second campaign using steganography
May 2017 – January 2018
chip-tuning.lg.ua is used for malicious payloads
January 2018
Third campaign using Quasar RAT
Palo Alto Networks writes about Vermin
March 2018
Samples of Vermin version 2.0 appear in the wild
April 2018
USB file stealer used as a standalone tool

Figure 2 // Timeline of these ongoing campaigns.

DISTRIBUTION METHODS

According to our telemetry, the attackers have been using email as the primary distribution channel for all three strains of malware. They have been quite successful in using social engineering to lure victims into downloading and executing the malware.

In most cases, filenames are in Ukrainian and refer to specific topics likely to be relevant to victims' occupations.

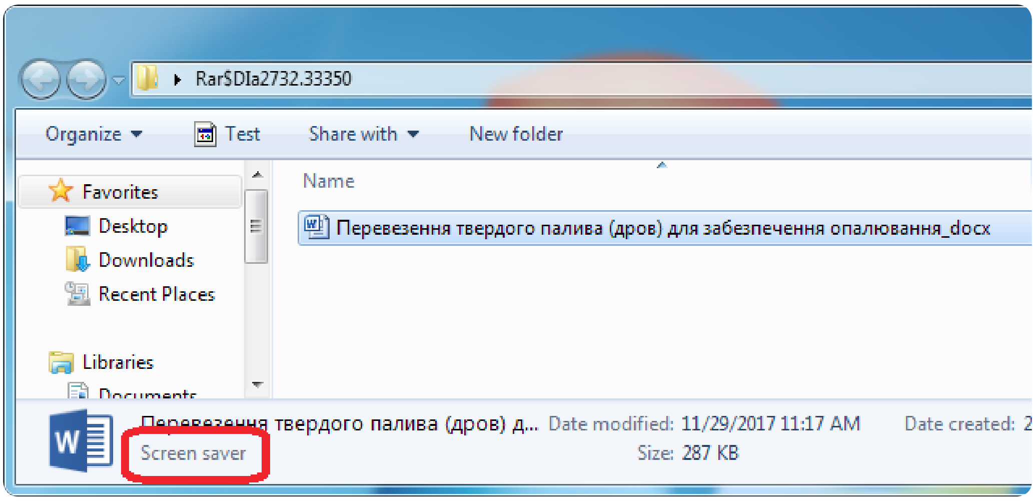
Examples of such filenames:

- “ІНСТРУКЦІЯ з організації забезпечення військовослужбовців Збройних Сил України та членів їх сімей” (roughly translates to “Directive on providing security for military personnel of Ukranian Army and their family members”)
- “новий проекту наказу, призначення перевірки вилучення” – (translates to “A new draft of directive regarding verification of seizure“)
- “Відділення забезпечення Дон ОВК. Збільшення ліміту” – (translates to “Purchasing department Don OVK. Increase of credit limit”)

Along with the basic social engineering technique of making email attachments attractive to their intended victims, three specific technical methods have been observed in use by these attackers. These presumably further improve the effectiveness of these campaigns.

Method #1: Email attachments using Unicode *right-to-left override* in their filenames to obscure their real extension. These are actually executable files using Word, Excel, PowerPoint or Acrobat Reader icons to appear more trustworthy.

Example file name: As seen in Figure 3 “Перевезення твердого палива (дров) для забезпечення опалювання_<<RLO>>xcod.scr” (translates to “Transport of firewood in order to provide heating”) will be seen with what the unwary may take to be a .DOCX extension.



Перевезення твердого палива (дров) для забезпечення опалювання_docx

Перевезення твердого палива (дров) д... Date modified: 11/29/2017 11:17 AM Date created: 2
Screen saver Size: 287 KB

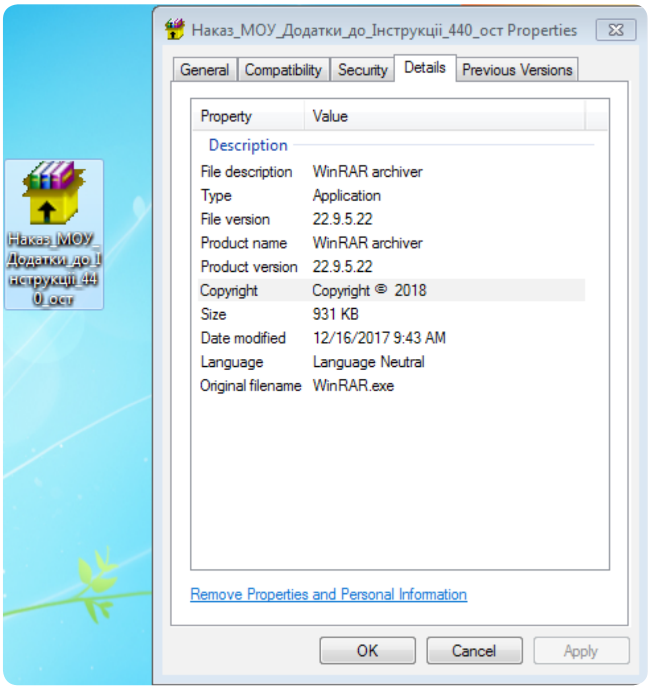
Figure 3 // Executable file disguised as a Word document.

Method #2: Email attachments disguised as RAR self-extracting archives.

Example: Email with the attachment “Наказ_МОУ_Додатки_до_Інструкції_440_ост.rar” (Translation - “Order of Ministry of Defense, Appendixes to Instruction No. 440”), as seen in Figure 4. Inside of the RAR archive, there is an executable file named “Наказ_МОУ_Додатки_до_Інструкції_440_ост.exe” that uses a RAR SFX icon. Victims would presumably run this file, thinking that further contents of a self-extracting archive would be extracted, but will inadvertently launch the malicious executable instead.

Method #3: Word document plus CVE-2017-0199 exploit. This vulnerability is triggered when the victim opens a specially crafted Word document. The Word process issues an HTTP request for an HTA file that contains a malicious script, located on a remote server. The malicious script will then be executed by `mshta.exe`. The first public information about this vulnerability *appeared in April 2017* and Microsoft fixed it by issuing Security updates for all versions of Windows and Office.

According to ESET’s telemetry, these threat actors started using this method in May 2017. Attackers used hxxp://chip-tuning.lg[.]ua/ to deliver the HTA files and the final payload.



Наказ_МОУ_Додатки_до_Інструкції_440_ост Properties

General
Compatibility
Security
Details
Previous Versions
Наказ_МОУ_Додатки_до_Інструкції_440_ост
Remove Properties and Personal Information
OK
Cancel
Apply

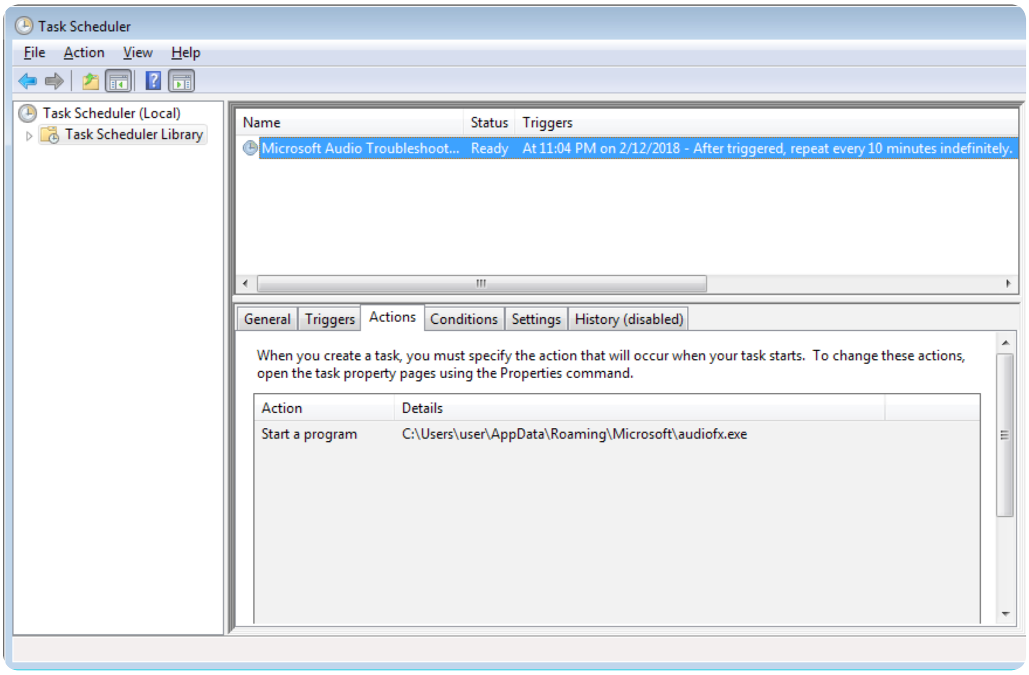
Property	Value
Description	
File description	WinRAR archiver
Type	Application
File version	22.9.5.22
Product name	WinRAR archiver
Product version	22.9.5.22
Copyright	Copyright © 2018
Size	931 KB
Date modified	12/16/2017 9:43 AM
Language	Language Neutral
Original filename	WinRAR.exe

Figure 4 // File disguised as a RAR self-extracting archive. Version information and copyright year reliably tell us that it's a fake.

INSTALLATION AND PERSISTENCE

The installation procedure is the same for all three malware strains used by these attackers. A dropper drops a malicious payload file (Vermin, Quasar or Sobaken malware) into the `%APPDATA%` folder, in a subfolder named after a legitimate company (usually Adobe, Intel or Microsoft). Then, as seen in Figure 5, it creates a scheduled task that runs the payload every 10 minutes to ensure its persistence.

Some versions also employ a trick of abusing the *Windows Control Panel* shortcut functionality to make their folders inaccessible from Windows Explorer. Such a folder will not open when clicked in Windows Explorer; instead it leads to the All Tasks page in the Windows Control Panel.



Task Scheduler

Name	Status	Triggers
Microsoft Audio Troubleshoot...	Ready	At 11:04 PM on 2/12/2018 - After triggered, repeat every 10 minutes indefinitely.

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	C:\Users\user\AppData\Roaming\Microsoft\audiofx.exe

Figure 5 // Task scheduled to run the dropped malicious payload every 10 minutes.

Examples:

`C:\Users\Admin\AppData\Roaming\Microsoft\Proof\Settings.{ED7BA470-8E54-465E-825C-99712043E01C}\TransactionBroker32.exe`

`C:\Users\Admin\AppData\Roaming\Adobe\SLStore\Setting.{ED7BA470-8E54-465E-825C-99712043E01C}\AdobeSLService.exe`

VICTIM TARGETING

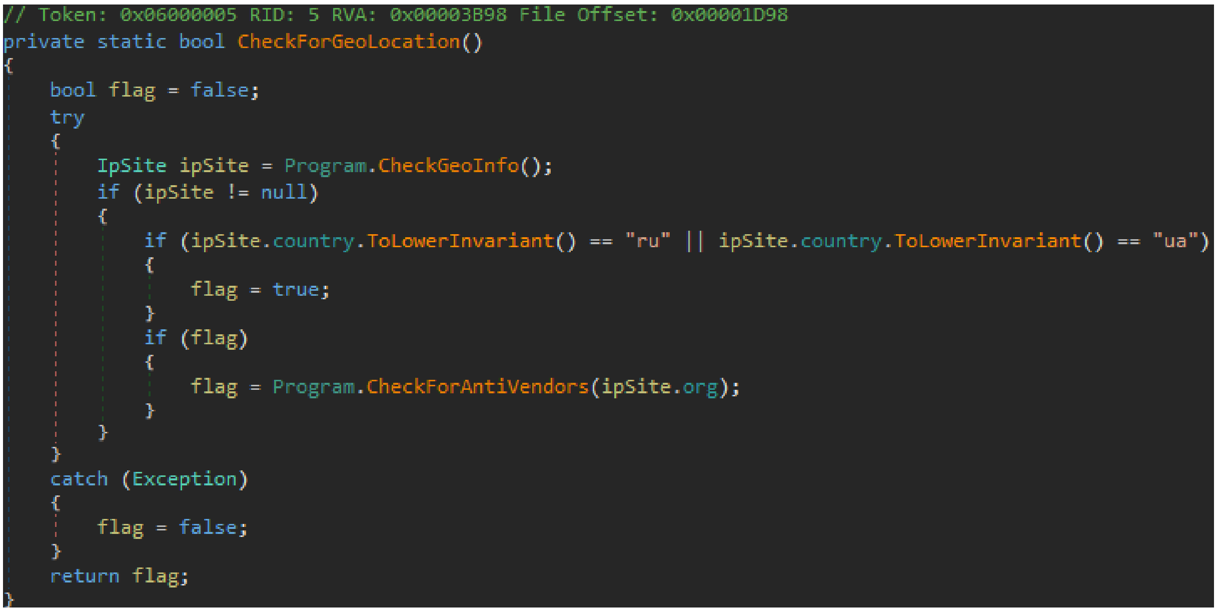
The attackers have been using quite a few tricks to ensure that the malware runs on targeted machines only, with special focus on avoiding automated analysis systems and sandboxes.

Method #1: Windows keyboard layout check.

The malware checks if Russian or Ukrainian keyboard layout is installed. If not, it terminates immediately.

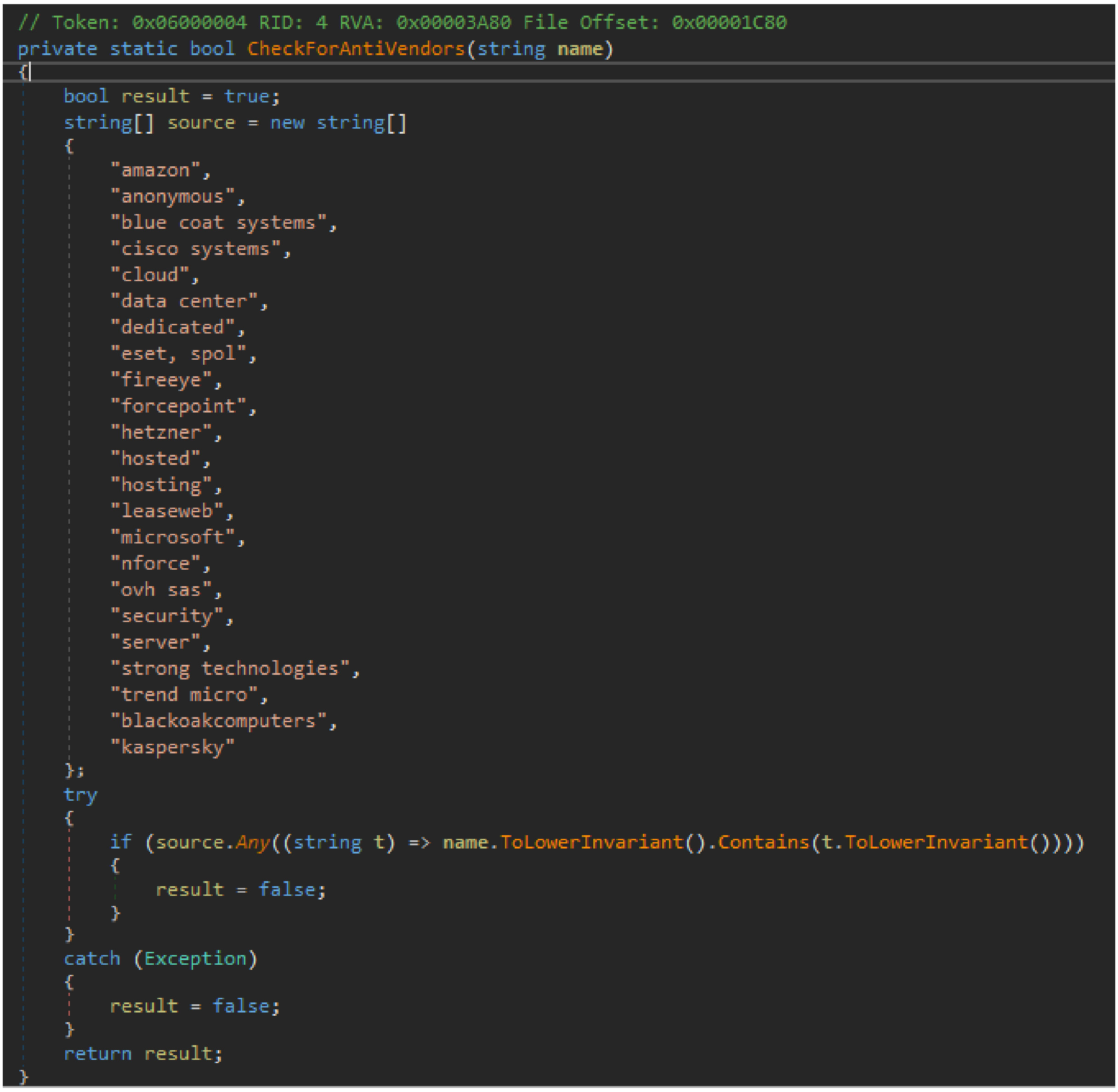
Method #2: IP address check.

The malware obtains its host computer's IP address via a request to the legitimate service ipinfo.io/json. The malware terminates if the IP address is not located in Ukraine or Russia, or if the IP address is registered to one of several selected antimalware vendors or cloud providers. Code related to these checks is seen in the disassemblies in Figures 6 and 7.



```
// Token: 0x06000005 RID: 5 RVA: 0x00003B98 File Offset: 0x00001D98
private static bool CheckForGeoLocation()
{
    bool flag = false;
    try
    {
        IpSite ipSite = Program.CheckGeoInfo();
        if (ipSite != null)
        {
            if (ipSite.country.ToLowerInvariant() == "ru" || ipSite.country.ToLowerInvariant() == "ua")
            {
                flag = true;
            }
            if (flag)
            {
                flag = Program.CheckForAntiVendors(ipSite.org);
            }
        }
    }
    catch (Exception)
    {
        flag = false;
    }
    return flag;
}
```

Figure 6 // Code checking geolocation of host IP address.

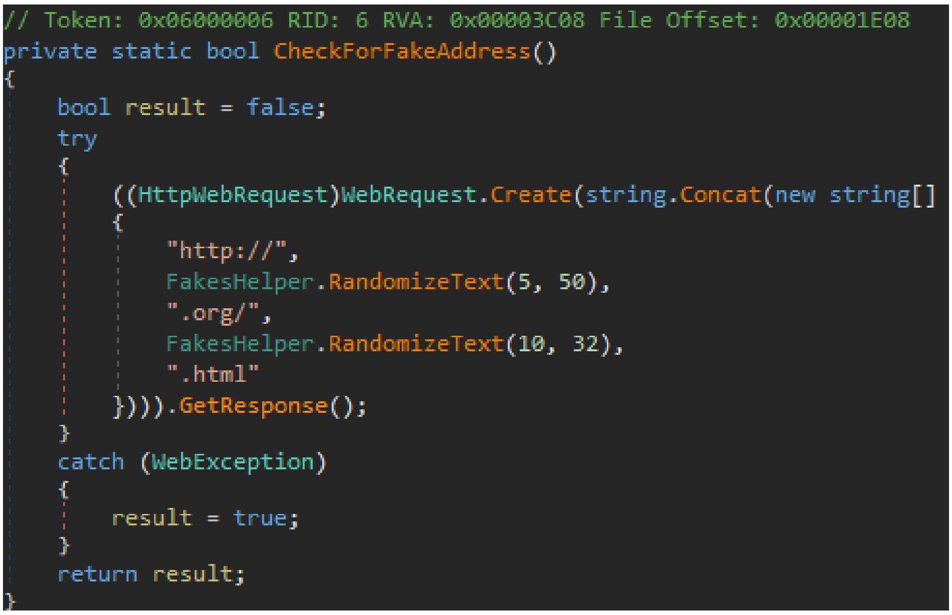


```
// Token: 0x06000004 RID: 4 RVA: 0x00003A80 File Offset: 0x00001C80
private static bool CheckForAntiVendors(string name)
{
    bool result = true;
    string[] source = new string[]
    {
        "amazon",
        "anonymous",
        "blue coat systems",
        "cisco systems",
        "cloud",
        "data center",
        "dedicated",
        "eset, spol",
        "fireeye",
        "forcepoint",
        "hetzner",
        "hosted",
        "hosting",
        "leaseweb",
        "microsoft",
        "nforce",
        "ovh sas",
        "security",
        "server",
        "strong technologies",
        "trend micro",
        "blackoakcomputers",
        "kaspersky"
    };
    try
    {
        if (source.Any((string t) => name.ToLowerInvariant().Contains(t.ToLowerInvariant())))
        {
            result = false;
        }
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 7 // Code checking IP address against a list of cloud providers and antimalware vendors.

Method #3: Emulated network environment check.

Automated analysis systems often use tools like Fakenet-NG where all DNS/HTTP communication succeeds and returns some result. Malware authors try to identify such systems by generating a random website name/URL and testing for connection to the URL to fail, such as in Figure 8, as would be expected on a real system.

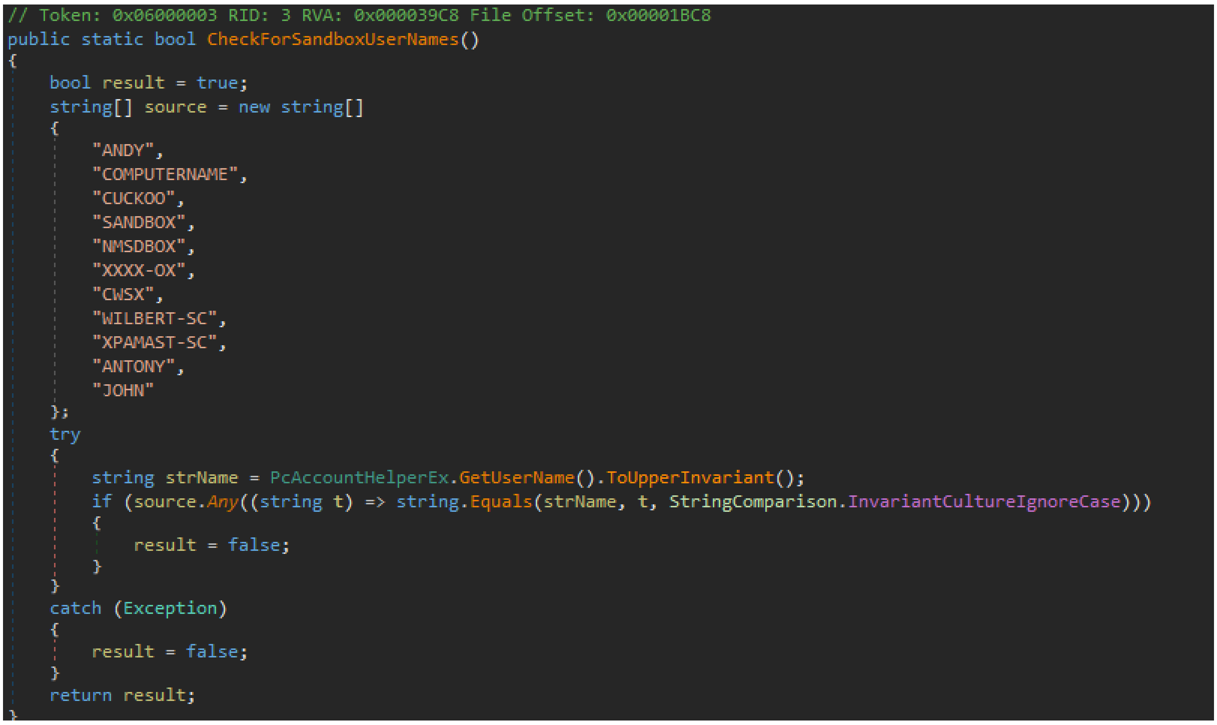


```
// Token: 0x06000006 RID: 6 RVA: 0x00003C08 File Offset: 0x00001E08
private static bool CheckForFakeAddress()
{
    bool result = false;
    try
    {
        ((HttpWebRequest)WebRequest.Create(string.Concat(new string[]
        {
            "http://",
            FakesHelper.RandomizeText(5, 50),
            ".org/",
            FakesHelper.RandomizeText(10, 32),
            ".html"
        }))).GetResponse();
    }
    catch (WebException)
    {
        result = true;
    }
    return result;
}
```

Figure 8 // Code generating random URL and attempting download.

Method #4: Specific username check.

The malware refuses to run under accounts with usernames typical of automated malware analysis systems, as seen in Figure 9.



```
// Token: 0x06000003 RID: 3 RVA: 0x000039C8 File Offset: 0x00001BC8
public static bool CheckForSandboxUserNames()
{
    bool result = true;
    string[] source = new string[]
    {
        "ANDY",
        "COMPUTERNAME",
        "CUCKOO",
        "SANDBOX",
        "NMSDBOX",
        "XXXX-OX",
        "CWSX",
        "WILBERT-SC",
        "XPAMAST-SC",
        "ANTONY",
        "JOHN"
    };
    try
    {
        string strName = PcAccountHelperEx.GetUserName().ToUpperInvariant();
        if (source.Any((string t) => string.Equals(strName, t, StringComparison.InvariantCultureIgnoreCase)))
        {
            result = false;
        }
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 9 // Checking current username against a list of known malware analysis systems.

USE OF STEGANOGRAPHY TO BYPASS CONTENT FILTERING

In mid-2017, the attackers explored the possibility of hiding payloads in images that were hosted on the free image hosting websites saveshot.net and ibb.co.

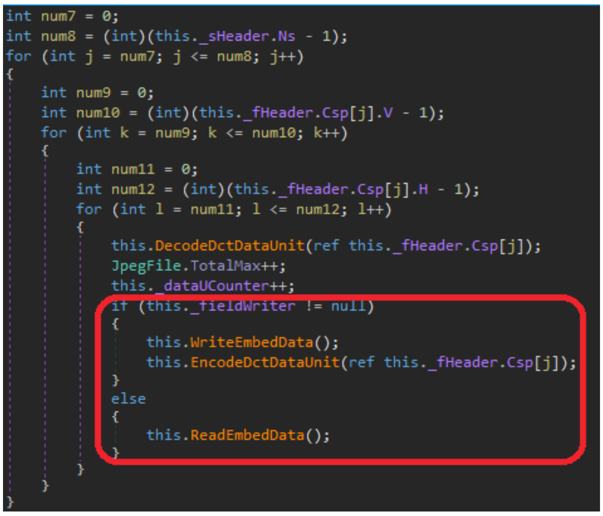
Steganography is the science of hiding data "in plain sight" – within other, non-secret data. In this case, a malicious EXE file was encrypted and hidden inside a valid JPEG file, such as the example in Figure 10. The malware downloaded and decoded the JPEG file, extracted the hidden data, decrypted the EXE file from that data, and launched it.

Figure 10 // Example of a JPEG image used for payload hiding (image resized and payload removed).

The decryption process is quite complex and can be described as follows:

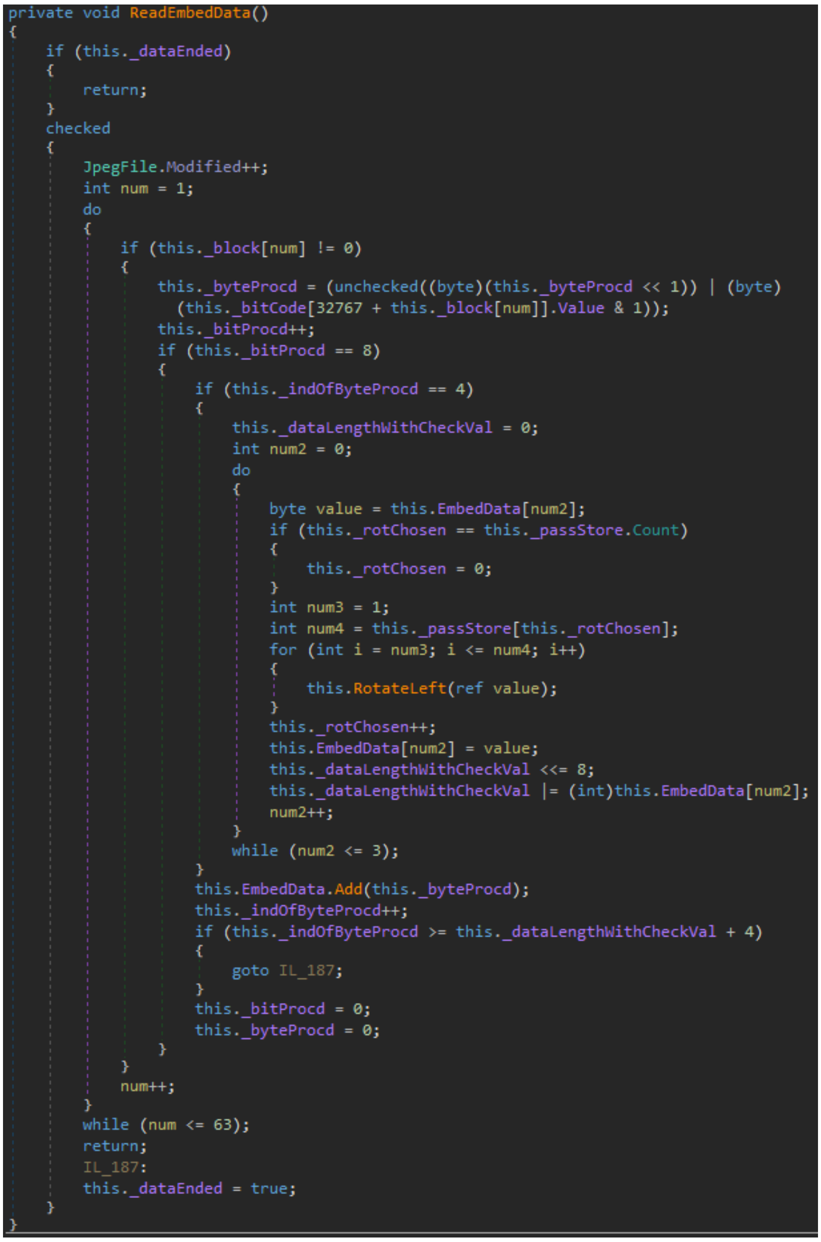
1. Download the JPEG from the URL hardcoded in the downloader binary.
2. Brute-force an 8-digit password by calculating its hash and verifying against the hash hardcoded in the downloader binary. This step is very CPU intensive and takes typical desktop computer 10+ minutes to complete. This is most likely another measure against automated malware analysis systems.
3. Process the JPEG file and extract data hidden in it, as seen in the code disassemblies in Figures 11 and 12. The algorithm used by the malware is very similar to *JSteg*, one of the oldest and simplest steganography algorithms for JPEG files that hides data in the LSB (least significant bit) of JPEG DCT coefficients. Such hidden data does not usually affect an image in a way that is visible to the naked eye, but the presence of hidden data is easily detectable with specialized algorithms. However, this steganography algorithm is very easy to implement, which is probably the reason it was chosen by the malware authors.
4. Extract data and decompress using GZip.
5. Decrypt the decompressed data using AES and the password obtained in step 2.
6. Decode decrypted data using Base64.
7. Write EXE file to disk and execute it.

Eventually, these threat authors abandoned the steganography idea and started using hxxp://chip-tuning.lg[.]ua to serve unencrypted malware executables.



```
int num7 = 0;
int num8 = (int)(this._sHeader.Ns - 1);
for (int j = num7; j <= num8; j++)
{
    int num9 = 0;
    int num10 = (int)(this._fHeader.Csp[j].V - 1);
    for (int k = num9; k <= num10; k++)
    {
        int num11 = 0;
        int num12 = (int)(this._fHeader.Csp[j].H - 1);
        for (int l = num11; l <= num12; l++)
        {
            this.DecodeDctDataUnit(ref this._fHeader.Csp[j]);
            JpegFile.TotalMax++;
            this._dataUCounter++;
            if (this._fieldWriter != null)
            {
                this.WriteEmbedData();
                this.EncodeDctDataUnit(ref this._fHeader.Csp[j]);
            }
            else
            {
                this.ReadEmbedData();
            }
        }
    }
}
```

Figure 11 // Steganography code inside JPEG decoder.



```
private void ReadEmbedData()
{
    if (this._dataEnded)
    {
        return;
    }
    checked
    {
        JpegFile.Modified++;
        int num = 1;
        do
        {
            if (this._block[num] != 0)
            {
                this._byteProcd = (unchecked((byte)(this._byteProcd << 1)) | (byte)
                  (this._bitCode[32767 + this._block[num]].Value & 1));
                this._bitProcd++;
                if (this._bitProcd == 8)
                {
                    if (this._indOfByteProcd == 4)
                    {
                        this._dataLengthWithCheckVal = 0;
                        int num2 = 0;
                        do
                        {
                            byte value = this.EmbedData[num2];
                            if (this._rotChosen == this._passStore.Count)
                            {
                                this._rotChosen = 0;
                            }
                            int num3 = 1;
                            int num4 = this._passStore[this._rotChosen];
                            for (int i = num3; i <= num4; i++)
                            {
                                this.RotateLeft(ref value);
                            }
                            this._rotChosen++;
                            this.EmbedData[num2] = value;
                            this._dataLengthWithCheckVal <<= 8;
                            this._dataLengthWithCheckVal |= (int)this.EmbedData[num2];
                            num2++;
                        }
                        while (num2 <= 3);
                    }
                    this.EmbedData.Add(this._byteProcd);
                    this._indOfByteProcd++;
                    if (this._indOfByteProcd >= this._dataLengthWithCheckVal + 4)
                    {
                        goto IL_187;
                    }
                    this._bitProcd = 0;
                    this._byteProcd = 0;
                }
            }
            num++;
        }
        while (num <= 63);
        return;
        IL_187:
        this._dataEnded = true;
    }
}
```

Figure 12 // Steganography code inside JPEG decoder.

MALWARE STRAINS

These threat actors are using three different malware strains in their attacks. We will provide a quick overview of each of them and focus on describing their unique features.

Quasar

Quasar is an open-source RAT (Remote Access Tool) which is freely available on GitHub. We've seen several campaigns where these threat actors used Quasar RAT binaries.

The first campaign we are aware of lasted from October 2015 to April 2016.

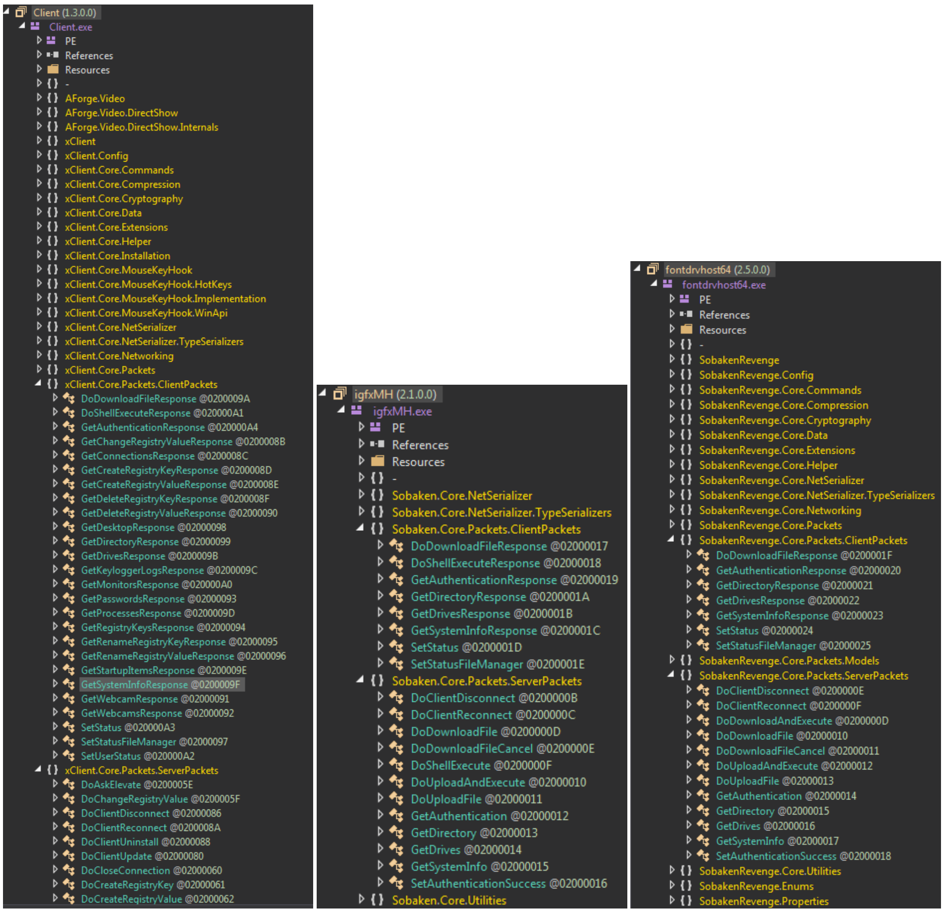
The next campaign utilizing the Quasar RAT took place in February 2017. Compilation artifacts show the PDB path `n:\projects\Viral\baybak_files_only\QRClient\QuasarRAT-master\Library\obj\Release\Library.pdb`

Another Quasar RAT campaign using these attackers' C&C servers (mailukr.net) occurred in July-September 2017. In this case, attackers used an old version of the Quasar RAT named "xRAT 2.0 RELEASE3". Compilation artifacts in the dropper show the PDB path `N:\shtorm\WinRARArchive\obj\Release\WinRAR.pdb`

Sobaken

Sobaken is a heavily modified version of the Quasar RAT. When comparing the program structure of Quasar and Sobaken, we can see quite a few similarities, such as in Figure 13.

The malware authors keep removing functionality, thus creating a much smaller executable, which is also easier to hide. They also added anti-sandbox, and other evasion, tricks described above.



```
Client (1.3.0.0)
  Client.exe
    PE
    References
    Resources
    { } -
    { } AForge.Video
    { } AForge.Video.DirectShow
    { } AForge.Video.DirectShow.Internals
    { } xClient
    { } xClient.Config
    { } xClient.Core.Commands
    { } xClient.Core.Compression
    { } xClient.Core.Cryptography
    { } xClient.Core.Data
    { } xClient.Core.Extensions
    { } xClient.Core.Helper
    { } xClient.Core.Installation
    { } xClient.Core.MouseKeyHook
    { } xClient.Core.MouseKeyHook.HotKeys
    { } xClient.Core.MouseKeyHook.Implementation
    { } xClient.Core.MouseKeyHook.WinApi
    { } xClient.Core.NetSerializer
    { } xClient.Core.NetSerializer.TypeSerializers
    { } xClient.Core.Networking
    { } xClient.Core.Packets
    { } xClient.Core.Packets.ClientPackets
      DoDownloadFileResponse @0200009A
      DoShellExecuteResponse @020000A1
      GetAuthenticationResponse @020000A4
      GetChangeRegistryValueResponse @0200008B
      GetConnectionsResponse @0200008C
      GetCreateRegistryKeyResponse @0200008D
      GetCreateRegistryValueResponse @0200008E
      GetDeleteRegistryKeyResponse @0200008F
      GetDeleteRegistryValueResponse @02000090
      GetDesktopResponse @02000098
      GetDirectoryResponse @02000099
      GetDrivesResponse @0200009B
      GetKeyloggerLogsResponse @0200009C
      GetMonitorsResponse @020000A0
      GetPasswordsResponse @02000093
      GetProcessesResponse @0200009D
      GetRegistryKeysResponse @02000094
      GetRenameRegistryKeyResponse @02000095
      GetRenameRegistryValueResponse @02000096
      GetStartupItemsResponse @0200009E
      GetSystemInfoResponse @0200009F
      GetWebcamResponse @02000091
      GetWebcamsResponse @02000092
      SetStatus @020000A3
      SetStatusFileManager @02000097
      SetUserStatus @020000A2
    { } xClient.Core.Packets.ServerPackets
      DoAskElevate @0200005E
      DoChangeRegistryValue @0200005F
      DoClientDisconnect @02000086
      DoClientReconnect @0200008A
      DoClientUninstall @02000088
      DoClientUpdate @02000080
      DoCloseConnection @02000060
      DoCreateRegistryKey @02000061
      DoCreateRegistryValue @02000062
```

```
igfxMH (2.1.0.0)
  igfxMH.exe
    PE
    References
    Resources
    { } -
    { } Sobaken.Core.NetSerializer
    { } Sobaken.Core.NetSerializer.TypeSerializers
    { } Sobaken.Core.Packets.ClientPackets
      DoDownloadFileResponse @02000017
      DoShellExecuteResponse @02000018
      GetAuthenticationResponse @02000019
      GetDirectoryResponse @0200001A
      GetDrivesResponse @0200001B
      GetSystemInfoResponse @0200001C
      SetStatus @0200001D
      SetStatusFileManager @0200001E
    { } Sobaken.Core.Packets.ServerPackets
      DoClientDisconnect @0200000B
      DoClientReconnect @0200000C
      DoDownloadFile @0200000D
      DoDownloadFileCancel @0200000E
      DoShellExecute @0200000F
      DoUploadAndExecute @02000010
      DoUploadFile @02000011
      GetAuthentication @02000012
      GetDirectory @02000013
      GetDrives @02000014
      GetSystemInfo @02000015
      SetAuthenticationSuccess @02000016
    { } Sobaken.Core.Utilities
```

```
fontdrvhost64 (2.5.0.0)
  fontdrvhost64.exe
    PE
    References
    Resources
    { } -
    { } SobakenRevenge
    { } SobakenRevenge.Config
    { } SobakenRevenge.Core.Commands
    { } SobakenRevenge.Core.Compression
    { } SobakenRevenge.Core.Cryptography
    { } SobakenRevenge.Core.Data
    { } SobakenRevenge.Core.Extensions
    { } SobakenRevenge.Core.Helper
    { } SobakenRevenge.Core.NetSerializer
    { } SobakenRevenge.Core.NetSerializer.TypeSerializers
    { } SobakenRevenge.Core.Networking
    { } SobakenRevenge.Core.Packets
    { } SobakenRevenge.Core.Packets.ClientPackets
      DoDownloadFileResponse @0200001F
      GetAuthenticationResponse @02000020
      GetDirectoryResponse @02000021
      GetDrivesResponse @02000022
      GetSystemInfoResponse @02000023
      SetStatus @02000024
      SetStatusFileManager @02000025
    { } SobakenRevenge.Core.Packets.Models
    { } SobakenRevenge.Core.Packets.ServerPackets
      DoClientDisconnect @0200000E
      DoClientReconnect @0200000F
      DoDownloadAndExecute @0200000D
      DoDownloadFile @02000010
      DoDownloadFileCancel @02000011
      DoUploadAndExecute @02000012
      DoUploadFile @02000013
      GetAuthentication @02000014
      GetDirectory @02000015
      GetDrives @02000016
      GetSystemInfo @02000017
      SetAuthenticationSuccess @02000018
    { } SobakenRevenge.Core.Utilities
    { } SobakenRevenge.Enums
    { } SobakenRevenge.Properties
```

Figure 13 // Evolution of Sobaken. Left: Quasar RAT v1.3, middle and right – 2 versions of Sobaken.

Vermin

Vermin is a custom-made backdoor that is used only by these threat actors, and was first documented in Palo Alto Networks report from *January 2018*. It first appeared in mid-2016 and is still in use. Just like Quasar and Sobaken, it is written in .NET. To slow down analysis, the program code is protected using the commercial .NET code protection system .NET Reactor or the open-source protector ConfuserEx.

Also, just like Sobaken, it uses Vitevic Assembly Embedder, free software for embedding required DLLs into the main executable, available from Visual Studio Marketplace.

Functionality

Vermin is a full-featured backdoor with several optional components. The latest known version of Vermin at the time of writing (Vermin 2.0) supports the following, self-explanatory, commands:

- StartCaptureScreen
- StopCaptureScreen
- ReadDirectory
- UploadFile
- DownloadFile
- CancelUploadFile
- CancelDownloadFile
- GetMonitors
- DeleteFiles
- ShellExec
- GetProcesses
- KillProcess
- CheckIfProcessIsRunning
- CheckIfTaskIsRunning
- RunKeyLogger
- CreateFolder
- RenameFolder
- DeleteFolder
- UpdateBot
- RenameFile
- ArchiveAndSplit
- StartAudioCapture
- StopAudioCapture
- SetMicVolume.

Most of the commands are implemented in the main payload, though several commands and additional functionality are implemented via optional components that attackers upload to the victim's machine.

Known optional components include:

- Audio recorder
- Keylogger
- Password stealer
- USB file stealer

Audio recorder (AudioManager)

This is a full-featured component of Vermin that can record audio from the microphone on the victim's computer. It implements three of Vermin's commands: StartAudioCapture, StopAudioCapture and SetMicVolume. Captured data is compressed using Speex codecs and uploaded in SOAP format to Vermin's C&C servers.

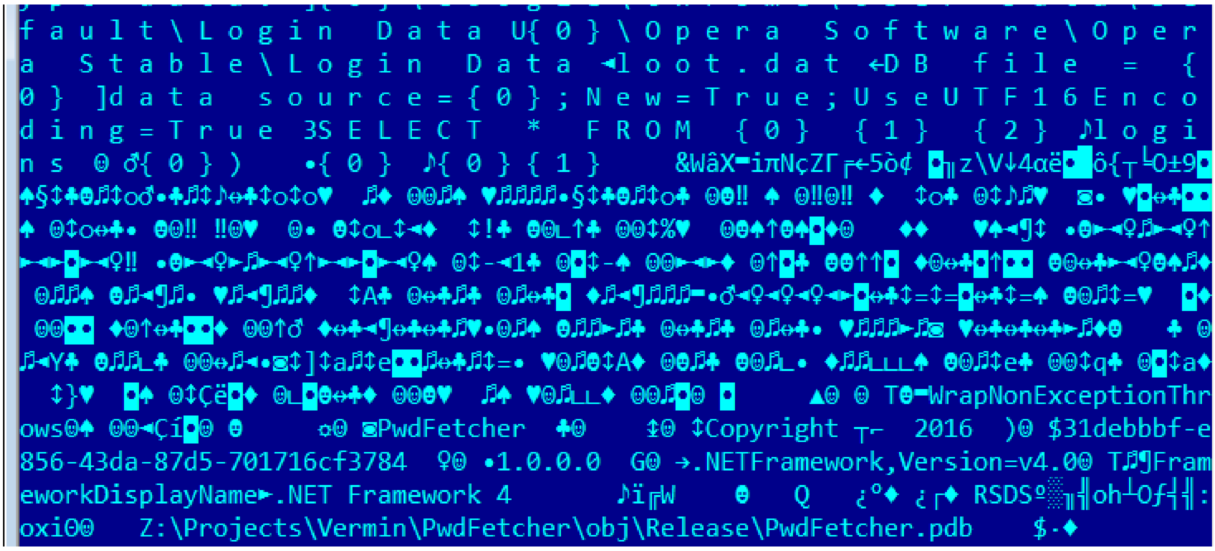
Keylogger (KeyboardHookLib)

Vermin's keylogger is a simple standalone executable that sets global keyboard hooks and writes all keystrokes into a file in an encrypted form. It also logs clipboard contents and active window titles. The keylogger cannot connect to Vermin's C&C servers by itself – the main backdoor component is used to transfer collected information to the attackers' servers.

The PDB path in the keylogger component confirms its association with the Vermin malware:
`Z:\Projects\Vermin\KeyboardHookLib\obj\Release\AdobePrintLib.pdb`

Password stealer (PwdFetcher)

Vermin's standalone password stealer is used to extract saved passwords from browsers (Chrome, Opera). The majority of its code appears to be copy-pasted from an article on the Russian forum Habrahabr. Some samples also contain code for extracting information from the Firefox browser, however, it appears to be unused. As seen in Figure 14, this component also contains PDB paths similar to that seen in the keylogger component, confirming its association with the Vermin malware.



```
f a u l t \ L o g i n   D a t a  U{ 0 } \ O p e r a   S o f t w a r e \ O p e r
a   S t a b l e \ L o g i n   D a t a  ◄l o o t . d a t  ←D B   f i l e   =   {
0 }  ]d a t a   s o u r c e = { 0 } ; N e w = T r u e ; U s e U T F 1 6 E n c o
d i n g = T r u e  3S E L E C T   *   F R O M   { 0 }   { 1 }   { 2 }  ♪l o g i
n s  ☺ ♂{ 0 } )    •{ 0 }  ♪{ 0 } { 1 }      &WâX━iπNçZΓ╒←5ò¢ ◘╖z\V↓4αë◘ ô{┬└O±9◘
♠§↕♣☻♫↕o♂•♣♫↕♪↔♣↕o↕o♥  ♫♦ ☺☻♫♠ ♥♫♫♫♫•§↕♣☻♫↕o♣ ☺☻‼ ♠ ☺‼☺‼ ♦  ↕o♣ ☺↕♪♫♥  ◙• ♥◘↔♣◘◘
♠ ☺↕o↔♣• ☺☻‼ ‼☺♥  ☺• ☻↕o└↕◄♦  ↕!♣ ☻☺└↑♣ ☺☻↕%♥  ☺☻♠↑☻♠◘♦☺   ♦♦   ♥♠◄¶↕ •☻►◄♀♫►◄♀↑
►◄►◘►◄♀‼ •☻►◄♀►♫►◄♀↑►◄►◘►◄♀♠ ☺↕-◄1♣ ☺◘↕-♠ ☺☺►◄►♦ ☺↑◘♣ ☻☻↑↑◘ ♦☻↔♣◘↑◘◘ ☻☺↔♣►◄♀☻♠♫♦
 ☺♫♫♠ ☻♫◄¶♫• ♥♫◄¶♫♫♦  ↕A♣ ☺↔♣♫♣ ☺♫↔♣◘ ♦♫◄¶♫♫♫━•♂◄♀◄♀◄♀◄►◘↔♣↕=↕=◘↔♣↕=♠ ☻☺♫↕=♥  ◘♦
 ☺☺◘◘ ♦☺↑↔♣◘◘♦ ☺☺↑♂ ♦↔♣◄¶↔♣↔♣♫♥•☺♫♠ ☻♫♫►♫♣ ☺↔♣♫♣ ☺♫↔♣• ♥♫♫♫►♫◙ ♥↔♣↔♣↔♣►♫♦☻   ♣ ☺
♫◄Y♣ ☻♫♫└♣ ☺☺↔♫◄•◙↕]↕a♫↕e◘◘♫↔♣♫↕=• ♥☺♫☻↕A♦ ☺☻♫♣ ☻☺♫└• ♦♫♫└└└♠ ☻☺♫↕e♣ ☺☻↕q♣ ☺◘↕a♦
  ↕}♥  ◘♠ ☺↕Çë◘♦ ☺└◘☻↔♣♦ ☺☺☻♥  ♫♠ ♥☺♫└└♦ ☺☺♫◘☺ ◘      ▲☺ ☺ T☻━WrapNonExceptionThr
ows☺♠ ☺☺◄Çí◘☺ ☻      ☼☺ ◙PwdFetcher  ♣☺    ↨☺ ↕Copyright ┬╓  2016  )☺ $31debbbf-e
856-43da-87d5-701716cf3784  ♀☺ •1.0.0.0  G☺ →.NETFramework,Version=v4.0☺ T♫¶Fram
eworkDisplayName►.NET Framework 4       ♪ï╓W    ☻    Q    ¿º♦ ¿┌♦ RSDSº▒╖╢oh┴Oƒ╡╢:
oxi0☺   Z:\Projects\Vermin\PwdFetcher\obj\Release\PwdFetcher.pdb    $·♦
```

Figure 14 // Compilation artifacts linking the password stealer to the Vermin malware.

USB file stealer (UsbGuard)

`UsbGuard.exe` is an optional component used by both Sobaken and Vermin. It is a small, standalone program that monitors USB drives connected to the computer and copies all files that match the filter chosen by the attackers. The stolen files were later exfiltrated using the main backdoor module. Many and various PDB paths clearly linking them to Vermin are found in samples of this component.

Since April 2018, the file stealer has been used as a standalone tool. It copies files and immediately uploads them to a server controlled by the attackers.

In the analyzed samples, attackers were after files with the following extensions:

- .doc
- .docx
- .xls
- .xlsx
- .zip
- .rar
- .7z
- .docm
- .txt
- .rtf
- .xlsm
- .pdf
- .jpg
- .jpeg
- .tif
- .odt
- .ods

CONCLUSION

Among the many different malware attacks targeted at high value assets in Ukraine, these attackers haven't received much public attention – perhaps because of their initial use of open-source-based malware before developing their own strain (Vermin).

Employing multiple malware families, as well as various infection mechanisms – including common social engineering techniques but also not-so-common steganography – over the past three years, could be explained by the attackers simply experimenting with various techniques and malware, or it may suggest operations by multiple subgroups.

The fact that the attackers successfully used relatively trivial techniques, such as sending RAR and EXE files by email (a bad practice, which still takes place among users) highlights the importance of securing the human factor in computer network protection.

IOCs

C&Cs

Sobaken C&C

- akamaicdn.ru
- akamainet021.info
- cdnakamai.ru
- windowsupdate.kiev.ua
- akamainet022.info
- akamainet066.info
- akamainet067.info
- notifymail.ru
- mailukr.net

- 188.227.16.73
- 212.116.121.46
- 206.54.179.160

Quasar C&C

- 188.227.75.189
- mailukr.net
- cdnakamai.ru
- notifymail.ru

Vermin C&C

- 185.158.153.222
- 188.227.17.68
- 195.78.105.23
- tech-adobe.dyndns.biz
- notifymail.ru
- akamainet023.info
- mailukr.net
- 185.125.46.24
- akamainet024.info
- 206.54.179.196

Payload delivery, data exfiltration

- chip-tuning.lg.ua
- www.chip-tuning.lg.ua
- olx.website
- news24ua.info
- rst.website
- 1ua.eu
- novaposhta.website

SHA1 hashes of files associated with this threat actor

Vermin

- 028EBDBEBAC7239B41A1F4CE8D2CC61B1E09983B
- 07E1AF6D3F7B42D2E26DF12A217DEBACEDB8B1B9
- 09457ACB28C754AA419AB6A7E8B1940201EF3FFE
- 0EEE92EC2723ED9623F84082DAD962778F4CF776
- 10128AB8770FBDECD81B8894208A760A3C266D78
- 131F99A2E18A358B60F09FD61EE312E74B02C07C
- 14F69C7BFAF1DF16E755CCF754017089238B0E7B
- 1509F85DE302BE83A47D5AFAD9BEE2542BA317FC
- 170CEE6523B6620124F52201D943D7D9CA7B95E5
- 191159F855A0E580290871C945245E3597A5F25C
- 1F12C32A41D82E978DE333CD4E93FDAA1396BE94
- 22B17966B597568DB46B94B253CD37CBCF561321
- 2C7332D8247376842BD1B1BD5298844307649C99

- 2E08BA5DF30C0718C1733A7836B5F4D98D84905E
- 2EDF808F8252A4CBCB92F47A0AEDC1AAAE79A777
- 360F54B33AC960EE29CA0557A28F6BB8417EF409
- 431FCE6A47D0A48A57F699AA084C9FF175A9D15F
- 45438834FDC5C690DA3BC1F60722BE86B871280D
- 4A8A8188E3A7A137651B24780DF37CB6F610CC19
- 4C1E4E136B7922F9E28D1B38E9760E28929E4F0B
- 5B6EA57FFC09593C3B65D903368EA5F7FAA2EB68
- 61D366939FE36861B2FECB38A4DFF6D86C925A00
- 6A72366D8AE09F72F0466FB59E8ED372F8B460D7
- 6FECA622B0FB282064F7DE42BA472A8EC908D0D6
- 70A772485C5ED330C6876FA901BA722CD44CA05E
- 70D97367A3DBD5D45482B6AF8C78C58B64D3F3B3
- 7803FD9753930522705F2B6B4E73622887892C28
- 7B11A84B18DC4B5F1F2826E7925F0B2DC1B936AE
- 889FD0BEB3197DDD6C88F5C40D6B8E4D74A892CE
- 9B6FBABFA2A77FA633F7A2EB352979D5C68CEBC6
- A451291F17489E3A59F440A1B693D691B053C531
- A53D77E55A06CF131D670339BACEC5AC0F0C6D66
- A925D0AFB5D4F5FAC65543C993BE4172F1DBF329
- B5F81C804E47B76C74C38DF03A5CBE8A4FE69A9A
- B99DE55043099E9506B304660B8E1374787AB195
- C00C104FC3E9F5977D11C67EF0C8C671D4DFC412
- CA0296FA9F48E83EA3F26988401B3F4C4E655F7A
- D4C6540E789BD3839D65E7EDA5CCA8832493649E
- D5EDE1BBB9A12757E24BE283AFC8D746ADC4A0D4
- DEFBFD98C74BEFF839EEB189F0F6C385AD6BA19B
- ECF152EB6417A069573F2C7D9A35B9CC31EC8F56
- EE2D40825C77C8DFEF67999F0C521919E6672A10
- EF09AC6BA08A116F2C4080CBEE8CEF9523E21265
- F414C49CF502D1B6CC46E08F3AC97D7846B30732

Sobaken

- 087F77998004207BCCFFBF3030B6789648930FA5
- 0A4A2BCB3EF4E19973D5C4BE4E141B665CC0BFE0
- 1CEEF0813C0F096E6DA5461DC4B3BF901C500C56
- 293DBFF0230DAB3C4C21428F90C8EF06E9F35608
- 37E2947BFB5FC0839087C5BCE194EC193F824C85
- 39525CBCA591F2A10946BA62A56E4C3382CD4FC0
- 3CE0A18E9A8A2B95827008DBFF16364B6FEDF361
- 3E869038080DAE006FF6B20DF9B0CD9CB3A5E1A1
- 400830AB6DD46789B00D081ADF0F82623472FB13
- 43F382A330A454FF83F4F35FB571ECF587A4694A
- 4449FBE2B28A81B760B284880ADBED43462C2030
- 4712AF28168FD728A13EFD520E0665FFD076B6FB
- 4F504D7B35660943B206D6034752C686365EA58D

- 53239A62E09BB0B4E49B7954D533258FEF3342C4
- 540292753FA0CC4ACB49E5F11FEDEA4B7DEF11D8
- 5589E8018DC7F934A8FDAB62670C9140AF31CAB6
- 57BBA7D8786D3B0C5F93BC20AB505DF3F69C72D4
- 630FE59D60F6882A0B9E35ED606BF06AD4BA048C
- 63EA7C844D86882F491812813AAAD746738A6BE9
- 64121FA2FD2E38AC85A911A9F7ADD8CA1E1A9820
- 64DBA711FDD52FECF534CAC0C6FE8848FE36F196
- 650AB5E674FEF431EBC8CF98141506DDC80C5E64
- 6EF13E9D5B0B6FCB5EB2A7439AAD7B21EA7FB7AC
- 7177F64362A504F3DF8AA815CEF7136D5A819C04
- 9B91EC03A09C4CF6DBEC637B3551BDCA11F04A9B
- A26764AFB1DAC34CAA2123F7BF3543D385147024
- A55319D3DBD7B9A587F5156CF201C327C803FBC9
- A841FF1EE379269F00261337A043448D3D72E6FD
- AAB5BAAAE8A2577E1036769F0D349F553E4D129B
- ACB989B3401780999474C5B1D7F9198ECA11549A
- B65372E41E7761A68AEF87001BBB698D8D8D5EC6
- BDB5E0B6CA0AA03E0BECA23B46A8420473091DFF
- C4421084C19423D311A94D7BB6CB0169C44CBECD
- C7E76993BB419DC755BD0C04255AB88E6C77B294
- CF5238C467EBE2704528EED18AB4259BFDC604E3
- D2334E161A1720E2DF048E4366150729B9395144
- D35FB6E031720876482E728A40532703EF02A305
- D82DF2903AA4BC5FD4274B5D1BFAF9E081771628
- E4B3CBCA9A53B7B93177A270C2A76F981D157C34
- E585AA2C5BFB9D42D2E58DB3833330D056713B9A
- F4A485696FC871307C22906701CBBB3FA522499B
- F5C75450108440D0BC9E7B210F072EF25A196D20

Quasar

- 0A4915B81D9A9ACF4E19181DEEEBBE244430C16B
- 323160C88A254127D9ADB2848AE044AFFF376A4D
- 395166835495B418773C9690227779D592F94F71
- 3EE410DD50FC64F39DFF0C4EE8CC676F0F7D5A74
- 5B665152F6596D4412267F9C490878455BA235F9
- 5FE8558EB8A3C244BE2DA8BE750221B9A9EE8539
- 61CB5E535F0AC90A1F904EC9937298F50E2B4974
- 6A1CD05F07B1024287CEA400237E1EA9D2FE1678
- 7676AFF05A3550E5BBFF78CF4D10C9E094447D72
- 86165F464EC1912A43445D80559D65C165E2CF76
- AB3CD05BE6B0BA8567B84D10EDE28ABF87E115AC
- BFD7158E1C2F6BA525E24F85ED8CCF8EF40FD370
- CFEBEFC92DCDF1687FD0BC1B50457EBDEA8672A2
- D21B8514990B0CEAC5EAE687DEAA60B447139B9D

Steganography

- 04DA3E81684E4963ABEC4C0F6D56DF9F00D2EF26
- 3C618A0C4BF4D3D24C9F2A84D191FC296ED22FA4
- 746155881D5AB2635566399ACC89E43F6F3DA91A
- CADBC40A4EFB10F4E9BD8F4EC3742FA8C37F4231
- E22CE72406B14EF32A469569FBE77839B56F2D69

HTA files

- 39F5B17471FD839CC6108266826A4AD8F6ECD6A3
- 751FBD034D63A5E0A3CA64F55045AE24E575384A
- 76433D1D13DF60EC0461ED6D8007A95C7A163FF9
- 89DF6A7551B00969E22DC1CAE7147447ACA10988
- D6D148050F03F5B14681A1BBF457572B9401B664

AudioManager

- 1F49946CA2CE51DC51615000BAA63F6C5A9961F1
- 98F62C2E6045D5A15D33C8383ADACF9232E5FBE3
- E7C4A69EBD7B41A6AF914DD3D3F64E1AA1ABE9B4
- F233A0F2997BB554D4F1A4B7AC77DAE4180850FA

Keylogger

- 21921864D2F1AB2761C36031A2E1D2C00C9B304A
- 3C2D0615BEF6F88FED6E308D4F45B6133080C74F
- 91E8346910E0E6783ACFC4F2B9A745C81BD7573A

PwdFetcher

- 2A5C9D4DAE5E53B2962FBE2B7FA8798A127BC9A6
- 9B1586766AF9885EF960F05F8606D1230B36AC15
- A2F0D5AF81D93752CFF1CF1E8BB9E6CAEE6D1B5E
- CE18467B33161E39C36FC6C5B52F68D49ABCFC2A

USBGuard

- 050EB7D20EE8EF1E1DAEE2F421E5BF648FB645DF
- 069A919B3BC8070BB2D71D3E1AD9F7642D8ECF0F
- 0D265E0BDA9DF83815759ABCA64938EC0FF65733
- 0D7DF910D0FB7B100F084BFB8DFA0A9F2371171A
- 2FF3F5DA2960BE95E50B751680F450896AD1ED67
- 3200ECC7503F184F72AB9DA1DC3E1F8D43DDFD48
- 46D256EF277328E803D2B15CA7C188267059949D
- 524EE1B7269D02F725E55254A015200BB472463A
- 53A0EFD3D448DA8E32CFDDA5848312D3CF802B06
- 6FC150A9CAFA75813E7473C687935E7E4A5DCE24
- 70559245303F99630A27CB47B328C20C9666F0BB
- 7D8044A5CBEFE3B016F2132A5750C30BB647E599
- 8FD919D531A7A80615517E1AC13C2D0F050AF20D
- 9D22421DA9696B535C708178C72323F64D31FC80
- BFD2DFA3D6AF31DF4B9CC2F6B31B239ADF1CECA1

- C08A6222B59A187F3CF27A7BAE4CACFACC97DDEE
- C2F6A65E14605828880927B9BA3C386507BD8161
- C562006D2FA53B15052A4B80C94B86355CCA7427
- CB43058D9EBB517832DF7058641AEDF6B303E736
- CC8A9C28E884FDA0E1B3F6CEAB12805FEA17D3C1
- D3CC27CA772E30C6260C5A3B6309D27F08A295CD
- E7A2DE3776BA7D939711E620C7D6AB25946C9881
- EE6EFA7A6A85A1B2FA6351787A1612F060086320
- EF0ABB3A0CD1E65B33C0F109DD18F156FC0F0CDE
- F63BE193C8A0FBB430F3B88CC8194D755BAD9CD1

ESET detection names

Most packed files were covered automatically by ESET’s generic detections. Detections directly related to the majority of files in the campaign:

- MSIL/Agent.AWB
- MSIL/Agent.AZG
- MSIL/Agent.AZJ
- MSIL/Agent.AZX
- MSIL/Agent.BCH
- MSIL/Agent.BCV
- MSIL/Agent.BCY
- MSIL/Agent.BFT
- MSIL/Agent.BGB
- MSIL/Agent.BGC
- MSIL/Agent.BGE
- MSIL/Agent.BGM
- MSIL/Agent.BJU
- MSIL/Agent.SCM
- MSIL/Spy.Agent.BBB
- MSIL/Spy.Agent.BIF
- MSIL/TrojanDownloader.Agent.DYV
- MSIL/TrojanDownloader.Small.BBM
- MSIL/TrojanDropper.Agent.DBE
- MSIL/TrojanDropper.Agent.DJQ
- MSIL/TrojanDropper.Agent.DJR